Introduction to Signal Processing, 2/e

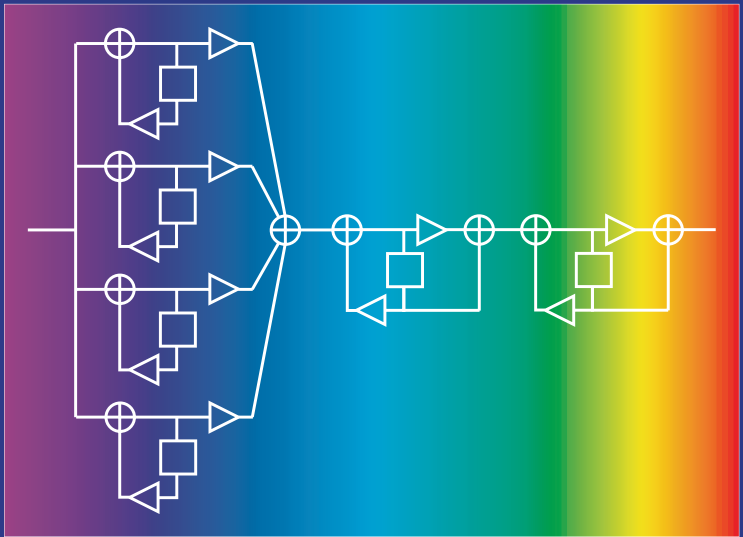
Sophocles J. Orfanidis

sampling and reconstruction
quantization
discrete-time systems
convolution
FIR filtering and circular delay-line buffers
z-transforms
transfer functions
digital filter realizations
lattice filters
DTFT and spectral analysis
DFT/FFT algorithms
FIR and IIR digital filter design
elliptic filter design
interpolation, decimation, oversampling
noise-shaping delta-sigma quantizers
noise reduction, signal enhancement
digital audio effects
high-order parametric equalizers
STFT and phase vocoder
DCT, MDCT, data compression
discrete wavelet transforms
discretization methods
control systems
local polynomial filters
exponential smoothing
filtering methods in financial markets
Whittaker-Henderson smoothing
sparse modeling
periodic signal extraction
neural networks
prolate spheroidal wave functions

www.ece.rutgers.edu/~orfanidi/intro2sp/2e

Orfanidis Introduction to Signal Processing 2023

Introduction to Signal Processing



Second Edition - 2023

Sophocles J. Orfanidis

Introduction to Signal Processing

Second Edition

Introduction to Signal Processing

Second Edition

Sophocles J. Orfanidis

Rutgers University

`http://www.ece.rutgers.edu/~orfanidi/intro2sp/2e`

To the memory of my dearest friend George Lazos
and to my family, Monica, John, Anna, and Owen

This book was previously published by Pearson Education, Inc.
Copyright © 1996–2009 by Prentice Hall, Inc. Previous ISBN 0-13-209172-0.

<u>Software tools:</u>

MATLAB® is a registered trademark of The MathWorks, Inc.

I2SP Toolbox – © S. J. Orfanidis 2010 – `www.ece.rutgers.edu/~orfanidi/intro2sp`
EWA Toolbox – © S. J. Orfanidis 2016 – `www.ece.rutgers.edu/~orfanidi/ewa`
AOSP Toolbox – © S. J. Orfanidis 2018 – `www.ece.rutgers.edu/~orfanidi/aosp`

CVX software by Michael Grant and Stephen Boyd, *CVX: Matlab software for disciplined convex programming*, version 2.0 beta, September 2013, `http://cvxr.com/cvx`.

<u>Audio files:</u>

Some example audio files were used from the following resources:

1. Robin N. Strickland, ECE 429/529, Digital Signal Processing course, Spring 2009.
   `https://uweb.engr.arizona.edu/~429rns/audiofiles/audiofiles.html`
   `https://uweb.engr.arizona.edu/~429rns/index.htm`

2. Udo Zölzer (ed.), *DAFX - Digital Audio Effects*, ISBN: 0-471-49078-4, Wiley, 2002.
   `https://dafx.de/DAFX_Book_Page/matlab.html`

Contents

Preface xv

Part I Basics

1 Sampling and Reconstruction 1

1.1 Introduction, 1
1.2 Review of Analog Signals, 1
1.3 Sampling Theorem, 4
1.3.1 Sampling Theorem, 5
1.3.2 Antialiasing Prefilters, 7
1.3.3 Hardware Limits, 8
1.4 Sampling of Sinusoids, 9
1.4.1 Analog Reconstruction and Aliasing, 10
1.4.2 Rotational Motion, 26
1.4.3 DSP Frequency Units, 28
1.5 Spectra of Sampled Signals, 29
1.5.1 Discrete-Time Fourier Transform, 30
1.5.2 Spectrum Replication, 32
1.5.3 Practical Antialiasing Prefilters, 38
1.6 Analog Reconstructors, 42
1.6.1 Ideal Reconstructors, 43
1.6.2 Staircase Reconstructors, 44
1.6.3 Anti-Image Postfilters, 45
1.7 Basic Components of DSP Systems, 53
1.8 Theory of Bandlimited Functions, 55
1.9 Problems, 56

2 Quantization 62

2.1 Quantization Process, 62
2.2 Oversampling and Noise Shaping, 66
2.3 D/A Converters, 71
2.4 A/D Converters, 76
2.5 Analog and Digital Dither, 84
2.6 Problems, 90

3 Discrete-Time Systems 95

3.1 Input/Output Rules, 96
3.2 Linearity and Time Invariance, 100
3.3 Impulse Response, 102
3.4 FIR and IIR Filters, 104
3.5 Causality and Stability, 111
3.6 Problems, 116

4 FIR Filtering and Convolution 120

4.1 Block Processing Methods, 121
4.1.1 Convolution, 121
4.1.2 Direct Form, 122
4.1.3 Convolution Table, 124
4.1.4 LTI Form, 126
4.1.5 Matrix Forms, 128
4.1.6 Flip-and-Slide Form, 130
4.1.7 Transient and Steady-State Behavior, 131
4.1.8 Convolution of Infinite Sequences, 133
4.1.9 Programming Considerations, 137
4.1.10 Overlap-Add Block Convolution Method, 141
4.2 Numerical Evaluation of Continuous-Time Convolution, 145
4.2.1 Computer Experiment – Numerical Approximation, 145
4.2.2 Computer Experiment – Transient and Steady-State Behavior, 149
4.3 Sample Processing Methods, 154
4.3.1 Pure Delays, 154
4.3.2 FIR Filtering in Direct Form, 159
4.3.3 Programming Considerations, 167
4.3.4 Hardware Realizations and Circular Buffers, 169
4.4 Problems, 185

5 z-Transforms 190

5.1 Basic Properties, 190
5.2 Region of Convergence, 193
5.3 Causality and Stability, 199
5.4 Frequency Spectrum, 202
5.5 Inverse z-Transforms, 208
5.6 Unilateral z-Transform, 215
5.7 Problems, 220

6 Transfer Functions 224

6.1 Equivalent Descriptions of Digital Filters, 224
6.2 Transfer Functions, 225
6.3 Sinusoidal Response, 239
6.3.1 Steady-State Response, 239
6.3.2 Transient Response, 242
6.4 Pole/Zero Designs, 252
6.4.1 First-Order Filters, 252

6.4.2 Parametric Resonators and Equalizers, 254
6.4.3 Notch and Comb Filters, 259
6.5 Deconvolution, Inverse Filters, and Stability, 264
6.6 Problems, 269

7 Digital Filter Realizations 275

7.1 Direct Form, 275
7.2 Canonical Form, 281
7.3 Transposed Form, 287
7.4 State-Space Realizations, 290
7.5 Cascade Form, 294
7.6 Cascade to Canonical, 301
7.7 Hardware Realizations and Circular Buffers, 310
7.8 Problems, 322

8 Lattice Realizations 331

8.1 Overview, 331
8.2 Applications of Lattice Structures, 332
8.3 Standard, Rearranged, and Normalized Lattice, 333
8.4 Direct to/from Lattice Transformations, 343
8.5 Filtering in Lattice Realizations, 344
8.6 Frequency Response of Lattice Forms, 346
8.7 Schur-Cohn Stability Test, 346
8.8 Lattice Filter Examples, 347
8.9 Quantization Effects in Digital Filters, 353
8.10 Computer Experiments – Coefficient Quantization Effects, 355

9 DTFT and Spectral Analysis 360

9.1 Frequency Resolution and Windowing, 360
9.2 DTFT Computation, 372
9.3 Window Parameters, 374
9.4 Additional Details on Windows, 382
9.4.1 Rectangular Window, 382
9.4.2 Hamming Window, 386
9.4.3 Kaiser Window, 387
9.4.4 DPSS Window, 388
9.4.5 Chebyshev Window, 390
9.5 Fourier Optics, Apertures, Spatial Arrays, 393
9.6 Periodogram and Its Improvements, 395
9.7 Filtering of Random Signals, 401
9.8 Computer Experiment – Sunspot Time Series, 403
9.9 Problems, 407

10 DFT/FFT Algorithms 410

10.1 Discrete Fourier Transform, 410
10.2 Zero Padding, 413
10.3 Physical versus Computational Resolution, 414

10.4 Matrix Form of DFT, 418
10.5 Modulo-*N* Reduction, 421
10.6 Inverse DFT, 429
10.7 Sampling of Periodic Signals and the DFT, 432
10.8 FFT, 436
10.9 Fast Convolution, 449
10.10 Circular Convolution, 449
10.11 Overlap-Add and Overlap-Save Methods, 453
10.12 Computer Experiment – Fast Convolution, 457
10.13 Computer Experiment – Matched Filtering, 460
10.14 Problems, 462

11 FIR Digital Filter Design 469

11.1 Window Method, 469
11.1.1 Ideal Filters, 469
11.1.2 Rectangular Window, 472
11.1.3 Hamming Window, 477
11.2 Gibbs Phenomenon, 477
11.3 Kaiser Window, 481
11.3.1 Kaiser Window for Filter Design, 481
11.3.2 Kaiser Window for Spectral Analysis, 495
11.4 Frequency Sampling Method, 498
11.5 Other FIR Design Methods, 499
11.6 Problems, 499

12 IIR Digital Filter Design 504

12.1 Bilinear Transformation, 504
12.2 First-Order Lowpass and Highpass Filters, 507
12.3 Second-Order Peaking and Notching Filters, 514
12.4 Parametric Equalizer Filters, 523
12.4.1 Shelving Equalizers, 530
12.4.2 Equalizers with Prescribed Nyquist-Frequency Gain, 532
12.5 Comb Filters, 534
12.6 Higher-Order Filters, 536
12.7 Analog Lowpass Butterworth Filters, 538
12.8 Digital Lowpass Filters, 543
12.9 Digital Highpass Filters, 547
12.10 Digital Bandpass Filters, 550
12.11 Digital Bandstop Filters, 555
12.12 Chebyshev Filter Design, 559
12.13 Problems, 572

13 Elliptic Filter Design 576

13.1 Introduction, 576
13.2 Jacobian Elliptic Functions, 580
13.3 Elliptic Rational Function and the Degree Equation, 587
13.4 Landen Transformations, 591
13.5 Analog Elliptic Filter Design, 593

13.6 Design Example, 594
13.7 Butterworth and Chebyshev Designs, 597
13.8 Highpass, Bandpass, and Bandstop Analog Filters, 600
13.9 Digital Filter Design, 605
13.10 Pole and Zero Transformations, 605
13.11 Transformation of Frequency Specifications, 609
13.12 MATLAB Implementation and Examples, 611
13.13 Frequency-Shifted Realizations, 614

14 Interpolation, Decimation, and Oversampling 622

14.1 Interpolation and Oversampling, 622
14.2 Interpolation Filter Design, 628
14.2.1 Direct Form, 628
14.2.2 Polyphase Form, 630
14.2.3 Frequency Domain Characteristics, 635
14.2.4 Kaiser Window Designs, 638
14.2.5 Multistage Designs, 639
14.3 Linear and Hold Interpolators, 647
14.4 Design Examples, 651
14.4.1 4-fold Interpolators, 651
14.4.2 Multistage 4-fold Interpolators, 657
14.4.3 DAC Equalization, 661
14.4.4 Postfilter Design and Equalization, 664
14.4.5 Multistage Equalization, 668
14.5 Decimation and Oversampling, 676
14.6 Sampling Rate Converters, 681
14.7 Noise Shaping Quantizers, 688
14.8 Problems, 696

15 Noise Reduction and Signal Enhancement 704

15.1 Noise Reduction and Signal Extraction, 704
15.2 IIR Exponential Smoother, 708
15.3 IIR Highpass Signal Extraction, 712
15.4 Bandpass Signal Extraction, 713
15.5 FIR Averaging Filters, 714
15.6 FIR Highpass Signal Extraction, 720
15.7 Noise Reduction, Time Constant, Group Delay, 720
15.8 Computer Experiment – Noise-Reduction vs. Group-Delay, 724
15.9 Computer Experiment – Time–Bandwidth Tradeoffs, 729
15.10 Computer Experiment – SMA, EMA, PMA, DEMA filters, 733
15.11 Notch and Comb Filters for Periodic Signals, 738
15.12 Line and Frame Combs for Digital TV, 749
15.13 Problems, 761

Part II Applications

16 Digital Audio Effects 767

16.1 Digital Waveform Generators, 767
16.1.1 Sinusoidal Generators, 767
16.1.2 Periodic Waveform Generators, 772
16.1.3 Wavetable Generators, 781
16.2 Digital Audio Effects, 800
16.2.1 Delays, Echoes, and Comb Filters, 801
16.2.2 Flanging, Chorusing, and Phasing, 806
16.2.3 Digital Reverberation, 813
16.2.4 Multitap Delays, 825
16.3 Dynamic Range Control, 829
16.3.1 Compressors, Limiters, Expanders, and Noise Gates, 829
16.3.2 Level Detectors and Gain Processors, 830
16.3.3 Attack and Release Time Constants and Gain Smoothing, 833
16.3.4 Computer Experiments, 835
16.3.5 Example Graphs, 838
16.4 Problems, 838

17 High-Order Digital Parametric Equalizers 849

17.1 Overview, 849
17.2 General Considerations, 850
17.3 Poles and Zeros, 854
17.4 Butterworth, Chebyshev, and Elliptic Designs, 857
17.5 Order Determination, 863
17.6 Bandwidth, 864
17.7 Realizations, 867
17.8 Decoupled Realizations, 871
17.9 Design Examples, 873
17.10 Appendix-1 State-Space Realizations, 879
17.11 Appendix-2 High-Order Analog Equalizer Design, 879
17.12 Appendix-3 MATLAB Functions, 880

18 STFT and Phase Vocoder 882

18.1 Introduction, 882
18.2 Short-Time Fourier Transform, 882
18.3 Spectrograms, 884
18.4 Inverse STFT and OLA Reconstruction, 886
18.5 STFT-Based Signal Processing System, 887
18.6 STFT Computation, 888
18.7 Phase Vocoder, 891
18.8 Time-Scale Modification, 891
18.9 Phase Vocoder Model, 892
18.10 Pitch-Scale Modification, 896
18.11 Computer Experiments, 897

19 DCT, MDCT, and Data Compression 905

19.1 DCT and MDCT Compression Systems, 905
19.2 Discrete Cosine Transform, 907
19.3 DCT Compression System, 910
19.4 MDCT and Time-Domain Aliasing Cancellation, 913
19.5 Princen-Bradley Windows, 916
19.6 Derivations and Computer Experiments, 918

20 Discrete Wavelet Transforms 928

20.1 Multiresolution Analysis, 928
20.2 Dilation Equations, 933
20.3 Wavelet Filter Properties, 939
20.4 Multiresolution and Filter Banks, 944
20.5 Discrete Wavelet Transform, 949
20.6 Multiresolution Decomposition, 962
20.7 Wavelet Denoising, 963
20.8 Undecimated Wavelet Transform, 966
20.9 MATLAB Functions, 975
20.10 Problems, 977

21 Discretization Methods 979

21.1 Continuous-Time Systems, 979
21.2 Mapping of Initial Conditions, 981
21.3 Forced Response, 983
21.4 Solution Procedures, 984
21.5 Steady-State Sinusoidal Response, 985
21.6 Continuous-Time Example, 986
21.7 Discretization Schemes – Summary, 994
21.8 Forward / Backward Euler, and Trapezoidal Rules, 996
21.9 Zero-Order and First-Order Holds, 997
21.10 Sample-by-Sample Processing, 999
21.11 Initialization Procedures, 1006
21.12 Forward / Backward Euler, and Trapezoidal Rules, 1014
21.13 Ideal Sampling, Starred Laplace Transform, z-Transform, 1015
21.14 Zero-Order Hold, 1018
21.15 Step Invariance and Impulse Invariance, 1023
21.16 First-Order Hold, 1025
21.17 Ramp Invariance, 1027
21.18 Appendix, 1027
21.19 MATLAB function – c2d2, 1029

22 Control Systems 1030

22.1 Feedback Control Systems, 1030
22.2 PID Control, 1032
22.3 Digital Control Systems, 1032
22.4 Examples, 1035
22.4.1 Cruise Control, 1035

22.4.2 Radar Tracking Antenna, 1037
22.4.3 Inverted Pendulum, 1048
22.4.4 Thermostat Model, 1053

23 Local Polynomial Filters 1058

23.1 Introduction, 1058
23.2 Local Polynomial Fitting, 1059
23.3 Exact Design Equations, 1068
23.4 Geometric Interpretation, 1073
23.5 Orthogonal Polynomial Bases, 1074
23.6 Polynomial Predictive and Interpolation Filters, 1075
23.7 Farrow Realization Structures, 1078
23.8 Minimum Variance Filters, 1082
23.9 Predictive Differentiation Filters, 1088
23.10 Filtering Implementations, 1093
23.11 Minimum Roughness Weighted Polynomial Filters, 1102
23.12 Henderson Filters, 1108
23.13 Hahn Orthogonal Polynomials, 1118
23.14 Maximally-Flat Filters, 1126
23.15 Missing Data and Outliers, 1131
23.16 Weighted Local Polynomial Modeling, 1136
23.17 Bandwidth Selection with CV and GCV, 1143
23.18 Local Polynomial Interpolation, 1145
23.19 Variable and Adaptive Bandwidth, 1150
23.20 Repeated Observations, 1156
23.21 Loess Smoothing, 1157
23.22 Problems, 1159

24 Exponential Moving Average Filters 1160

24.1 Mean Tracking, 1160
24.2 Forecasting and State-Space Models, 1169
24.3 Higher-Order Polynomial Smoothing Filters, 1170
24.4 Linear Trend FIR Filters, 1172
24.5 Higher-Order Exponential Smoothing, 1174
24.6 Steady-State Exponential Smoothing, 1180
24.7 Smoothing Parameter Selection, 1186
24.8 Single, Double, Triple Exponential Smoothing, 1191
24.9 Tukey’s Twicing Operation, 1193
24.10 Zero-Lag Filters and Twicing, 1194
24.11 Local Level, Local Slope, Local Acceleration Filters, 1196
24.12 Basis Transformations and EMA Initialization, 1198
24.13 Holt’s Exponential Smoothing, 1203
24.14 Problems, 1205

25 Filtering Methods in Financial Markets 1209

25.1 Technical Analysis of Financial Markets, 1209
25.2 Moving Average Filters – SMA, WMA, TMA, EMA, 1209
25.3 Predictive Moving Average Filters, 1212
25.4 Single, Double, Triple EMA Indicators, 1215
25.5 Linear Regression and R-Square Indicators, 1217
25.6 Initialization Schemes, 1222
25.7 Butterworth Moving Average Filters, 1228
25.8 Moving Average Filters with Reduced Lag, 1231
25.9 Zigzag Indicator, 1237
25.10 L_0 Trend Indicator, 1239
25.11 Envelopes, Bands, and Channels, 1242
25.12 Momentum, Oscillators, and Other Indicators, 1251
25.13 MATLAB Functions, 1257
25.14 Computer Project – Markowitz Portfolio Theory, 1259

26 Whittaker-Henderson Smoothing 1271

26.1 Smoothing Splines, 1271
26.2 Whittaker-Henderson Smoothing Methods, 1272
26.3 Regularization Filters, 1277
26.4 Hodrick-Prescott Filters, 1279
26.5 Poles and Impulse Response, 1282
26.6 Regularization and Kernel Machines, 1283
26.7 Sparse Whittaker-Henderson Methods, 1289
26.8 Computer Experiments, 1291
26.8.1 Total Variation Minimization, 1293
26.8.2 Local Linear Trends, 1294
26.8.3 Global Warming Trends, 1295
26.8.4 US GDP Macroeconomic Data, 1298
26.9 Sparse Modeling – LASSO and BPDN, 1302
26.9.1 Sparse Spike Deconvolution Example, 1309
26.9.2 Sparse Signal Recovery Example, 1314
26.10 Problems, 1318

27 Periodic Signal Extraction 1319

27.1 Introduction, 1319
27.2 Notch and Comb Filters for Periodic Signals, 1320
27.3 Notch and Comb Filters with Fractional Delay, 1326
27.4 Parallel and Cascade Realizations, 1331
27.5 Signal Averaging, 1336
27.6 Ideal Seasonal Decomposition Filters, 1342
27.7 Classical Seasonal Decomposition, 1344
27.8 Seasonal Moving-Average Filters, 1352
27.9 Census X-11 Decomposition Filters, 1358
27.10 Musgrave Asymmetric Filters, 1362
27.11 Seasonal Whittaker-Henderson Decomposition, 1368
27.12 Sparse Seasonal Whittaker-Henderson Decomposition, 1370
27.13 Problems, 1375

28 Neural Networks 1376

28.1 Introduction, 1376
28.2 Multilayer Feedforward Neural Networks, 1376
28.3 Backpropagation Algorithm, 1379
28.4 Computer Experiments, 1382
28.4.1 3:3:2 network for 3-bit parity problem, 1382
28.4.2 3:3:2:2 network for 3-bit parity problem, 1384
28.4.3 4:4:1 network for prediction of sunspot time series, 1384

29 Appendices 1389

A Random Number Generators, 1389
A.1 Uniform and Gaussian Generators, 1389
A.2 Low-Frequency Noise Generators, 1394
A.3 $1/f$ Noise Generators, 1399
A.4 Problems, 1403
B Prolate Spheroidal Wave Functions, 1406
B.1 Definition, 1407
B.2 Fourier Transform, 1409
B.3 Orthogonality and Completeness Properties, 1411
B.4 Signal Restoration, 1413
B.5 Representation and Extrapolation of Bandlimited Functions, 1415
B.6 Energy Concentration Properties, 1417
B.7 Computation, 1419
C MATLAB Toolbox, 1432

References 1433

Index 1505

Preface

Preface to the First Edition

This book provides an applications-oriented introduction to digital signal processing written primarily for electrical engineering undergraduates. Practicing engineers and graduate students may also find it useful as a first text on the subject.

Digital signal processing is everywhere. Today's college students hear "DSP" all the time in their everyday life—from their CD players, to their electronic music synthesizers, to the sound cards in their PCs. They hear all about "DSP chips", "oversampling digital filters", "1-bit A/D and D/A converters", "wavetable sound synthesis", "audio effects processors", "all-digital audio studios". By the time they reach their junior year, they are already very eager to learn more about DSP.

Approach

The learning of DSP can be made into a rewarding, interesting, and fun experience for the student by weaving into the material several applications, such as the above, that serve as vehicles for teaching the basic DSP concepts, while generating and maintaining student interest. This has been the guiding philosophy and objective in writing this text. As a result, the book's emphasis is more on signal processing than discrete-time system theory, although the basic principles of the latter are adequately covered.

The book teaches by example and takes a hands-on practical approach that emphasizes the *algorithmic, computational, and programming* aspects of DSP. It contains a large number of worked examples, computer simulations and applications, and several C and MATLAB functions for implementing various DSP operations. The practical slant of the book makes the concepts more concrete.

Use

The book may be used at the *junior or senior* level. It is based on a junior-level DSP course that I have taught at Rutgers since 1988. The assumed background is only a first course on linear systems. Sections marked with an asterisk (*) are more appropriate for a second or senior elective course on DSP. The rest can be covered at the junior level. The included computer experiments can form the basis of an accompanying DSP lab course, as is done at Rutgers.

A solutions manual, which also contains the results of the computer experiments, is available from the publisher. The C and MATLAB functions may be obtained via anonymous FTP from the Internet site `ece.rutgers.edu` in the directory `/pub/sjo` or by pointing a Web browser to the book's WWW home page at the URL:
`http://www.ece.rutgers.edu/~orfanidi/intro2sp`

Contents and Highlights

Chapters 1 and 2 contain a discussion of the two key DSP concepts of *sampling and quantization*. The first part of Chapter 1 covers the basic issues of sampling, aliasing, and *analog reconstruction* at a level appropriate for juniors. The second part is more advanced and discusses the practical issues of choosing and defining specifications for *antialiasing prefilters and anti-image postfilters*.

Chapter 2 discusses the *quantization process* and some practical implementations of A/D and D/A converters, such as the conversion algorithm for bipolar two's complement successive approximation converters. The standard model of quantization noise is presented, as well as the techniques of *oversampling, noise shaping, and dithering*. The tradeoff between oversampling ratio and savings in bits is derived. This material is continued in Section 12.7 where the implementation and operation of delta-sigma noise shaping quantizers is considered.

Chapter 3 serves as a review of basic *discrete-time systems* concepts, such as linearity, time-invariance, impulse response, convolution, FIR and IIR filters, causality, and stability. It can be covered quickly as most of this material is assumed known from a prerequisite linear systems course.

Chapter 4 focuses on FIR filters and its purpose is to introduce two basic signal processing methods: *block-by-block* processing and *sample-by-sample* processing. In the block processing part, we discuss various approaches to convolution, transient and steady-state behavior of filters, and real-time processing on a block-by-block basis using the overlap-add method and its software implementation. This is further discussed in Section 9.9 using the FFT.

In the sample processing part, we introduce the basic building blocks of filters: adders, multipliers, and delays. We discuss *block diagrams* for FIR filters and their time-domain operation on a sample-by-sample basis. We put a lot of emphasis on the concept of *sample processing algorithm*, which is the repetitive series of computations that must be carried out on each input sample.

We discuss the concept of *circular buffers* and their use in implementing delays and FIR filters. We present a systematic treatment of the subject and carry it on to the remainder of the book. The use of circular delay-line buffers is old, dating back at least 25 years with its application to computer music. However, it has not been treated systematically in DSP texts. It has acquired a new relevance because all modern DSP chips use it to minimize the number of hardware instructions.

Chapter 5 covers the basics of z-transforms. We emphasize the z-domain view of causality, stability, and frequency spectrum. Much of this material may be known from an earlier linear system course.

Chapter 6 shows the equivalence of various ways of characterizing a linear filter and illustrates their use by example. It also discusses topics such as sinusoidal and steady-state responses, time constants of filters, simple pole/zero designs of first- and second-order filters as well as comb and notch filters. The issues of inverse filtering and causality are also considered.

Chapter 7 develops the standard *filter realizations* of canonical, direct, and cascade forms, and their implementation with *linear and circular* buffers. Quantization effects are briefly discussed.

Chapter 8 presents three DSP application areas. The first is on digital *waveform generation*, with particular emphasis on *wavetable generators*. The second is on *digital audio effects*, such as flanging, chorusing, reverberation, multitap delays, and dynamics processors, such as compressors, limiters, expanders, and gates. These areas were chosen for their appeal to undergraduates and because they provide concrete illustrations of the use of delays, circular buffers, and filtering concepts in the context of audio signal processing.

The third area is on *noise reduction/signal enhancement*, which is one of the most important applications of DSP and is of interest to practicing engineers and scientists who remove noise from data on a routine basis. Here, we develop the basic principles for designing noise reduction and signal enhancement filters both in the frequency and time domains. We discuss the design and circular buffer implementation of *notch and comb* filters for removing periodic interference, enhancing periodic signals, signal averaging, and separating the luminance and chrominance components in digital color TV systems. We also discuss *Savitzky-Golay* filters for data smoothing and differentiation.

Chapter 9 covers *DFT/FFT algorithms*. The first part emphasizes the issues of spectral analysis, frequency resolution, windowing, and leakage. The second part discusses the computational aspects of the DFT and some of its pitfalls, the difference between physical and computational frequency resolution, the FFT, and fast convolution.

Chapter 10 covers *FIR filter design* using the window method, with particular emphasis on the *Kaiser window*. We also discuss the use of the Kaiser window in spectral analysis.

Chapter 11 discusses *IIR filter design* using the bilinear transformation based on Butterworth and Chebyshev filters. By way of introducing the bilinear transformation, we show how to design practical second-order digital audio *parametric equalizer* filters having prescribed widths, center frequencies, and gains. We also discuss the design of periodic notch and comb filters with prescribed widths.

In the two filter design chapters, we have chosen to present only a few design methods that are simple enough for our intended level of presentation and effective enough to be of practical use.

Chapter 12 discusses *interpolation, decimation, oversampling DSP systems, sample rate converters, and delta-sigma quantizers*. We discuss the use of oversampling for alleviating the need for high quality analog prefilters and postfilters. We present several practical design examples of interpolation filters, including *polyphase and multistage* designs. We consider the design of sample rate converters and study the operation of oversampled delta-sigma quantizers by simulation. This material is too advanced for juniors but not seniors. All undergraduates, however, have a strong interest in it because of its use in digital audio systems such as CD and DAT players.

The Appendix has four parts: (a) a review section on *random signals*; (b) a discussion of random number generators, including uniform, Gaussian, low frequency, and $1/f$ noise generators; (c) C functions for performing the complex arithmetic in the DFT routines; (d) listings of MATLAB functions.

Paths

Several course paths are possible through the text depending on the desired level of presentation. For example, in the 14-week junior course at Rutgers we cover Sections 1.1–1.4, 2.1–2.4, Chapters 3–7, Sections 8.1–8.2, Chapter 9, and Sections 10.1–10.2 and 11.1–11.4. One may omit certain of these sections and/or add others depending on the available time and student interest and background. In a second DSP course at the senior year, one may add Sections 1.5–1.7, 2.5, 8.3, 11.5–11.6, and Chapter 12. In a graduate course, the entire text can be covered comfortably in one semester.

Acknowledgments

I am indebted to the many generations of students who tried earlier versions of the book and helped me refine it. In particular, I would like to thank Mr. Cem Saraydar for his thorough proofreading of the manuscript. I would like to thank my colleagues Drs. Zoran Gajic, Mark Kahrs, James Kaiser, Dino Lelic, Tom Marshall, Peter Meer, and Nader Moayeri for their feedback and encouragement. I am especially indebted to Dr. James Kaiser for enriching my classes over the past eight years with his inspiring yearly lectures on the Kaiser window. I would like to thank the book's reviewers Drs. A. V. Oppenheim, J. A. Fleming, Y-C. Jenq, W. B. Mikhael, S. J. Reeves, A. Sekey, and J. Weitzen, whose comments helped improve the book. And I would like to thank Rutgers for providing me with a sabbatical leave to finish up the project. I welcome any feedback from readers—it may be sent to `orfanidi@ece.rutgers.edu`.

Finally, I would like to thank my wife Monica and son John for their love, patience, encouragement, and support.

Sophocles J. Orfanidis

Preface to the Second Edition

The contents of the first edition have been left essentially unchanged with some minor changes. Several new topics have been added covering additional theory, as well as a range of signal processing applications. New topics include:

- Lattice filters
- Elliptic filter design
- High-order digital parametric audio equalizers
- Short-time Fourier transform (STFT) and applications
- Phase vocoder, time-scale and pitch-scale modification
- DCT, modified DCT, and data compression
- Discrete wavelet transforms
- Discretization methods for continuous-time systems
- Brief introduction to analog and digital PID control systems
- Local polynomial filters
- Minimum-roughness Henderson filters
- Weighted local polynomial modeling and LOESS
- Local polynomial interpolation
- Exponential moving average filters
- Zero-lag filters
- Filtering methods in financial markets
- Moving Average Filters - SMA, WMA, TMA, EMA
- Predictive moving average filters
- Single, double, triple EMA indicators
- Linear regression and R-square indicators
- Moving average filters with reduced lag
- Envelopes, bands, and channels
- Momentum, oscillators, and other market indicators
- Whittaker-Henderson smoothing including sparse versions
- Hodrick-Prescott filters
- Sparse modeling – LASSO and BPDN
- Periodic signal extraction
- Fractional delay filters
- Signal averaging
- Ideal and classical seasonal decomposition
- Census X-11 decomposition filters
- Seasonal Whittaker-Henderson decomposition including sparse versions
- Brief introduction to neural networks
- Prolate spheroidal wave functions

Sophocles J. Orfanidis

Part I

Basics

1

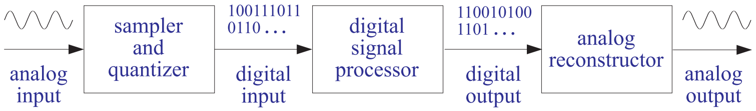
Sampling and Reconstruction

1.1 Introduction

Digital processing of analog signals proceeds in three stages:

1. The analog signal is *digitized*, that is, it is *sampled* and each sample *quantized* to a finite number of bits. This process is called A/D conversion.
2. The digitized samples are processed by a *digital signal processor*.
3. The resulting output samples may be converted back into *analog* form by an analog reconstructor (D/A conversion).

A typical digital signal processing system is shown below.



The digital signal processor can be programmed to perform a variety of signal processing operations, such as filtering, spectrum estimation, and other DSP algorithms. Depending on the speed and computational requirements of the application, the digital signal processor may be realized by a general purpose computer, minicomputer, special purpose DSP chip, or other digital hardware dedicated to performing a particular signal processing task.

The design and implementation of DSP algorithms will be considered in the rest of this text. In the first two chapters we discuss the two key concepts of *sampling* and *quantization*, which are prerequisites to every DSP operation.

1.2 Review of Analog Signals

We begin by reviewing some pertinent topics from analog system theory. An *analog* signal is described by a function of time, say, $x(t)$. The *Fourier transform* $X(\Omega)$ of $x(t)$

is the *frequency spectrum* of the signal:

$$\boxed{X(\Omega)=\int_{-\infty}^{\infty}x(t)\,e^{-j\Omega t}\,dt} \tag{1.2.1}$$

where Ω is the radian frequency[†] in [radians/second]. The ordinary frequency f in [Hertz] or [cycles/sec] is related to Ω by

$$\boxed{\Omega = 2\pi f} \tag{1.2.2}$$

The physical meaning of $X(\Omega)$ is brought out by the *inverse* Fourier transform, which expresses the arbitrary signal $x(t)$ as a linear superposition of *sinusoids* of different frequencies:

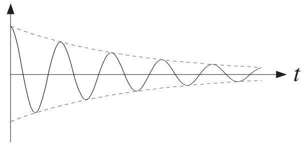
$$\boxed{x(t)=\int_{-\infty}^{\infty}X(\Omega)\,e^{j\Omega t}\,\frac{d\Omega}{2\pi}} \tag{1.2.3}$$

The relative importance of each sinusoidal component is given by the quantity $X(\Omega)$. The *Laplace transform* is defined by

$$X(s)=\int_{-\infty}^{\infty}x(t)\,e^{-st}\,dt$$

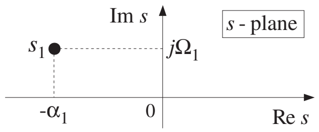
It reduces to the Fourier transform, Eq. (1.2.1), under the substitution $s = j\Omega$. The s-plane pole/zero properties of transforms provide additional insight into the nature of signals. For example, a typical exponentially decaying sinusoid of the form

$$x(t)=e^{-\alpha_1 t}e^{j\Omega_1 t}u(t)=e^{s_1 t}u(t)$$

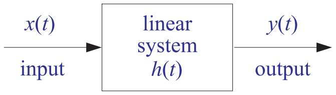


where $s_1 = -\alpha_1 + j\Omega_1$, has Laplace transform

$$X(s)=\frac{1}{s-s_1}$$



with a pole at $s = s_1$, which lies in the left-hand s-plane. Next, consider the response of a *linear system* to an input signal $x(t)$:



[†] We use the notation Ω to denote the physical frequency in units of [radians/sec], and reserve the notation ω to denote digital frequency in [radians/sample].

The system is characterized completely by the *impulse response* function $h(t)$. The output $y(t)$ is obtained in the time domain by *convolution*:

$$y(t)=\int_{-\infty}^{\infty} h(t-t')\,x(t')\,dt'$$

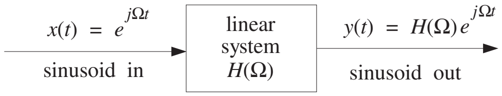
or, in the frequency domain by *multiplication*:

$$Y(\Omega)=H(\Omega)X(\Omega) \tag{1.2.4}$$

where $H(\Omega)$ is the *frequency response* of the system, defined as the Fourier transform of the impulse response $h(t)$:

$$H(\Omega)=\int_{-\infty}^{\infty} h(t)\,e^{-j\Omega t}\,dt \tag{1.2.5}$$

The *steady-state* sinusoidal response of the filter, defined as its response to sinusoidal inputs, is summarized below:



This figure illustrates the *filtering* action of linear filters, that is, a given frequency component Ω is attenuated (or, magnified) by an amount $H(\Omega)$ by the filter. More precisely, an input sinusoid of frequency Ω will reappear at the output modified in *magnitude* by a factor $|H(\Omega)|$ and shifted in *phase* by an amount $\arg H(\Omega)$:

$$x(t)=e^{j\Omega t} \qquad\Rightarrow\qquad y(t)=H(\Omega)e^{j\Omega t}=|H(\Omega)|e^{j\Omega t+j\arg H(\Omega)}$$

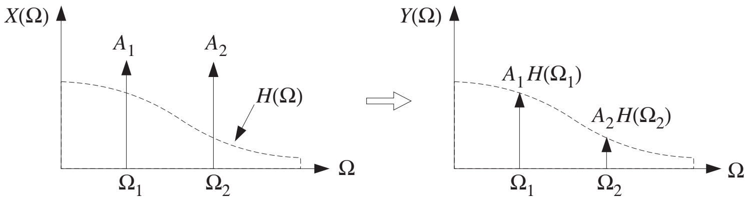
By linear superposition, if the input consists of the sum of two sinusoids of frequencies Ω_1 and Ω_2 and relative amplitudes A_1 and A_2,

$$x(t)=A_1e^{j\Omega_1 t}+A_2e^{j\Omega_2 t}$$

then, after filtering, the steady-state output will be

$$y(t)=A_1H(\Omega_1)e^{j\Omega_1 t}+A_2H(\Omega_2)e^{j\Omega_2 t}$$

Notice how the filter changes the *relative* amplitudes of the sinusoids, but not their frequencies. The filtering effect may also be seen in the frequency domain using Eq. (1.2.4), as shown below:



The input spectrum $X(\Omega)$ consists of two sharp spectral lines at frequencies Ω_1 and Ω_2, as can be seen by taking the Fourier transform of $x(t)$:

$$X(\Omega)=2\pi A_1\delta(\Omega-\Omega_1)+2\pi A_2\delta(\Omega-\Omega_2)$$

The corresponding output spectrum $Y(\Omega)$ is obtained from Eq. (1.2.4):

$$\begin{aligned}Y(\Omega)&=H(\Omega)X(\Omega)=H(\Omega)\big(2\pi A_1\delta(\Omega-\Omega_1)+2\pi A_2\delta(\Omega-\Omega_2)\big)\\&=2\pi A_1H(\Omega_1)\delta(\Omega-\Omega_1)+2\pi A_2H(\Omega_2)\delta(\Omega-\Omega_2)\end{aligned}$$

What makes the subject of linear filtering useful is that the designer has complete *control* over the shape of the frequency response $H(\Omega)$ of the filter. For example, if the sinusoidal component Ω_1 represents a desired signal and Ω_2 an unwanted interference, then a filter may be designed that lets Ω_1 pass through, while at the same time it filters out the Ω_2 component. Such a filter must have $H(\Omega_1)=1$ and $H(\Omega_2)=0$.

1.3 Sampling Theorem

Next, we study the sampling process, illustrated in Fig. 1.3.1, where the analog signal $x(t)$ is *periodically measured* every T seconds. Thus, time is discretized in units of the *sampling interval* T:

$$t=nT,\qquad n=0,1,2,\dots$$

Considering the resulting stream of samples as an *analog* signal, we observe that the sampling process represents a very drastic chopping operation on the original signal $x(t)$, and therefore, it will introduce a lot of spurious *high-frequency* components into the frequency spectrum. Thus, for system design purposes, two questions must be answered:

1. What is the effect of sampling on the original frequency spectrum?

2. How should one choose the sampling interval T?

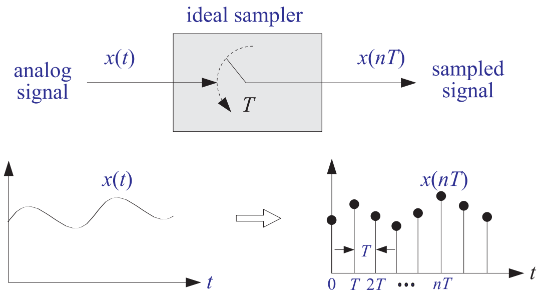


Fig. 1.3.1 Ideal sampler.

We will try to answer these questions intuitively, and then more formally using Fourier transforms. We will see that although the sampling process generates high frequency components, these components appear in a very regular fashion, that is, *every* frequency component of the original signal is *periodically replicated* over the entire frequency axis, with period given by the *sampling rate*:

$$\boxed{f_s = \frac{1}{T}} \tag{1.3.1}$$

This replication property will be justified first for simple sinusoidal signals and then for arbitrary signals. Consider, for example, a single sinusoid $x(t)= e^{2\pi jft}$ of frequency f. Before sampling, its spectrum consists of a single sharp spectral line at f. But after sampling, the spectrum of the sampled sinusoid $x(nT)= e^{2\pi jfnT}$ will be the periodic replication of the original spectral line at intervals of f_s, as shown in Fig. 1.3.2.

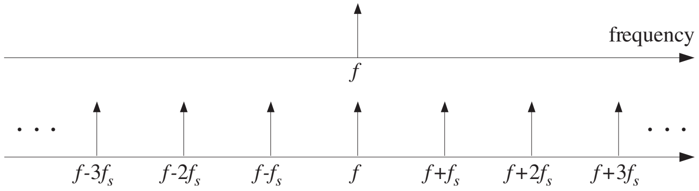


Fig. 1.3.2 Spectrum replication caused by sampling.

Note also that starting with the replicated spectrum of the sampled signal, one cannot tell uniquely what the original frequency was. It could be *any* one of the replicated frequencies, namely, $f' = f + mf_s$, $m = 0, \pm 1, \pm 2, \dots$. That is so because any one of them has the same periodic replication when sampled. This potential confusion of the original frequency with another is known as *aliasing* and can be avoided if one satisfies the conditions of the sampling theorem.

The sampling theorem provides a quantitative answer to the question of how to choose the sampling time interval T. Clearly, T must be *small enough* so that signal variations that occur between samples are not lost. But how small is small enough? It would be very impractical to choose T too small because then there would be too many samples to be processed. This is illustrated in Fig. 1.3.3, where T is small enough to resolve the details of signal 1, but is unnecessarily small for signal 2.

Another way to say the same thing is in terms of the sampling rate f_s, which is measured in units of [samples/sec] or [Hertz] and represents the "density" of samples per unit time. Thus, a rapidly varying signal must be sampled at a high sampling rate f_s, whereas a slowly varying signal may be sampled at a lower rate.

1.3.1 Sampling Theorem

A more quantitative criterion is provided by the *sampling theorem* which states that for *accurate* representation of a signal $x(t)$ by its time samples $x(nT)$, two conditions must be met:

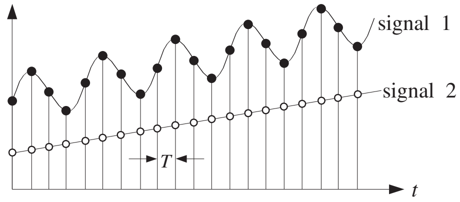


Fig. 1.3.3 Signal 2 is oversampled.

1. The signal $x(t)$ must be *bandlimited*, that is, its frequency spectrum must be limited to contain frequencies up to some maximum frequency, say $f_{\max}$, and no frequencies beyond that. A typical bandlimited spectrum is shown in Fig. 1.3.4.

2. The sampling rate f_s must be chosen to be at least *twice* the maximum frequency $f_{\max}$, that is,

$$\boxed{f_s \geq 2f_{\max}} \tag{1.3.2}$$

or, expressed in terms of the sampling time interval,

$$\boxed{T \leq \frac{1}{2f_{\max}}}$$

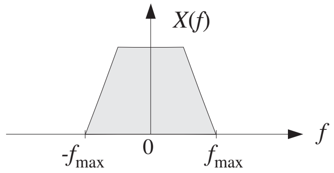


Fig. 1.3.4 Typical bandlimited spectrum.

The *minimum* sampling rate allowed by the sampling theorem, that is, $f_s = 2f_{\max}$, is called the *Nyquist rate*. For arbitrary values of f_s, the quantity $f_s/2$ is called the *Nyquist frequency*, or the folding frequency. It defines the endpoints of the *Nyquist frequency interval*:

$$(f_s)_{\max} = 2f_{\max} = \text{Nyquist rate}$$

$$\frac{f_s}{2} = \text{Nyquist frequency}$$

$$\left[-\frac{f_s}{2}, \frac{f_s}{2}\right] = \text{Nyquist Interval}$$

The Nyquist frequency $f_s/2$ also defines the *cutoff frequencies* of the lowpass analog prefilters and postfilters that are required in DSP operations. The values of $f_{\max}$ and f_s

depend on the application. Typical sampling rates for some common DSP applications are shown in the following table.

application	$f_{\max}$	f_s
geophysical	500 Hz	1 kHz
biomedical	1 kHz	2 kHz
mechanical	2 kHz	4 kHz
speech	4 kHz	8 kHz
audio	20 kHz	40 kHz
video	4 MHz	8 MHz

1.3.2 Antialiasing Prefilters

The practical implications of the sampling theorem are quite important. Since most signals are not bandlimited, they must be made so by lowpass filtering *before* sampling.

In order to sample a signal at a desired rate f_s and satisfy the conditions of the sampling theorem, the signal must be *prefiltered* by a lowpass *analog* filter, known as an *antialiasing* prefilter. The cutoff frequency of the prefilter, $f_{\max}$, must be taken to be at most equal to the Nyquist frequency $f_s/2$, that is, $f_{\max} \le f_s/2$. This operation is shown in Fig. 1.3.5.

The output of the analog prefilter will then be *bandlimited* to maximum frequency $f_{\max}$ and may be sampled properly at the desired rate f_s. The spectrum replication caused by the sampling process can also be seen in Fig. 1.3.5. It will be discussed in detail in Section 1.5.

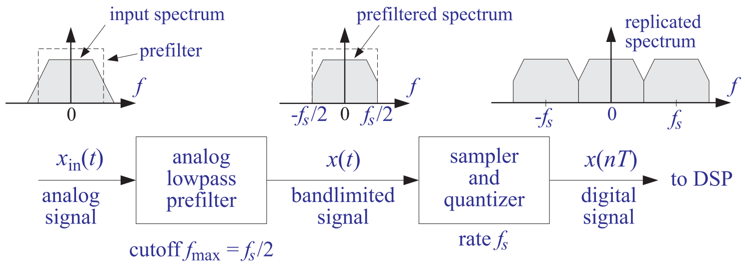


Fig. 1.3.5 Antialiasing prefilter.

It should be emphasized that the rate f_s must be chosen to be high enough so that, after the prefiltering operation, the surviving signal spectrum within the Nyquist interval $[-f_s/2, f_s/2]$ contains all the *significant* frequency components for the application at hand.

Example 1.3.1: In a hi-fi digital audio application, we wish to digitize a music piece using a sampling rate of 40 kHz. Thus, the piece must be prefiltered to contain frequencies up to 20 kHz. After the prefiltering operation, the resulting spectrum of frequencies is more than adequate for this application because the human ear can hear frequencies only up to 20 kHz. □

Example 1.3.2: Similarly, the spectrum of speech prefiltered to about 4 kHz results in very intelligible speech. Therefore, in digital speech applications it is adequate to use sampling rates of about 8 kHz and prefilter the speech waveform to about 4 kHz. □

What happens if we do not sample in accordance with the sampling theorem? If we undersample, we may be missing important time variations between sampling instants and may arrive at the erroneous conclusion that the samples represent a signal which is smoother than it actually is. In other words, we will be confusing the true frequency content of the signal with a lower frequency content. Such confusion of signals is called *aliasing* and is depicted in Fig. 1.3.6.

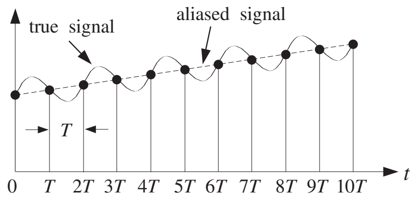


Fig. 1.3.6 Aliasing in the time domain.

1.3.3 Hardware Limits

Next, we consider the restrictions imposed on the choice of the sampling rate f_s by the hardware. The sampling theorem provides a *lower* bound on the allowed values of f_s. The hardware used in the application imposes an *upper* bound.

In real-time applications, each input sample must be acquired, quantized, and processed by the DSP, and the output sample converted back into analog format. Many of these operations can be pipelined to reduce the total processing time. For example, as the DSP is processing the present sample, the D/A may be converting the previous output sample, while the A/D may be acquiring the next input sample.

In any case, there is a total processing or computation time, say T_{proc} seconds, required for each sample. The time interval T between input samples must be greater than T_{proc}; otherwise, the processor would not be able to keep up with the incoming samples. Thus,

$$\boxed{T \ge T_{\text{proc}}}$$

or, expressed in terms of the computation or processing rate, $f_{\text{proc}} = 1/T_{\text{proc}}$, we obtain the upper bound $f_s \le f_{\text{proc}}$, which combined with Eq. (1.3.2) restricts the choice of f_s to the range:

$$\boxed{2f_{\max} \le f_s \le f_{\text{proc}}}$$

In succeeding sections we will discuss the phenomenon of aliasing in more detail, provide a quantitative proof of the sampling theorem, discuss the spectrum replication property, and consider the issues of practical sampling and reconstruction and their effect on the overall quality of a digital signal processing system. Quantization will be considered later on.

1.4 Sampling of Sinusoids

The two conditions of the sampling theorem, namely, that $x(t)$ be bandlimited and the requirement $f_s \geq 2f_{\max}$, can be derived intuitively by considering the sampling of sinusoidal signals only. Figure 1.4.1 shows a sinusoid of frequency f,

$$x(t)=\cos(2\pi f t)$$

that has been sampled at the three rates: $f_s = 8f$, $f_s = 4f$, and $f_s = 2f$. These rates correspond to taking 8, 4, and 2 samples in each cycle of the sinusoid.

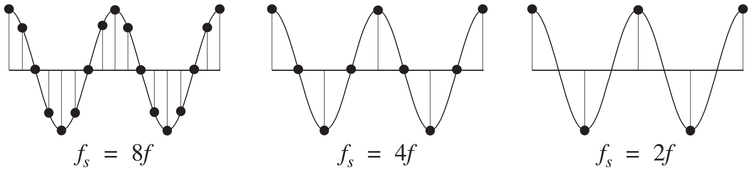


Fig. 1.4.1 Sinusoid sampled at rates $f_s = 8f, 4f, 2f$.

Simple inspection of these figures leads to the conclusion that the *minimum* acceptable number of samples per cycle is *two*. The representation of a sinusoid by two samples per cycle is hardly adequate,[†] but at least it does incorporate the basic *up-down* nature of the sinusoid. The number of samples per cycle is given by the quantity f_s/f:

$$\frac{f_s}{f} = \frac{\text{samples/sec}}{\text{cycles/sec}} = \frac{\text{samples}}{\text{cycle}}$$

Thus, to sample a single sinusoid properly, we must require

$$\frac{f_s}{f} \geq 2 \text{ samples/cycle} \qquad \Rightarrow \qquad f_s \geq 2f \tag{1.4.1}$$

Next, consider the case of an arbitrary signal $x(t)$. According to the inverse Fourier transform of Eq. (1.2.3), $x(t)$ can be expressed as a linear combination of sinusoids. Proper sampling of $x(t)$ will be achieved only if *every* sinusoidal component of $x(t)$ is properly sampled.

This requires that the signal $x(t)$ be bandlimited. Otherwise, it would contain sinusoidal components of arbitrarily high frequency f, and to sample those accurately, we would need, by Eq. (1.4.1), arbitrarily high rates f_s. If the signal is bandlimited to some maximum frequency $f_{\max}$, then by choosing $f_s \geq 2f_{\max}$, we are accurately sampling the *fastest-varying* component of $x(t)$, and thus a fortiori, all the slower ones. As an example, consider the special case:

$$x(t)= A_1\cos(2\pi f_1 t)+A_2\cos(2\pi f_2 t)+\cdots+A_{\max}\cos(2\pi f_{\max} t)$$

where f_i are listed in increasing order. Then, the conditions

$$2f_1 \leq 2f_2 \leq \cdots \leq 2f_{\max} \leq f_s$$

imply that every component of $x(t)$, and hence $x(t)$ itself, is properly sampled.

[†] It also depends on the phase of the sinusoid. For example, sampling at the zero crossings instead of at the peaks, would result in zero values for the samples.

1.4.1 Analog Reconstruction and Aliasing

Next, we discuss the aliasing effects that result if one violates the sampling theorem conditions (1.3.2) or (1.4.1). Consider the complex version of a sinusoid:

$$x(t)= e^{j\Omega t} = e^{2\pi j f t}$$

and its sampled version obtained by setting $t = nT$,

$$x(nT)= e^{j\Omega T n} = e^{2\pi j f T n}$$

Define also the following family of sinusoids, for $m = 0, \pm1, \pm2, \dots$,

$$x_m(t)= e^{2\pi j(f+mf_s)t}$$

and their sampled versions,

$$x_m(nT)= e^{2\pi j(f+mf_s)Tn}$$

Using the property $f_sT = 1$ and the trigonometric identity,

$$e^{2\pi j m f_s T n} = e^{2\pi j m n} = 1$$

we find that, although the signals $x_m(t)$ are different from each other, their *sampled* values are the *same*; indeed,

$$x_m(nT)= e^{2\pi j(f+mf_s)Tn} = e^{2\pi j f T n}e^{2\pi j m f_s T n} = e^{2\pi j f T n} = x(nT)$$

In terms of their sampled values, the signals $x_m(t)$ are indistinguishable, or aliased. Knowledge of the sample values $x(nT)= x_m(nT)$ is not enough to determine which among them was the original signal that was sampled. It could have been any one of the $x_m(t)$. In other words, the set of frequencies,

$$f,\ f\pm f_s,\ f\pm 2f_s,\ \dots,\ f\pm mf_s,\ \dots \tag{1.4.2}$$

are equivalent to each other. The effect of sampling was to replace the original frequency f with the replicated set (1.4.2). This is the intuitive explanation of the spectrum replication property depicted in Fig. 1.3.2. A more mathematical explanation will be given later using Fourier transforms.

Given that the sample values $x(nT)$ do not uniquely determine the analog signal they came from, the question arises: What analog signal would result if these samples were fed into an analog reconstructor, as shown in Fig. 1.4.2?

We will see later that an *ideal* analog reconstructor *extracts* from a sampled signal all the frequency components that lie *within* the Nyquist interval $[-f_s/2, f_s/2]$ and *removes* all frequencies outside that interval. In other words, an ideal reconstructor acts as a *lowpass filter* with cutoff frequency equal to the Nyquist frequency $f_s/2$.

Among the frequencies in the replicated set (1.4.2), there is a *unique* one that lies *within* the Nyquist interval.[†] It is obtained by reducing the original f modulo-f_s, that is,

[†] The only exception is when it falls exactly on the left or right edge of the interval, $f = \pm f_s/2$.

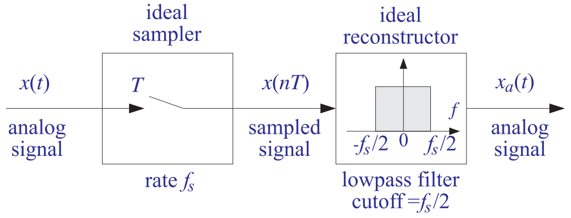


Fig. 1.4.2 Ideal reconstructor as a lowpass filter.

adding to or subtracting from f enough *multiples* of f_s until it lies within the *symmetric* Nyquist interval $[-f_s/2, f_s/2]$. We denote this operation by[‡]

$$\boxed{f_a = f \bmod (f_s)} \tag{1.4.3}$$

This is the frequency, in the replicated set (1.4.2), that will be extracted by the analog reconstructor. Therefore, the reconstructed sinusoid will be:

$$x_a(t) = e^{2\pi j f_a t}$$

It is easy to see that $f_a = f$ only if f lies within the Nyquist interval, that is, only if $|f| \le f_s/2$, which is equivalent to the sampling theorem requirement. If f lies outside the Nyquist interval, that is, $|f| > f_s/2$, violating the sampling theorem condition, then the "aliased" frequency f_a will be different from f and the reconstructed analog signal $x_a(t)$ will be different from $x(t)$, even though the two agree at the sampling times, $x_a(nT) = x(nT)$.

It is instructive also to plot in Fig. 1.4.3 the aliased frequency $f_a = f \bmod (f_s)$ versus the true frequency f. Observe how the straight line $f_{\text{true}} = f$ is brought down in segments by parallel translation of the Nyquist periods by multiples of f_s.

In summary, potential aliasing effects that can arise at the reconstruction phase of DSP operations can be avoided if one makes sure that *all* frequency components of the signal to be sampled satisfy the sampling theorem condition, $|f| \le f_s/2$, that is, all frequency components lie *within* the Nyquist interval. This is ensured by the lowpass antialiasing prefilter, which removes all frequencies beyond the Nyquist frequency $f_s/2$, as shown in Fig. 1.3.5.

Example 1.4.1: Consider a sinusoid of frequency $f = 10$ Hz sampled at a rate of $f_s = 12$ Hz. The sampled signal will contain all the replicated frequencies $f_m = 10 + 12m$ Hz, $m = 0, \pm 1, \pm 2, \dots$, or,

$$\dots, -26,\ -14,\ -2,\ 10,\ 22,\ 34,\ 46, \dots$$

and among these only $f_a = 10 \bmod (12) = 10 - 12 = -2$ Hz lies within the Nyquist interval $[-6, 6]$ Hz. This sinusoid will appear at the output of a reconstructor as a -2 Hz sinusoid instead of a 10 Hz one.

[‡] This differs slightly from a true modulo operation; the latter would bring f into the right-sided Nyquist interval $[0, f_s]$.

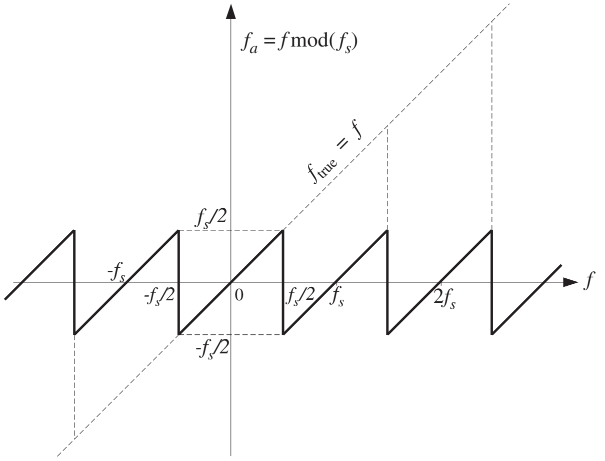


Fig. 1.4.3 $f \bmod (f_s)$ versus f.

On the other hand, had we sampled at a proper rate, that is, greater than $2f = 20$ Hz, say at $f_s = 22$ Hz, then no aliasing would result because the given frequency of 10 Hz already lies within the corresponding Nyquist interval of $[-11, 11]$ Hz. □

Example 1.4.2: Suppose a music piece is sampled at rate of 40 kHz without using a prefilter with cutoff of 20 kHz. Then, inaudible components having frequencies greater than 20 kHz can be aliased into the Nyquist interval $[-20, 20]$ distorting the true frequency components in that interval. For example, all components in the inaudible frequency range $20 \le f \le 60$ kHz will be aliased with $-20 = 20 - 40 \le f - f_s \le 60 - 40 = 20$ kHz, which are audible. □

Example 1.4.3: The following five signals, where t is in seconds, are sampled at a rate of 4 Hz:

$$-\sin(14\pi t),\quad -\sin(6\pi t),\quad \sin(2\pi t),\quad \sin(10\pi t),\quad \sin(18\pi t)$$

Show that they are all aliased with each other in the sense that their sampled values are the same.

Solution: The frequencies of the five sinusoids are:

$$-7,\quad -3,\quad 1,\quad 5,\quad 9\quad \text{Hz}$$

They differ from each other by multiples of $f_s = 4$ Hz. Their sampled spectra will be indistinguishable from each other because each of these frequencies has the *same* periodic replication in multiples of 4 Hz.

Writing the five frequencies compactly:

$$f_m = 1 + 4m, \qquad m = -2, -1, 0, 1, 2$$

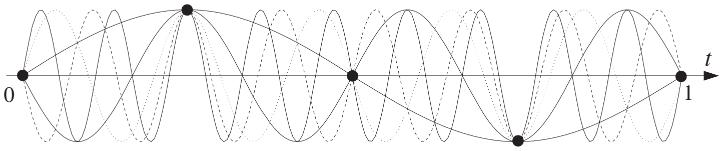
we can express the five sinusoids as:

$$x_m(t) = \sin(2\pi f_m t) = \sin(2\pi(1 + 4m)t), \qquad m = -2, -1, 0, 1, 2$$

Replacing $t = nT = n/f_s = n/4$ sec, we obtain the sampled signals:

$$\begin{aligned} x_m(nT) &= \sin\big(2\pi(1+4m)nT\big) = \sin\big(2\pi(1+4m)n/4\big) \\ &= \sin(2\pi n/4 + 2\pi mn) = \sin(2\pi n/4) \end{aligned}$$

which are the same, independently of m. The following figure shows the five sinusoids over the interval $0 \le t \le 1$ sec.



They all intersect at the sampling time instants $t = nT = n/4$ sec. We will reconsider this example in terms of rotating wheels in Section 1.4.2. □

Example 1.4.4: Let $x(t)$ be the sum of sinusoidal signals

$$x(t) = 4 + 3\cos(\pi t) + 2\cos(2\pi t) + \cos(3\pi t)$$

where t is in milliseconds. Determine the minimum sampling rate that will not cause any aliasing effects, that is, the Nyquist rate. To observe such aliasing effects, suppose this signal is sampled at half its Nyquist rate. Determine the signal $x_a(t)$ that would be aliased with $x(t)$.

Solution: The frequencies of the four terms are: $f_1 = 0$, $f_2 = 0.5$ kHz, $f_3 = 1$ kHz, and $f_4 = 1.5$ kHz (they are in kHz because t is in msec). Thus, $f_{\max} = f_4 = 1.5$ kHz and the Nyquist rate will be $2f_{\max} = 3$ kHz. If $x(t)$ is now sampled at half this rate, that is, at $f_s = 1.5$ kHz, then aliasing will occur.

The corresponding Nyquist interval is $[-0.75, 0.75]$ kHz. The frequencies f_1 and f_2 are already in it, and hence they are not aliased, in the sense that $f_{1a} = f_1$ and $f_{2a} = f_2$. But f_3 and f_4 lie outside the Nyquist interval and they will be aliased with

$$\begin{aligned} f_{3a} &= f_3 \bmod(f_s) = 1 \bmod(1.5) = 1 - 1.5 = -0.5 \text{ kHz} \\ f_{4a} &= f_4 \bmod(f_s) = 1.5 \bmod(1.5) = 1.5 - 1.5 = 0 \text{ kHz} \end{aligned}$$

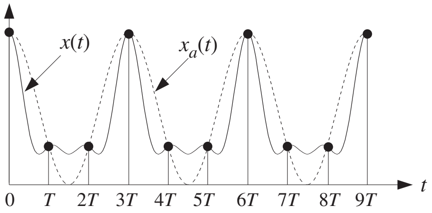
The aliased signal $x_a(t)$ is obtained from $x(t)$ by replacing f_1, f_2, f_3, f_4 by $f_{1a}, f_{2a}, f_{3a}, f_{4a}$. Thus, the signal

$$x(t) = 4\cos(2\pi f_1 t) + 3\cos(2\pi f_2 t) + 2\cos(2\pi f_3 t) + \cos(2\pi f_4 t)$$

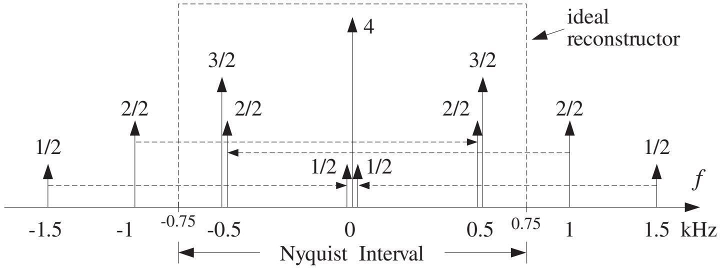
will be aliased with

$$\begin{aligned} x_a(t) &= 4\cos(2\pi f_{1a} t) + 3\cos(2\pi f_{2a} t) + 2\cos(2\pi f_{3a} t) + \cos(2\pi f_{4a} t) \\ &= 4 + 3\cos(\pi t) + 2\cos(-\pi t) + \cos(0) \\ &= 5 + 5\cos(\pi t) \end{aligned}$$

The signals $x(t)$ and $x_a(t)$ are shown below. Note that they agree only at their sampled values, that is, $x_a(nT) = x(nT)$. The aliased signal $x_a(t)$ is smoother, that is, it has lower frequency content than $x(t)$ because its spectrum lies entirely within the Nyquist interval, as shown below:



The form of $x_a(t)$ can also be derived in the frequency domain by replicating the spectrum of $x(t)$ at intervals of $f_s = 1.5$ kHz, and then extracting whatever part of the spectrum lies within the Nyquist interval. The following figure shows this procedure.



Each spectral line of $x(t)$ is replicated in the fashion of Fig. 1.3.2. The two spectral lines of strength 1/2 at $f_4 = \pm 1.5$ kHz replicate onto $f = 0$ and the amplitudes add up to give a total amplitude of $(4 + 1/2 + 1/2) = 5$. Similarly, the two spectral lines of strength 2/2 at $f_3 = \pm 1$ kHz replicate onto $f = \mp 0.5$ kHz and the amplitudes add to give $(3/2 + 2/2) = 2.5$ at $f = \pm 0.5$ kHz. Thus, the ideal reconstructor will extract $f_1 = 0$ of strength 5 and $f_2 = \pm 0.5$ of equal strengths 2.5, which recombine to give:

$$5 + 2.5e^{2\pi j 0.5 t} + 2.5e^{-2\pi j 0.5 t} = 5 + 5\cos(\pi t)$$

This example shows how aliasing can distort irreversibly the amplitudes of the original frequency components within the Nyquist interval. □

Example 1.4.5: The signal

$$x(t) = \sin(\pi t) + 4\sin(3\pi t)\cos(2\pi t)$$

where t is in msec, is sampled at a rate of 3 kHz. Determine the signal $x_a(t)$ aliased with $x(t)$. Then, determine two other signals $x_1(t)$ and $x_2(t)$ that are aliased with the same $x_a(t)$, that is, such that $x_1(nT) = x_2(nT) = x_a(nT)$.

Solution: To determine the frequency content of $x(t)$, we must express it as a sum of sinusoids. Using the trigonometric identity $2\sin a\cos b = \sin(a+b) + \sin(a-b)$, we find:

$$x(t) = \sin(\pi t) + 2\big[\sin(3\pi t + 2\pi t) + \sin(3\pi t - 2\pi t)\big] = 3\sin(\pi t) + 2\sin(5\pi t)$$

Thus, the frequencies present in $x(t)$ are $f_1 = 0.5$ kHz and $f_2 = 2.5$ kHz. The first already lies in the Nyquist interval $[-1.5, 1, 5]$ kHz so that $f_{1a} = f_1$. The second lies outside and can be reduced mod f_s to give $f_{2a} = f_2 \bmod (f_s) = 2.5 \bmod (3) = 2.5 - 3 = -0.5$. Thus, the given signal will "appear" as:

$$\begin{aligned} x_a(t) &= 3\sin(2\pi f_{1a}t) + 2\sin(2\pi f_{2a}t) \\ &= 3\sin(\pi t) + 2\sin(-\pi t) = 3\sin(\pi t) - 2\sin(\pi t) \\ &= \sin(\pi t) \end{aligned}$$

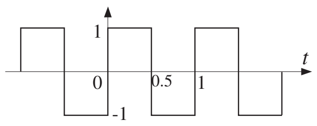
To find two other signals that are aliased with $x_a(t)$, we may shift the original frequencies f_1, f_2 by multiples of f_s. For example,

$$\begin{aligned} x_1(t) &= 3\sin(7\pi t) + 2\sin(5\pi t) \\ x_2(t) &= 3\sin(13\pi t) + 2\sin(11\pi t) \end{aligned}$$

where we replaced $\{f_1, f_2\}$ by $\{f_1 + f_s, f_2\} = \{3.5, 2.5\}$ for $x_1(t)$, and by $\{f_1 + 2f_s, f_2 + f_s\} = \{6.5, 5.5\}$ for $x_2(t)$. □

Example 1.4.6: Consider a periodic square wave with period $T_0 = 1$ sec, defined within its basic period $0 \le t \le 1$ by

$$x(t) = \begin{cases} 1, & \text{for } 0 < t < 0.5 \\ -1, & \text{for } 0.5 < t < 1 \\ 0, & \text{for } t = 0,\, 0.5,\, 1 \end{cases}$$



where t is in seconds. The square wave is sampled at rate f_s and the resulting samples are reconstructed by an *ideal* reconstructor as in Fig. 1.4.2. Determine the signal $x_a(t)$ that will appear at the output of the reconstructor for the two cases $f_s = 4$ Hz and $f_s = 8$ Hz. Verify that $x_a(t)$ and $x(t)$ agree at the sampling times $t = nT$.

Solution: The Fourier series expansion of the square wave contains odd harmonics at frequencies $f_m = m/T_0 = m$ Hz, $m = 1, 3, 5, 7, \dots$. It is given by

$$\begin{aligned} x(t) &= \sum_{m=1,3,5,\dots} b_m \sin(2\pi m t) = \\ &= b_1 \sin(2\pi t) + b_3 \sin(6\pi t) + b_5 \sin(10\pi t) + \cdots \end{aligned} \tag{1.4.4}$$

where $b_m = 4/(\pi m)$, $m = 1, 3, 5, \dots$. Because of the presence of an infinite number of harmonics, the square wave is not bandlimited and, thus, cannot be sampled properly at any rate. For the rate $f_s = 4$ Hz, only the $f_1 = 1$ harmonic lies within the Nyquist interval $[-2, 2]$ Hz. For the rate $f_s = 8$ Hz, only $f_1 = 1$ and $f_3 = 3$ Hz lie in $[-4, 4]$ Hz. The following table shows the true frequencies and the corresponding aliased frequencies in the two cases:

f_s	f	1	3	5	7	9	11	13	15	$\cdots$
4 Hz	$f \bmod (4)$	1	−1	1	−1	1	−1	1	−1	$\cdots$
8 Hz	$f \bmod (8)$	1	3	−3	−1	1	3	−3	−1	$\cdots$

Note the repeated patterns of aliased frequencies in the two cases. If a harmonic is aliased with $\pm f_1 = \pm 1$, then the corresponding term in Eq. (1.4.4) will appear (at the output of the reconstructor) as $\sin(\pm 2\pi f_1 t) = \pm\sin(2\pi t)$. And, if it is aliased with $\pm f_3 = \pm 3$, the term will appear as $\sin(\pm 2\pi f_3 t) = \pm\sin(6\pi t)$. Thus, for $f_s = 4$, the aliased signal will be

$$\begin{aligned} x_a(t) &= b_1\sin(2\pi t) - b_3\sin(2\pi t) + b_5\sin(2\pi t) - b_7\sin(2\pi t) + \cdots \\ &= (b_1 - b_3 + b_5 - b_7 + b_9 - b_{11} + \cdots)\sin(2\pi t) \\ &= A\sin(2\pi t) \end{aligned}$$

where

$$A = \sum_{k=0}^{\infty}(b_{1+4k} - b_{3+4k}) = \frac{4}{\pi}\sum_{k=0}^{\infty}\left[\frac{1}{1+4k} - \frac{1}{3+4k}\right] \tag{1.4.5}$$

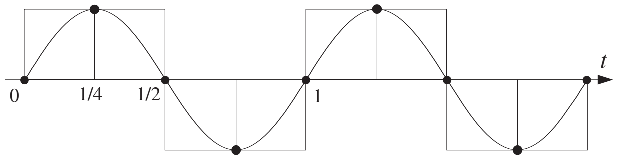
Similarly, for $f_s = 8$, grouping together the 1 and 3 Hz terms, we find the aliased signal

$$\begin{aligned} x_a(t) &= (b_1 - b_7 + b_9 - b_{15} + \cdots)\sin(2\pi t) + \\ &\quad + (b_3 - b_5 + b_{11} - b_{13} + \cdots)\sin(6\pi t) \\ &= B\sin(2\pi t) + C\sin(6\pi t) \end{aligned}$$

where

$$\begin{aligned} B &= \sum_{k=0}^{\infty}(b_{1+8k} - b_{7+8k}) = \frac{4}{\pi}\sum_{k=0}^{\infty}\left[\frac{1}{1+8k} - \frac{1}{7+8k}\right] \\ C &= \sum_{k=0}^{\infty}(b_{3+8k} - b_{5+8k}) = \frac{4}{\pi}\sum_{k=0}^{\infty}\left[\frac{1}{3+8k} - \frac{1}{5+8k}\right] \end{aligned} \tag{1.4.6}$$

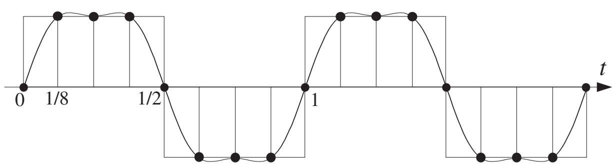
There are two ways to determine the aliased coefficients A, B, C. One is to demand that the *sampled* signals $x_a(nT)$ and $x(nT)$ agree. For example, in the first case we have $T = 1/f_s = 1/4$, and therefore, $x_a(nT) = A\sin(2\pi n/4) = A\sin(\pi n/2)$. The condition $x_a(nT) = x(nT)$ evaluated at $n = 1$ implies $A = 1$. The following figure shows $x(t)$, $x_a(t)$, and their samples:



Similarly, in the second case we have $T = 1/f_s = 1/8$, resulting in the sampled aliased signal $x_a(nT) = B\sin(\pi n/4) + C\sin(3\pi n/4)$. Demanding the condition $x_a(nT) = x(nT)$ at $n = 1, 2$ gives the two equations

$$\left.\begin{aligned} B\sin(\pi/4) + C\sin(3\pi/4) &= 1 \\ B\sin(\pi/2) + C\sin(3\pi/2) &= 1 \end{aligned}\right. \quad\Rightarrow\quad \begin{aligned} B + C &= \sqrt{2} \\ B - C &= 1 \end{aligned}$$

which can be solved to give $B = (\sqrt{2} + 1)/2$ and $C = (\sqrt{2} - 1)/2$. The following figure shows $x(t)$, $x_a(t)$, and their samples:



The second way of determining A, B, C is by evaluating the infinite sums of Eqs. (1.4.5) and (1.4.6). All three are special cases of the more general sum:

$$b(m, M) \equiv \frac{4}{\pi}\sum_{k=0}^{\infty}\left[\frac{1}{m+Mk} - \frac{1}{M-m+Mk}\right]$$

with $M > m > 0$. It can be computed as follows. Write

$$\frac{1}{m+Mk} - \frac{1}{M-m+Mk} = \int_0^{\infty}\left(e^{-mx} - e^{-(M-m)x}\right)e^{-Mkx}\,dx$$

then, interchange summation and integration and use the geometric series sum (for $x > 0$)

$$\sum_{k=0}^{\infty} e^{-Mkx} = \frac{1}{1-e^{-Mx}}$$

to get

$$b(m, M) = \frac{4}{\pi}\int_0^{\infty}\frac{e^{-mx} - e^{-(M-m)x}}{1-e^{-Mx}}\,dx$$

Looking this integral up in a table of integrals [38], we find:

$$b(m, M) = \frac{4}{M}\cot\left(\frac{m\pi}{M}\right)$$

The desired coefficients A, B, C are then:

$$A = b(1, 4) = \cot\left(\frac{\pi}{4}\right) = 1$$

$$B = b(1, 8) = \frac{1}{2}\cot\left(\frac{\pi}{8}\right) = \frac{\sqrt{2}+1}{2}$$

$$C = b(3, 8) = \frac{1}{2}\cot\left(\frac{3\pi}{8}\right) = \frac{\sqrt{2}-1}{2}$$

The above results generalize to any sampling rate $f_s = M$ Hz, where M is a multiple of 4. For example, if $f_s = 12$, we obtain

$$x_a(t) = b(1, 12)\sin(2\pi t) + b(3, 12)\sin(6\pi t) + b(5, 12)\sin(10\pi t)$$

and more generally

$$x_a(t) = \sum_{m=1,3,\dots,(M/2)-1} b(m, M)\sin(2\pi m t)$$

The coefficients $b(m, M)$ tend to the original Fourier series coefficients b_m in the continuous-time limit, $M \to \infty$. Indeed, using the approximation $\cot(x) \approx 1/x$, valid for small x, we obtain the limit

$$\lim_{M\to\infty} b(m, M) = \frac{4}{M}\cdot\frac{1}{\pi m/M} = \frac{4}{\pi m} = b_m$$

The table below shows the successive improvement of the values of the aliased harmonic coefficients as the sampling rate increases:

coefficients	4 Hz	8 Hz	12 Hz	16 Hz	∞
b_1	1	1.207	1.244	1.257	1.273
b_3	–	0.207	0.333	0.374	0.424
b_5	–	–	0.089	0.167	0.255
b_7	–	–	–	0.050	0.182

In this example, the sampling rates of 4 and 8 Hz, and any multiple of 4, were chosen so that all the harmonics outside the Nyquist intervals got aliased onto *harmonics* within the intervals. For other values of f_s, such as $f_s = 13$ Hz, it is possible for the aliased harmonics to fall on non-harmonic frequencies within the Nyquist interval; thus, changing not only the relative balance of the Nyquist interval harmonics, but also the frequency values. □

When we develop DFT algorithms, we will see that the aliased Fourier series coefficients for the above type of problem can be obtained by performing a DFT, provided that the *periodic analog signal* remains a *periodic discrete-time signal* after sampling.

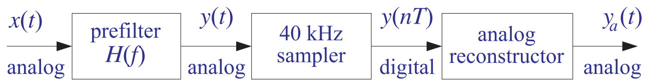
This requires that the sampling frequency f_s be an integral multiple of the fundamental harmonic of the given signal, that is, $f_s = Nf_1$. In such a case, the aliased coefficients can be obtained by an N-point DFT of the first N time samples $x(nT)$, $n = 0, 1, \dots, N-1$ of the analog signal. See Section 10.7.

Example 1.4.7: A sound wave has the form:

$$\begin{aligned} x(t) &= 2A\cos(10\pi t) + 2B\cos(30\pi t) \\ &\quad + 2C\cos(50\pi t) + 2D\cos(60\pi t) + 2E\cos(90\pi t) + 2F\cos(125\pi t) \end{aligned}$$

where t is in milliseconds. What is the frequency content of this signal? Which parts of it are audible and why?

This signal is prefiltered by an analog prefilter $H(f)$. Then, the output $y(t)$ of the prefilter is sampled at a rate of 40 kHz and immediately reconstructed by an ideal analog reconstructor, resulting into the final analog output $y_a(t)$, as shown below:

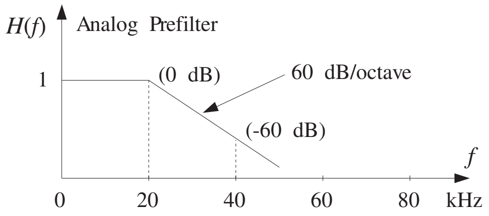


Determine the output signals $y(t)$ and $y_a(t)$ in the following cases:

(a) When there is no prefilter, that is, $H(f) = 1$ for all f.

(b) When $H(f)$ is the ideal prefilter with cutoff $f_s/2 = 20$ kHz.

(c) When $H(f)$ is a practical prefilter with specifications as shown below:



That is, it has a flat passband over the 20 kHz audio range and drops monotonically at a rate of 60 dB per octave beyond 20 kHz. Thus, at 40 kHz, which is an octave away, the filter's response will be down by 60 dB.

For the purposes of this problem, the filter's *phase response* may be ignored in determining the output $y(t)$. Does this filter help in removing the aliased components? What happens if the filter's attenuation rate is reduced to 30 dB/octave?

Solution: The six terms of $x(t)$ have frequencies:

$$\begin{array}{lll} f_A = 5\text{ kHz} & f_C = 25\text{ kHz} & f_E = 45\text{ kHz} \\ f_B = 15\text{ kHz} & f_D = 30\text{ kHz} & f_F = 62.5\text{ kHz} \end{array}$$

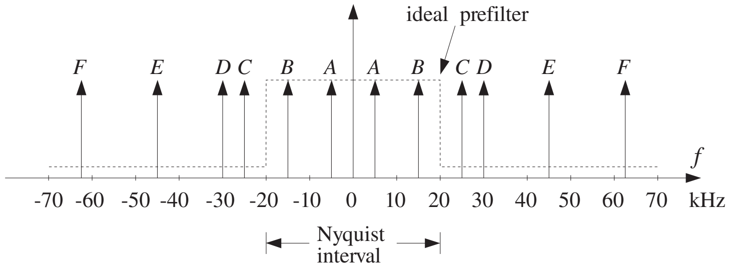
Only f_A and f_B are audible; the rest are inaudible. Our ears filter out all frequencies beyond 20 kHz, and we hear $x(t)$ as though it were the signal:

$$x_1(t) = 2A\cos(10\pi t) + 2B\cos(30\pi t)$$

Each term of $x(t)$ is represented in the frequency domain by two peaks at positive and negative frequencies, for example, the A-term has spectrum:

$$2A\cos(2\pi f_A t) = A\,e^{2\pi j f_A t} + A\,e^{-2\pi j f_A t} \quad \longrightarrow \quad A\,\delta(f - f_A) + A\,\delta(f + f_A)$$

Therefore, the spectrum of the input $x(t)$ will be as shown below:



The sampling process will replicate each of these peaks at multiples of $f_s = 40$ kHz. The four terms C, D, E, F lie outside the $[-20, 20]$ kHz Nyquist interval and therefore will be aliased with the following frequencies inside the interval:

$$\begin{array}{lll} f_C = 25 & \Rightarrow & f_{C,a} = f_C \bmod (f_s) = f_C - f_s = 25 - 40 = -15 \\ f_D = 30 & \Rightarrow & f_{D,a} = f_D \bmod (f_s) = f_D - f_s = 30 - 40 = -10 \\ f_E = 45 & \Rightarrow & f_{E,a} = f_E \bmod (f_s) = f_E - f_s = 45 - 40 = 5 \\ f_F = 62.5 & \Rightarrow & f_{F,a} = f_F \bmod (f_s) = f_F - 2f_s = 62.5 - 2\times 40 = -17.5 \end{array}$$

In case (a), if we do not use any prefilter at all, we will have $y(t) = x(t)$ and the reconstructed signal will be:

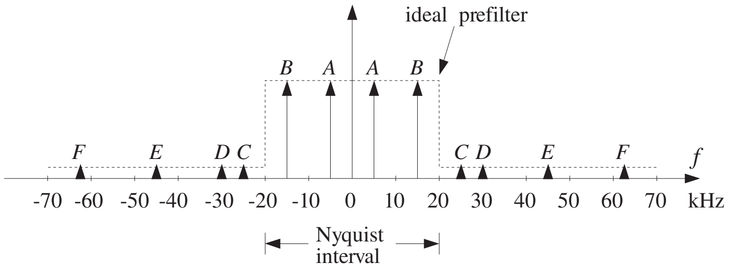
$$\begin{aligned} y_a(t) &= 2A\cos(10\pi t) + 2B\cos(30\pi t) \\ &\quad + 2C\cos(-2\pi 15t) + 2D\cos(-2\pi 10t) \\ &\quad + 2E\cos(2\pi 5t) + 2F\cos(-2\pi 17.5t) \\ &= 2(A+E)\cos(10\pi t) + 2(B+C)\cos(30\pi t) \\ &\quad + 2D\cos(20\pi t) + 2F\cos(35\pi t) \end{aligned}$$

where we replaced each out-of-band frequency with its aliased self, for example,

$$2C\cos(2\pi f_C t) \to 2C\cos(2\pi f_{C,a} t)$$

The *relative* amplitudes of the 5 and 15 kHz audible components have changed and, in addition, two *new* audible components at 10 and 17.5 kHz have been introduced. Thus, $y_a(t)$ will sound very different from $x(t)$.

In case (b), if an ideal prefilter with cutoff $f_s/2 = 20$ kHz is used, then its output will be the same as the audible part of $x(t)$, that is, $y(t) = x_1(t)$. The filter's effect on the input spectrum is to remove completely all components beyond the 20 kHz Nyquist frequency, as shown below:



Because the prefilter's output contains no frequencies beyond the Nyquist frequency, there will be no aliasing and after reconstruction the output would sound the same as the input, $y_a(t) = y(t) = x_1(t)$.

In case (c), if the practical prefilter $H(f)$ is used, then its output $y(t)$ will be:

$$\begin{aligned} y(t) &= 2A|H(f_A)|\cos(10\pi t) + 2B|H(f_B)|\cos(30\pi t) \\ &\quad + 2C|H(f_C)|\cos(50\pi t) + 2D|H(f_D)|\cos(60\pi t) \\ &\quad + 2E|H(f_E)|\cos(90\pi t) + 2F|H(f_F)|\cos(125\pi t) \end{aligned} \tag{1.4.7}$$

This follows from the steady-state sinusoidal response of a filter applied to the individual sinusoidal terms of $x(t)$, for example, the effect of $H(f)$ on A is:

$$2A\cos(2\pi f_A t)\xrightarrow{H} 2A|H(f_A)|\cos\big(2\pi f_A t+\theta(f_A)\big)$$

where in Eq. (1.4.7) we ignored the phase response $\theta(f_A)=\arg H(f_A)$. The basic conclusions of this example are not affected by this simplification.

Note that Eq. (1.4.7) applies also to cases (a) and (b). In case (a), we can replace:

$$|H(f_A)| = |H(f_B)| = |H(f_C)| = |H(f_D)| = |H(f_E)| = |H(f_F)| = 1$$

and in case (b):

$$|H(f_A)| = |H(f_B)| = 1,\quad |H(f_C)| = |H(f_D)| = |H(f_E)| = |H(f_F)| = 0$$

In case (c), because f_A and f_B are in the filter's passband, we still have

$$|H(f_A)| = |H(f_B)| = 1$$

To determine $|H(f_C)|, |H(f_D)|, |H(f_E)|, |H(f_F)|$, we must find how many octaves[†] away the frequencies f_C, f_D, f_E, f_F are from the $f_s/2 = 20$ kHz edge of the passband. These are given by:

$$\log_2\left(\frac{f_C}{f_s/2}\right) = \log_2\left(\frac{25}{20}\right) = 0.322$$

$$\log_2\left(\frac{f_D}{f_s/2}\right) = \log_2\left(\frac{30}{20}\right) = 0.585$$

$$\log_2\left(\frac{f_E}{f_s/2}\right) = \log_2\left(\frac{45}{20}\right) = 1.170$$

$$\log_2\left(\frac{f_F}{f_s/2}\right) = \log_2\left(\frac{62.5}{20}\right) = 1.644$$

and therefore, the corresponding filter attenuations will be:

$$\begin{aligned}
&\text{at } f_C: && 60\ \text{dB/octave}\times 0.322\ \text{octaves} = 19.3\ \text{dB}\\
&\text{at } f_D: && 60\ \text{dB/octave}\times 0.585\ \text{octaves} = 35.1\ \text{dB}\\
&\text{at } f_E: && 60\ \text{dB/octave}\times 1.170\ \text{octaves} = 70.1\ \text{dB}\\
&\text{at } f_F: && 60\ \text{dB/octave}\times 1.644\ \text{octaves} = 98.6\ \text{dB}
\end{aligned}$$

By definition, an amount of A dB attenuation corresponds to reducing $|H(f)|$ by a factor $10^{-A/20}$. For example, the relative drop of $|H(f)|$ with respect to the edge of the passband $|H(f_s/2)|$ is A dB if:

$$\frac{|H(f)|}{|H(f_s/2)|} = 10^{-A/20}$$

Assuming that the passband has 0 dB normalization, $|H(f_s/2)| = 1$, we find the following values for the filter responses:

[†] The number of octaves is the number of powers of two, that is, if $f_2 = 2^\nu f_1 \Rightarrow \nu = \log_2(f_2/f_1)$.

$$|H(f_C)| = 10^{-19.3/20} = \frac{1}{9}$$

$$|H(f_D)| = 10^{-35.1/20} = \frac{1}{57}$$

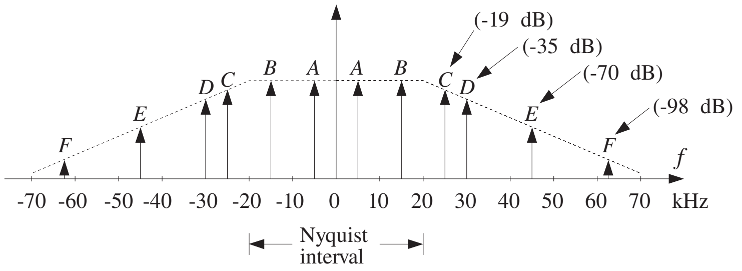
$$|H(f_E)| = 10^{-70.1/20} = \frac{1}{3234}$$

$$|H(f_F)| = 10^{-98.6/20} = \frac{1}{85114}$$

It follows from Eq. (1.4.7) that the output $y(t)$ of the prefilter will be:

$$\begin{aligned}
y(t) &= 2A\cos(10\pi t) + 2B\cos(30\pi t)\\
&\quad + \frac{2C}{9}\cos(50\pi t) + \frac{2D}{57}\cos(60\pi t)\\
&\quad + \frac{2E}{3234}\cos(90\pi t) + \frac{2F}{85114}\cos(125\pi t)
\end{aligned}\tag{1.4.8}$$

Its spectrum is shown below:



Notice how the inaudible out-of-band components have been attenuated by the prefilter, so that when they get aliased back into the Nyquist interval because of sampling, their distorting effect will be *much less*. The wrapping of frequencies into the Nyquist interval is the same as in case (a). Therefore, after sampling and reconstruction we will get:

$$\begin{aligned}
y_a(t) &= 2\left(A + \frac{E}{3234}\right)\cos(10\pi t) + 2\left(B + \frac{C}{9}\right)\cos(30\pi t)\\
&\quad + \frac{2D}{57}\cos(20\pi t) + \frac{2F}{85114}\cos(35\pi t)
\end{aligned}$$

Now, all aliased components have been reduced in magnitude. The component closest to the Nyquist frequency, namely f_C, causes the most distortion because it does not get attenuated much by the filter.

We will see in Section 1.5.3 that the prefilter's rate of attenuation in dB/octave is related to the filter's order N by $\alpha = 6N$ so that $\alpha = 60$ dB/octave corresponds to $60 = 6N$ or $N = 10$. Therefore, the given filter is already a fairly complex analog filter. Decreasing the filter's complexity to $\alpha = 30$ dB/octave, corresponding to filter order $N = 5$, would reduce all the attenuations by half, that is,

$$
\begin{aligned}
&\text{at } f_C: \quad 30\text{ dB/octave} \times 0.322 \text{ octaves} = 9.7\text{ dB}\\
&\text{at } f_D: \quad 30\text{ dB/octave} \times 0.585 \text{ octaves} = 17.6\text{ dB}\\
&\text{at } f_E: \quad 30\text{ dB/octave} \times 1.170 \text{ octaves} = 35.1\text{ dB}\\
&\text{at } f_F: \quad 30\text{ dB/octave} \times 1.644 \text{ octaves} = 49.3\text{ dB}
\end{aligned}
$$

and, in absolute units:

$$
\begin{aligned}
|H(f_C)| &= 10^{-9.7/20} = \frac{1}{3}\\
|H(f_D)| &= 10^{-17.6/20} = \frac{1}{7.5}\\
|H(f_E)| &= 10^{-35.1/20} = \frac{1}{57}\\
|H(f_F)| &= 10^{-49.3/20} = \frac{1}{292}
\end{aligned}
$$

Therefore, the resulting signal after reconstruction would be:

$$
\begin{aligned}
y_a(t) = 2\left(A + \frac{E}{57}\right)&\cos(10\pi t) + 2\left(B + \frac{C}{3}\right)\cos(30\pi t)\\
&+ \frac{2D}{7.5}\cos(20\pi t) + \frac{2F}{292}\cos(35\pi t)
\end{aligned}
\tag{1.4.9}
$$

Now the C and D terms are not as small and aliasing would still be significant. The situation can be remedied by oversampling, as discussed in the next example. □

Example 1.4.8: *Oversampling* can be used to reduce the attenuation requirements of the prefilter, and thus its order. Oversampling increases the gap between spectral replicas reducing aliasing and allowing less sharp cutoffs for the prefilter.

For the previous example, if we oversample by a factor of 2, $f_s = 2 \times 40 = 80$ kHz, the new Nyquist interval will be $[-40, 40]$ kHz. Only the $f_E = 45$ kHz and $f_F = 62.5$ kHz components lie outside this interval, and they will be aliased with

$$
\begin{aligned}
f_{E,a} &= f_E - f_s = 45 - 80 = -35\text{ kHz}\\
f_{F,a} &= f_F - f_s = 62.5 - 80 = -17.5\text{ kHz}
\end{aligned}
$$

Only $f_{F,a}$ lies in the audio band and will cause distortions, unless we attenuate f_F using a prefilter before it gets wrapped into the audio band. Without a prefilter, the reconstructed signal will be:

$$
\begin{aligned}
y_a(t) &= 2A\cos(10\pi t) + 2B\cos(30\pi t)\\
&\quad + 2C\cos(50\pi t) + 2D\cos(60\pi t)\\
&\quad + 2E\cos(-2\pi 35 t) + 2F\cos(-2\pi 17.5 t)\\
&= 2A\cos(10\pi t) + 2B\cos(30\pi t)\\
&\quad + 2C\cos(50\pi t) + 2D\cos(60\pi t) + 2E\cos(70\pi t) + 2F\cos(35\pi t)
\end{aligned}
$$

The audible components in $y_a(t)$ are:

$$
y_1(t) = 2A\cos(10\pi t) + 2B\cos(30\pi t) + 2F\cos(35\pi t)
$$

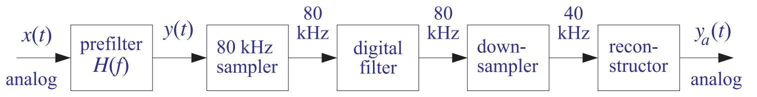
Thus, oversampling eliminated almost all the aliasing from the desired audio band. Note that two types of aliasing took place here, namely, the aliasing of the E component which remained *outside* the relevant audio band, and the aliasing of the F component which does represent distortion in the audio band.

Of course, one would not want to feed the signal $y_a(t)$ into an amplifier/speaker system because the high frequencies beyond the audio band might damage the system or cause nonlinearities. (But even if they were filtered out, the F component would still be there.) □

Example 1.4.9: *Oversampling and Decimation.* Example 1.4.8 assumed that sampling at 80 kHz could be maintained throughout the digital processing stages up to reconstruction. There are applications however, where the sampling rate must eventually be dropped down to its original value. This is the case, for example, in digital audio, where the rate must be reduced eventually to the standardized value of 44.1 kHz (for CDs) or 48 kHz (for DATs).

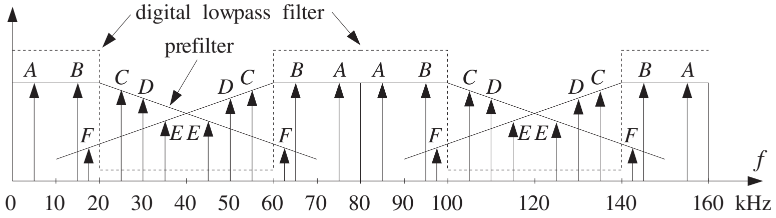
When the sampling rate is dropped, one must make sure that aliasing will not be reintroduced. In our example, if the rate is reduced back to 40 kHz, the C and D components, which were *inside* the $[-40, 40]$ kHz Nyquist interval with respect to the 80 kHz rate, would find themselves *outside* the $[-20, 20]$ kHz Nyquist interval with respect to the 40 kHz rate, and therefore would be aliased inside that interval, as in Example 1.4.7.

To prevent C and D, as well as E, from getting aliased into the audio band, one must *remove them* by a lowpass *digital* filter *before* the sampling rate is dropped to 40 kHz. Such a filter is called a digital *decimation* filter. The overall system is shown below.

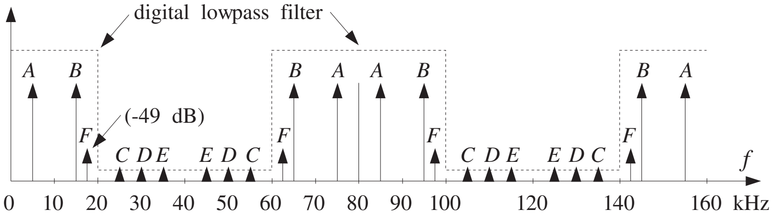


The downsampler in this diagram reduces the sampling rate from 80 down to 40 kHz by throwing away every other sample, thus, keeping only half the samples. This is equivalent to sampling at a 40 kHz rate.

The input to the digital filter is the sampled spectrum of $y(t)$, which is replicated at multiples of 80 kHz as shown below.



We have also assumed that the 30 dB/octave prefilter is present. The output of the digital filter will have spectrum as shown below.



The digital filter operates at the *oversampled rate* of 80 kHz and acts as a lowpass filter within the $[-40, 40]$ kHz Nyquist interval, with a cutoff of 20 kHz. Thus, it will remove the C, D, and E components, as well as any other component that lies between $20 \le |f| \le 60$ kHz.

However, because the digital filter is periodic in f with period $f_s = 80$ kHz, it *cannot* remove any components from the interval $60 \le f \le 100$. Any components of the analog input $y(t)$ that lie in that interval would be aliased into the interval $60-80 \le f - f_s \le 100-80$, which is the desired audio band $-20 \le f - f_s \le 20$. This is what happened to the F component, as can be seen in the above figure.

The frequency components of $y(t)$ in $60 \le |f| \le 100$ can be removed *only by a prefilter*, prior to sampling and replicating the spectrum. For example, our low-complexity 30 dB/octave prefilter would provide 47.6 dB attenuation at 60 kHz. Indeed, the number of octaves from 20 to 60 kHz is $\log_2(60/20) = 1.585$ and the attenuation there will be 30 dB/octave $\times$ 1.584 octaves = 47.6 dB.

The prefilter, being monotonic beyond 60 kHz, would suppress all potential aliased components beyond 60 kHz by more than 47.6 dB. At 100 kHz, it would provide $30\times\log_2(100/20) = 69.7$ dB attenuation. At $f_F = 62.5$ kHz, it provides 49.3 dB suppression, as was calculated in Example 1.4.7, that is, $|H(f_F)| = 10^{-49.3/20} = 1/292$.

Therefore, assuming that the digital filter has *already removed* the C, D, and E components, and that the aliased F component has been *sufficiently attenuated* by the prefilter, we can now drop the sampling rate down to 40 kHz.

At the reduced 40 kHz rate, if we use an ideal reconstructor, it would extract only the components within the $[-20, 20]$ kHz band and the reconstructed output will be:

$$y_a(t) = 2A\cos(10\pi t) + 2B\cos(30\pi t) + \frac{2F}{292}\cos(35\pi t)$$

which has a much attenuated aliased component F. This is to be compared with Eq. (1.4.9), which used the *same* prefilter but no oversampling. Oversampling in conjunction with digital decimation helped eliminate the most severe aliased components, C and D.

In summary, with oversampling, the complexity of the analog prefilter can be reduced and *traded off* for the complexity of a digital filter which is much easier to design and cheaper to implement with programmable DSPs. As we will see in Chapter 2, another benefit of oversampling is to reduce the *number of bits* representing each quantized sample. The connection between sampling rate and the savings in bits is discussed in Section 2.2. The subject of oversampling, decimation, interpolation, and the design and implementation of digital decimation and interpolation filters will be discussed in detail in Chapter 14. □

1.4.2 Rotational Motion

A more intuitive way to understand the sampling properties of sinusoids is to consider a representation of the complex sinusoid $x(t) = e^{2\pi jft}$ as a wheel rotating with a frequency of f revolutions per second. The wheel is seen in a dark room by means of a strobe light flashing at a rate of f_s flashes per second. The rotational frequency in [radians/sec] is $\Omega = 2\pi f$. During the time interval T between flashes, the wheel turns by an angle:

$$\boxed{\omega = \Omega T = 2\pi f T = \frac{2\pi f}{f_s}} \tag{1.4.10}$$

This quantity is called the *digital frequency* and is measured in units of [radians/sample]. It represents a convenient normalization of the physical frequency f. In terms of ω, the sampled sinusoid reads simply

$$x(nT) = e^{2\pi jfTn} = e^{j\omega n}$$

In units of ω, the Nyquist frequency $f = f_s/2$ becomes $\omega = \pi$ and the Nyquist interval becomes $[-\pi, \pi]$. The replicated set $f + mf_s$ becomes

$$\frac{2\pi(f + mf_s)}{f_s} = \frac{2\pi f}{f_s} + 2\pi m = \omega + 2\pi m$$

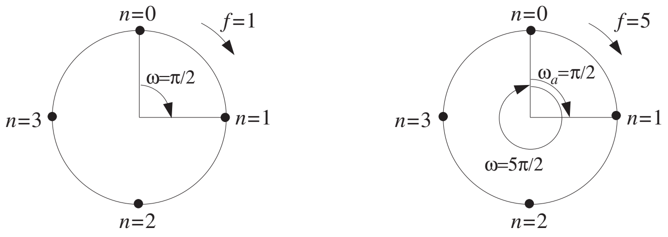
Because the frequency $f = f_s$ corresponds to $\omega = 2\pi$, the aliased frequency given in Eq. (1.4.3) becomes in units of ω:

$$\omega_a = \omega \bmod (2\pi)$$

The quantity $f/f_s = fT$ is also called the *digital frequency* and is measured in units of [cycles/sample]. It represents another convenient normalization of the physical frequency axis, with the Nyquist interval corresponding to $[-0.5, 0.5]$.

In terms of the rotating wheel, fT represents the number of revolutions turned during the flashing interval T. If the wheel were actually turning at the higher frequency $f + mf_s$, then during time T it would turn by $(f + mf_s)T = fT + mf_sT = fT + m$ revolutions, that is, it would cover m whole additional revolutions. An observer would miss these extra m revolutions completely. The *perceived* rotational speed for an observer is always given by $f_a = f \bmod (f_s)$. The next two examples illustrate these remarks.

Example 1.4.10: Consider two wheels turning clockwise, one at $f_1 = 1$ Hz and the other at $f_2 = 5$ Hz, as shown below. Both are sampled with a strobe light flashing at $f_s = 4$ Hz. Note that the second one is turning at $f_2 = f_1 + f_s$.



The first wheel covers $f_1T = f_1/f_s = 1/4$ of a revolution during $T = 1/4$ second. Its angle of rotation during that time interval is $\omega_1 = 2\pi f_1/f_s = 2\pi/4 = \pi/2$ radians. During the sampled motion, an observer would observe the sequence of points $n = 0, 1, 2, 3, \dots$ and would conclude that the wheel is turning at a speed of 1/4 of a revolution in 1/4 second, or,

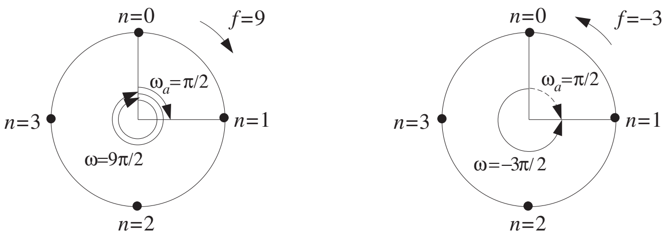
$$\frac{1/4\text{ cycles}}{1/4\text{ sec}} = 1\text{ Hz}$$

Thus, the observer would perceive the correct speed and sense of rotation. The second wheel, on the other hand, is actually turning by $f_2T = f_2/f_s = 5/4$ revolutions in 1/4 second, with an angle of rotation $\omega_2 = 5\pi/2$. Thus, it covers one whole extra revolution compared to the first one. However, the observer would still observe the same sequence of points $n = 0, 1, 2, 3, \dots$, and would conclude again that the wheel is turning at 1/4 revolution in 1/4 second, or, 1 Hz. This result can be obtained quickly using Eq. (1.4.3):

$$f_{2a} = f_2 \bmod(f_s) = 5 \bmod(4) = 5 - 4 = 1$$

Thus, in this case the perceived speed is wrong, but the sense of rotation is still correct.

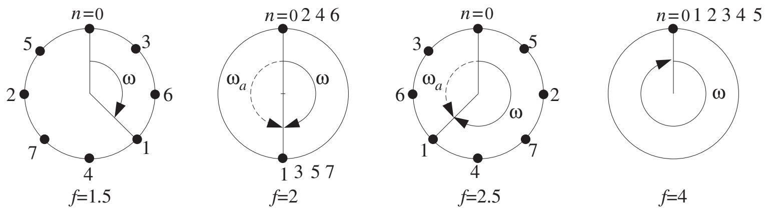
In the next figure, we see two more wheels, one turning clockwise at $f_3 = 9$ Hz and the other counterclockwise at $f_4 = -3$ Hz.



The negative sign signifies here the sense of rotation. During $T = 1/4$ sec, the third wheel covers $f_3T = 9/4$ revolutions, that is, two whole extra revolutions over the f_1 wheel. An observer would again see the sequence of points $n = 0, 1, 2, 3, \dots$, and would conclude that f_3 is turning at 1 Hz. Again, we can quickly compute, $f_{3a} = f_3 \bmod(f_s) = 9 \bmod(4) = 9 - 2\cdot 4 = 1$ Hz.

The fourth wheel is more interesting. It covers $f_4T = -3/4$ of a revolution in the counterclockwise direction. An observer captures the motion every 3/4 of a counterclockwise revolution. Thus, she will see the sequence of points $n = 0, 1, 2, 3, \dots$, arriving at the conclusion that the wheel is turning at 1 Hz in the clockwise direction. In this case, both the perceived speed and sense of rotation are wrong. Again, the same conclusion can be reached quickly using $f_{4a} = f_4 \bmod(f_s) = (-3) \bmod(4) = -3 + 4 = 1$ Hz. Here, we added one f_s in order to bring f_4 within the Nyquist interval $[-2, 2]$. □

Example 1.4.11: The following figure shows four wheels rotating clockwise at $f = 1.5, 2, 2.5, 4$ Hz and sampled at $f_s = 4$ Hz by a strobe light.



This example is meant to show that if a wheel is turning by less than *half* of a revolution between sampling instants, that is, $fT < 1/2$ or $\omega = 2\pi fT < \pi$, then the motion is perceived correctly and there is no aliasing. The conditions $fT < 1/2$ or $\omega < \pi$ are equivalent to the sampling theorem condition $f_s > 2f$. But if the wheel is turning by more than half of a revolution, it will be perceived as turning in the opposite direction and aliasing will occur.

The first wheel turns by $fT = 3/8$ of a revolution every T seconds. Thus, an observer would see the sequence of points $n = 0, 1, 2, 3, \dots$ and perceive the right motion.

The second wheel is turning by exactly half of a revolution $fT = 1/2$ or angle $\omega = 2\pi fT = \pi$ radians. An observer would perceive an up-down motion and lose sense of direction, not being able to tell which way the wheel is turning.

The third wheel turns by more than half of a revolution, $fT = 5/8$. An observer would see the sequence of points $n = 0, 1, 2, 3, \dots$, corresponding to successive rotations by $\omega = 5\pi/4$ radians. An observer always perceives the motion in terms of the lesser angle of rotation, and therefore will think that the wheel is turning the other way by an angle $\omega_a = \omega \bmod(2\pi) = (5\pi/4)\bmod(2\pi) = 5\pi/4 - 2\pi = -3\pi/4$ or frequency $f_a = -(3/8\text{ cycle})/(1/4\text{ sec}) = -1.5$ Hz.

The fourth wheel will appear to be stationary because $f = f_s = 4$ and the motion is sampled once every revolution, $\omega = 2\pi$. The perceived frequency will be $f_a = f \bmod(f_s) = 4 \bmod(4) = 4 - 4 = 0$. □

1.4.3 DSP Frequency Units

Figure 1.4.4 compares the various frequency scales that are commonly used in DSP, and the corresponding Nyquist intervals. A sampled sinusoid takes the form in these units:

$$e^{2\pi jfTn} = e^{2\pi j(f/f_s)n} = e^{j\Omega Tn} = e^{j\omega n}$$

being expressed more simply in terms of ω. Sometimes f is normalized with respect to the Nyquist frequency $f_N = f_s/2$, that is, in units of f/f_N. In this case, the Nyquist interval becomes $[-1, 1]$. In multirate applications, where successive digital processing stages operate at different sampling rates, the most convenient set of units is simply in terms of f. In fixed-rate applications, the units of ω or f/f_s are the most convenient.

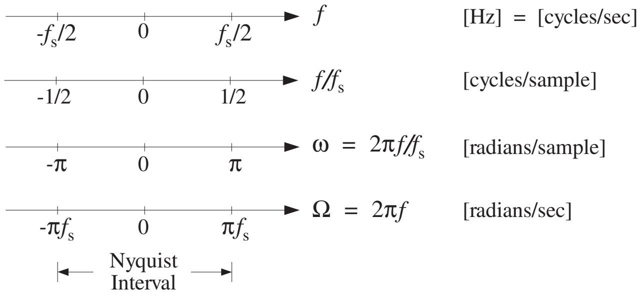


Fig. 1.4.4 Commonly used frequency units.

1.5 Spectra of Sampled Signals

Next, we discuss the effects of sampling using Fourier transforms. Figure 1.3.1 shows an ideal sampler that *instantaneously* measures the analog signal $x(t)$ at the sampling instants $t = nT$. The output of the sampler can be considered to be an *analog* signal consisting of the linear superposition of impulses occurring at the sampling times, with each impulse weighted by the corresponding sample value. Thus, the *sampled signal* is

$$\hat{x}(t) = \sum_{n=-\infty}^{\infty} x(nT)\,\delta(t - nT) \tag{1.5.1}$$

In practical sampling, each sample must be held constant for a short period of time, say τ seconds, in order for the A/D converter to accurately convert the sample to digital format. This holding operation may be achieved by a sample/hold circuit. In this case, the sampled signal will be:

$$x_{\text{flat}}(t) = \sum_{n=-\infty}^{\infty} x(nT)\,p(t - nT) \tag{1.5.2}$$

where $p(t)$ is a flat-top pulse of duration of τ seconds such that $\tau \ll T$. Ideal sampling corresponds to the limit $\tau \to 0$. Figure 1.5.1 illustrates the ideal and practical cases.

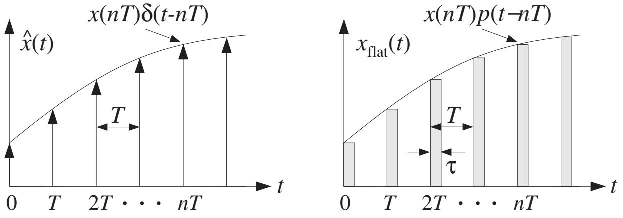


Fig. 1.5.1 Ideal and practical sampling.

We will consider only the ideal case, Eq. (1.5.1), because it captures all the essential features of the sampling process. Our objective is to determine the spectrum of the sampled signal $\hat{x}(t)$ and compare it with the spectrum of the original signal $x(t)$. Problem 1.21 explores practical sampling.

Our main result will be to express the spectrum of $\hat{x}(t)$ in two ways. The first relates the sampled spectrum to the *discrete-time samples* $x(nT)$ and leads to the discrete-time Fourier transform. The second relates it to the *original spectrum* and implies the spectrum replication property that was mentioned earlier.

1.5.1 Discrete-Time Fourier Transform

The spectrum of the sampled signal $\hat{x}(t)$ is the Fourier transform:

$$\hat{X}(f) = \int_{-\infty}^{\infty} \hat{x}(t)\,e^{-2\pi jft}\,dt \tag{1.5.3}$$

Inserting Eq. (1.5.1) into Eq. (1.5.3) and interchanging integration and summation, we obtain:

$$\begin{aligned}\hat{X}(f) &= \int_{-\infty}^{\infty} \sum_{n=-\infty}^{\infty} x(nT)\,\delta(t - nT)\,e^{-2\pi jft}\,dt \\ &= \sum_{n=-\infty}^{\infty} x(nT) \int_{-\infty}^{\infty} \delta(t - nT)\,e^{-2\pi jft}\,dt \qquad \text{or,}\end{aligned}$$

$$\boxed{\hat{X}(f) = \sum_{n=-\infty}^{\infty} x(nT)\,e^{-2\pi jfTn}} \tag{1.5.4}$$

This is the first way of expressing $\hat{X}(f)$. Several remarks are in order:

1. *DTFT.* Eq. (1.5.4) is known as the *Discrete-Time Fourier Transform* (DTFT)[†] of the sequence of samples $x(nT)$. $\hat{X}(f)$ is computable only from the knowledge of the sample values $x(nT)$.

2. *Periodicity.* $\hat{X}(f)$ is a *periodic* function of f with period f_s, hence, $\hat{X}(f+f_s) = \hat{X}(f)$. This follows from the fact that $e^{-2\pi jfTn}$ is periodic in f. Because of this periodicity, one may restrict the frequency interval to just one period, namely, the Nyquist interval, $[-f_s/2, f_s/2]$.

   The periodicity in f implies that $\hat{X}(f)$ will extend over the entire frequency axis, in accordance with our expectation that the sampling process introduces high frequencies into the original spectrum. Although not obvious yet, the periodicity in f is related to the periodic replication of the original spectrum.

3. *Fourier Series.* Mathematically, Eq. (1.5.4) may be thought of as the *Fourier series* expansion of the periodic function $\hat{X}(f)$, with the samples $x(nT)$ being the corresponding Fourier series *coefficients*. Thus, $x(nT)$ may be recovered from $\hat{X}(f)$ by the inverse Fourier series:

$$\boxed{x(nT) = \frac{1}{f_s}\int_{-f_s/2}^{f_s/2} \hat{X}(f)\,e^{2\pi jfTn}\,df = \int_{-\pi}^{\pi} \hat{X}(\omega)\,e^{j\omega n}\,\frac{d\omega}{2\pi}} \tag{1.5.5}$$

[†] Not to be confused with the *Discrete Fourier Transform* (DFT), which is a special case of the DTFT.

where in the second equation we changed variables from f to $\omega = 2\pi f/f_s$.[‡] Eq. (1.5.5) is the *inverse DTFT* and expresses the discrete-time signal $x(nT)$ as a superposition of discrete-time sinusoids $e^{j\omega n}$.

4. *Numerical Approximation.* Eq. (1.5.4) may be thought of as a *numerical approximation* to the frequency spectrum of the original analog signal $x(t)$. Indeed, using the definition of integrals, we may write approximately,

$$X(f)=\int_{-\infty}^{\infty}x(t)e^{-2\pi jft}\,dt\,\simeq\sum_{n=-\infty}^{\infty}x(nT)e^{-2\pi jfnT}\cdot T\qquad\text{or,}$$

$$X(f)\simeq T\hat{X}(f)\tag{1.5.6}$$

This approximation becomes exact in the continuous-time limit:

$$X(f)=\lim_{T\to 0}T\hat{X}(f)\tag{1.5.7}$$

It is precisely this limiting result and the approximation of Eq. (1.5.6) that *justify* the use of discrete Fourier transforms to compute actual spectra of analog signals.

5. *Practical Approximations.* In an actual spectrum computation, two additional approximations must be made before anything can be computed:

   (a) We must keep only a *finite* number of time samples $x(nT)$, say L samples, $n = 0, 1, 2, \dots, L-1$, so that Eq. (1.5.4) is computed approximately by the truncated sum:

   $$\hat{X}(f)\simeq\hat{X}_L(f)=\sum_{n=0}^{L-1}x(nT)e^{-2\pi jfTn}\tag{1.5.8}$$

   This approximation leads to the concept of a *time window* and the related effects of *smearing* and *leakage* of the spectrum. These concepts are central in the area of spectral analysis and will be discussed in Chapter 10.

   (b) We must decide on a *finite* set of frequencies f at which to evaluate $\hat{X}(f)$. Proper choice of this set allows the development of various efficient computational algorithms for the DFT, such as the Fast Fourier Transform (FFT), presented also in Chapter 10.

6. *z-transform.* Finally, we note that Eq. (1.5.4) leads to the concept of the *z-transform*, much like the ordinary Fourier transform leads to the Laplace transform. Setting $z = e^{j\omega} = e^{2\pi jfT}$, we may write Eq. (1.5.4) as the z-transform[†]

$$\hat{X}(z)=\sum_{n=-\infty}^{\infty}x(nT)z^{-n}\tag{1.5.9}$$

[‡] Abusing the notation slightly, we wrote $\hat{X}(\omega)$ for $\hat{X}(f)$.

[†] Again, abusing the notation, we wrote $\hat{X}(z)$ for $\hat{X}(f)$.

7. *Starred Laplace Transform.* Often, especially in the control systems literature, the ideally sample signal $\hat{x}(t)$, and its Laplace and Fourier transforms, and the resulting z-transform, are denoted by the following "star" notation, not to be confused with complex conjugation,

$$\begin{aligned}
x^*(t)&=\sum_{n=-\infty}^{\infty}x(nT)\delta(t-nT)\\
X^*(s)&=\int_{-\infty}^{\infty}x^*(t)\,e^{-st}\,dt=\sum_{n=-\infty}^{\infty}x(nT)\,e^{-sTn}\\
X^*(f)&=X^*(s)\Big|_{s=2\pi jf}=\sum_{n=-\infty}^{\infty}x(nT)e^{-2\pi jfTn}\\
X^*(z)&=X^*(s)\Big|_{z=e^{sT}}=\sum_{n=-\infty}^{\infty}x(nT)z^{-n}
\end{aligned}\tag{1.5.10}$$

and the following abused notation is used for this z-transform, $X^*(z)=\mathcal{Z}[X(s)]$, referred to as the *z-transform of a Laplace transform*, that is,

$$X^*(z)=\mathcal{Z}[X(s)]=X^*(s)\Big|_{z=e^{sT}}=\sum_{n=-\infty}^{\infty}x(nT)\,z^{-n}\tag{1.5.11}$$

which consists of the following series of steps going from $X(s)$ to $X^*(z)$, first perform an inverse Laplace transform on $X(s)$ to get the analog time signal $x(t)$, then, sample $x(t)$ at the sampling instants $t_n = nT$ to obtain the sampled signal $x(nT)$, and finally, perform a z-transform,

$$\boxed{X(s)\xrightarrow{\mathcal{L}^{-1}}x(t)\xrightarrow{\text{sample}}x(nT)\xrightarrow{\mathcal{Z}}X^*(z)=\sum_{n=-\infty}^{\infty}x(nT)z^{-n}}\tag{1.5.12}$$

and these can be combined into the more accurate but awkward notation,

$$X^*(z)=\mathcal{Z}\left[\mathcal{L}^{-1}[X(s)]\Big|_{\text{sampled}}\right]\tag{1.5.13}$$

We will use this notation in Chap. 22 in discussing digital control systems.

1.5.2 Spectrum Replication

Next, we show the spectrum replication property by deriving the precise relationship between the spectrum $\hat{X}(f)$ of the sampled signal $\hat{x}(t)$ and the original spectrum $X(f)$ of the analog signal $x(t)$.

The nth term $x(nT)\delta(t-nT)$ in Eq. (1.5.1) may be replaced by $x(t)\delta(t-nT)$ because the term is nonzero only at $t = nT$. Then, $x(t)$ can be factored out of the sum in Eq. (1.5.1) as a common factor:

$$\hat{x}(t)=x(t)\sum_{n=-\infty}^{\infty}\delta(t-nT)\equiv x(t)s(t)\tag{1.5.14}$$

Thinking of this as the *modulation* of the “carrier” $s(t)$ by the “baseband” signal $x(t)$, we expect to get frequency translations of the original spectrum, much like the AM modulation of a sinusoidal carrier. The frequency translation effect may be seen by expanding the (periodic in time) sampling function $s(t)$ into its Fourier series representation as a linear combination of harmonics. It is easily shown that

$$s(t)=\sum_{n=-\infty}^{\infty}\delta(t-nT)=\frac{1}{T}\sum_{m=-\infty}^{\infty}e^{2\pi jmf_s t} \tag{1.5.15}$$

which expresses the sampling function $s(t)$ as a linear combination of sinusoidal carriers, each causing its own frequency shift. Writing Eq. (1.5.14) as

$$\hat{x}(t)=x(t)s(t)=\frac{1}{T}\sum_{m=-\infty}^{\infty}x(t)e^{2\pi jmf_s t}$$

and using the modulation property of Fourier transforms, which states that if $X(f)$ is the transform of $x(t)$ then $X(f-f_c)$ is the transform of $x(t)e^{2\pi jf_c t}$, we obtain by taking Fourier transforms of both sides,

$$\boxed{\hat{X}(f)=\frac{1}{T}\sum_{m=-\infty}^{\infty}X(f-mf_s)} \tag{1.5.16}$$

This represents the *periodic replication* of the original spectrum $X(f)$ at intervals of the sampling rate f_s. Fig. 1.5.2 shows $T\hat{X}(f)$ as the sum of the periodic replicas of $X(f)$.

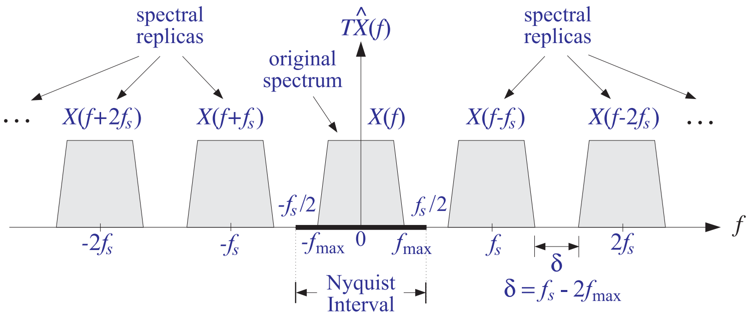


Fig. 1.5.2 Spectrum replication caused by sampling.

Another way to prove Eq. (1.5.16) is as follows. Because $\hat{x}(t)$ is the product of $x(t)$ and $s(t)$, its Fourier transform will be the convolution of the corresponding transforms, that is,

$$\hat{X}(f)=\int_{-\infty}^{\infty}X(f-f')S(f')\,df' \tag{1.5.17}$$

On the other hand, it follows from Eq. (1.5.15) that the Fourier transform of $s(t)$ will be the sum of the transforms of the individual harmonics:

$$S(f)=\frac{1}{T}\sum_{m=-\infty}^{\infty}\delta(f-mf_s) \tag{1.5.18}$$

Inserting this into Eq. (1.5.17) and interchanging the summation over m with the integration over f', we obtain

$$\hat{X}(f)=\frac{1}{T}\sum_{m=-\infty}^{\infty}\int_{-\infty}^{\infty}X(f-f')\,\delta(f'-mf_s)\,df' = \frac{1}{T}\sum_{m=-\infty}^{\infty}X(f-mf_s)$$

Combining Eqs. (1.5.4) and (1.5.16), we obtain the two alternative expressions for the spectrum $\hat{X}(f)$

$$\boxed{\hat{X}(f)=\sum_{n=-\infty}^{\infty}x(nT)e^{-2\pi jfTn}=\frac{1}{T}\sum_{m=-\infty}^{\infty}X(f-mf_s)} \tag{1.5.19}$$

This is known as the *Poisson summation* formula. We also see from Fig. 1.5.2 that as we let $T\to 0$, or equivalently, $f_s\to\infty$, the replicas move out to infinity leaving behind only the original spectrum $X(f)$. Therefore, Eq. (1.5.7) follows.

We emphasize that Eq. (1.5.19) holds for *arbitrary* signals $x(t)$, not necessarily bandlimited ones. In the special case when $x(t)$ is bandlimited to some maximum frequency $f_{\max}$, as suggested by Fig. 1.5.2, we immediately obtain the sampling theorem condition, Eq. (1.3.2).

It is seen in Fig. 1.5.2 that the replicas are separated from each other by a distance $\delta = f_s - 2f_{\max}$, known as the *guard band*. It follows that the replicas will *not* overlap if $\delta\geq 0$, or equivalently, $f_s\geq 2f_{\max}$. But they will overlap if $f_s < 2f_{\max}$ or $\delta < 0$ and aliasing of frequencies will take place as the tails of the replicas enter into the Nyquist interval and add to the original spectrum, distorting it. This case is shown in Fig. 1.5.3.

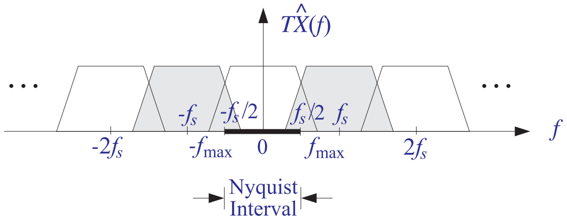


Fig. 1.5.3 Aliasing caused by overlapping spectral replicas.

It is evident by inspecting Fig. 1.5.2 that if the signal is bandlimited and f_s is large enough so that the replicas do not overlap, then the portion of the sampled signal spectrum $\hat{X}(f)$ that lies *within* the Nyquist interval $[-f_s/2, f_s/2]$ will be identical to the original spectrum $X(f)$, that is,

$$\boxed{T\hat{X}(f)=X(f),\qquad \text{for}\quad -\frac{f_s}{2}\leq f\leq\frac{f_s}{2}} \tag{1.5.20}$$

This is an important result for DSP. Not only does it make possible the analog reconstruction of the sampled signal, but it also guarantees that any *subsequent* digital processing of the sampled signal will be applied to the original spectrum $X(f)$ and not to some aliased and distorted version thereof.

For example, a subsequent digital filtering operation will transform the input samples $x(nT)$ into a sequence of output samples $y(nT)$. Just like analog filtering, digital filtering is equivalent to spectral shaping in the frequency domain. If the digital filter has frequency response $H_{\text{DSP}}(f)$, the spectrum $\hat{X}(f)$ of the input sequence will be reshaped into the output spectrum:

$$\hat{Y}(f) = H_{\text{DSP}}(f)\hat{X}(f)$$

If Eq. (1.5.20) holds, then the digital filter will reshape the original spectrum $X(f)$. Note that because all digital filters have periodic frequency responses, the periodicity of the sampled spectrum is preserved by the digital filtering operation. Therefore, the output samples could be recovered from Eq. (1.5.5)

$$y(nT) = \int_{-\pi}^{\pi} \hat{Y}(\omega)e^{j\omega n}\frac{d\omega}{2\pi} = \int_{-\pi}^{\pi} H_{\text{DSP}}(\omega)\hat{X}(\omega)e^{j\omega n}\frac{d\omega}{2\pi}$$

If the spectrum $X(f)$ is not bandlimited, or, if it is bandlimited but the sampling rate f_s is so low that the replicas overlap, then Eq. (1.5.20) does not hold. Any subsequent filtering will reshape the *wrong* spectrum. Therefore, it is essential to use a *lowpass antialiasing prefilter*, as shown in Fig. 1.3.5, to bandlimit the input spectrum to within the Nyquist interval, so that the resulting replicas after sampling will not overlap.

Example 1.5.1: Consider a pure sinusoid of frequency f_0, $x(t) = e^{2\pi jf_0 t}$. Its Fourier transform is the spectral line $X(f) = \delta(f - f_0)$. It follows from Eq. (1.5.16) that the sampled sinusoid:

$$\hat{x}(t) = \sum_{n=-\infty}^{\infty} x(nT)\delta(t - nT) = \sum_{n=-\infty}^{\infty} e^{2\pi jf_0 Tn}\delta(t - nT)$$

will have Fourier spectrum

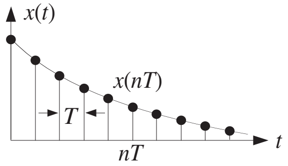
$$\hat{X}(f) = \frac{1}{T}\sum_{m=-\infty}^{\infty}\delta(f - f_0 - mf_s)$$

Thus, the spectrum of the sampled sinusoid consists of all the frequencies in the replicated set $\{f_0 + mf_s, m = 0, \pm 1, \pm 2, \dots\}$ in accordance with Fig. 1.3.2 and our remarks in Sections 1.4 and 1.3. □

Example 1.5.2: This example illustrates the effect of sampling on a non-bandlimited signal and the degree to which the portion of the spectrum $\hat{X}(f)$ within the Nyquist interval approximates the original spectrum $X(f)$. Consider the exponentially decaying signal and its spectrum:

$$x(t) = e^{-at}u(t)$$

$$X(f) = \frac{1}{a + 2\pi jf}$$



The frequency spectrum of the sampled signal $\hat{x}(t)$ may be obtained in two ways. Using Eq. (1.5.4)

$$\hat{X}(f) = \sum_{n=-\infty}^{\infty} x(nT)e^{-2\pi jfTn} = \sum_{n=0}^{\infty} e^{-aTn}e^{-2\pi jfTn}$$

and summing the geometric series, we get

$$\hat{X}(f) = \frac{1}{1 - e^{-aT}e^{-2\pi jfT}} = \frac{1}{1 - e^{-aT}e^{-j\omega}}$$

Its magnitude square is

$$|\hat{X}(f)|^2 = \frac{1}{1 - 2e^{-aT}\cos(2\pi fT) + e^{-2aT}}$$

The periodicity in f is evident because the dependence on f comes through the periodic cosine function. Alternatively, we may use Eq. (1.5.16) and sum the replicas of the original spectrum to get

$$\hat{X}(f) = \frac{1}{T}\sum_{m=-\infty}^{\infty} X(f - mf_s) = \frac{1}{T}\sum_{m=-\infty}^{\infty}\frac{1}{a + 2\pi j(f - mf_s)}$$

Combining the two expression for $\hat{X}(f)$, we obtain the not-so-obvious identity in the parameters a, f, T:[†]

$$\frac{1}{T}\sum_{m=-\infty}^{\infty}\frac{1}{a + 2\pi j(f - mf_s)} = \frac{1}{1 - e^{-aT}e^{-2\pi jfT}} \qquad (*)$$

The left graph in Fig. 1.5.4 compares the periodic spectrum $|T\hat{X}(f)|^2$ with the original analog spectrum $|X(f)|^2 = 1/\big(a^2 + (2\pi f)^2\big)$. The spectra are shown in decibels, that is, $20\log_{10}|X(f)|$. The parameter a was $a = 0.2$ sec^{-1}. Two values of the sampling rate $f_s = 1/T$ are shown, $f_s = 1$ Hz and $f_s = 2$ Hz. The two Nyquist intervals are $[-0.5, 0.5]$ Hz and $[-1, 1]$ Hz, respectively. Outside these intervals, the sampled spectra repeat periodically.

Notice that even with the scale factor T taken into account, the two spectra $X(f)$ and $T\hat{X}(f)$ are very different from each other. However, within the central Nyquist interval $[-f_s/2, f_s/2]$, they agree approximately, especially at low frequencies. This approximation gets better as f_s increases.

The limit as $T \to 0$ or $f_s \to \infty$ can be seen explicitly in this example. Using the approximation $e^{-x} \simeq 1 - x$, valid for small x, or L'Hospital's rule, we obtain

$$\lim_{T\to 0} T\hat{X}(f) = \lim_{T\to 0}\frac{T}{1 - e^{-aT}e^{-2\pi jfT}} = \frac{1}{a + 2\pi jf} = X(f)$$

In the right graph of Fig. 1.5.4, we show the effect of using a length-L *time window* and approximating the spectrum by Eq. (1.5.8). The parameter values were $a = 0.2$, $f_s = 2$, and $L = 10$ samples.

That figure compares what we would *like* to compute, that is, $|X(f)|^2$, with what we can *at best* hope to compute based on our sampled signal, $|T\hat{X}(f)|^2$, and with what we can *actually* compute based on a finite record of samples, $|T\hat{X}_L(f)|^2$.

[†] Please see a corrected discussion at the end of this example.

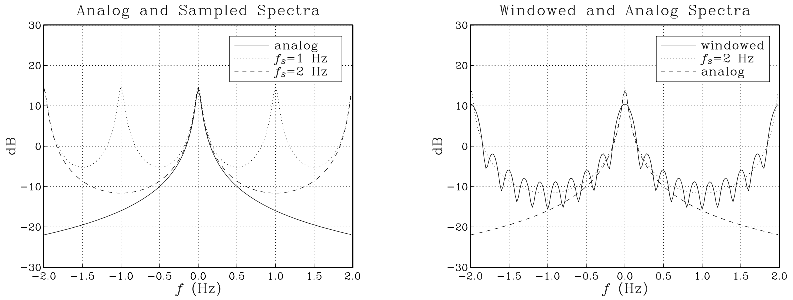


Fig. 1.5.4 Spectra of analog, sampled, and windowed signals.

The windowed spectrum $|T\hat{X}_L(f)|^2$ can be improved by taking longer L and using a non-rectangular window, such as a Hamming window. At best, however, it will approach the sampled spectrum $|T\hat{X}(f)|^2$ and not $|X(f)|^2$. The approximation of $X(f)$ by $T\hat{X}(f)$ can be improved only by increasing the sampling rate f_s.

The quantity $\hat{X}_L(f)$ can be computed by sending the L samples $x(nT)= e^{-anT}$, $n = 0,1,\dots,L-1$ into a general DFT routine. In this particular example, $\hat{X}_L(f)$ can also be computed in closed form. Using the finite geometric series:

$$\sum_{n=0}^{L-1} x^n = \frac{1-x^L}{1-x}$$

we obtain:

$$\hat{X}_L(f)=\sum_{n=0}^{L-1} e^{-aTn}e^{-2\pi jfTn} = \frac{1-e^{-aTL}e^{-2\pi jfTL}}{1-e^{-aT}e^{-2\pi jfT}}$$

It is evident that $\hat{X}_L(f)\to\hat{X}(f)$ as $L\to\infty$.

Correction. The Poisson sum identity in Eq. (*) should be corrected to read:

$$\frac{1}{T}\sum_{m=-\infty}^{\infty}\frac{1}{a+2\pi j(f-mf_s)} = \frac{1}{1-e^{-aT}e^{-2\pi jfT}}-\frac{1}{2}$$

This requires that the sampled signal $x(nT)$ in this example be redefined such that $x(0)=1/2$ instead of $x(0)=1$, which gives,

$$\hat{X}(f)=\sum_{n=0}^{\infty}x(nT)e^{-2\pi jfTn} = \sum_{n=0}^{\infty}\big(e^{-aTn}-\frac{1}{2}\delta(n)\big)e^{-2\pi jfTn} = \frac{1}{1-e^{-aT}e^{-2\pi jfT}}-\frac{1}{2}$$

The Poisson summation formula requires that the signal $x(t)$ be continuous for all t. The present example has a discontinuity at $t=0$ arising from the unit step $u(t)$. Choosing $x(0)$ to be equal to $\big(x(0+)+x(0-)\big)/2 = (1+0)/2 = 1/2$, enables the use of the Poisson summation formula. The issue has been discussed in connection with the impulse-invariance filter design method.[†] Except for a level shift, the basic conclusions of this example remain the same. Similar redefinitions must be introduced also in Problem 1.13.

The above Poisson sum identity can be found in standard mathematical tables.[‡] Indeed, setting $z=(2\pi fT-jaT)/2$, it is not hard to verify that the identity takes on the standard form:

$$\cot z = \sum_{m=-\infty}^{\infty}\frac{1}{z-\pi m} = \frac{1}{z}+2z\sum_{m=1}^{\infty}\frac{1}{z^2-\pi^2m^2}$$

1.5.3 Practical Antialiasing Prefilters

An ideal analog prefilter is shown in Fig. 1.5.5. It acts as an ideal lowpass filter removing all frequency components of the analog input signal that lie beyond the Nyquist frequency $f_s/2$.

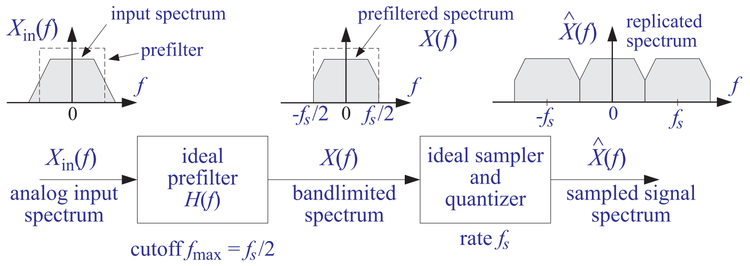


Fig. 1.5.5 Ideal antialiasing prefilter.

The antialiasing prefilters used in practice are not ideal and do not completely remove all the frequency components outside the Nyquist interval. Thus, some aliasing will take place. However, by proper design the prefilters may be made as good as desired and the amount of aliasing reduced to tolerable levels. A practical antialiasing lowpass filter is shown in Fig. 1.5.6. Its passband $[-f_{\text{pass}}, f_{\text{pass}}]$ is usually taken to be the *frequency range of interest* for the application at hand and must lie entirely within the Nyquist interval.

The prefilter must be essentially *flat* over this passband in order not to distort the frequencies of interest. Even if it is not completely flat over the passband, it can be "equalized" *digitally* at a subsequent processing stage by a digital filter, say $H_{\text{EQ}}(f)$, whose frequency response is the *inverse* of the response of the prefilter *over* the pass-

[†] R. A. Gabel and R. A. Roberts, *Signals and Linear Systems*, 3/e, Wiley, New York, 1987; L. B. Jackson, "A Correction to Impulse Invariance," *IEEE Signal Process. Lett.*, **7**, 273 (2000); W. F. G. Mecklenbräuker, "Remarks on and Correction to the Impulse Invariant Method for the Design of IIR Digital Filters," *Signal Processing*, **80**, 1687 (2000); E. Eitelberg, "Convolution invariance and corrected impulse invariance,", *ibid.*, **86**, 1116 (2006); S. R. Nelatury, "Additional correction to the impulse invariance method for the design of IIR digital filters," *Digital Signal Processing*, **17**, 530 (2007).

[‡] M. Abramowitz and I. A. Stegun, Eds., *Handbook of Mathematical Functions*, Dover, New York, 1968, p.75.

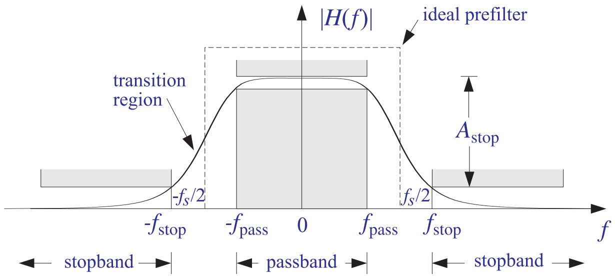


Fig. 1.5.6 Practical antialiasing lowpass prefilter.

band range:

$$\boxed{H_{\rm EQ}(f)=\frac{1}{H(f)}}\,,\qquad \text{for}\quad -f_{\rm pass}\le f\le f_{\rm pass}$$

The digital filter $H_{\rm EQ}(f)$, being periodic with period f_s, cannot be the inverse of the prefilter over the entire frequency axis, but it can be the inverse over the passband.

The *stopband* frequency $f_{\rm stop}$ of the prefilter and the *minimum stopband attenuation* $A_{\rm stop}$ in dB must be chosen appropriately to minimize aliasing effects. It will become evident from the examples below that $f_{\rm stop}$ must be chosen as

$$\boxed{f_{\rm stop}=f_s-f_{\rm pass}} \qquad (1.5.21)$$

or, equivalently,

$$f_s=f_{\rm pass}+f_{\rm stop}$$

This places the Nyquist frequency $f_s/2$ exactly in the middle of the transition region of the prefilter, as shown in Fig. 1.5.6. The attenuation of the filter in *decibels* is defined in terms of its magnitude response by:

$$\boxed{A(f)=-20\log_{10}\left|\frac{H(f)}{H(f_0)}\right|}\qquad \text{(attenuation in dB)}$$

where f_0 is a convenient reference frequency, typically taken to be at DC for a lowpass filter. Therefore, the stopband specification of the filter, depicted in this figure, is $A(f)\ge A_{\rm stop}$, for $|f|\ge f_{\rm stop}$.

Transfer functions of analog filters typically drop like a power $H(s)\sim 1/s^N$ for large s, where N is the filter order. Thus, their magnitude response drops like $|H(f)|\sim 1/f^N$ for large f, and their attenuation will be, up to an *additive* constant,

$$A(f)=-20\log_{10}\left|1/f^N\right| = \alpha_{10}\log_{10}f\,,\qquad \text{(for large } f\text{)} \qquad (1.5.22)$$

where α_{10} is the attenuation in *dB per decade* defined by:

$$\alpha_{10}=20N \qquad \text{(dB per decade)}$$

It represents the increase in attenuation when f is changed by a factor of ten, that is, $A(10f)-A(f)=\alpha_{10}$. Engineers also like to measure attenuation in *dB per octave*, that is, the amount of change per *doubling* of f. This is obtained by using logs in base two, that is, writing Eq. (1.5.22) in the form:

$$A(f)=\alpha_2\log_2 f=\alpha_{10}\log_{10}f$$

where α_2 is in dB/octave and is related to α_{10} by:

$$\alpha_2=\alpha_{10}\log_{10}2=6N \qquad \text{(dB per octave)}$$

Figure 1.5.5 shows the effect on the input spectrum $X_{\rm in}(f)$ of an ideal prefilter with a sharp cutoff. For a practical prefilter, the output spectrum is given by:

$$X(f)=H(f)X_{\rm in}(f)$$

or, in terms of attenuations in dB:

$$A_X(f)=A(f)+A_{X_{\rm in}}(f) \qquad (1.5.23)$$

where $A_X(f)=-20\log_{10}|X(f)/X(f_0)|$ and similarly for $A_{X_{\rm in}}(f)$. Thus, attenuations are *additive*. The spectrum $X(f)$ will be replicated by the subsequent sampling operation and therefore, the amount of attenuation in $A_X(f)$ will determine the degree of overlapping of the spectral replicas, that is, the degree of aliasing.

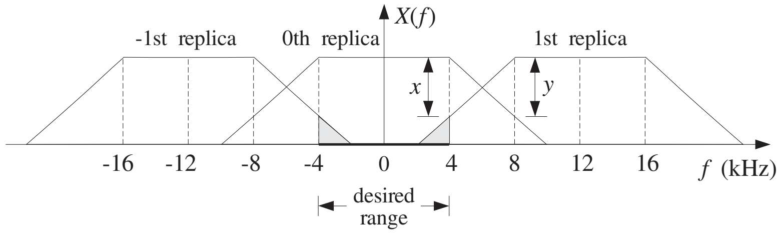
The specifications of the prefilter can be adjusted so that its attenuation $A(f)$, in *combination* with the attenuation $A_{X_{\rm in}}(f)$ of the input spectrum, will result in sufficient attenuation of $X(f)$ to reduce the amount of aliasing within the desired frequency band. The next few examples illustrate these remarks.

Example 1.5.3: The frequency *range of interest* of an analog signal extends to 4 kHz. Beyond 4 kHz, the spectrum attenuates at a rate of 15 dB per octave. Ideally, we would sample at a rate of 8 kHz provided the sampling operation is preceded by a perfect lowpass antialiasing prefilter with cutoff of 4 kHz. As a practical alternative to designing a perfect prefilter, we decide to sample at the higher rate of 12 kHz.

(a) If we do not use any prefilter at all, determine the amount of aliasing that will be introduced by the sampling process into the frequency range of interest, that is, into the 4 kHz range.

(b) We wish to suppress the aliased components *within* the frequency range of interest by more than 50 dB. Determine the least stringent specifications of the lowpass antialiasing prefilter that must be used.

Solution: Both parts are answered with the help of the figure below, which shows the original spectrum and its first replicas centered at $\pm f_s=\pm 12$ kHz.



By the even symmetry of the spectra, it follows that the left tail of the 1st replica will be the same as the right tail of the 0th replica. Thus, the indicated attenuations x and y at frequencies 4 and 8 kHz will be equal, $x = y$.

If we do not use any prefilter, the attenuation at 8 kHz will be $y = 15$ dB because the 0th replica attenuates by 15 dB per octave starting at 4 kHz. The aliased components within the desired 4 kHz range correspond to the shaded portion of the left side of the 1st replica that has entered into the 4 kHz interval. They are suppressed by more than x dB. Thus, $x = y = 15$ dB. This probably represents too much aliasing to be tolerable.

If we use a prefilter, its passband must extend over the desired 4 kHz range. Therefore, $f_{\text{pass}} = 4$ kHz and $f_{\text{stop}} = f_s - f_{\text{pass}} = 12 - 4 = 8$ kHz. Because attenuations are additive in dB, the total attenuation y at 8 kHz will now be the sum of the attenuation due to the signal, that is, 15 dB, and the attenuation due to the prefilter, say A_{stop} dB. The equality $y = x$ and the requirement that $x \geq 50$ dB lead to

$$y = 15 + A_{\text{stop}} = x \geq 50 \qquad \Rightarrow \qquad A_{\text{stop}} \geq 50 - 15 = 35 \text{ dB}$$

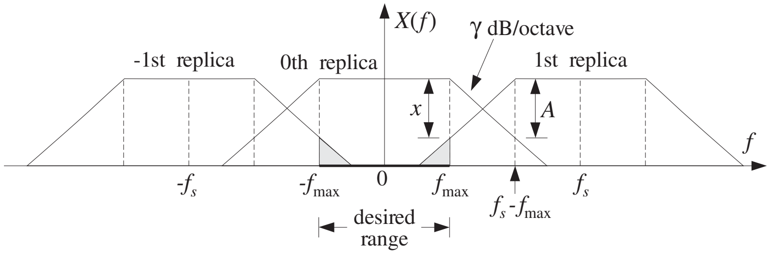
Thus, the specifications of the prefilter are a fairly flat passband over the ±4 kHz range and a stopband starting at 8 kHz with minimum attenuation of 35 dB. □

Example 1.5.4: The significant frequency range of a signal extends to $f_{\max}$. Beyond $f_{\max}$, the spectrum attenuates by α dB/octave. We have available an off-the-shelf antialiasing prefilter that has a flat passband up to $f_{\max}$ and attenuates by β dB/octave beyond that. It is required that within the $f_{\max}$ range of interest, the aliased components due to sampling be suppressed by more than A dB. Show that the *minimum* sampling rate that we should use is given by

$$f_s = f_{\max} + 2^{A/\gamma} f_{\max}$$

where $\gamma = \alpha + \beta$.

Solution: We refer to the following figure, which shows the 0th and ±1st replicas.



The passband edge is at $f_{\text{pass}} = f_{\max}$ and the stopband edge at $f_{\text{stop}} = f_s - f_{\max}$. Beyond the desired $f_{\max}$ range, the total attenuation (in dB) of the 0th replica will be the sum of the attenuations of the signal and the prefilter. In the notation of Eq. (1.5.23), it will be given as function of frequency by

$$A_X(f) = \alpha \log_2\left(\frac{f}{f_{\max}}\right) + \beta \log_2\left(\frac{f}{f_{\max}}\right) = \gamma \log_2\left(\frac{f}{f_{\max}}\right)$$

where we have normalized the attenuation to 0 dB at $f = f_{\max}$. This is the mathematical expression of the statement that the total attenuation will be γ dB per octave.

By the even symmetry of the spectra, we have $x = A_X(f_{\text{stop}}) = A_X(f_s - f_{\max})$. Thus, the requirement that $x \geq A$ gives the condition

$$A_X(f_s - f_{\max}) \geq A \qquad \Rightarrow \qquad \gamma \log_2\left(\frac{f_s - f_{\max}}{f_{\max}}\right) \geq A$$

Solving this as an equality gives the minimum acceptable rate f_s. If α and β had been given in dB/decade instead of dB/octave, the above condition would be valid with $\log_{10}$ instead of $\log_2$ resulting in $f_s = f_{\max} + 10^{A/\gamma} f_{\max}$. Note that the previous example corresponds to the case $A = \gamma$ giving $f_s = f_{\max} + 2f_{\max} = 3f_{\max}$. □

The above examples show that to accommodate practical specifications for antialiasing prefilters, the sampling rates must be somewhat *higher* than the minimum Nyquist rate. The higher the rate, the less complex the prefilter. This idea is carried further in the method of *oversampling*, whereby the input is sampled at rates that are many times higher than the Nyquist rate. The replicas become very far separated, allowing the use of low quality, inexpensive analog prefilters. Oversampling methods will be discussed in Chapter 14.

1.6 Analog Reconstructors

We saw in Section 1.4.1 that an ideal reconstructor is an ideal lowpass filter with cutoff the Nyquist frequency $f_s/2$. Here, we derive this result and also consider practical reconstructors.

Analog reconstruction represents some sort of *lowpass* filtering of the sampled signal. This can be seen in Fig. 1.6.1, where practical reconstruction has been accomplished by filling the gaps between samples by holding the current sample value constant till the next sample. This is the staircase or sample/hold reconstructor.

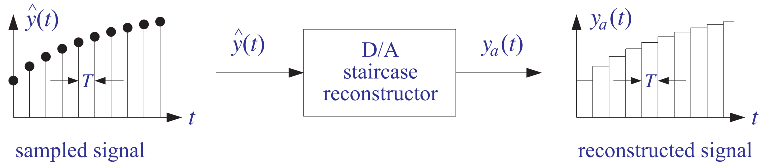


Fig. 1.6.1 Staircase reconstructor.

It must be clear from this figure that *any* reasonable way of filling the gaps between samples will result in some sort of reconstruction. Filling the gaps results in a *smoother* signal than the sampled signal. In frequency-domain language, the higher frequencies in the sampled signal are removed, that is, the sampled signal is lowpass filtered. Thus, any reconstructor may be viewed as an analog *lowpass* filter, as shown in Fig. 1.6.2.

We will determine the form of the impulse response $h(t)$ of the reconstructor both for ideal and practical reconstruction. The relationship of the reconstructed output $y_a(t)$ to the input samples $y(nT)$ can be found by inserting the sampled input signal

$$\hat{y}(t) = \sum_{n=-\infty}^{\infty} y(nT)\,\delta(t - nT)$$

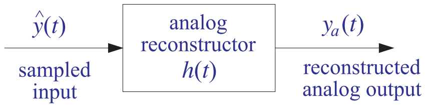


Fig. 1.6.2 Analog reconstructor as a lowpass filter.

into the convolutional equation of the reconstructor

$$y_a(t)=\int_{-\infty}^{\infty} h(t-t')\hat{y}(t')\,dt'$$

It then follows that:

$$\boxed{y_a(t)=\sum_{n=-\infty}^{\infty} y(nT)h(t-nT)} \tag{1.6.1}$$

It states that the way to fill the gaps between samples is to start at the current sample $y(nT)$ and interpolate from it following the shape of $h(t)$ until the next sample. More precisely, a copy of $h(t)$ must be attached at each sample $y(nT)$, and all such contributions must be summed over—the resulting curve being the reconstructed analog signal. In the frequency domain, Eq. (1.6.1) becomes

$$Y_a(f)=H(f)\hat{Y}(f) \tag{1.6.2}$$

where $\hat{Y}(f)$ is the replicated spectrum given by Eq. (1.5.16)

$$\hat{Y}(f)=\frac{1}{T}\sum_{m=-\infty}^{\infty} Y(f-mf_s)$$

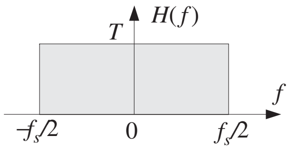
1.6.1 Ideal Reconstructors

For *perfect* or ideal reconstruction one must require that $Y_a(f)$ be identical to the original analog spectrum $Y(f)$. If the spectrum $Y(f)$ is bandlimited and its replicas do not overlap, then within the Nyquist interval, $T\hat{Y}(f)$ will agree with $Y(f)$ in accordance with Eq. (1.5.20), that is,

$$\hat{Y}(f)=\frac{1}{T}Y(f)\,,\qquad \text{for}\quad -\frac{f_s}{2}\le f\le\frac{f_s}{2} \tag{1.6.3}$$

The ideal reconstruction filter $H(f)$ is an ideal lowpass filter with cutoff $f_s/2$, defined as follows:

$$\boxed{H(f)=\begin{cases} T, & \text{if } |f|\le f_s/2 \\ 0, & \text{otherwise}\end{cases}}$$



The value T for the passband gain is justified below. As shown in Fig. 1.6.3, such a filter will extract the central replica and remove all other replicas. Using Eq. (1.6.3), we have within the Nyquist interval:

$$Y_a(f)=H(f)\hat{Y}(f)=T\cdot\frac{1}{T}Y(f)=Y(f)$$

where the filter's gain factor T canceled the $1/T$ factor in the spectrum.

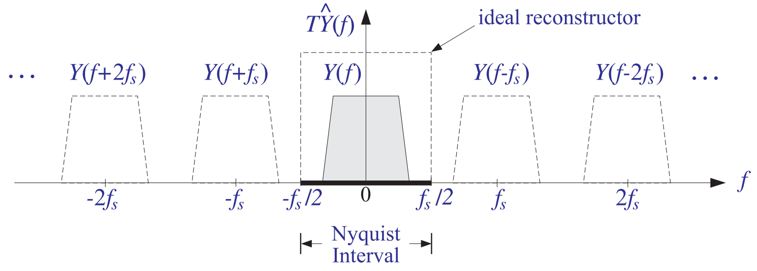


Fig. 1.6.3 Ideal reconstructor in frequency domain.

The same relationship also holds trivially ($0\equiv 0$) outside the Nyquist interval. Thus, we have $Y_a(f)=Y(f)$, for all f, which implies that the reconstructed analog signal $y_a(t)$ will be identical to the original signal that was sampled, $y_a(t)=y(t)$. Combining this with Eq. (1.6.1), we obtain the *Shannon sampling theorem* [47–55] expressing the bandlimited signal $y(t)$ in terms of its samples $y(nT)$:

$$\boxed{y(t)=\sum_{n=-\infty}^{\infty} y(nT)h(t-nT)} \tag{1.6.4}$$

The impulse response of the ideal reconstructor can be obtained from the inverse Fourier transform of $H(f)$:

$$h(t)=\int_{-\infty}^{\infty} H(f)e^{2\pi jft}\,df=\int_{-f_s/2}^{f_s/2} Te^{2\pi jft}\,df,\qquad \text{or,}$$

$$\boxed{h(t)=\frac{\sin(\pi t/T)}{\pi t/T}=\frac{\sin(\pi f_s t)}{\pi f_s t}}\qquad\text{(ideal reconstructor)} \tag{1.6.5}$$

It is shown in Fig. 1.6.4. Unfortunately, the ideal reconstructor is not realizable. Its impulse response is not causal, having an infinite anticausal part. Therefore, alternative reconstructors, such as the staircase one, are used in practice.

An *approximation* to the ideal reconstructor, obtained by truncating it to finite length, is used in the design of digital FIR interpolation filters for oversampling and sample-rate conversion applications. We will discuss it in Chapter 14.

1.6.2 Staircase Reconstructors

The staircase reconstructor shown in Fig. 1.6.1 is the simplest and most widely used reconstructor in practice. It generates a staircase approximation to the original signal.

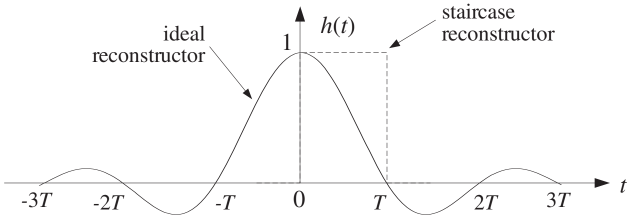
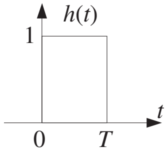


Fig. 1.6.4 Impulse response of ideal reconstructor.

Note the similarity of this operation to practical sampling, where $h(t)$ is a sampling pulse $p(t)$ having a very narrow width $\tau \ll T$. By contrast, the impulse response of the staircase reconstructor must have duration of T seconds in order to fill the entire gap between samples. Thus, $h(t)$ is given by:

$$\boxed{h(t)=u(t)-u(t-T)=\begin{cases}1, & \text{if } 0\le t\le T\\ 0, & \text{otherwise}\end{cases}}$$



where $u(t)$ is the unit step. The staircase output, although smoother than the sampled input, still contains spurious high-frequency components arising from the sudden jumps in the staircase levels from sample to sample. This spurious frequency content may be seen by computing the frequency response of the reconstructor. The Laplace transform of $h(t)=u(t)-u(t-T)$ is

$$H(s)=\frac{1}{s}-\frac{1}{s}e^{-sT}$$

from which we obtain the Fourier transform by setting $s=2\pi jf$:

$$\boxed{H(f)=\frac{1}{2\pi jf}\big(1-e^{-2\pi jfT}\big)=T\,\frac{\sin(\pi fT)}{\pi fT}\,e^{-\pi jfT}} \tag{1.6.6}$$

It is shown in Fig. 1.6.5 in comparison to the ideal reconstructor. Notice that it vanishes at integral *multiples* of f_s — exactly where the replicas caused by sampling are centered. The spurious high frequencies mentioned above are those beyond the Nyquist frequency $f_s/2$.

Thus, the reconstructor does not completely eliminate the replicated spectral images as the ideal reconstructor does. Figure 1.6.6 compares the spectra before and after the staircase reconstructor, that is, the effect of the multiplication $Y_a(f)=H(f)\hat{Y}(f)$.

1.6.3 Anti-Image Postfilters

The surviving spectral replicas may be removed by an additional *lowpass postfilter*, called an *anti-image* postfilter, whose cutoff is the Nyquist frequency $f_s/2$. This operation is shown in Fig. 1.6.7.

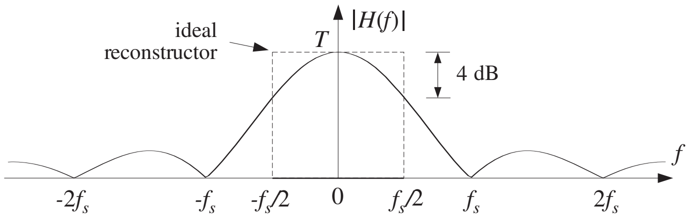


Fig. 1.6.5 Frequency response of staircase reconstructor.

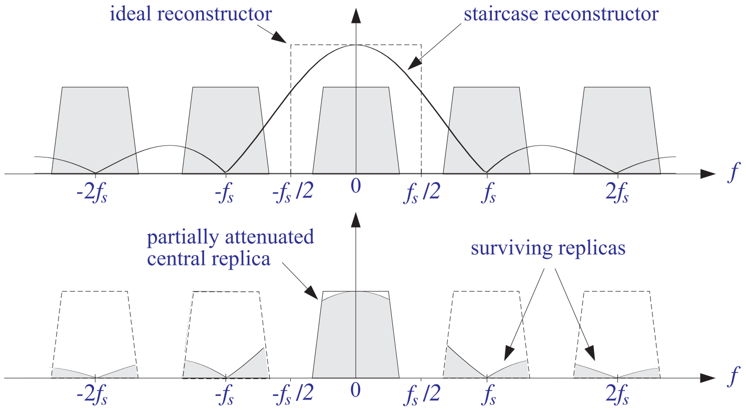


Fig. 1.6.6 Frequency response of staircase reconstructor.

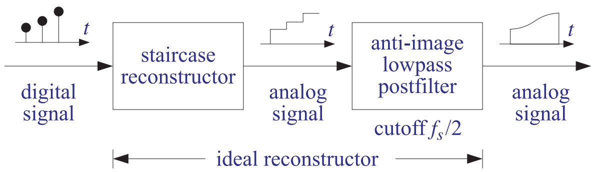


Fig. 1.6.7 Analog anti-image postfilter.

In the time domain, the postfilter has the effect of rounding off the corners of the staircase output making it smoother. In the frequency domain, the combined effect of the staircase reconstructor followed by the anti-image postfilter is to remove the spectral replicas as much as possible, that is, to emulate the ideal reconstructor. The final reconstructed spectrum at the output of the postfilter is shown in Fig. 1.6.8.

The reason for using this two-stage reconstruction procedure is the *simplicity* of implementation of the staircase reconstruction part. A typical D/A converter will act as such a reconstructor. The digital code for each sample is applied to the DAC for T seconds generating an analog output that remains constant during T.

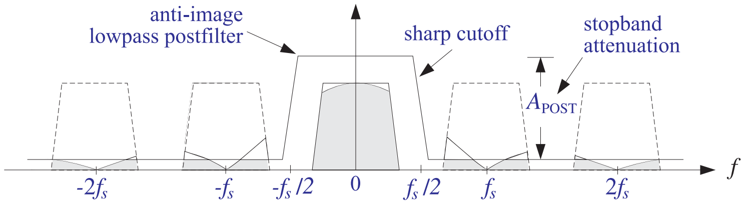


Fig. 1.6.8 Spectrum after postfilter.

The specifications of the postfilter are similar to those of an antialiasing prefilter, namely, a flat passband and cutoff frequency equal to the Nyquist frequency $f_s/2$. High-quality DSP applications, such as digital audio, require the use of postfilters (and prefilters) with very stringent specifications. In deciding the specifications of a postfilter, one must take into account the effect of the staircase D/A which does part of the reconstruction.

The main function of the postfilter is to remove the remnants of the spectral images that survived the staircase D/A reconstructor. It can also be used to equalize the rolloff of the staircase response within the Nyquist interval. As shown in Fig. 1.6.5, the staircase reconstructor is not flat within the Nyquist interval, tending to attenuate more near the Nyquist frequency $f_s/2$. The maximum attenuation suffered at $f_s/2$ is about 4 dB. This can be seen as follows:

$$-20\log_{10}\left|\frac{H(f_s/2)}{H(0)}\right| = -20\log_{10}\left|\frac{\sin(\pi/2)}{\pi/2}\right| = 3.9\text{ dB}$$

This attenuation can be compensated by proper design of the passband of the anti-image postfilter. But more conveniently, it can be compensated *digitally* before analog reconstruction, by designing an equalizing digital filter whose response matches the *inverse* of $H(f)$ *over* the Nyquist interval.

Similar techniques were mentioned in Section 1.5.3 for equalizing the imperfect passband of the antialiasing prefilter. The use of high-quality digital filters to perform these equalizations improves the overall quality of the digital processing system. By contrast, analog compensation techniques would be more cumbersome and expensive. The combined equalizer, DAC, and postfilter are shown in Fig. 1.6.9. The frequency response of the equalizer is defined as the inverse of the DAC, as given by Eq. (1.6.6):

$$H_{EQ}(f)=\frac{T}{H(f)}=\frac{\pi f T}{\sin(\pi f T)}\,e^{\pi j f T}, \qquad \text{for} \quad -\frac{f_s}{2}\le f\le\frac{f_s}{2} \tag{1.6.7}$$

It is shown in Fig. 1.6.10. As a digital filter, $H_{EQ}(f)$ is periodic outside the Nyquist interval with period f_s. We will design such inverse filters later using the frequency sampling design method of Section 11.4. Some designs are presented in Chapter 14.

The equalizer filter transforms the sequence $y(nT)$ into the "equalized" sequence $y_{EQ}(nT)$, which is then fed into the DAC and postfilter. The frequency spectrum of $y_{EQ}(nT)$ is $\hat{Y}_{EQ}(f)=H_{EQ}(f)\hat{Y}(f)$. The spectrum of the staircase output of the DAC will be $Y_a(f)=H(f)\hat{Y}_{EQ}(f)$. Therefore, the final reconstructed spectrum at the output of the postfilter will be,

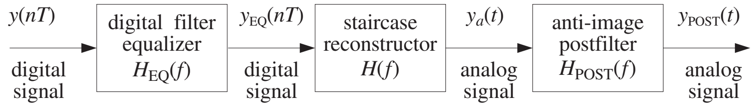


Fig. 1.6.9 Digital equalization filter for D/A conversion.

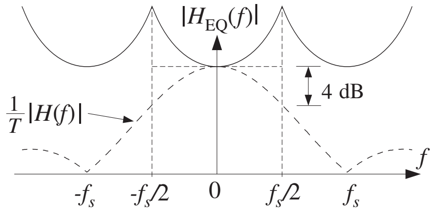


Fig. 1.6.10 Frequency response of DAC equalizer.

$$\begin{aligned} Y_{POST}(f) &= H_{POST}(f)\,Y_a(f) \\ &= H_{POST}(f)\,H(f)\,\hat{Y}_{EQ}(f) \\ &= H_{POST}(f)\,H(f)\,H_{EQ}(f)\,\hat{Y}(f) \end{aligned}$$

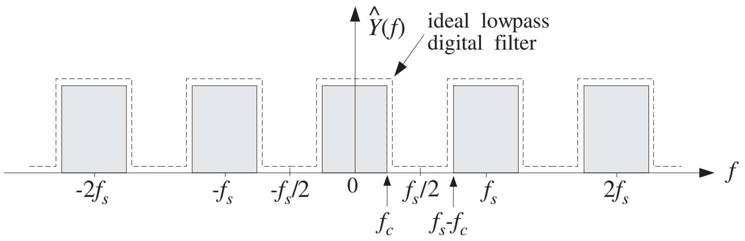
Within the Nyquist interval, using Eqs. (1.6.7) and (1.5.20) and assuming a flat postfilter there, $H_{POST}(f)\simeq 1$, we have,

$$Y_{POST}(f)=H_{POST}(f)\,H(f)\,H_{EQ}(f)\,\hat{Y}(f)=1\cdot T\cdot\frac{1}{T}Y(f)=Y(f)$$

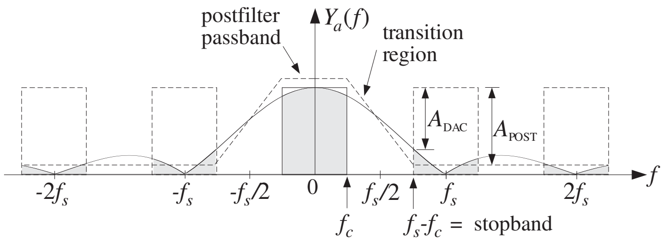
Outside the Nyquist interval, assuming $H_{POST}(f)\simeq 0$, we have $Y_{POST}(f)=0$. Thus, the combination of equalizer, DAC, and postfilter acts like an ideal reconstructor.

Example 1.6.1: The signal of Example 1.5.3 that was sampled at $f_s = 12$ kHz is filtered by a digital filter designed to act as an ideal lowpass filter with cutoff frequency of $f_c = 2$ kHz. The filtered digital signal is then fed into a staircase D/A and then into a lowpass anti-image postfilter. The overall reconstructor is required to suppress the spectral images caused by sampling by more than $A = 80$ dB. Determine the *least stringent* specifications for the analog postfilter that will satisfy this requirement.

Solution: The digital lowpass filter is, by construction, periodic in f with period f_s. Thus, the spectrum of the signal after the digital filter will look as follows:



The spectral images are separated now by a distance $f_s - 2f_c = 12 - 2\cdot 2 = 8$ kHz. After passing through the staircase reconstructor, the spectrum will be as shown below:



The postfilter must have a flat passband over $[-f_c, f_c]$. Its stopband must begin at $f_{\rm stop} = f_s - f_c = 12 - 2 = 10$ kHz because the first replica is largest there. The wide transition region between f_c and $f_s - f_c$ allows the use of a less stringent postfilter.

The required stopband attenuation of the postfilter can be determined as follows. The total attenuation caused by the cascade of the DAC and postfilter is the sum of the corresponding attenuations:

$$A(f) = A_{\rm DAC}(f) + A_{\rm POST}(f)$$

where

$$A_{\rm DAC}(f) = -20\log_{10}\left|\frac{H(f)}{H(0)}\right| = -20\log_{10}\left|\frac{\sin(\pi f/f_s)}{\pi f/f_s}\right|$$

At $f = f_{\rm stop} = f_s - f_c$, the total attenuation must be greater than A

$$A_{\rm DAC} + A_{\rm POST} \geq A \quad\Rightarrow\quad A_{\rm POST} \geq A - A_{\rm DAC}$$

Numerically, we find at $f_{\rm stop} = 10$ kHz

$$A_{\rm DAC} = -20\log_{10}\left|\frac{\sin(\pi 10/12)}{\pi 10/12}\right| = 14.4$$

resulting in $A_{\rm POST} \geq 80 - 14.4 = 65.6$ dB. □

The key idea in this example was to use the separation between spectral replicas as the *transition region* of the postfilter. The wider this separation, the less stringent the postfilter. Oversampling and digital interpolation techniques exploit this idea to its fullest and can be used to alleviate the need for expensive high-quality analog postfilters. Such oversampling systems are routinely used in CD and DAT players. They will be discussed in Chapter 14.

Example 1.6.2: A sinusoid of frequency f_0 is sampled at a rate f_s, such that $|f_0| \leq f_s/2$. The resulting sampled sinusoid is then reconstructed by an arbitrary reconstructor $H(f)$. Determine the analog signal at the output of the reconstructor when $H(f)$ is: (a) the ideal reconstructor, (b) the staircase reconstructor, (c) the staircase reconstructor followed by a very good anti-image postfilter, and (d) a staircase reconstructor equalized by the digital filter defined in Eq. (1.6.7).

Solution: Let $y(t) = e^{2\pi jf_0t}$. Its spectrum is $Y(f) = \delta(f - f_0)$ and the spectrum of the sampled sinusoid will be the replication of $Y(f)$, as in Example 1.5.1:

$$\hat{Y}(f) = \frac{1}{T}\sum_{m=-\infty}^{\infty}\delta(f - f_0 - mf_s)$$

The spectrum of the reconstructed signal will be:

$$\begin{aligned} Y_a(f) = H(f)\hat{Y}(f) &= \frac{1}{T}\sum_{m=-\infty}^{\infty} H(f)\delta(f - f_0 - mf_s) \\ &= \frac{1}{T}\sum_{m=-\infty}^{\infty} H(f_0 + mf_s)\delta(f - f_0 - mf_s) \end{aligned}$$

Taking inverse Fourier transforms, we obtain:

$$y_a(t) = \frac{1}{T}\sum_{m=-\infty}^{\infty} H(f_m)e^{2\pi jf_mt} \tag{1.6.8}$$

where $f_m = f_0 + mf_s$. If $H(f)$ is the ideal reconstructor, then $H(f_m)$ will be zero if f_m does not lie in the Nyquist interval. Because f_0 was assumed to lie in the interval, only the $m = 0$ term will survive the sum giving:

$$y_a(t) = \frac{1}{T}H(f_0)e^{2\pi jf_0t} = \frac{1}{T}\cdot T\,e^{2\pi jf_0t} = e^{2\pi jf_0t}$$

thus, the sinusoid is reconstructed perfectly. If f_0 lies outside the interval, $|f_0| > f_s/2$, then there exists a unique integer m_0 such that $|f_0 + m_0f_s| < f_s/2$, where m_0 is negative if $f_0 > 0$. In this case, only the $m = m_0$ term will survive the sum giving:

$$y_a(t) = \frac{1}{T}H(f_{m_0})e^{2\pi jf_{m_0}t} = e^{2\pi jf_{m_0}t}$$

where $f_{m_0} = f_0 + m_0f_s = f_0 \bmod (f_s)$. The sinusoid f_0 will be confused with the sinusoid f_{m_0}, as we discussed qualitatively in Section 1.4.1.

For the staircase reconstructor of Eq. (1.6.6), the reconstructed signal will be given by Eq. (1.6.8), which should sum up to generate the staircase approximation to the sinusoid. This is demonstrated in Example 1.6.3.

In case (c), a good postfilter will remove all frequencies outside the Nyquist interval, that is, only the $m = 0$ term will survive in Eq. (1.6.8), giving:

$$y_a(t) = \frac{1}{T}H(f_0)e^{2\pi jf_0t}$$

where we assumed that the postfilter has unity gain over the Nyquist interval. Using Eq. (1.6.6) evaluated at $f = f_0$, we get:

$$y_a(t) = \frac{\sin(\pi f_0 T)}{\pi f_0 T}\, e^{-\pi j f_0 T}\, e^{2\pi j f_0 t}$$

Thus, there is amplitude attenuation and phase shift, which both become worse as f_0 increases towards the Nyquist frequency $f_s/2$. A digital filter that equalizes the staircase response, would anticipate this attenuation and phase shift and undo them. Indeed, in case (d), the effective reconstructor is $H_{\text{EQ}}(f)H(f)$. Therefore, Eq. (1.6.8) becomes:

$$y_a(t) = \frac{1}{T}\sum_{m=-\infty}^{\infty} H_{\text{EQ}}(f_m)H(f_m)e^{2\pi j f_m t}$$

But because of the periodicity of $H_{\text{EQ}}(f)$, we can replace $H_{\text{EQ}}(f_m) = H_{\text{EQ}}(f_0) = T/H(f_0)$, giving:

$$y_a(t) = \sum_{m=-\infty}^{\infty} \frac{H(f_m)}{H(f_0)}\, e^{2\pi j f_m t} \tag{1.6.9}$$

A good postfilter, extracting the $m = 0$ term, would result in the final reconstructed output $y_{\text{post}}(t) = \dfrac{H(f_0)}{H(f_0)} e^{2\pi j f_0 t} = e^{2\pi j f_0 t}$. □

Example 1.6.3: The cosinusoid $y(t) = \cos(2\pi f_0 t)$ is sampled at a rate f_s and the samples are reconstructed by a staircase reconstructor $H(f)$. The reconstructed signal will be:

$$y_a(t) = \sum_{m=-\infty}^{\infty} G(f_m)\cos\big(2\pi f_m t + \phi(f_m)\big) \tag{1.6.10}$$

where $G(f)$ and $\phi(f)$ are defined as

$$G(f) = \frac{\sin(\pi f T)}{\pi f T}, \quad \phi(f) = -\pi f T \qquad \Rightarrow \qquad H(f) = TG(f)e^{j\phi(f)}$$

Note that $TG(f)$ and $\phi(f)$ are not quite the magnitude and phase responses of $H(f)$; those are $|H(f)| = T|G(f)|$ and $\arg H(f) = \phi(f) + \pi\big(1 - \text{sign}\,G(f)\big)/2$. Eq. (1.6.10) is obtained by substituting $H(f) = TG(f)e^{j\phi(f)}$ into Eq. (1.6.8) and taking real parts. A computable approximation of Eq. (1.6.10) is obtained by truncating the sum to $2M + 1$ terms, that is, keeping the terms $-M \le m \le M$:

$$y_a(t) = \sum_{m=-M}^{M} w(m)G(f_m)\cos\big(2\pi f_m t + \phi(f_m)\big) \tag{1.6.11}$$

where we have also introduced appropriate weights $w(m)$ to reduce the Gibbs ripples resulting from the truncation of the sum. For example, the *Hamming weights* are:

$$w(m) = 0.54 + 0.46\cos\Big(\frac{\pi m}{M}\Big), \qquad -M \le m \le M$$

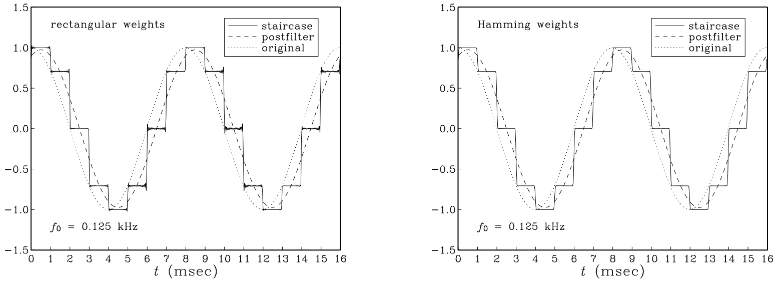
whereas the *rectangular* weights are $w(m) = 1$.

For the numerical values $f_0 = 0.125$ kHz, $f_s = 1$ kHz, $M = 15$, we have computed the original analog signal $y(t)$ and the reconstructed signal $y_a(t)$ as given by the approximation of Eq. (1.6.11), over the time interval $0 \le t \le 16$ msec, that is, over 16 sampling instants.

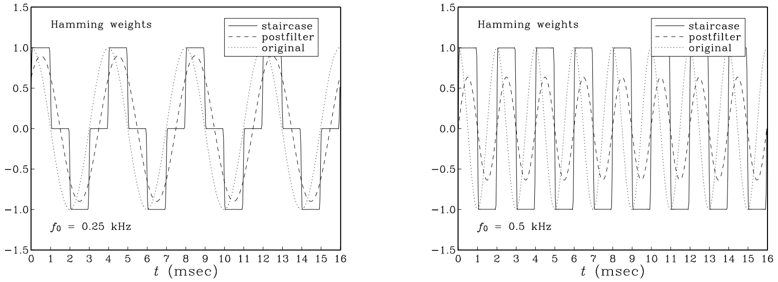
If the signal $y_a(t)$ were postfiltered by a good postfilter, only the $m = 0$ term would survive the sum, and the resulting signal would be the original f_0 sinusoid with some attenuation and phase shift:

$$y_{\text{post}}(t) = G(f_0)\cos\big(2\pi f_0 t + \phi(f_0)\big)$$

The following figure compares the three signals $y(t)$, $y_a(t)$, and $y_{\text{post}}(t)$ in the two cases of using rectangular weights and Hamming weights $w(m)$.



Notice how the postfilter output $y_{\text{post}}(t)$ is essentially an averaged or smoothed version of the staircase output $y_a(t)$. To see the dependence on the value of f_0 of the attenuation and phase shift of $y_{\text{post}}(t)$, the next two graphs show the cases $f_0 = 0.25$ and $f_0 = 0.5$ kHz.



The case $f_0 = 0.5$ kHz corresponds to the Nyquist frequency $f_s/2$, having the maximum amplitude and phase distortion. In all cases, however, $y_{\text{post}}(t)$ is a smoothed version of the staircase output.

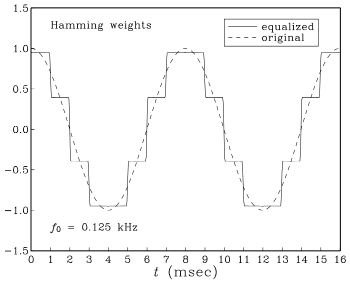
If the staircase reconstructor is preceded by an equalizer filter, as shown in Fig. 1.6.9, then the staircase output will be given by the real part of Eq. (1.6.9). We have:

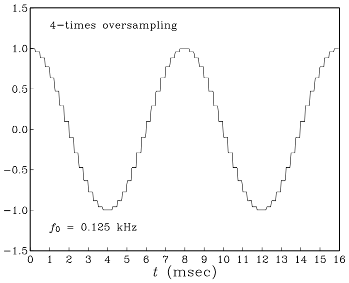
$$\begin{aligned}
\frac{H(f_m)}{H(f_0)} &= \frac{\sin(\pi f_m T)}{\pi f_m T}\,\frac{\pi f_0 T}{\sin(\pi f_0 T)}\, e^{-j\pi(f_m - f_0)T} \\
&= \frac{\sin(\pi f_0 T + \pi m)}{\sin(\pi f_0 T)}\,\frac{f_0}{f_m}\, e^{-j\pi m} \\
&= \frac{(-1)^m \sin(\pi f_0 T)}{\sin(\pi f_0 T)}\,\frac{f_0}{f_m}\,(-1)^m = \frac{f_0}{f_m}
\end{aligned}$$

where we used the property $\cos(x+\pi m)=(-1)^m\cos x$. Thus,

$$y_a(t)=\sum_{m=-M}^{M} w(m)\,\frac{f_0}{f_m}\,\cos(2\pi f_m t) \qquad (1.6.12)$$

This signal is shown below for the case $f_0 = 0.125$ kHz.





It is superimposed on the original sinusoid, corresponding to the $m = 0$ term, which is what would be extracted by a good postfilter. Notice again the smoothing effect of the postfilter. In order to remove completely all the $m \neq 0$ terms, the postfilter must be a high-quality lowpass filter with sharp cutoff at the Nyquist frequency.

To illustrate the beneficial effect of oversampling on such a reconstructor, we have also plotted the digitally equalized staircase output in the case of 4-times oversampling, as given by Eq. (1.6.12) with $f_s = 4$ kHz. Now there are four times as many staircase levels. They follow the original sinusoid more closely and can be smoothed more easily. Therefore, a lower quality, less expensive lowpass postfilter would be sufficient in this case. □

1.7 *Basic Components of DSP Systems*

It follows from the discussion in Sections 1.5 and 1.6 that the minimum number of necessary components in a typical digital signal processing system must be:

1. A lowpass analog antialiasing prefilter that bandlimits the signal to be sampled to within the Nyquist interval.
2. An A/D converter (sampler and quantizer).
3. A digital signal processor.
4. A D/A converter (staircase reconstructor), possibly preceded by an equalizing digital filter.
5. A lowpass analog anti-image postfilter to complete the job of the staircase reconstructor and further remove the spectral images introduced by the sampling process.

These are shown in Fig. 1.7.1. Here, we review briefly the function of each stage and its impact on the quality of the overall digital processing system. With the exception of the sampling stage, every stage in this system may be thought of as a filtering operation by an appropriate transfer function. The sampling stage, through its spectrum replication, is a spectrum expansion operation.

The function of the antialiasing prefilter $H_{\rm PRE}(f)$ is to bandlimit the overall analog input signal $x_a(t)$ to within the Nyquist interval $[-f_s/2, f_s/2]$. The output of the prefilter is now a bandlimited signal $x(t)$ and may be sampled at a rate of f_s samples per second. By design, the spectral replicas generated by the sampling process will not overlap. The sampling rate must be high enough so that the surviving input spectrum after the prefiltering operation, that is, the spectrum of $x(t)$, contains all the frequencies of interest for the application at hand.

The quality of the prefilter affects critically the quality of the overall system, that is, the degree of overlap of the spectral replicas depends on the rolloff characteristics of the prefilter.

The sampled (and quantized) signal $\hat{x}(t)$ or $x(nT)$ is then processed by a digital signal processor whose effect is to *reshape* the spectrum by means of a transfer function, say $H_{\rm DSP}(f)$, so that $\hat{Y}(f)=H_{\rm DSP}(f)\hat{X}(f)$.

The resulting output samples $\hat{y}(t)$ or $y(nT)$ are then reconstructed by the DAC into the staircase analog signal $y(t)$. Finally, the signal $y(t)$ is smoothed further by the postfilter, resulting in the overall analog output signal $y_a(t)$. Separating in Eq. (1.5.16) the central replica from the other replicas, we write

$$\hat{X}(f)=\frac{1}{T}\sum_{m=-\infty}^{\infty}X(f-mf_s)=\frac{1}{T}\big[X(f)+\text{replicas}\big]$$

and following backwards all the transfer function relationships, we find for the spectrum of $y_a(t)$:

$$\begin{aligned}
Y_a(f) &= H_{\rm POST}(f)\,Y(f)=H_{\rm POST}(f)\,H_{\rm DAC}(f)\,\hat{Y}(f)\\
&= H_{\rm POST}(f)\,H_{\rm DAC}(f)\,H_{\rm DSP}(f)\,\hat{X}(f)\\
&= H_{\rm POST}(f)\,H_{\rm DAC}(f)\,H_{\rm DSP}(f)\,\frac{1}{T}\big[X(f)+\text{replicas}\big]\\
&= H_{\rm POST}(f)\,H_{\rm DAC}(f)\,H_{\rm DSP}(f)\,\frac{1}{T}\big[H_{\rm PRE}(f)X_a(f)+\text{replicas}\big]
\end{aligned}$$

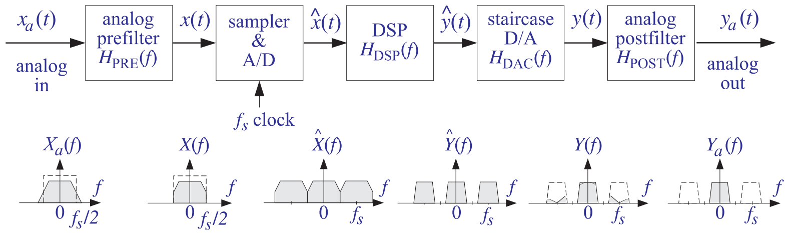


Fig. 1.7.1 Components of typical DSP system.

In a well-designed system, the product of the staircase DAC and the postfilter transfer functions should be effectively equal to the ideal reconstructor. Therefore, for frequencies outside the Nyquist interval the output spectrum will vanish, that is, the spectral images will be removed. But for frequencies within the Nyquist interval, that product should be equal to the gain T canceling the $1/T$ factor.

Furthermore, because the prefilter $H_{\rm PRE}(f)$ ensures that the replicas do not overlap, the terms labeled “replicas” will vanish for all frequencies within the Nyquist interval. Finally, because the prefilter approximates an ideal lowpass filter, its passband gain will be approximately one. The upshot of all this is that *within* the Nyquist interval one has approximately

$$H_{\rm POST}(f)\,H_{\rm DAC}(f) \simeq T$$

$$\text{replicas} \simeq 0$$

$$H_{\rm PRE}(f) \simeq 1$$

To the extent that these approximations are good—and this determines the quality of the overall system—we finally find

$$Y_a(f) = T\cdot H_{\rm DSP}(f)\,\frac{1}{T}\big[1\cdot X_a(f) + 0\big], \qquad \text{or,}$$

$$\boxed{Y_a(f) = H_{\rm DSP}(f)\,X_a(f)}\,, \qquad \text{for}\quad |f| \le \frac{f_s}{2} \tag{1.7.1}$$

Thus, the above arrangement works exactly as expected, that is, it is equivalent to *linear filtering* of the analog input, with an effective transfer function $H_{\rm DSP}(f)$ defined by the digital signal processor. This is, of course, the ultimate goal of the DSP system. The primary reasons for using digital signal processing are the *programmability, reliability, accuracy, availability, and cost* of the digital hardware.

1.8 Theory of Bandlimited Functions

The Shannon sampling theorem of Eq. (1.6.4) expresses a bandlimited signal $y(t)$ in terms of its time samples $y(nT)$ using the ideal analog reconstructor’s sinc-function impulse response,

$$y(t) = \sum_{n=-\infty}^{\infty} y(nT)h(t-nT)\,, \quad h(t-nT) = \frac{\sin\big(\pi f_s(t-nT)\big)}{\pi f_s(t-nT)}$$

The shifted-sinc functions, $f_n(t) = h(n-nT)$, may be thought of as a basis of functions for the representation of bandlimited functions.

There are other function bases for representing bandlimited signals, such as spherical Bessel functions, and, most notably, *prolate spheroidal wave functions*, which are discussed in some detail in Appendix B, with several references and applications. Additional references on sampling may be found in [47–55].

1.9 Problems

1.1 A wheel, rotating at 6 Hz, is seen in a dark room by means of a strobe light flashing at a rate of 8 Hz. Determine the apparent rotational speed and sense of rotation of the wheel. Repeat the question if the flashes occur at 12 Hz, 16 Hz, or 24 Hz.

1.2 The analog signal $x(t) = 10\sin(2\pi t) + 10\sin(8\pi t) + 5\sin(12\pi t)$, where t is in seconds, is sampled at a rate of $f_s = 5$ Hz. Determine the signal $x_a(t)$ aliased with $x(t)$. Show that the two signals have the *same* sample values, that is, show that $x(nT) = x_a(nT)$. Repeat the above questions if the sampling rate is $f_s = 10$ Hz.

1.3 The signal $x(t) = \cos(5\pi t) + 4\sin(2\pi t)\sin(3\pi t)$, where t is in milliseconds, is sampled at a rate of 3 kHz. Determine the signal $x_a(t)$ aliased with $x(t)$.

Determine two other signals $x_1(t)$ and $x_2(t)$ that are different from each other and from $x(t)$, yet they are aliased with the same $x_a(t)$ that you found.

1.4 Let $x(t) = \cos(8\pi t) + 2\cos(4\pi t)\cos(6\pi t)$, where t is in seconds. Determine the signal $x_a(t)$ aliased with $x(t)$, if the sampling rate is 5 Hz. Repeat for a sampling rate of 9 Hz.

1.5 The analog signal $x(t) = \sin(6\pi t)\big[1 + 2\cos(4\pi t)\big]$, where t is in milliseconds, is sampled at a rate of 4 kHz. The resulting samples are immediately reconstructed by an ideal reconstructor. Determine the analog signal $x_a(t)$ at the output of the reconstructor.

1.6 The analog signal $x(t) = 4\cos(2\pi t)\cos(8\pi t)\cos(12\pi t)$, where t is in seconds, is sampled at a rate of $f_s = 10$ Hz. Determine the signal $x_a(t)$ aliased with $x(t)$. Show that the two signals have the *same* sample values, that is, show that $x(nT) = x_a(nT)$. Repeat the above questions if the sampling rate is $f_s = 12$ Hz. [*Hint*: Express $x(t)$ as a *sum* of sines and cosines.]

1.7 Consider the periodic triangular waveform with period $T_0 = 1$ sec shown in Fig. 1.9.1. The waveform is sampled at rate $f_s = 8$ Hz and the resulting samples are reconstructed by an *ideal* reconstructor. Show that the signal $x_{\rm rec}(t)$ that will appear at the output of the reconstructor will have the form:

$$x_{\rm rec}(t) = A\sin(2\pi f_1 t) + B\sin(2\pi f_2 t)$$

and determine the numerical values of the frequencies f_1, f_2 and amplitudes A, B.

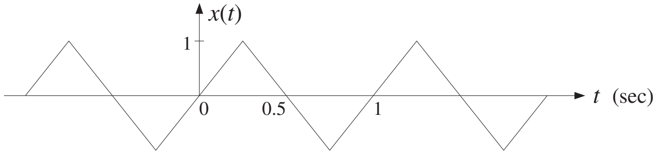


Fig. 1.9.1 Triangular waveform of Problem 1.7.

1.8 *Computer Experiment: Aliasing.* Consider an analog signal $x(t)$ consisting of three sinusoids of frequencies $f_1 = 1$ kHz, $f_2 = 4$ kHz, and $f_3 = 6$ kHz, where t is in milliseconds:

$$x(t) = 2\sin(2\pi f_1 t) + 2\sin(2\pi f_2 t) + \sin(2\pi f_3 t)$$

a. The signal is sampled at a rate of 5 kHz. Determine the signal $x_a(t)$ that would be aliased with $x(t)$. On the same graph, plot the two signals $x(t)$ and $x_a(t)$ versus t in the range $0 \le t \le 2$ msec. Show both analytically and graphically that the two signals have the *same* sampled values, which occur at intervals of $T = 1/f_s = 0.2$ msec.

b. Repeat with a sampling rate of $f_s = 10$ kHz.

c. On the same graph, plot the signals $x(t)$ and $x_a(t)$ of Problem 1.7, over the range $0 \le t \le 2$ sec, and verify that they intersect at the sampling instants at multiples of $T = 1/f_s = 0.125$ sec. In plotting, $x(t)$, you need to define it as a triangular function of t.

Repeat this part, but with sampling rate $f_s = 4$ Hz. What is $x_a(t)$ now?

1.9 Consider the following sound wave, where t is in milliseconds:

$$x(t) = \sin(10\pi t) + \sin(20\pi t) + \sin(60\pi t) + \sin(90\pi t)$$

This signal is prefiltered by an analog antialiasing prefilter $H(f)$ and then sampled at an audio rate of 40 kHz. The resulting samples are immediately reconstructed using an ideal reconstructor. Determine the output $y_a(t)$ of the reconstructor in the following cases and compare it with the *audible part* of $x(t)$:

a. When there is no prefilter, that is, $H(f) \equiv 1$.

b. When $H(f)$ is an *ideal* prefilter with cutoff of 20 kHz.

c. When $H(f)$ is a *practical* prefilter that has a flat passband up to 20 kHz and attenuates at a rate of 48 dB/octave beyond 20 kHz. (You may ignore the effects of the phase response of the filter.)

1.10 Prove the Fourier series expansion of the ideal sampling function $s(t)$ given in Eq. (1.5.15). Then, prove its Fourier transform expression (1.5.18).

1.11 Given Eq. (1.5.4), prove the inverse DTFT property (1.5.5), that is,

$$\hat{X}(f) = \sum_{n=-\infty}^{\infty} x(nT)e^{-2\pi j f T n} \quad \Rightarrow \quad x(nT) = \frac{1}{f_s}\int_{-f_s/2}^{f_s/2} \hat{X}(f)e^{2\pi j f T n}\, df$$

1.12 Consider a pure sinusoid of frequency f_0, $x(t) = \cos(2\pi f_0 t)$. Show that the spectrum of the sampled sinusoid $x(nT)$ is:

$$\hat{X}(f) = \frac{1}{2T}\sum_{m=-\infty}^{\infty}\big[\delta(f - f_0 - mf_s) + \delta(f + f_0 + mf_s)\big]$$

1.13 *Computer Experiment: Sampling of Non-Bandlimited Signals.* Consider the exponentially decaying sinusoid $x(t) = e^{-at}\cos(2\pi f_0 t)$ sampled at a rate $f_s = 1/T$. For convenience, replace it by its complex-valued version: $x(t) = e^{-at}e^{2\pi j f_0 t}$. Let $x(nT) = e^{-aTn}e^{2\pi j f_0 Tn}$ be its samples, and let $x_L(nT) = x(nT)$, $n = 0, 1, \dots, L-1$ be its windowed version to length L. Show that the magnitude spectra of the analog, sampled, and windowed signals are given by:

$$|X(f)|^2 = \frac{1}{a^2 + \big(2\pi(f - f_0)\big)^2}$$

$$|\hat{X}(f)|^2 = \frac{1}{1 - 2e^{-aT}\cos\big(2\pi(f - f_0)T\big) + e^{-2aT}}$$

$$|\hat{X}_L(f)|^2 = \frac{1 - 2e^{-aTL}\cos\big(2\pi(f - f_0)LT\big) + e^{-2aTL}}{1 - 2e^{-aT}\cos\big(2\pi(f - f_0)T\big) + e^{-2aT}}$$

Show the limits:

$$\lim_{L\to\infty}\hat{X}_L(f) = \hat{X}(f), \qquad \lim_{f_s\to\infty} T\hat{X}(f) = X(f)$$

For the numerical values $a = 0.2$ sec^{-1}, $f_0 = 0.5$ Hz, and the two rates $f_s = 1$ Hz and $f_s = 2$ Hz, plot on the same graph the analog spectrum $|X(f)|^2$ and the sampled spectrum $|T\hat{X}(f)|^2$, over the frequency range $0 \le f \le 3$ Hz.

For $f_s = 2$, plot on another graph, the three spectra $|X(f)|^2$, $|T\hat{X}(f)|^2$, $|T\hat{X}_L(f)|^2$, over the range $0 \le f \le 3$ Hz.

What conclusions do you draw from these graphs? What are the implications of the above limits? What are the essential differences if we work with the real-valued signal?

1.14 The frequency range of interest of a signal extends to $f_{\max}$. Beyond $f_{\max}$, the spectrum attenuates by α dB per decade. We have available an off-the-shelf antialiasing prefilter that has a flat passband up to $f_{\max}$ and attenuates by β dB per decade beyond that. It is required that within the $f_{\max}$ range of interest, the aliased components due to sampling be suppressed by more than A dB. Show that the *minimum* sampling rate that we should use is given by

$$f_s = f_{\max} + 10^{A/\gamma}f_{\max}, \qquad \text{where } \gamma = \alpha + \beta$$

1.15 An analog input signal to a DSP system has spectrum:

$$|X_{\text{in}}(f)| = \frac{1}{\sqrt{1 + (0.1f)^8}}$$

where f is in kHz. The highest frequency of interest is 20 kHz. The signal is to be sampled at a rate f_s. It is required that the aliased spectral components within the frequency range of interest be suppressed by more than 60 dB *relative* to the signal components, that is, they must be at least 60 dB below the value of the signal components throughout the 20 kHz range of interest.

a. Determine the minimum sampling rate f_s, if no antialiasing prefilter is used.

b. Suppose a simple third-order Butterworth antialiasing prefilter is used having magnitude response

$$|H(f)| = \frac{1}{\sqrt{1 + (f/f_0)^6}}$$

It is required that the prefilter's attenuation within the 20 kHz band of interest remain less than 1 dB. What is the value of the normalization frequency f_0 in this case? What is the minimum value of f_s that may be used? Compare your exact calculation of f_s with the approximate one using the method of Problem 1.14.

1.16 For the above example, suppose we are constrained to use a particular sampling rate, which is less than the minimum we determined above (and greater than $2f_{\max}$), such as $f_s = 70$ kHz. In order to achieve the required 60 dB suppression of the aliased replicas, we must now use a more complex prefilter—one that has a steeper transition width, such as a higher-order Butterworth. An Nth order Butterworth filter has magnitude response

$$|H(f)|^2 = \frac{1}{1 + (f/f_0)^{2N}}$$

Given f_s, determine the minimum filter order N in order for the filter to attenuate less than $A_{\text{pass}} = 1$ dB in the passband and the total suppression of the spectral images to be greater than $A = 60$ dB.

1.17 *Computer Experiment: Butterworth Prefilter Design.* Using the methods of the previous problem, derive a "design curve" for the prefilter, that is, an expression for the Butterworth filter order N as a function of the sampling rate f_s and stopband attenuation A. Assume $f_{\max} = 20$ kHz and $A_{\text{pass}} = 1$ dB for the passband attenuation.

For each of the attenuation values $A = 40, 50, 60, 70, 80$ dB, plot the filter order N versus f_s in the range $50 \le f_s \le 120$ kHz. Identify on these graphs the design points of the Problems 1.15 and 1.16.

1.18 The significant frequency range of an analog signal extends to 10 kHz. Beyond 10 kHz, the signal spectrum attenuates at a rate of 80 dB per decade.

The signal is to be sampled at a rate of 30 kHz. The aliased frequency components introduced into the 10 kHz range of interest must be kept below 60 dB, as compared to the signal components.

Suppose we use an antialiasing prefilter whose passband is flat over the 10 kHz interval. Beyond 10 kHz, it attenuates at a certain rate that must be steep enough to satisfy the above sampling requirements. What is this attenuation rate in *dB per decade*? Explain your reasoning. What is the minimum *filter order* that we must use?

What is the prefilter's attenuation rate if we increase the sampling rate to 50 kHz? What is the filter order in this case?

1.19 An analog input signal to a DSP system has spectrum:

$$|X_{\text{in}}(f)| = \frac{1}{\sqrt{1 + (f/f_a)^{2N_a}}}$$

where f_a and N_a are given. The highest frequency of interest is $f_{\max} = 2f_a$. The signal is to be sampled at a rate f_s. It is required that the aliased spectral components within the frequency range of interest be suppressed by more than A dB *relative* to the signal components, that is, they must be at least A dB below the value of the signal components throughout the $0 \le f \le f_{\max}$ range of interest.

a. Assuming that no antialiasing prefilter is used, set up and solve an equation for the *minimum* sampling rate f_s, in terms of the quantities f_a, N_a, A.

b. Next, suppose that an Nth order Butterworth analog prefilter is to be used to aid the sampling process. Let f_0 be the filter's 3-dB normalization frequency. It is required that the prefilter's attenuation within the $0 \le f \le f_{\max}$ band of interest remain less than B dB.

Set up an equation for f_0 that would guarantee this condition.

Then, set up an equation for the minimum f_s that would guarantee the desired A dB suppression of the spectral images.

c. Show that f_s is given approximately by

$$f_s = f_{\max}\left[1 + 10^{A/20(N+N_a)}\right]$$

When is this approximation valid? Show that this expression also covers part (a) if you set $N = 0$. Discuss the meaning of the limit $N \to \infty$ in terms of the sampling theorem.

1.20 In Problem 1.19, we implicitly assumed that the prefilter's order N was given, and we determined f_0 and f_s. Here, we assume that f_s is given and is equal to some value above $2f_{\max}$. Show that the *minimum* prefilter order that must be used to guarantee A dB suppression of the spectral images is approximately linearly related to A via an equation of the form:

$$N = aA + b$$

Determine expressions for a and b in terms of the given quantities.

1.21 The operation of flat-top practical sampling depicted in Fig. 1.5.1 may be thought of as filtering the ideally sampled signal $\hat{x}(t)$ through an analog linear filter whose impulse response is the sampling pulse $p(t)$, as shown in Fig. 1.9.2. Show that Eq. (1.5.2) can be written as the I/O convolutional equation of such a filter:

$$x_{\text{flat}}(t) = \int_{-\infty}^{\infty} p(t - t')\hat{x}(t')\,dt' = \sum_{n=-\infty}^{\infty} x(nT)p(t - nT)$$

where $\hat{x}(t)$ is given by Eq. (1.5.1). In the frequency domain, this translates to $X_{\text{flat}}(f) = P(f)\hat{X}(f)$, where $P(f)$ is the spectrum of sampling pulse $p(t)$.

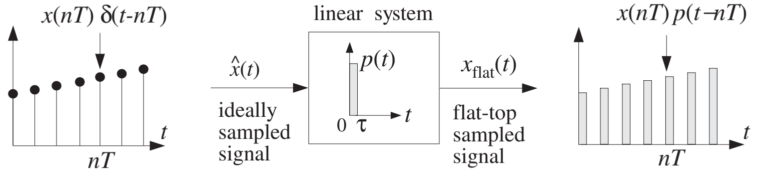


Fig. 1.9.2 Flat-top sampling as filtering.

Determine $P(f)$ for a flat pulse $p(t)$ of duration τ seconds. For the case $\tau = T/5$, make a sketch of $X_{\text{flat}}(f)$ over the range $-6f_s \le f \le 6f_s$.

1.22 After having been properly prefiltered by an antialiasing filter, an analog signal is sampled at a rate of 6 kHz. The digital signal is then filtered by a digital filter designed to act as an ideal lowpass filter with cutoff frequency of 1 kHz. The filtered digital signal is then fed into a staircase D/A reconstructor and then into a lowpass anti-image postfilter.

The overall reconstructor is required to suppress the spectral images caused by sampling by more than $A = 40$ dB. Determine the *least stringent* specifications for the analog postfilter that will satisfy this requirement.

1.23 Consider an arbitrary D/A reconstructing filter with impulse response $h(t)$ and corresponding frequency response $H(f)$. The analog signal at the output of the reconstructor is related to the incoming time samples $x(nT)$ by

$$x_a(t) = \sum_n x(nT)h(t - nT)$$

Show this result in two ways:

a. Using convolution in the time domain.

b. Starting with $X_a(f) = H(f)\hat{X}(f)$ and taking inverse Fourier transforms.

1.24 The sinusoidal signal $x(t) = \sin(2\pi f_0 t)$ is sampled at a rate f_s and the resulting samples are then reconstructed by an *arbitrary* analog reconstructing filter $H(f)$. Show that the analog signal at the output of the reconstructor will have the form:

$$x_{\text{rec}}(t)=\sum_{m=-\infty}^{\infty}A_m\sin(2\pi f_m t+\theta_m)$$

What are the frequencies f_m? How are the quantities A_m and θ_m related to the frequency response $H(f)$? Determine the quantities A_m and θ_m for the two cases of a staircase reconstructor and an ideal reconstructor.

1.25 The sum of sinusoids

$$y(t)=A_1e^{2\pi jf_1t}+A_2e^{2\pi jf_2t}$$

is sampled at a rate f_s such that $f_s > 2|f_1|$ and $f_s > 2|f_2|$. The resulting samples are then filtered digitally by a *staircase-equalizing* digital filter and then reconstructed by a staircase reconstructor, as shown in Fig. 1.6.9. If a final postfilter is not used, show that the resulting analog signal at the output of the reconstructor will be

$$y_a(t)=\sum_{m=-\infty}^{\infty}\left[A_{1m}e^{2\pi jf_{1m}t}+A_{2m}e^{2\pi jf_{2m}t}\right]$$

where $A_{1m}=A_1f_1/f_{1m}$, $A_{2m}=A_2f_2/f_{2m}$, and $f_{1m}=f_1+mf_s$, $f_{2m}=f_2+mf_s$. What would a final postfilter do to each of these terms?

2

Quantization

2.1 Quantization Process

Sampling and quantization are the necessary prerequisites for any digital signal processing operation on analog signals. A sampler and quantizer are shown in Fig. 2.1.1 [56–61]. The hold capacitor in the sampler holds each measured sample $x(nT)$ for at most T seconds during which time the A/D converter must convert it to a quantized sample, $x_Q(nT)$, which is representable by a finite number of bits, say B bits. The B-bit word is then shipped over to the digital signal processor.

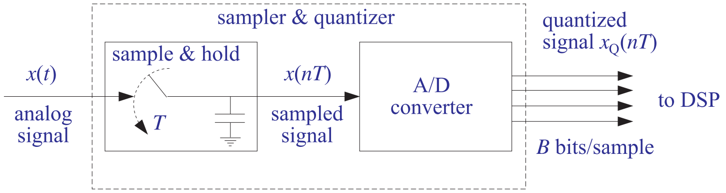


Fig. 2.1.1 Analog to digital conversion.

After digital processing, the resulting B-bit word is applied to a D/A converter which converts it back to analog format generating a staircase output. In practice, the sample/hold and ADC may be separate modules or may reside on board the same chip.

The quantized sample $x_Q(nT)$, being represented by B bits, can take only one of 2^B possible values. An A/D converter is characterized by a *full-scale range* R, which is divided equally (for a uniform quantizer) into 2^B *quantization levels*, as shown in Fig. 2.1.2. The spacing between levels, called the *quantization width* or the quantizer resolution, is given by:

$$\boxed{Q=\frac{R}{2^B}} \tag{2.1.1}$$

This equation can also be written in the form:

$$\boxed{\frac{R}{Q}=2^B} \tag{2.1.2}$$

which gives the number of quantization levels.

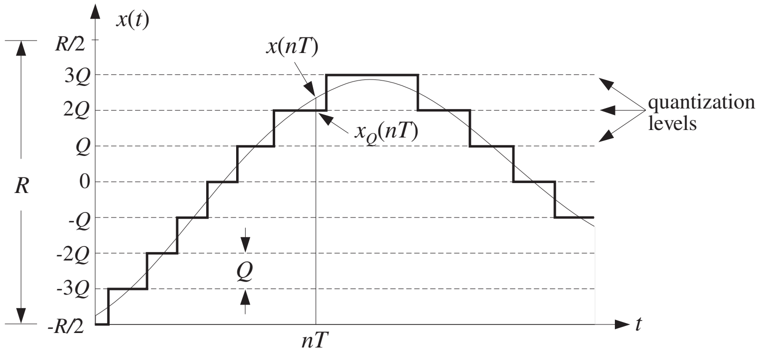


Fig. 2.1.2 Signal quantization.

Typical values of R in practice are between 1–10 volts. Figure 2.1.2 shows the case of $B = 3$ or $2^B = 8$ levels, and assumes a *bipolar* ADC for which the possible quantized values lie within the symmetric range:

$$-\frac{R}{2} \leq x_Q(nT) < \frac{R}{2}$$

For a *unipolar* ADC, we would have instead $0 \leq x_Q(nT) < R$. In practice, the input signal $x(t)$ must be preconditioned by analog means to lie within the full-scale range of the quantizer, that is, $-R/2 \leq x(t) < R/2$, before it is sent to the sampler and quantizer. The upper end, $R/2$, of the full-scale range is not realized as one of the levels; rather, the maximum level is $R/2 - Q$.

In Fig. 2.1.2, quantization of $x(t)$ was done by *rounding*, that is, replacing each value $x(t)$ by the value of the *nearest* quantization level. Quantization can also be done by *truncation* whereby each value is replaced by the value of the level below it. Rounding is preferred in practice because it produces a less biased quantized representation of the analog signal.

The *quantization error* is the error that results from using the quantized signal $x_Q(nT)$ instead of the true signal $x(nT)$, that is,[†]

$$e(nT) = x_Q(nT) - x(nT) \tag{2.1.3}$$

In general, the error in quantizing a number x that lies in $[-R/2, R/2)$ is:

$$e = x_Q - x$$

where x_Q is the quantized value. If x lies between two levels, it will be rounded up or down depending on which is the closest level. If x lies in the upper (lower) half between the two levels, it will be rounded up (down). Thus, the error e can only take the values[†]

$$-\frac{Q}{2} \leq e \leq \frac{Q}{2} \tag{2.1.4}$$

Therefore, the maximum error is $e_{\max} = Q/2$ in magnitude. This is an overestimate for the typical error that occurs. To obtain a more representative value for the average error, we consider the *mean* and *mean-square* values of e defined by:

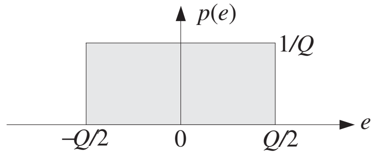
$$\bar{e} = \frac{1}{Q}\int_{-Q/2}^{Q/2} e\,de = 0, \qquad \text{and} \qquad \overline{e^2} = \frac{1}{Q}\int_{-Q/2}^{Q/2} e^2\,de = \frac{Q^2}{12} \tag{2.1.5}$$

The result $\bar{e} = 0$ states that on the average half of the values are rounded up and half down. Thus, $\bar{e}$ cannot be used as a representative error. A more typical value is the *root-mean-square* (rms) error defined by:

$$\boxed{e_{\text{rms}} = \sqrt{\overline{e^2}} = \frac{Q}{\sqrt{12}} = \frac{R}{\sqrt{12}}\,2^{-B}} \tag{2.1.6}$$

Equations (2.1.5) can be given a probabilistic interpretation by assuming that the quantization error e is a *random variable* which is distributed *uniformly* over the range (2.1.4), that is, having probability density:

$$p(e) = \begin{cases} \dfrac{1}{Q} & \text{if } -\dfrac{Q}{2} \leq e \leq \dfrac{Q}{2} \\ 0 & \text{otherwise} \end{cases}$$



The normalization $1/Q$ is needed to guarantee:

$$\int_{-Q/2}^{Q/2} p(e)\,de = 1$$

It follows that Eqs. (2.1.5) represent the *statistical expectations*:

$$E[e] = \int_{-Q/2}^{Q/2} ep(e)\,de \qquad \text{and} \qquad E[e^2] = \int_{-Q/2}^{Q/2} e^2 p(e)\,de$$

Thinking of R and Q as the ranges of the signal and quantization noise, the ratio in Eq. (2.1.2) is a *signal-to-noise ratio* (SNR). It can be expressed in dB:

$$20\log_{10}\left(\frac{R}{Q}\right) = 20\log_{10}(2^B) = B \cdot 20\log_{10}2, \qquad \text{or,}$$

$$\boxed{SNR = 20\log_{10}\left(\frac{R}{Q}\right) = 6B \ \text{dB}} \quad \text{(6-dB-per-bit-rule)} \tag{2.1.7}$$

which is referred to as the 6 *dB per bit* rule. Eq. (2.1.7) is called the *dynamic range* of the quantizer. Equations (2.1.1) and (2.1.6) can be used to determine the wordlength B if the full-scale range and desired rms error are given.

[†] A more natural definition would have been $e(nT) = x(nT) - x_Q(nT)$. The choice (2.1.3) is more convenient for making quantizer models.

[†] If the midpoint between levels is rounded up, then we should have more strictly, $-Q/2 < e \leq Q/2$.

Example 2.1.1: In a digital audio application, the signal is sampled at a rate of 44 kHz and each sample quantized using an A/D converter having a full-scale range of 10 volts. Determine the number of bits B if the rms quantization error must be kept below 50 microvolts. Then, determine the actual rms error and the bit rate in bits per second.

Solution: Write Eq. (2.1.6) in terms of B, $e_{\text{rms}} = Q/\sqrt{12} = R2^{-B}/\sqrt{12}$ and solve for B:

$$B = \log_2\left[\frac{R}{e_{\text{rms}}\sqrt{12}}\right] = \log_2\left[\frac{10}{50\cdot 10^{-6}\sqrt{12}}\right] = 15.82$$

which is rounded to $B = 16$ bits, corresponding to $2^B = 65536$ quantization levels. With this value of B, we find $e_{\text{rms}} = R2^{-B}/\sqrt{12} = 44$ microvolts. The bit rate will be $Bf_s = 16\cdot 44 = 704$ kbits/sec. This is a typical bit rate for CD players.

The dynamic range of the quantizer is $6B = 6\cdot 16 = 96$ dB. Note that the dynamic range of the human ear is about 100 dB. Therefore, the quantization noise from 16-bit quantizers is about at the threshold of hearing. This is the reason why “CD quality” digital audio requires at least 16-bit quantization. □

Example 2.1.2: By comparison, in digital speech processing the typical sampling rate is 8 kHz and the quantizer's wordlength 8 bits, corresponding to 256 levels. An 8-bit ADC with full-scale range of 10 volts, would generate an rms quantization noise $e_{\text{rms}} = R2^{-B}/\sqrt{12} = 11$ millivolts. The bit rate in this case is $Bf_s = 8\cdot 8 = 64$ kbits/sec. □

The probabilistic interpretation of the quantization noise is very useful for determining the effects of quantization as they propagate through the rest of the digital processing system. Writing Eq. (2.1.3) in the form[†]

$$\boxed{x_Q(n) = x(n) + e(n)} \tag{2.1.8}$$

we may think of the quantized signal $x_Q(n)$ as a noisy version of the original unquantized signal $x(n)$ to which a noise component $e(n)$ has been added. Such an additive noise model of a quantizer is shown in Fig. 2.1.3.

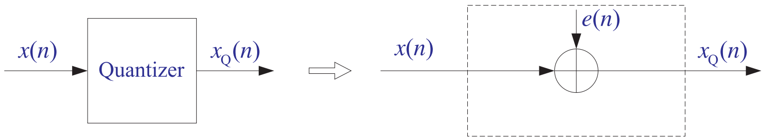


Fig. 2.1.3 Additive noise model of a quantizer.

In general, the statistical properties of the noise sequence $e(n)$ are very complicated [62–67,70]. However, for so-called *wide-amplitude wide-band* signals, that is, signals that vary through the entire full-scale range R crossing often all the quantization levels, the sequence $e(n)$ may be assumed to be a *stationary* zero-mean *white noise* sequence with *uniform* probability density over the range $[-Q/2, Q/2]$. Moreover, $e(n)$ is assumed to be *uncorrelated* with the signal $x(n)$. The average *power* or variance of $e(n)$ has already been computed above:

[†] For simplicity, we denoted $x(nT)$ by $x(n)$, etc.

$$\boxed{\sigma_e^2 = E[e^2(n)] = \frac{Q^2}{12}} \tag{2.1.9}$$

The assumption that $e(n)$ is white noise means that it has a delta-function autocorrelation:

$$R_{ee}(k) = E[e(n+k)e(n)] = \sigma_e^2\delta(k) \tag{2.1.10}$$

for all lags k. Similarly, that it is uncorrelated with $x(n)$ means that it has zero cross-correlation:

$$R_{ex}(k) = E[e(n+k)x(n)] = 0 \tag{2.1.11}$$

for all k. Later on we will illustrate this statistical model for $e(n)$ with a simulation example and verify equations (2.1.9)–(2.1.11), as well as the uniform distribution for the density $p(e)$.

The model is not accurate for low-amplitude slowly varying signals. For example, a sinusoid that happens to lie exactly in the middle between two levels and has amplitude less than $Q/2$ will be quantized to be a square wave, with all the upper humps of the sinusoid being rounded up and all the lower ones rounded down. The resulting error $e(n)$ will be highly periodic, that is, correlated from sample to sample, and not resembling random white noise. It will also be highly correlated with input sinusoid $x(n)$.

In digital audio, quantization distortions arising from low-level signals are referred to as *granulation* noise and correspond to unpleasant sounds. They can be virtually eliminated by the use of *dither*, which is low-level noise added to the signal before quantization.

The beneficial effect of dithering is to make the overall quantization error behave as a white noise signal, which is *perceptually* much more preferable and acceptable than the gross granulation distortions of the undithered signal [68–85]. On the negative side, dithering reduces the signal-to-noise ratio somewhat—between 3 to 6 dB depending on the type of dither used. Dither will be discussed in Section 2.5.

2.2 Oversampling and Noise Shaping

In the frequency domain, the assumption that $e(n)$ is a white noise sequence means that it has a flat spectrum. More precisely, the total average power σ_e^2 of $e(n)$ is distributed *equally* over the Nyquist interval $[-f_s/2, f_s/2]$, as shown in Fig. 2.2.1.

Thus, the power per unit frequency interval or *power spectral density* of $e(n)$ will be[†]

$$S_{ee}(f) = \frac{\sigma_e^2}{f_s}, \qquad \text{for} \quad -\frac{f_s}{2} \le f \le \frac{f_s}{2} \tag{2.2.1}$$

[†] In units of digital frequency $\omega = 2\pi f/f_s$, it is $S_{ee}(\omega) = \sigma_e^2/2\pi$.

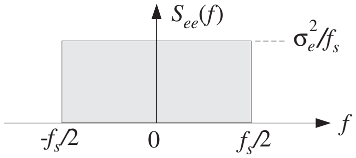


Fig. 2.2.1 Power spectrum of white quantization noise.

and it is periodic outside this interval with period f_s. The noise power within any Nyquist subinterval $[f_a, f_b]$ of width $\Delta f = f_b - f_a$ is given by

$$S_{ee}(f)\,\Delta f = \sigma_e^2 \frac{\Delta f}{f_s} = \sigma_e^2 \frac{f_b - f_a}{f_s}$$

As expected, the total power over the entire interval $\Delta f = f_s$ will be

$$\frac{\sigma_e^2}{f_s} f_s = \sigma_e^2$$

Noise shaping quantizers reshape the spectrum of the quantization noise into a more convenient shape. This is accomplished by filtering the white noise sequence $e(n)$ by a noise shaping filter $H_{\mathrm{NS}}(f)$. The *equivalent noise model* for the quantization process is shown in Fig. 2.2.2. The corresponding quantization equation, replacing Eq. (2.1.8), becomes:

$$\boxed{x_{\mathrm{Q}}(n) = x(n) + \varepsilon(n)} \tag{2.2.2}$$

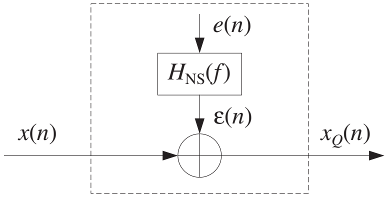


Fig. 2.2.2 Model of noise shaping quantizer.

where $\varepsilon(n)$ denotes the filtered noise. The sequence $\varepsilon(n)$ is no longer white. Its power spectral density is not flat, but acquires the shape of the filter $H_{\mathrm{NS}}(f)$:

$$S_{\varepsilon\varepsilon}(f) = |H_{\mathrm{NS}}(f)|^2 S_{ee}(f) = \frac{\sigma_e^2}{f_s}\,|H_{\mathrm{NS}}(f)|^2 \tag{2.2.3}$$

The noise power within a given subinterval $[f_a, f_b]$ is obtained by integrating $S_{\varepsilon\varepsilon}(f)$ over that subinterval:

$$\text{Power in } [f_a, f_b] = \int_{f_a}^{f_b} S_{\varepsilon\varepsilon}(f)\,df = \frac{\sigma_e^2}{f_s}\int_{f_a}^{f_b} |H_{\mathrm{NS}}(f)|^2\,df \tag{2.2.4}$$

Noise shaping quantizers are implemented by the so-called *delta-sigma* A/D converters [350], which are increasingly being used in commercial products such as digital audio sampling systems for hard disk or digital tape recording. We will discuss implementation details in Chapter 14. Here, we give a broad overview of the advantages of such quantizers.

The concepts of sampling and quantization are independent of each other. The first corresponds to the quantization of the time axis and the second to the quantization of the amplitude axis. Nevertheless, it is possible to trade off one for the other. Oversampling was mentioned earlier as a technique to alleviate the need for high quality prefilters and postfilters. It can also be used to trade off bits for samples. In other words, if we sample at a higher rate, we can use a coarser quantizer. Each sample will be less accurate, but there will be many more of them and their effect will average out to recover the lost accuracy.

The idea is similar to performing multiple measurements of a quantity, say x. Let σ_x^2 be the mean-square error in a single measurement. If L independent measurements of x are made, it follows from the law of large numbers that the measurement error will be reduced to σ_x^2/L, improving the accuracy of measurement. Similarly, if σ_x^2 is increased, making each individual measurement worse, one can maintain the *same* level of quality as long as the number of measurements L is also increased commensurately to keep the ratio σ_x^2/L constant.

Consider two cases, one with sampling rate f_s and B bits per sample, and the other with higher sampling rate f_s' and B' bits per sample. The quantity:

$$\boxed{L = \frac{f_s'}{f_s}}$$

is called the *oversampling ratio* and is usually an integer. We would like to show that B' can be less than B and still maintain the same level of quality. Assuming the same full-scale range R for the two quantizers, we have the following quantization widths:

$$Q = R2^{-B}, \qquad Q' = R2^{-B'}$$

and quantization noise powers:

$$\sigma_e^2 = \frac{Q^2}{12}, \qquad \sigma_e'^2 = \frac{Q'^2}{12}$$

To maintain the same quality in the two cases, we require that the power spectral densities remain the same, that is, using Eq. (2.2.1):

$$\frac{\sigma_e^2}{f_s} = \frac{\sigma_e'^2}{f_s'}$$

which can be rewritten as,

$$\sigma_e^2 = f_s\frac{\sigma_e'^2}{f_s'} = \frac{\sigma_e'^2}{L} \tag{2.2.5}$$

Thus, the total quantization power σ_e^2 is less than $\sigma_e'^2$ by a factor of L, making B greater than B'. The meaning of this result is shown pictorially in Fig. 2.2.3. If sampling

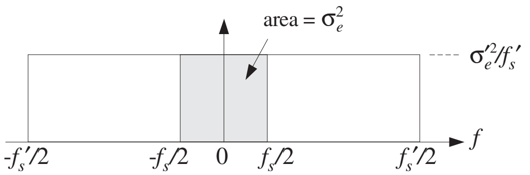


Fig. 2.2.3 Oversampled quantization noise power, without noise shaping.

is done at the higher rate f_s', then the total power $\sigma_e'^2$ of the quantization noise is spread evenly over the f_s' Nyquist interval.

The shaded area in Fig. 2.2.3 gives the *proportion* of the $\sigma_e'^2$ power that lies within the smaller f_s interval. Solving Eq. (2.2.5) for L and expressing it in terms of the difference $\Delta B = B - B'$, we find:

$$L = \frac{\sigma_e'^2}{\sigma_e^2} = 2^{2(B-B')} = 2^{2\Delta B}$$

or, equivalently,

$$\boxed{\Delta B = 0.5\log_2 L} \tag{2.2.6}$$

that is, a saving of half a bit per doubling of L. This is too small to be useful. For example, in order to reduce a 16-bit quantizer for digital audio to a 1-bit quantizer, that is, $\Delta B = 15$, one would need the unreasonable oversampling ratio of $L = 2^{30}$.

A *noise shaping* quantizer operating at the higher rate f_s' can reshape the flat noise spectrum so that most of the power is squeezed out of the f_s Nyquist interval and moved into the outside of that interval. Fig. 2.2.4 shows the power spectrum of such a quantizer.

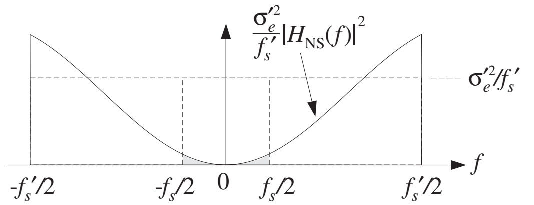


Fig. 2.2.4 Spectrum of oversampling noise shaping quantizer.

The total quantization noise power that resides within the original f_s Nyquist interval is the shaded area in this figure. It can be calculated by integrating Eq. (2.2.4) over $[-f_s/2, f_s/2]$:

$$\sigma_e^2 = \frac{\sigma_e'^2}{f_s'}\int_{-f_s/2}^{f_s/2} |H_{\text{NS}}(f)|^2\, df \tag{2.2.7}$$

Note that it reduces to Eq. (2.2.5) if there is no noise shaping, that is, $H_{\text{NS}}(f) = 1$. We will see in Section 14.7 that a typical pth order noise shaping filter operating at the high rate f_s' has magnitude response:

$$|H_{\text{NS}}(f)|^2 = \left|2\sin\left(\frac{\pi f}{f_s'}\right)\right|^{2p}, \qquad \text{for} \quad -\frac{f_s'}{2} \le f \le \frac{f_s'}{2} \tag{2.2.8}$$

For small frequencies f, we may use the approximation, $\sin x \simeq x$, to obtain:

$$|H_{\text{NS}}(f)|^2 = \left(\frac{2\pi f}{f_s'}\right)^{2p}, \qquad \text{for } |f| \ll f_s'/2 \tag{2.2.9}$$

Assuming a large oversampling ratio L, we will have $f_s \ll f_s'$, and therefore, we can use the approximation (2.2.9) in the integrand of Eq. (2.2.7). This gives:

$$\begin{aligned}\sigma_e^2 &= \frac{\sigma_e'^2}{f_s'}\int_{-f_s/2}^{f_s/2}\left(\frac{2\pi f}{f_s'}\right)^{2p} df = \sigma_e'^2\,\frac{\pi^{2p}}{2p+1}\left(\frac{f_s}{f_s'}\right)^{2p+1}\\ &= \sigma_e'^2\,\frac{\pi^{2p}}{2p+1}\,\frac{1}{L^{2p+1}}\end{aligned}$$

Using $\sigma_e^2/\sigma_e'^2 = 2^{-2(B-B')} = 2^{-2\Delta B}$, we obtain:

$$2^{-2\Delta B} = \frac{\pi^{2p}}{2p+1}\,\frac{1}{L^{2p+1}}$$

Solving for ΔB, we find the gain in bits:

$$\boxed{\Delta B = (p + 0.5)\log_2 L - 0.5\log_2\left(\frac{\pi^{2p}}{2p+1}\right)} \tag{2.2.10}$$

Now, the savings are $(p + 0.5)$ bits per doubling of L. Note that Eq. (2.2.10) reduces to Eq. (2.2.6) if there is no noise shaping, that is, $p = 0$. Practical values for the order p are at present $p = 1, 2, 3$, with $p = 4, 5$ becoming available. Table 2.2.1 compares the gain in bits ΔB versus oversampling ratio L for various quantizer orders.

p	L	4	8	16	32	64	128
0	$\Delta B = 0.5\log_2 L$	1.0	1.5	2.0	2.5	3.0	3.5
1	$\Delta B = 1.5\log_2 L - 0.86$	2.1	3.6	5.1	6.6	8.1	9.6
2	$\Delta B = 2.5\log_2 L - 2.14$	2.9	5.4	7.9	10.4	12.9	15.4
3	$\Delta B = 3.5\log_2 L - 3.55$	3.5	7.0	10.5	14.0	17.5	21.0
4	$\Delta B = 4.5\log_2 L - 5.02$	4.0	8.5	13.0	17.5	22.0	26.5
5	$\Delta B = 5.5\log_2 L - 6.53$	4.5	10.0	15.5	21.0	26.5	32.0

Table 2.2.1 Performance of oversampling noise shaping quantizers.

The first CD player built by Philips used a first-order noise shaper with 4-times oversampling, that is, $p = 1$, $L = 4$, which according to the table, achieves a savings of $\Delta B = 2.1$ bits. Because of that, the Philips CD player used a 14-bit, instead of a 16-bit, D/A converter at the analog reconstructing stage [353].

We also see from the table that to achieve 16-bit CD-quality resolution using 1-bit quantizers, that is, $\Delta B = 15$, we may use a second-order 128-times oversampling quantizer. For digital audio rates $f_s = 44.1$ kHz, this would imply oversampling at $f_s' = Lf_s = 5.6$ MHz, which is feasible with the present state of the art. Alternatively, we may use third-order noise shaping with 64-times oversampling.

An overall DSP system that uses oversampling quantizers with noise shaping is shown in Fig. 2.2.5. Sampling and reconstruction are done at the fast rate f_s' and at the reduced resolution of B' bits. Intermediate processing by the DSP is done at the low rate f_s and increased resolution of B bits. The overall quality remains the same through all the processing stages. Such a system replaces the traditional DSP system, shown in Fig. 1.7.1.

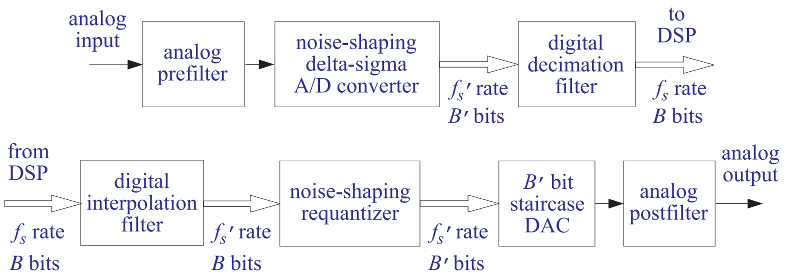


Fig. 2.2.5 Oversampling DSP system.

The faster sampling rate f_s' also allows the use of a less expensive, lower quality, antialiasing prefilter. The digital decimation filter converts the fast rate f_s' back to the desired low rate f_s at the higher resolution of B bits and removes the out-of-band quantization noise that was introduced by the noise shaping quantizer into the outside of the f_s Nyquist interval.

After digital processing by the DSP, the interpolation filter increases the sampling rate digitally back up to the fast rate f_s'. The noise shaping requantizer rounds the B-bit samples to B' bits, without reducing quality. Finally, an ordinary B'-bit staircase D/A converter reconstructs the samples to analog format and the postfilter smooths out the final output. Again, the fast rate f_s' allows the use of a low-quality postfilter.

Oversampling DSP systems are used in a variety of applications, such as digital transmission and coding of speech, the playback systems of CD players, and the sampling/playback systems of DATs. We will discuss the design of oversampling digital interpolation and decimation filters and the structure of noise shaping quantizers and $\Delta\Sigma$ converters in Chapter 14.

2.3 D/A Converters

Next, we discuss some coding details for standard A/D and D/A converters, such as binary representations of quantized samples and the successive approximation method of A/D conversion. We begin with D/A converters, because they are used as the building blocks of successive approximation ADCs. We take a functional view of such converters without getting into the electrical details of their construction.

Consider a B-bit DAC with full-scale range R, as shown in Fig. 2.3.1. Given B input bits of zeros and ones, $\mathbf{b} = [b_1, b_2, \dots, b_B]$, the converter outputs an analog value x_Q, that lies on one of the 2^B quantization levels within the range R. If the converter is unipolar, the output x_Q falls in the range $[0, R)$. If it is bipolar, it falls in $[-R/2, R/2)$.

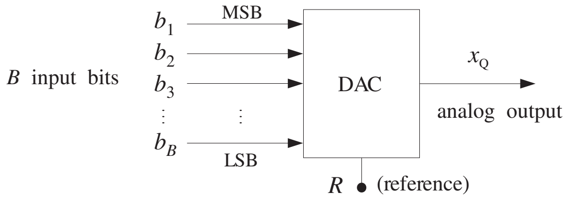


Fig. 2.3.1 B-bit D/A converter.

The manner in which the B bits $[b_1, b_2, \dots, b_B]$ are associated with the analog value x_Q depends on the type of converter and the coding convention used. We will discuss the three widely used types: (a) unipolar natural binary, (b) bipolar offset binary, and (c) bipolar two's complement converters.

The *unipolar natural binary* converter is the simplest. Its output x_Q is computed in terms of the B bits by:

$$\boxed{x_Q = R(b_1 2^{-1} + b_2 2^{-2} + \cdots + b_B 2^{-B})} \tag{2.3.1}$$

The minimum level is $x_Q = 0$ and is reached when all the bits are zero, $\mathbf{b} = [0, 0, \dots, 0]$. The smallest nonzero level is $x_Q = Q = R2^{-B}$ and corresponds to the least significant bit (LSB) pattern $\mathbf{b} = [0, 0, \dots, 0, 1]$. The most significant bit (MSB) pattern is $\mathbf{b} = [1, 0, 0, \dots, 0]$ and corresponds to the output value $x_Q = R/2$. The maximum level is reached when all bits are one, that is, $\mathbf{b} = [1, 1, \dots, 1]$ and corresponds to the analog output:

$$x_Q = R(2^{-1} + 2^{-2} + \cdots + 2^{-B}) = R(1 - 2^{-B}) = R - Q$$

where we used the geometric series

$$\begin{aligned} 2^{-1} + 2^{-2} + \cdots + 2^{-B} &= 2^{-1}\left(1 + 2^{-1} + 2^{-2} + \cdots + 2^{-(B-1)}\right) \\ &= 2^{-1}\left(\frac{1 - 2^{-B}}{1 - 2^{-1}}\right) = 1 - 2^{-B} \end{aligned}$$

Eq. (2.3.1) can be written also in terms of the quantization width Q, as follows:

$$x_Q = R2^{-B}(b_1 2^{B-1} + b_2 2^{B-2} + \cdots + b_{B-1} 2^1 + b_B) \qquad \text{or,}$$

$$\boxed{x_Q = Qm} \tag{2.3.2}$$

where m is the integer whose binary representation is $(b_1 b_2 \cdots b_B)$, that is,

$$m = b_1 2^{B-1} + b_2 2^{B-2} + \cdots + b_{B-1} 2^1 + b_B$$

As the integer m takes on the 2^B consecutive values $m = 0, 1, 2, \dots, 2^B - 1$, the analog output x_Q runs through the quantizer's consecutive levels. The *bipolar offset binary* converter is obtained by shifting Eq. (2.3.1) down by half-scale, $R/2$, giving the rule:

$$\boxed{x_Q = R(b_1 2^{-1} + b_2 2^{-2} + \cdots + b_B 2^{-B} - 0.5)} \tag{2.3.3}$$

The minimum and maximum attainable levels are obtained by shifting the corresponding natural binary values by $R/2$:

$$x_Q = 0 - \frac{R}{2} = -\frac{R}{2} \qquad \text{and} \qquad x_Q = (R - Q) - \frac{R}{2} = \frac{R}{2} - Q$$

The analog value x_Q can also be expressed in terms of Q, as in Eq. (2.3.2). In this case we have:

$$\boxed{x_Q = Qm'} \tag{2.3.4}$$

where m' is the integer m shifted by half the maximum scale, that is,

$$m' = m - \frac{1}{2} 2^B = m - 2^{B-1}$$

It takes on the sequence of 2^B values

$$m' = -2^{B-1}, \dots, -2, -1, 0, 1, 2, \dots, 2^{B-1} - 1$$

One unnatural property of the offset binary code is that the level $x_Q = 0$ is represented by the nonzero bit pattern $\mathbf{b} = [1, 0, \dots, 0]$. This is remedied by the *two's complement* code, which is the most commonly used code. It is obtained from the offset binary code by *complementing* the most significant bit, that is, replacing b_1 by $\overline{b}_1 = 1 - b_1$, so that

$$\boxed{x_Q = R(\overline{b}_1 2^{-1} + b_2 2^{-2} + \cdots + b_B 2^{-B} - 0.5)} \tag{2.3.5}$$

Table 2.3.1 summarizes the three converter types and their input/output coding conventions and Table 2.3.2 compares the three coding conventions for the case $B = 4$ and $R = 10$ volts. The level spacing is $Q = R/2^B = 10/2^4 = 0.625$ volts. The codes $[b_1, b_2, b_3, b_4]$ in the first column, apply to both the natural and offset binary cases, but the quantized analog values that they represent are different.

For the natural binary case, the values x_Q are positive, spanning the range $[0, 10)$ volts, with the maximum value being $R - Q = 10 - 0.625 = 9.375$. For offset binary, the level values are offset by half scale, $R/2 = 5$ volts, and span the range $[-5, 5)$ volts, with the maximum being $R/2 - Q = 5 - 0.625 = 4.375$ volts. Note that the upper ends of the full-scale range, $R = 10$ and $R/2 = 5$ volts, are shown in the table for reference and do not represent a level.

The last column shows the two's complement codes. They are obtained from the first column by complementing the MSB, b_1. The quantized values x_Q represented by these codes are the *same* as in the offset binary case, that is, given in the fifth column of the table.

Converter type	I/O relationship
natural binary	$x_Q = R(b_1 2^{-1} + b_2 2^{-2} + \cdots + b_B 2^{-B})$
offset binary	$x_Q = R(b_1 2^{-1} + b_2 2^{-2} + \cdots + b_B 2^{-B} - 0.5)$
two's complement	$x_Q = R(\overline{b}_1 2^{-1} + b_2 2^{-2} + \cdots + b_B 2^{-B} - 0.5)$

Table 2.3.1 Converter types.

	natural binary		offset binary		2's C
$b_1b_2b_3b_4$	m	$x_Q = Qm$	m'	$x_Q = Qm'$	$b_1b_2b_3b_4$
—	16	10.000	8	5.000	—
1 1 1 1	15	9.375	7	4.375	0 1 1 1
1 1 1 0	14	8.750	6	3.750	0 1 1 0
1 1 0 1	13	8.125	5	3.125	0 1 0 1
1 1 0 0	12	7.500	4	2.500	0 1 0 0
1 0 1 1	11	6.875	3	1.875	0 0 1 1
1 0 1 0	10	6.250	2	1.250	0 0 1 0
1 0 0 1	9	5.625	1	0.625	0 0 0 1
1 0 0 0	8	5.000	0	0.000	0 0 0 0
0 1 1 1	7	4.375	−1	−0.625	1 1 1 1
0 1 1 0	6	3.750	−2	−1.250	1 1 1 0
0 1 0 1	5	3.125	−3	−1.875	1 1 0 1
0 1 0 0	4	2.500	−4	−2.500	1 1 0 0
0 0 1 1	3	1.875	−5	−3.125	1 0 1 1
0 0 1 0	2	1.250	−6	−3.750	1 0 1 0
0 0 0 1	1	0.625	−7	−4.375	1 0 0 1
0 0 0 0	0	0.000	−8	−5.000	1 0 0 0

Table 2.3.2 Converter codes for $B = 4$ bits, $R = 10$ volts.

The two's complement code can be understood by wrapping the linear natural binary code around in a circle, as shown in Fig. 2.3.2. This figure shows the natural binary integers m and their negative values in the lower half of the circle. The negative of any positive m in the upper semicircle can be obtained by the usual rule of complementing all its bits and adding one, that is, $m_{2c} = \overline{m} + 1$.

Example 2.3.1: In Table 2.3.2 or Fig. 2.3.2, the level $m = 3$ corresponds to the natural binary quantized value of $x_Q = 1.875$ volts. The two's complement of m is obtained by the rule

$$m_{2c} = \overline{m} + 1 = \overline{(0011)} + (0001) = (1100) + (0001) = (1101) = -3$$

which, according to the fifth column of the table, corresponds to the two's complement

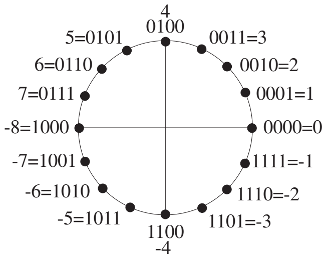


Fig. 2.3.2 Two's complement code.

quantized value $x_Q = -1.875$ volts. □

The following C routine `dac.c` simulates the operation of the *bipolar two's complement* converter. Its inputs are the B bits $[b_1, b_2, \dots, b_B]$, and the full-scale range R, and its output is the analog value x_Q computed by Eq. (2.3.5).

```
/* dac.c - bipolar two's complement D/A converter */

double dac(b, B, R)
int *b, B;                            bits are dimensioned as b[0], b[1], ... , b[B − 1]
double R;
{
       int i;
       double dac = 0;

       b[0] = 1 - b[0];                         complement MSB

       for (i = B-1; i >= 0; i--)               Hörner's rule
          dac = 0.5 * (dac + b[i]);

       dac = R * (dac - 0.5);                   shift and scale

       b[0] = 1 - b[0];                         restore MSB

       return dac;
}
```

Its usage is:

```
xQ = dac(b, B, R);
```

Because of the default indexing of arrays in C, the B-dimensional bit vector $b[i]$ is indexed for $i = 0, 1, \dots, B-1$. The declaration and dimensioning of $b[i]$ should be done in the main program. For example, if $B = 4$, the main program must include a line:

```
int b[4];
```

The array $b[i]$ can also be allocated dynamically for any desired value of B using `calloc`. The main program must include the lines:

```
int *b;                                         b is a pointer to int
B = 4;                                          B can also be read from stdin
b = (int *) calloc(B, sizeof(int));             allocates B int slots
```

The internal for-loop in `dac.c` implements a variant of Hörner's rule for evaluating a polynomial. The result is the binary sum $\bar{b}_1 2^{-1} + b_2 2^{-2} + \cdots + b_B 2^{-B}$ which is then shifted by 0.5 and scaled by R. We leave the details of Hörner's algorithm for Problems 2.10–2.13. This algorithm will be used again later for the evaluation of z-transforms and DTFTs. (See the MATLAB function `dtft.m` in Appendix C.)

The routine `dac` may be modified easily to implement the natural binary and offset binary converter types, given by Eqs. (2.3.1) and (2.3.3).

2.4 A/D Converters

A/D converters quantize an analog value x so that it is represented by B bits $[b_1, b_2, \dots, b_B]$, as shown in Fig. 2.4.1. ADCs come in many varieties, depending on how the conversion process is implemented. One of the most popular ones is the *successive approximation* A/D converter whose main building block is a D/A converter in a feedback loop. It is shown in Fig. 2.4.2.

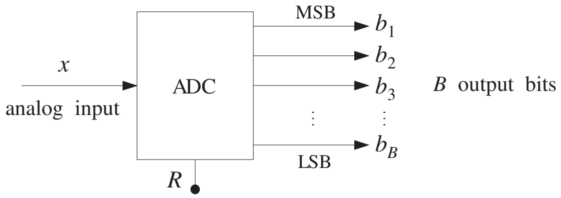


Fig. 2.4.1 B-bit A/D converter.

The conversion algorithm is as follows. Initially all B bits are cleared to zero, $\mathbf{b} = [0, 0, \dots, 0]$, in the successive approximation register (SAR). Then, starting with the MSB b_1, each bit is turned *on* in sequence and a test is performed to determine whether that bit should be left *on* or turned *off*.

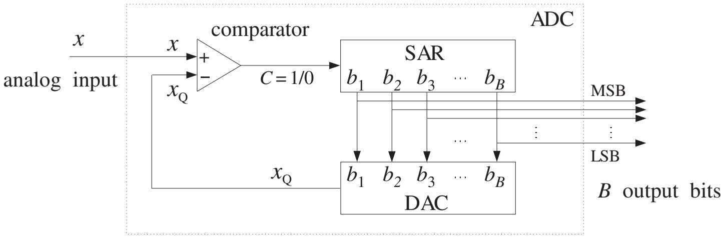


Fig. 2.4.2 Successive approximation A/D converter.

The control logic puts the correct value of that bit in the right slot in the SAR register. Then, leaving all the tested bits set at their correct values, the next bit is turned *on* in the SAR and the process repeated. After B tests, the SAR will hold the correct bit vector $\mathbf{b} = [b_1, b_2, \dots, b_B]$, which can be sent to the output.

At each test, the SAR bit vector $\mathbf{b}$ is applied to the DAC which produces the analog quantized value x_Q. When a given bit is turned *on*, the output x_Q of the DAC is compared with the analog input x to the ADC. If $x \geq x_Q$, that bit is kept *on*; else, it is turned *off.* The output C of the comparator is the correct value of the bit being tested. The algorithm is summarized below:

```
for each x to be converted, do:
    initialize  $\mathbf{b} = [0, 0, \dots, 0]$
    for  $i = 1, 2, \dots, B$  do:
        $b_i = 1$
        $x_Q = \text{dac}(\mathbf{b}, B, R)$
        if  $(x \geq x_Q)$
            $C = 1$
        else
            $C = 0$
        $b_i = C$
```

Therefore, C becomes a *serial representation* of the bit vector $\mathbf{b}$. The algorithm imitates the operations shown in Fig. 2.4.2. It can be written more compactly as follows:

```
for each x to be converted, do:
    initialize  $\mathbf{b} = [0, 0, \dots, 0]$
    for  $i = 1, 2, \dots, B$  do:
        $b_i = 1$
        $x_Q = \text{dac}(\mathbf{b}, B, R)$
        $b_i = u(x - x_Q)$
```

where $u(x)$ is the unit-step function, defined by:

$$u(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

As stated above, the algorithm applies to the natural and offset binary cases (with corresponding versions of `dac`). It implements *truncation* of x to the quantization level just below, instead of rounding to the nearest level.

The algorithm converges to the right quantization level by performing a *binary search* through the quantization levels. This can be seen with the help of the first column of Table 2.3.2. The test $b_1 = 1$ or 0 determines whether x lies in the upper or lower *half* of the levels. Then, the test $b_2 = 1/0$ determines whether x lies in the upper/lower half of the first half, and so on. Some examples will make this clear.

Example 2.4.1: Convert the analog values $x = 3.5$ and $x = -1.5$ volts to their offset binary representation, assuming $B = 4$ bits and $R = 10$ volts, as in Table 2.3.2.

Solution: The following table shows the successive tests of the bits, the corresponding DAC output x_Q at each test, and the comparator output $C = u(x - x_Q)$.

test	$b_1b_2b_3b_4$	x_Q	$C = u(x - x_Q)$
b_1	1 0 0 0	0.000	1
b_2	1 1 0 0	2.500	1
b_3	1 1 1 0	3.750	0
b_4	1 1 0 1	3.125	1
	1 1 0 1	3.125	

For each bit pattern, the DAC inputs/outputs were looked up in the first/fifth columns of Table 2.3.2, instead of computing them via Eq. (2.3.3). When b_1 is tested, the DAC output is $x_Q = 0$ which is less than x; therefore, b_1 passes the test. Similarly, b_2 passes the test and stays on. On the other hand bit b_3 fails the test because $x < x_Q = 3.75$; thus, b_3 is turned off. Finally, b_4 passes.

The last row gives the final content of the SAR register and the corresponding quantized value $x_Q = 3.125$. Even though $x = 3.5$ lies in the upper half between the two levels $x_Q = 3.75$ and $x_Q = 3.125$, it gets truncated down to the lower level. The C column is a serial representation of the final answer.

Note also the binary searching taking place: $b_1 = 1$ selects the upper half of the levels, $b_2 = 1$ selects the upper half of the upper half, and of these, $b_3 = 0$ selects the lower half, and of those, $b_4 = 1$ selects the upper half. For the case $x = -1.5$ we have the testing table

test	$b_1b_2b_3b_4$	x_Q	$C = u(x - x_Q)$
b_1	1 0 0 0	0.000	0
b_2	0 1 0 0	−2.500	1
b_3	0 1 1 0	−1.250	0
b_4	0 1 0 1	−1.875	1
	0 1 0 1	−1.875	

Bit b_1 fails the test because $x < x_Q = 0$, and therefore, $b_1 = 0$, and so on. Again, the final quantized value $x_Q = -1.875$ is that obtained by truncating $x = -1.5$ to the level below it, even though x lies nearer the level above it. □

In order to quantize by *rounding* to the nearest level, we must shift x by half the spacing between levels, that is, use:

$$\boxed{y = x + \frac{1}{2}Q}$$

in place of x and perform *truncation* on y. If x is already in the upper half between two levels, then y will be brought above the upper level and will be truncated down to that level. The conversion algorithm for rounding is:

```
for each x to be converted, do:
    $y = x + Q/2$
    initialize $\mathbf{b} = [0, 0, \dots, 0]$
    for $i = 1, 2, \dots, B$ do:
        $b_i = 1$
        $x_Q = \mathrm{dac}(\mathbf{b}, B, R)$
        $b_i = u(y - x_Q)$
```

Example 2.4.2: To quantize the value $x = 3.5$ by rounding, we shift it to $y = x + Q/2 = 3.5 + 0.625/2 = 3.8125$. The corresponding test table will be

test	$b_1b_2b_3b_4$	x_Q	$C = u(y - x_Q)$
b_1	1 0 0 0	0.000	1
b_2	1 1 0 0	2.500	1
b_3	1 1 1 0	3.750	1
b_4	1 1 1 1	4.375	0
	1 1 1 0	3.750	

Only b_4 fails the test because with it on, the DAC output $x_Q = 4.375$ exceeds y. The final value $x_Q = 3.750$ is the rounded up version of $x = 3.5$. For the case $x = -1.5$, we have $y = -1.5 + 0.625/2 = -1.1875$. The corresponding test table is

test	$b_1b_2b_3b_4$	x_Q	$C = u(y - x_Q)$
b_1	1 0 0 0	0.000	0
b_2	0 1 0 0	−2.500	1
b_3	0 1 1 0	−1.250	1
b_4	0 1 1 1	−0.625	0
	0 1 1 0	−1.250	

The value $x_Q = -1.250$ is the rounded up version of $x = -1.5$. □

The successive approximation algorithm for the *two's complement* case is slightly different. Because the MSB is complemented, it must be treated separately from the other bits. As seen in the last column of Table 2.3.2, the bit b_1 determines whether the number x is positive or negative. If $x \geq 0$ then, we must have $b_1 = 0$; else $b_1 = 1$. We can express this result by $b_1 = 1 - u(x)$, or, $b_1 = 1 - u(y)$ if we are quantizing by rounding. The remaining bits, $\{b_2, b_3, \dots, b_B\}$, are tested in the usual manner. This leads to the following two's complement conversion algorithm with rounding:

```
for each x to be converted, do:
    $y = x + Q/2$
    initialize $\mathbf{b} = [0, 0, \dots, 0]$
    $b_1 = 1 - u(y)$
    for $i = 2, 3, \dots, B$ do:
        $b_i = 1$
        $x_Q = \mathrm{dac}(\mathbf{b}, B, R)$
        $b_i = u(y - x_Q)$
```

Example 2.4.3: The two's complement rounded 4-bit representations of $x = 3.5$ and $x = -1.5$ are:

$$
\begin{array}{lllll}
x = 3.5 & \Rightarrow & x_Q = 3.750 & \Rightarrow & \mathbf{b} = [0, 1, 1, 0] \\
x = -1.5 & \Rightarrow & x_Q = -1.250 & \Rightarrow & \mathbf{b} = [1, 1, 1, 0]
\end{array}
$$

They are obtained from the offset binary by complementing the MSB. The quantized values x_Q are the same as in the offset binary case — only the binary codes change. □

Example 2.4.4: Consider the sampled sinusoid $x(n) = A\cos(2\pi f n)$, where $A = 3$ volts and $f = 0.04$ cycles/sample. The sinusoid is evaluated at the ten sampling times $n = 0, 1, \dots, 9$ and $x(n)$ is quantized using a 4-bit successive approximation ADC with full-scale range $R = 10$ volts. The following table shows the *sampled and quantized* values $x_Q(n)$ and the offset binary and two's complement binary codes representing them.

n	$x(n)$	$x_Q(n)$	2's C	offset
0	3.000	3.125	0101	1101
1	2.906	3.125	0101	1101
2	2.629	2.500	0100	1100
3	2.187	1.875	0011	1011
4	1.607	1.875	0011	1011
5	0.927	0.625	0001	1001
6	0.188	0.000	0000	1000
7	−0.562	−0.625	1111	0111
8	−1.277	−1.250	1110	0110
9	−1.912	−1.875	1101	0101

The 2's complement and offset binary codes differ only in their MSB. The quantized values they represent are the same. □

The following routine `adc.c` simulates the operation of a *bipolar two's complement* successive approximation ADC. It makes successive calls to `dac.c` to determine each bit.

```
/* adc.c - successive approximation A/D converter */

#include <math.h>

double dac();
int u();

void adc(x, b, B, R)
double x, R;
int *b, B;
{
          int i;
          double y, xQ, Q;

          Q = R / pow(2, B);                           quantization width Q = R/2^B
          y = x + Q/2;                                 rounding

          for (i = 0; i < B; i++)                      initialize bit vector
                 b[i] = 0;

          b[0] = 1 - u(y);                             determine MSB
```

```
        for (i = 1; i < B; i++) {                    loop starts with i = 1
               b[i] = 1;                             turn ith bit ON
               xQ = dac(b, B, R);                    compute DAC output
               b[i] = u(y-xQ);                       test and correct bit
               }
}
```

The inputs to the routine are the analog value x to be converted and the full-scale range R. The outputs are the B bits $\mathbf{b} = [b_1, b_2, \dots, b_B]$ representing x in the two's complement representation. The unit-step function $u(x)$ is implemented by the routine:

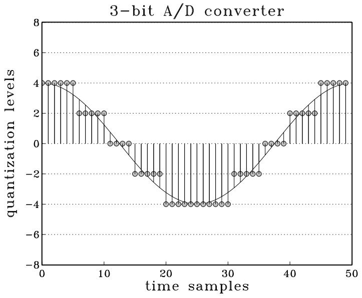
```
/* u.c - unit step function */

int u(x)
double x;
{
       if (x >= 0)
              return 1;
       else
              return 0;
}
```

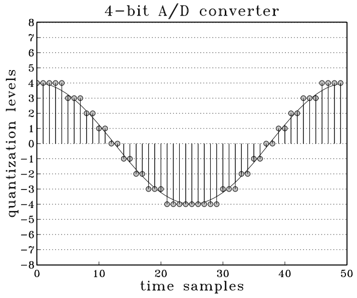
Example 2.4.5: This example illustrates the usage of the routines adc and dac. Consider $L = 50$ samples of a sinusoid $x(n) = A\cos(2\pi f n)$ of digital frequency $f = 0.02$ cycles/sample and amplitude $A = 4$. The signal $x(n)$ is quantized using a successive approximation two's complement converter with rounding, as implemented by the routine adc. The following for-loop was used in the main program for calculating $x_Q(n)$:

```
for (n=0; n<L; n++) {
       x[n]  = A * cos(2 * pi * f * n);
       adc(x[n], b, B, R);
       xQ[n] = dac(b, B, R);
       }
```

where each call to adc determines the bit vector $\mathbf{b}$, which is then passed to dac to calculate the quantized value.

The following figure shows the sampled and quantized signal $x_Q(n)$ plotted together with the exact values $x(n)$ for the two cases of a 3-bit and a 4-bit converter. The full-scale range was $R = 16$.





Example 2.4.6: This example illustrates the statistical properties of the quantization error. Consider L samples of the noisy sinusoidal signal:

$$x(n) = A\cos(2\pi f_0 n + \phi) + v(n)\,, \qquad n = 0, 1, \dots, L-1 \tag{2.4.1}$$

where ϕ is a random phase distributed uniformly in the interval $[0, 2\pi]$ and $v(n)$ is white noise of variance σ_v^2. Using a B-bit two's complement converter with full-scale range R, these samples are quantized to give $x_Q(n)$ and the quantization error is computed:

$$e(n) = x_Q(n) - x(n)\,, \qquad n = 0, 1, \dots, L-1$$

According to the standard statistical model discussed in Section 2.1, the quantization noise samples $e(n)$ should be distributed uniformly over the interval $-Q/2 \le e \le Q/2$. This can be tested by computing the histogram of the L values of $e(n)$.

The theoretical statistical quantities given in Eqs. (2.1.9–2.1.11) can be calculated experimentally by the *time-average* approximations:

$$\sigma_e^2 = \frac{1}{L}\sum_{n=0}^{L-1} e^2(n) \tag{2.4.2}$$

$$R_{ee}(k) = \frac{1}{L}\sum_{n=0}^{L-1-k} e(n+k)e(n) \tag{2.4.3}$$

$$R_{ex}(k) = \frac{1}{L}\sum_{n=0}^{L-1-k} e(n+k)x(n) \tag{2.4.4}$$

We can also compute the autocorrelation of $x(n)$ itself:

$$R_{xx}(k) = \frac{1}{L}\sum_{n=0}^{L-1-k} x(n+k)x(n) \tag{2.4.5}$$

where in the last three equations, k ranges over a few lags $0 \le k \le M$, with M typically being much less than $L-1$. Note also that $\sigma_e^2 = R_{ee}(0)$. All four of the above expressions are special cases of the cross correlation, Eq. (2.4.4), which is implemented by the correlation routine `corr.c`. For this experiment, we chose the following numerical values:

$B = 10$ bits
$R = 1024$ volts, so that $Q = 1$ volt
$L = 1000$ samples
$M = 50$ lags
$f_0 = 1/\sqrt{131} \simeq 0.08737$ cycles/sample
$A = R/4 = 256$ volts and $\phi = 0$

The white noise $v(n)$ was distributed uniformly over the interval $[-R/4, R/4]$. Such numbers can be generated by:

$$v = 0.5R(u - 0.5) \tag{2.4.6}$$

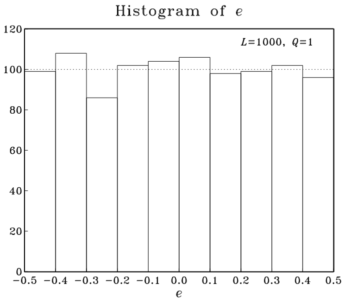
where u is a *uniform* random number in the standardized interval $[0, 1]$. The quantity $(u - 0.5)$ is uniform over $[-0.5, 0.5]$, making v uniform over $[-R/4, R/4]$. We used the routine ran of Appendix A.1 to generate the u's, but any other uniform random number generator could have been used. The samples $v(n)$ were generated by a for-loop of the form:

```
for (n=0; n<L; n++)
        v[n] = 0.5 * R * (ran(&iseed) - 0.5);
```

where the initial seed[†] was picked arbitrarily. With these choices, the sinusoidal and noise terms in $x(n)$ vary over half of the full-scale range, so that their sum varies over the full range $[-R/2, R/2]$, as is necessary for the model.

With $Q = 1$, the theoretical value of the noise variance is $\sigma_e = Q/\sqrt{12} = 1/\sqrt{12} = 0.289$. The experimental value computed using Eq. (2.4.2) was $\sigma_e = 0.287$.

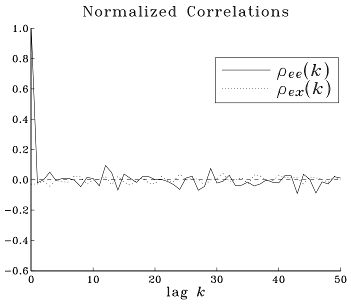
The histogram of the computed $e(n)$ values was computed by dividing the interval $[-Q/2, Q/2] = [-0.5, 0.5]$ into 10 bins. It is shown below. Theoretically, for a uniform distribution, 1000 samples would distribute themselves evenly over the 10 bins giving $1000/10 = 100$ samples per bin.

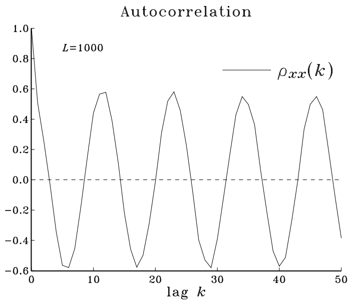


The next two figures show the standard *normalized* correlation functions:

$$\rho_{ee}(k) = \frac{R_{ee}(k)}{R_{ee}(0)}\,, \quad \rho_{ex}(k) = \frac{R_{ex}(k)}{\sqrt{R_{ee}(0)R_{xx}(0)}}\,, \quad \rho_{xx}(k) = \frac{R_{xx}(k)}{R_{xx}(0)}$$

computed for lags $k = 0, 1, \dots, M$ using Eqs. (2.4.3–2.4.5).





Theoretically, $\rho_{ee}(k)$ should be $\delta(k)$ and $\rho_{ex}(k)$ should be zero. Using the results of Problem 2.20, the theoretical expression for $\rho_{xx}(k)$ will be, for the particular numerical values of the parameters:

$$\rho_{xx}(k) = 0.6\cos(2\pi f_0 k) + 0.4\,\delta(k)$$

Thus, although $x(n)$ itself is highly self-correlated, the quantization noise $e(n)$ is not. The above figures show the closeness of the experimental quantities to the theoretical ones, confirming the reasonableness of the standard statistical model. □

[†] Note that iseed is passed by address in ran(&iseed).

Successive approximation A/D converters are used routinely in applications with sampling rates of 1 MHz or less. Slower converters also exist, the so-called counter or integrating type. They convert by searching through the quantization levels in a *linear* fashion, comparing each level with the value x to be converted. Therefore, they may require up to 2^B tests to perform a conversion. This is to be compared with the B binary searching steps of the successive approximation type.

For higher rates, *parallel* or flash A/D converters must be used. They determine all the bits simultaneously, in parallel, but they are electrically complex requiring $2^B - 1$ comparators internally. For this reason they are limited at present to $B \le 12$ bits, achieving conversion rates of 500 MHz with 8 bits, or 50 MHz with 10 bits [57]. As discussed in Problem 2.21, two or more flash A/D converters can be cascaded together, in a so-called subranging configuration, to increase the effective quantization resolution.

2.5 Analog and Digital Dither

Dither is a low-level white noise signal added to the input *before* quantization for the purpose of eliminating granulation or quantization distortions and making the total quantization error behave like white noise [68–85].

Analog dither can be added to the analog input signal before the A/D converter, but perhaps after the sample/hold operation. It is depicted in Fig. 2.5.1. In many applications, such as digital audio recordings, the inherent analog system noise of the microphones or mixers may already provide some degree of dithering and therefore artificial dithering may not be necessary.

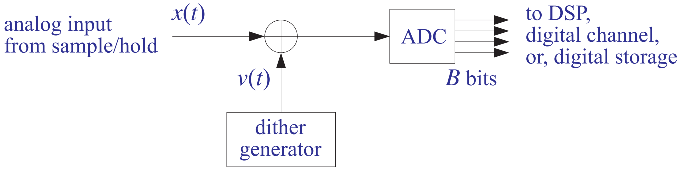


Fig. 2.5.1 Analog dither.

Digital dither can be added to a digital signal prior to a *requantization* operation that *reduces* the number of bits representing the signal.

This circumstance arises, for example, when an audio signal has been sampled and quantized with 20 bits for the purpose of high-quality digital mixing and processing, which then must be reduced to 16 bits for storing it on a CD. Another example is the noise shaping requantization required in oversampling D/A converters used in the playback systems of CD players and DAT decks.

Figure 2.5.2 shows a general model of the analog or digital dither process followed by the quantization operation. It represents a type of dither known as *nonsubtractive*.

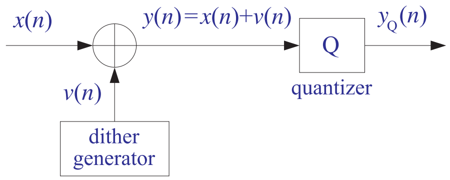


Fig. 2.5.2 Nonsubtractive dither process and quantization.

The input to the quantizer is the sum of the input signal $x(n)$ to be quantized or requantized and the dither noise $v(n)$, that is,

$$y(n)=x(n)+v(n)$$

The output of the quantizer is $y_Q(n)$, the quantized version of $y(n)$. The quantization error is,

$$e(n)=y_Q(n)-y(n)$$

Thus, the *total* error resulting from dithering *and* quantization will be:

$$\epsilon(n)=y_Q(n)-x(n) \tag{2.5.1}$$

which can be written as

$$\epsilon(n)=\big(y(n)+e(n)\big)-x(n)=x(n)+v(n)+e(n)-x(n)$$

or,

$$\epsilon(n)=y_Q(n)-x(n)=e(n)+v(n) \tag{2.5.2}$$

that is, the sum of the dither noise plus the quantization error. Proper choice of the dither process $v(n)$ can guarantee that $e(n)$ and $v(n)$ will be uncorrelated from each other, and therefore the total error noise power will be

$$\boxed{\sigma_\epsilon^2=\sigma_e^2+\sigma_v^2=\frac{1}{12}Q^2+\sigma_v^2} \tag{2.5.3}$$

The statistical properties of the dither signal $v(n)$, such as its probability density function (pdf), can affect drastically the nature of the total error (2.5.2). In practice, there are three commonly used types of dithers, namely, those with Gaussian, rectangular, or triangular pdf's. The pdf's of the rectangular and triangular cases are shown in Fig. 2.5.3.

Note that the areas under the curves are equal to unity. In the Gaussian case, the zero-mean pdf is:

$$p(v)=\frac{1}{\sqrt{2\pi\sigma_v^2}}e^{-v^2/2\sigma_v^2} \tag{2.5.4}$$

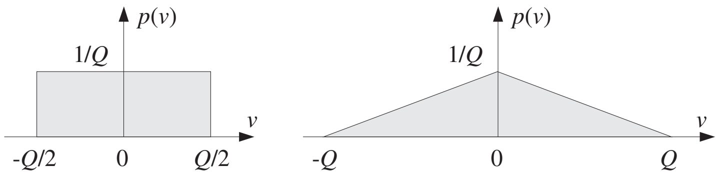


Fig. 2.5.3 Rectangular and triangular dither probability densities.

with the recommended value for the variance:

$$\sigma_v^2=\frac{1}{4}Q^2 \tag{2.5.5}$$

which corresponds to the rms value $v_{\rm rms}=Q/2$, or half-LSB. It follows from Eq. (2.5.3) that the total error variance will be:

$$\sigma_\epsilon^2=\frac{1}{12}Q^2+\frac{1}{4}Q^2=4\cdot\frac{1}{12}Q^2=\frac{1}{3}Q^2$$

In the rectangular case, the pdf is taken to have width Q, that is, 1-LSB. Therefore, the dither signal can only take values in the interval:

$$-\frac{1}{2}Q\le v\le\frac{1}{2}Q$$

The corresponding pdf and variance are in this case:

$$p(v)=\begin{cases}\dfrac{1}{Q}, & \text{if } -\dfrac{1}{2}Q\le v\le\dfrac{1}{2}Q\\ 0, & \text{otherwise}\end{cases}\quad\text{and}\quad \sigma_v^2=\frac{1}{12}Q^2 \tag{2.5.6}$$

Therefore, the total error variance will be:

$$\sigma_\epsilon^2=\frac{1}{12}Q^2+\frac{1}{12}Q^2=2\cdot\frac{1}{12}Q^2=\frac{1}{6}Q^2$$

Similarly, the width of the triangular dither pdf is taken to be $2Q$, that is, 2-LSB, and therefore, the corresponding pdf and variance are:

$$p(v)=\begin{cases}\dfrac{Q-|v|}{Q^2}, & \text{if } -Q\le v\le Q\\ 0, & \text{otherwise}\end{cases}\quad\text{and}\quad \sigma_v^2=\frac{1}{6}Q^2 \tag{2.5.7}$$

and, the total error variance will be:

$$\sigma_\epsilon^2=\frac{1}{12}Q^2+\frac{1}{6}Q^2=3\cdot\frac{1}{12}Q^2=\frac{1}{4}Q^2$$

In summary, the total error variance in the three cases and the undithered case $(v=0)$ will be:

$$\sigma_\epsilon^2=\begin{cases}Q^2/12, & \text{undithered}\\ 2Q^2/12, & \text{rectangular dither}\\ 3Q^2/12, & \text{triangular dither}\\ 4Q^2/12, & \text{Gaussian dither}\end{cases} \tag{2.5.8}$$

Thus, the noise penalty in using dither is to double, triple, or quadruple the noise of the undithered case. This corresponds to a decrease of the SNR by:

$$10\log_{10} 2 = 3 \text{ dB}$$

$$10\log_{10} 3 = 4.8 \text{ dB}$$

$$10\log_{10} 4 = 6 \text{ dB}$$

which is quite acceptable in digital audio systems that have total SNRs of the order of 95 dB.

It has been shown that the *triangular* dither is the best (of the nonsubtractive types) in the sense that it accomplishes the main objective of the dithering process, namely, to eliminate the quantization distortions of the undithered case and to render the total error (2.5.2) equivalent to white noise [72].

Rectangular, uniformly distributed dither can be generated very simply by using a uniform random number generator such as ran. For example,

$$v = Q(u - 0.5)$$

where u is the random number returned by ran, that is, uniformly distributed over the interval $[0, 1)$. The shifting and scaling of u imply that v will be uniformly distributed within $-Q/2 \le v < Q/2$.

Triangular dither can be generated just as simply by noting that the triangular pdf is the convolution of two rectangular ones and therefore v can be obtained as the *sum* of two independent rectangular random numbers, that is,

$$v = v_1 + v_2 \tag{2.5.9}$$

where v_1, v_2 are generated from two independent uniform u_1, u_2, by

$$v_1 = Q(u_1 - 0.5) \qquad \text{and} \qquad v_2 = Q(u_2 - 0.5)$$

Example 2.5.1: This simulation example illustrates the impact of dither on the quantization of a low-amplitude sinusoid and the removal of quantization distortions. Consider the following dithered sinusoid:

$$y(n) = x(n) + v(n) = A\cos(2\pi f_0 n) + v(n)$$

where A is taken to be below 1-LSB; for example, $A = 0.75Q$. The frequency f_0 is taken to be $f_0 = 0.025$ cycles/sample, which corresponds to $1/f_0 = 40$ samples per cycle. At an audio rate of $f_s = 40$ kHz, this frequency would correspond to a 1 kHz sinusoid.

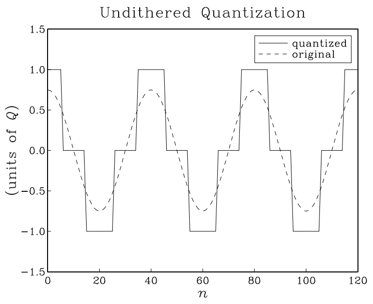
The signal $y(n)$ is quantized using a 3-bit A/D converter, $(B = 3)$, with full-scale range of $R = 8$ volts. Therefore, $Q = R/2^B = 8/2^3 = 1$, and $A = 0.75Q = 0.75$. Triangular dither was used, generated by Eq. (2.5.9). The dithered signal $y(n)$ and its quantized version $y_Q(n)$ were generated by the following loop:

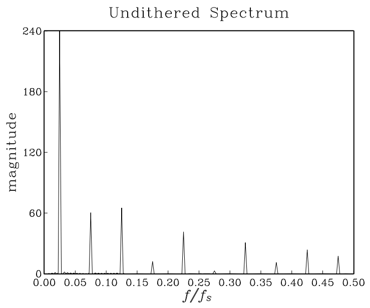
```
for (n=0; n<Ntot; n++) {
      v1 = Q * (ran(&iseed) - 0.5);
      v2 = Q * (ran(&iseed) - 0.5);
      v = v1 + v2;
```

```
      y[n] = A * cos(2 * pi * f0 * n) + v;
      adc(y[n], b, B, R);
      yQ[n] = dac(b, B, R);
      }
```

Note that v1 and v2 are independent of each other because each call to ran updates the seed to a new value.

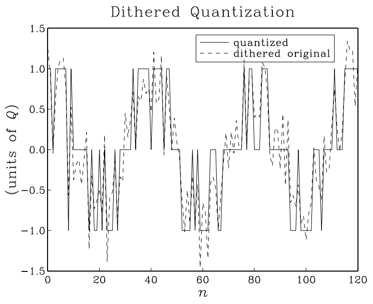
The following graphs show the undithered sinusoid $x(n)$ and its quantized version $x_Q(n)$, together with its Fourier spectrum $|X_Q(f)|$ plotted over the right half of the Nyquist interval, that is, $0 \le f \le 0.5$, in units of cycles/sample.

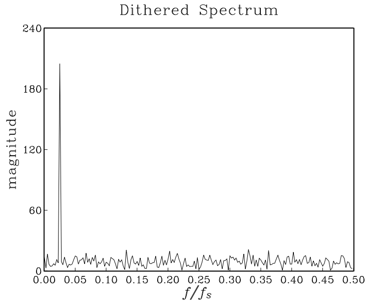




The spectrum of $x_Q(n)$ has peaks at f_0 and the odd harmonics $3f_0$, $5f_0$, and so forth. These harmonics were not present in $x(n)$. They are the artifacts of the quantization process which replaced the sinusoid by a square-wave-like signal.

The next two graphs show the dithered signal $y(n)$ and its quantized version $y_Q(n)$, together with its spectrum $|Y_Q(f)|$.





The main peak at f_0 is still there, but the odd harmonics have been eliminated by the dithering process and replaced by a typical featureless background noise spectrum. For digital audio, this noise is perceptually far more acceptable than the artificial harmonics introduced by the quantizer.

The above spectra were computed in the following way: The sequences $x_Q(n)$ and $y_Q(n)$ were generated for $0 \le n \le N_{\text{tot}} - 1$, with $N_{\text{tot}} = 1000$. Then, they were windowed using a length-N_{tot} Hamming window, that is,

$$y'_Q(n) = w(n)y_Q(n), \qquad n = 0, 1, \dots, N_{\text{tot}} - 1$$

where,

$$w(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{N_{\text{tot}} - 1}\right), \qquad n = 0, 1, \dots, N_{\text{tot}} - 1$$

And, their DTFT

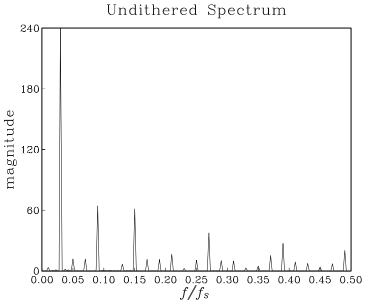
$$Y_Q(f) = \sum_{n=0}^{N_{\text{tot}}-1} y'_Q(n)e^{-2\pi jfn}$$

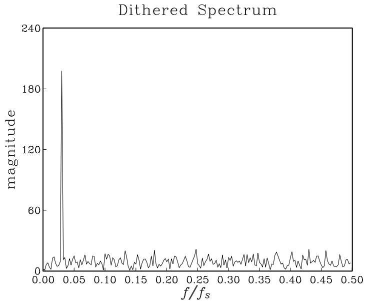
was evaluated at 200 equally spaced frequencies f over the interval $0 \le f \le 0.5$ [cycles/sample], that is, at $f_i = 0.5i/200$, $i = 0, 1, \dots, 199$.

This example is somewhat special in that the undithered spectrum $X_Q(f)$ contained only odd harmonics of the fundamental frequency f_0. This is what one would expect if the quantized square-wave-like signal $x_Q(n)$ were an unsampled, analog, signal.

In general, the sampling process will cause all the odd harmonics that lie outside the Nyquist interval to be aliased back into the interval, onto frequencies that may or may not be odd harmonics. In the above example, because the sampling rate is an even multiple of f_0, that is, $f_s = 40f_0$, one can show that any odd harmonic of f_0 that lies outside the Nyquist interval will be wrapped onto one of the odd harmonics inside the interval.

But, for other values of f_0, the out-of-band odd harmonics may be aliased onto in-band non-harmonic frequencies. For example, the following graphs show the undithered and dithered spectra in the case of $f_0 = 0.03$ cycles/sample.





In addition to the odd harmonics at $3f_0 = 0.09$, $5f_0 = 0.15$, and so forth, one sees non-harmonic peaks at:

$$f = 0.01, 0.05, 0.07, 0.13, 0.17, 0.19, 0.23, 0.25, \dots$$

which are the aliased versions of the following out-of-band odd harmonics:

$$33f_0, 35f_0, 31f_0, 29f_0, 39f_0, 27f_0, 41f_0, 25f_0, \dots$$

The beneficial effect of dithering works, of course, for any value of f_0. □

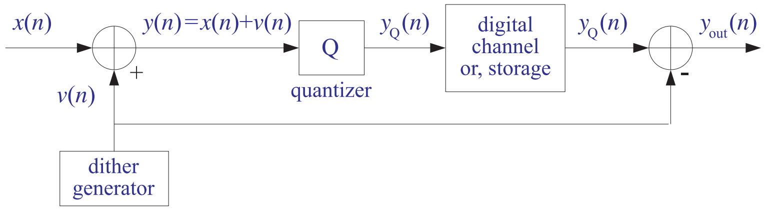


Fig. 2.5.4 Subtractive dither.

An alternative dithering strategy is to use the so-called *subtractive* dither, as shown in Fig. 2.5.4. Here, the dither noise $v(n)$ that was added during the recording or transmission phase prior to quantization is subtracted at the playback or receiving end. The total error in this case can be determined as follows:

$$\epsilon(n) = y_{\text{out}}(n) - x(n) = \big(y_Q(n) - v(n)\big) - x(n) = y_Q(n) - \big(x(n) + v(n)\big)$$

or,

$$\epsilon(n) = y_Q(n) - y(n) = e(n)$$

that is, only the quantizer error. Therefore, its variance remains the same as the undithered case, $\sigma_\epsilon^2 = Q^2/12$, and the SNR remains the same.

It can be shown [72] that the *best* type of dither is subtractive rectangularly distributed dither with 1-LSB width, in the sense that it not only removes the quantization distortions but also renders the total error completely *independent* of the input signal. However, its practical implementation in digital audio and other applications is difficult because it requires a copy of the dither signal at the playback or receiving end.

By contrast, the triangular nonsubtractive dither that we considered earlier does not make the total error independent of the input—it only makes the power spectrum of the error independent of the input. In digital audio, this whitening of the total error appears to be enough perceptually. Therefore, triangular nonsubtractive dither is the best choice for practical use [72].

In summary, triangular nonsubtractive dither improves the quality of a digital processing system by removing the artifacts of the quantization process with only a modest decrease in the signal-to-noise ratio. It may be applied at any intermediate processing stage that involves reduction in the number of bits and, therefore, potential quantization distortions.

2.6 Problems

2.1 Consider a 3-bit successive approximation two's complement bipolar A/D converter with full scale range of $R = 16$ volts. Using the successive approximation algorithm, determine the quantized value as well as the corresponding 3-bit representation of the following analog input values: $x = 2.9, 3.1, 3.7, 4, -2.9, -3.1, -3.7, -4$.

Repeat using an offset binary converter.

2.2 Consider the signal $x(n) = 5\sin(2\pi f n)$, where $f = 0.04$ [cycles/sample]. This signal is to be quantized using a 4-bit successive approximation bipolar ADC whose full-scale range is $R = 16$ volts. For $n = 0, 1, \dots, 19$, compute the numerical value of $x(n)$ and its quantized version $x_Q(n)$ as well as the corresponding bit representation at the output of the converter. Do this both for an *offset binary* converter and a *two's complement* converter.

2.3 It is desired to pick an A/D converter for a DSP application that meets the following specifications: The full-scale range of the converter should be 10 volts and the rms quantization error should be kept below 1 millivolt. How many bits should the converter have? What is the actual rms error of the converter? What is the dynamic range in dB of the converter?

2.4 Hard disk recording systems for digital audio are becoming widely available. It is often quoted that to record 1 minute of "CD quality" digital audio in *stereo*, one needs about 10 Megabytes of hard disk space. Please, derive this result, explaining your reasoning.

2.5 A digital audio mixing system uses 16 separate recording channels, each sampling at a 48 kHz rate and quantizing each sample with 20 bits. The digitized samples are saved on a hard disk for further processing.

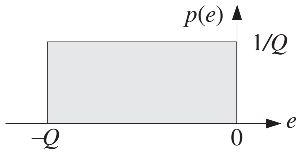
a. How many megabytes of hard disk space are required to record a 3-minute song for a 16-channel recording?

b. Each channel requires about 35 multiplier/accumulation (MAC) instructions to perform the processing of each input sample. (This corresponds to about 7 second-order parametric EQ filters covering the audio band.)

In how many nanoseconds should each MAC instruction be executed for: (i) each channel? (ii) all 16 channels, assuming they are handled by a single processor? Is this within the capability of present day DSP chips?

2.6 If the quantized value x_Q is obtained by *truncation* of x instead of rounding, show that the truncation error $e = x_Q - x$ will be in the interval $-Q < e \le 0$. Assume a uniform probability density $p(e)$ over this interval, that is,

$$p(e) = \begin{cases} \dfrac{1}{Q} & \text{if } -Q < e \le 0 \\ 0 & \text{otherwise} \end{cases}$$



Determine the mean $m_e = E[e]$ and variance $\sigma_e^2 = E[(e - m_e)^2]$ in terms of Q.

2.7 Using Eq. (2.2.10), determine the value of the oversampling ratio L to achieve 16-bit resolution using 1-bit quantizers for the cases of first-, second-, and third-order noise shaping quantizers. What would be the corresponding oversampled rate Lf_s for digital audio?

2.8 In a speech codec, it is desired to maintain quality of 8-bit resolution at 8 kHz sampling rates using a 1-bit oversampled noise shaping quantizer. For quantizer orders $p = 1, 2, 3$, determine the corresponding oversampling ratio L and oversampling rate Lf_s in Hz.

2.9 Show that the two's complement expression defined in Eq. (2.3.5) can be written in the alternative form:

$$x_Q = R\left(-b_1 2^{-1} + b_2 2^{-2} + \cdots + b_B 2^{-B}\right)$$

2.10 Hörner's rule is an efficient algorithm for polynomial evaluation. Consider a polynomial of degree M

$$B(z) = b_0 + b_1 z + b_2 z^2 + \cdots + b_M z^M$$

Hörner's algorithm for evaluating $B(z)$ at some value of z, say $z = a$, can be stated as follows:

```
initialize $p = 0$
for $i = M, M-1, \dots, 0$ do:
    $p = ap + b_i$
```

Verify that upon exit, p will be the value of the polynomial at $z = a$, that is, $p = B(a)$.

2.11 *Computer Experiment: Hörner's Rule.* Write a polynomial evaluation C routine `pol.c` that implements the algorithm of Problem 2.10. The routine should return the value $B(a)$ of the polynomial and should be dimensioned as follows:

```
double pol(M, b, a)
int M;                          order of polynomial
double *b, a;                   b is (M+1)-dimensional
```

2.12 Consider the following variation of Hörner's algorithm:

```
initialize $q_{M-1} = b_M$
for $i = M-1, M-2, \dots, 1$ do:
    $q_{i-1} = aq_i + b_i$
$p = aq_0 + b_0$
```

where the final computation yields $p = B(a)$. Show that it is equivalent to the algorithm of Problem 2.10. This version is equivalent to "synthetic division" in the following sense.

The computed coefficients $\{q_0, q_1, \dots, q_{M-1}\}$ define a polynomial of degree $(M-1)$, namely, $Q(z) = q_0 + q_1 z + \cdots + q_{M-1} z^{M-1}$.

Show that $Q(z)$ is the *quotient* polynomial of the division of $B(z)$ by the monomial $z - a$. Moreover, the *last* computed $p = B(a)$ is the *remainder* of that division. That is, show as an identity in z

$$B(z) = (z - a)Q(z) + p$$

2.13 In the `dac` routines, the polynomial to be evaluated is of the form

$$B(z) = b_1 z + b_2 z^2 + \cdots + b_M z^M$$

The `dac` routines evaluate it at the specific value $z = 2^{-1}$. Show that the polynomial $B(z)$ can be evaluated at $z = a$ by the following modified version of Hörner's rule:

```
initialize $p = 0$
for $i = M, M-1, \dots, 1$ do:
    $p = a(p + b_i)$
```

2.14 Consider a 4-bit successive approximation A/D converter with full-scale range of 8 volts. Using the successive approximation algorithm (with rounding), determine the 4-bit codes of the voltage values $x = 1.2, 5.2, -3.2$ volts, for the following types of converters:

a. Natural binary.

a. Bipolar two's complement.

In each case, show all the steps of the successive approximation algorithm. Explain what happens if the analog voltage to be converted lies *outside* the full-scale range of the converter. This happens for $x = 5.2$ in two's complement, and $x = -3.2$ in natural binary representations.

2.15 Carry out, by hand, the successive approximation conversion of all the signal values shown in the table of Example 2.4.4, for both the offset binary and two's complement cases.

2.16 *Computer Experiment: DAC and ADC Routines.* Write C versions of the routines `dac` and `adc` for the natural binary, offset binary, and two's complement cases that implement *truncation.*

For the natural and offset binary cases, write another set of such routines that implement *rounding.*

2.17 *Computer Experiment: Simulating DAC and ADC Operations.* Generate $L = 50$ samples of a sinusoidal signal $x(n) = A\cos(2\pi f n)$, $n = 0, 1, \dots, L-1$ of frequency $f = 0.02$ [cycles/sample] and amplitude $A = 8$.

a. Using a 3-bit ($B = 3$) bipolar two's complement successive approximation A/D converter, as implemented by the routine `adc`, with full-scale range $R = 32$, quantize $x(n)$ and denote the quantized signal by $x_Q(n)$.

For $n = 0, 1, \dots, L-1$, print in three parallel columns the true analog value $x(n)$, the quantized value $x_Q(n)$, and the corresponding two's complement bit vector $\mathbf{b}$.

On the same graph, plot the two signals $x(n)$ and $x_Q(n)$ versus n. Scale the vertical scales from $[-16, 16]$ and use 8 y-grid lines to indicate the 8 quantization levels.

b. Repeat part (a) using a $B = 4$ bit A/D converter. In plotting $x(n)$ and $x_Q(n)$, use the *same* vertical scales as above, namely, from $[-16, 16]$, but use 16 y-grid lines to show the 16 quantization levels.

c. What happens if the analog signal to be quantized has amplitude that *exceeds* the full-scale range of the quantizer? Most D/A converters will *saturate* to their largest (positive or negative) levels. To see this, repeat part (a) by taking the amplitude of the sinusoid to be $A = 20$.

d. What happens if we use truncation instead of rounding? Repeat part (a) using the two's complement truncation routines `adc` and `dac` that you developed in the previous problem.

2.18 *Computer Experiment: Quantization Noise Model.* Reproduce the results of Example 2.4.6.

2.19 Show that the mean and variance of the random variable v defined by Eq. (2.4.6) of Example 2.4.6 are $m_v = 0$ and $\sigma_v^2 = R^2/48$.

2.20 Show that the normalized autocorrelation function $\rho_{xx}(k)$ of the signal $x(n)$ given by Eq. (2.4.1) in Example 2.4.6, is given by

$$\rho_{xx}(k) = \frac{R_{xx}(k)}{R_{xx}(0)} = a\cos(2\pi f_0 k) + (1-a)\delta(k)\,, \quad \text{where} \quad a = \frac{SNR}{SNR+1}$$

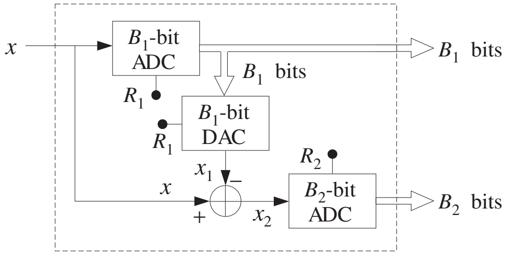
where $R_{xx}(k)$ defined as the statistical expectation value

$$R_{xx}(k) = E[x(n+k)x(n)]$$

Assume that phase of the sinusoid ϕ is not correlated with $v(n)$. The quantity SNR is the signal-to-noise ratio $SNR = A^2/(2\sigma_v^2)$. For the numerical values of Example 2.4.6, show $a = 0.6$.

2.21 *Computer Experiment: Subranging Converters.* It was mentioned that parallel A/D converters are at present limited in their bits. However, it is possible to use two of them in cascade. For example, using two identical 6-bit flash ADCs, one can achieve effective resolution of 12 bits at conversion rates of 10 MHz.

Consider a B-bit ADC and write B as the sum of two integers $B = B_1 + B_2$. The conversion of an analog value x to its B-bit representation can be done in two stages: First, convert x into its B_1-bit representation. This is equivalent to keeping the first B_1 most significant bits of its B-bit representation. Let x_1 be the quantized B_1-bit value. Then, form the difference $x_2 = x - x_1$ and quantize it to B_2 bits. These operations are shown in the following figure:



The B_1-bit word from the first ADC is sent to the output and also to a B_1-bit DAC whose output is x_1. The analog subtracter forms x_2, which is sent to the B_2-bit ADC producing the remaining B_2 bits.

a. What should be the full-scale ranges R_1 and R_2 of the two ADCs in order for this arrangement to be *equivalent* to a single $(B_1 + B_2)$-bit ADC with full-scale range R? What is the relationship of R_1 and R_2 in terms of R?

b. Using the routines `adc` and `dac` as building blocks, write a routine that implements this block diagram. Test your routine on the signal:

$$x(n) = A\cos(2\pi f_0 n)\,, \qquad n = 0, 1, \dots, L-1$$

where $A = 4$, $f_0 = 0.02$, and $L = 50$. Take $B_1 = 5$, $B_2 = 5$, $B = B_1 + B_2 = 10$, and $R = 16$. Compare the results of your routine with the results of an equivalent single B-bit ADC with full-scale range R.

c. How does the block diagram generalize in the case of cascading three such converters, such that $B = B_1 + B_2 + B_3$?

2.22 *Computer Experiment: Triangular Dither.* Reproduce the results and graphs of Example 2.5.1.

3

Discrete-Time Systems

In this and the next chapter, we discuss discrete-time systems and, in particular, linear time-invariant (LTI) systems. The input/output (I/O) relationship of LTI systems is given by the discrete-time convolution of the system's impulse response with the input signal.

LTI systems can be classified into *finite impulse response* (FIR) or *infinite impulse response* (IIR) types depending on whether their impulse response has finite or infinite duration. Our main objective in these two chapters is to develop practical *computational algorithms* for the FIR case. The IIR case is considered in Chapter 7, although we do present a few simple IIR examples here.

Depending on the application and hardware, an FIR digital filtering operation can be organized to operate either on a *block* basis or a *sample-by-sample* basis.

In the *block processing* case, the input signal is considered to be a single block of signal samples. The block is filtered by *convolving* it with the filter, generating the output signal as another block of samples.

If the input signal is very long or infinite in duration, this method requires modification—for example, breaking up the input into multiple blocks of manageable size, filtering the blocks one at a time, and piecing together the resulting output blocks to form the overall output. The filtering of each block can be implemented in various ways, such as by ordinary convolution, or fast convolution via the FFT.

In the *sample processing* case, the input samples are processed *one at a time* as they arrive at the input. The filter operates as a state machine; that is, each input sample is used in conjunction with the current *internal state* of the filter to compute the current output sample and also to *update* the internal state of the filter in preparation for processing the *next* input sample.

This approach is useful in *real-time applications* involving very long input signals. It is also useful in adaptive filtering applications where the filter itself changes after processing each sample. Moreover, it is efficiently implemented with present day DSP chip families, such as the Texas Instruments TMS320, the Bell Labs AT&T DSP16/32, the Motorola DSP56K/96K, and the Analog Devices ADSP2101 families. The architectures and instruction sets of these chips are optimized for such sample-by-sample processing operations.

3.1 Input/Output Rules

A *discrete-time system*, shown in Fig. 3.1.1, is a processor that transforms an input sequence of discrete-time samples $x(n)$ into an output sequence of samples $y(n)$, according to some *input/output rule* that specifies how to compute the output sequence $y(n)$ from the knowledge of the input sequence $x(n)$. In sample-by-sample processing methods, we may think of the I/O rule as processing the input samples one at a time:[†]

$$\{x_0, x_1, x_2, \dots, x_n, \dots\} \xrightarrow{H} \{y_0, y_1, y_2, \dots, y_n, \dots\}$$

that is, $x_0 \xrightarrow{H} y_0$, $x_1 \xrightarrow{H} y_1$, $x_2 \xrightarrow{H} y_2$, and so on. In block processing methods, we think of the input sequence as a block or vector of signal samples being processed as a whole by the system, producing the corresponding output block:

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \end{bmatrix} \xrightarrow{H} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \end{bmatrix} = \mathbf{y}$$

Thus, the I/O rule maps the input vector $\mathbf{x}$ into the output vector $\mathbf{y}$ according to some functional mapping:

$$\mathbf{y} = H[\mathbf{x}] \tag{3.1.1}$$

For linear systems, this mapping becomes a linear transformation by a matrix H, $\mathbf{y} = H\mathbf{x}$. For linear and time-invariant systems, the matrix H has a special structure being built in terms of the impulse response of the system.

Some examples of discrete-time systems illustrating the wide variety of possible I/O rules are given below.

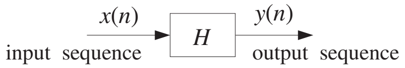


Fig. 3.1.1 Discrete-time system.

Example 3.1.1: $y(n) = 2x(n)$. It corresponds to simple scaling of the input:

$$\{x_0, x_1, x_2, x_3, x_4, \dots\} \xrightarrow{H} \{2x_0, 2x_1, 2x_2, 2x_3, 2x_4, \dots\}$$

Example 3.1.2: $y(n) = 2x(n) + 3x(n-1) + 4x(n-2)$. A weighted average of three successive input samples. At each time instant n, the system must remember the *previous* input samples $x(n-1)$ and $x(n-2)$ in order to use them.

[†] For brevity, we denoted $\{x(0), x(1), x(2), \dots\}$ by subscripts $\{x_0, x_1, x_2, \dots\}$.

Example 3.1.3: Here, the I/O rule is specified as a block processing operation by a linear transformation, transforming a length-4 input block $\{x_0, x_1, x_2, x_3\}$ into a length-6 output block:

$$\mathbf{y} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 3 & 2 & 0 & 0 \\ 4 & 3 & 2 & 0 \\ 0 & 4 & 3 & 2 \\ 0 & 0 & 4 & 3 \\ 0 & 0 & 0 & 4 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = H\mathbf{x}$$

It is equivalent to the convolutional form of Example 3.1.2. The output block is longer than the input block by two samples because this filter has memory two — the last two outputs being the input-off transients generated after the input is turned off. If we had to filter length-5 input blocks $\{x_0, x_1, x_2, x_3, x_4\}$, the linear transformation would have one more column and row:

$$\mathbf{y} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 3 & 2 & 0 & 0 & 0 \\ 4 & 3 & 2 & 0 & 0 \\ 0 & 4 & 3 & 2 & 0 \\ 0 & 0 & 4 & 3 & 2 \\ 0 & 0 & 0 & 4 & 3 \\ 0 & 0 & 0 & 0 & 4 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = H\mathbf{x}$$

Example 3.1.4: Example 3.1.2 can also be cast in an equivalent sample-by-sample processing form described by the following system of three equations:

$$\begin{aligned} y(n) &= 2x(n) + 3w_1(n) + 4w_2(n) \\ w_2(n+1) &= w_1(n) \\ w_1(n+1) &= x(n) \end{aligned}$$

The auxiliary signals $w_1(n)$ and $w_2(n)$ can be thought of as the internal states of the system. The present input sample $x(n)$ together with the knowledge of the present internal states $\{w_1(n), w_2(n)\}$ is sufficient to compute the present output $y(n)$. The next output $y(n+1)$ due to the next input $x(n+1)$ requires knowledge of the updated states $\{w_1(n+1), w_2(n+1)\}$, but these are already available from the nth time step; thus, at time $n+1$ we have:

$$\begin{aligned} y(n+1) &= 2x(n+1) + 3w_1(n+1) + 4w_2(n+1) \\ w_2(n+2) &= w_1(n+1) \\ w_1(n+2) &= x(n+1) \end{aligned}$$

The computations are repetitive from one time instant to the next and can be summarized by the following I/O *sample-by-sample processing algorithm* which tells how to process each arriving input sample x producing the corresponding output sample y and updating the internal states:[†]

```
for each new input sample $x$ do:
    $y := 2x + 3w_1 + 4w_2$
    $w_2 := w_1$
    $w_1 := x$
```

[†] The symbol := denotes *assignment* not equation, that is, $a := b$ means "a takes on the value b."

Once the current values of the internal states $\{w_1, w_2\}$ are used in the computation of the output y, they may be updated by the last two assignment equations to the values they must have for processing the *next* input sample. Therefore, $\{w_1, w_2\}$ must be saved from call to call of the algorithm. The order in which $\{w_1, w_2\}$ are updated is important, that is, w_2 is updated first and w_1 second, to prevent overwriting of the correct values.

This and the previous two examples represent equivalent formulations of the same discrete-time system. Deciding which form to use depends on the nature of the application—that is, whether the input signals are finite or infinite sequences and the samples must be processed one at a time as they arrive.

This example is a special case of more general *state-space* representations of discrete-time systems described by the following I/O sample processing algorithm:

$$\begin{aligned} y(n) &= g\big(x(n), \mathbf{s}(n)\big) && \text{(output equation)} \\ \mathbf{s}(n+1) &= \mathbf{f}\big(x(n), \mathbf{s}(n)\big) && \text{(state updating equation)} \end{aligned}$$

where $\mathbf{s}(n)$ is an internal state vector of appropriate dimension, like $\mathbf{s}(n) = \begin{bmatrix} w_1(n) \\ w_2(n) \end{bmatrix}$ of the previous example. The I/O algorithm calculates both the output $y(n)$ and the next state $\mathbf{s}(n+1)$ from the knowledge of the present input $x(n)$ and the present state $\mathbf{s}(n)$. It can be rephrased in the repetitive algorithmic form:

```
for each new input sample $x$ do:
    $y := g(x, \mathbf{s})$
    $\mathbf{s} := \mathbf{f}(x, \mathbf{s})$
```

State-space realizations of LTI systems are described by functions $\mathbf{f}$ and g that are *linear functions* of their arguments, that is, $\mathbf{f}(x, \mathbf{s}) = A\mathbf{s} + Bx$, $g(x, \mathbf{s}) = C\mathbf{s} + Dx$, where A, B, C, D have appropriate dimensions. In particular, for the above example we have

$$y := 2x + 3w_1 + 4w_2 = [3, 4]\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + 2x = [3, 4]\mathbf{s} + 2x \equiv g(x, \mathbf{s})$$

$$\mathbf{s} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} := \begin{bmatrix} x \\ w_1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} x = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}\mathbf{s} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} x \equiv \mathbf{f}(x, \mathbf{s})$$

Example 3.1.5: $y(n) = 0.5y(n-1) + 2x(n) + 3x(n-1)$. The output is computed recursively by a constant-coefficient difference equation. At each time instant n, the system must remember the previous input *and* output samples $x(n-1), y(n-1)$.

Example 3.1.6: Example 3.1.5 can also be described by stating its I/O rule as a sample-by-sample processing algorithm:

```
for each new input sample $x$ do:
    $y := 0.5w_1 + 2x + 3v_1$
    $w_1 := y$
    $v_1 := x$
```

It corresponds to the so-called *direct form* realization of the difference equation and requires the computation and updating of the auxiliary quantities $\{w_1, v_1\}$. Its equivalence to Example 3.1.5 will be seen later.

An alternative I/O computational rule for Example 3.1.5, corresponding to the so-called *canonical* realization of the system, is as follows:

```
for each new input sample x do:
    $w_0 := x + 0.5w_1$
    $y := 2w_0 + 3w_1$
    $w_1 := w_0$
```

It uses the auxiliary quantities $\{w_0, w_1\}$.

Example 3.1.7: $y(n) = \frac{1}{5}\big[x(n+2)+x(n+1)+x(n)+x(n-1)+x(n-2)\big]$. Smoother or averager of five successive samples. The operation is slightly non-causal, because at each time n, the system must know the next samples $x(n+1)$ and $x(n+2)$. We will see later how to handle such cases in real time.

Example 3.1.8: $y(n) = 2x(n)+3$. Scaling and shifting of the input.

Example 3.1.9: $y(n) = x^2(n)$. Squaring the input.

Example 3.1.10: $y(n) = 2x(n)+3x(n-1)+x(n)x(n-1)$. It contains a nonlinear cross-product term $x(n)x(n-1)$.

Example 3.1.11: $y(n) = \text{med}[x(n+1), x(n), x(n-1)]$. A simple *median filter*, where the operation $\text{med}[a,b,c]$ represents the median of the three numbers a,b,c obtained by sorting the three numbers in increasing order and picking the middle one.

Example 3.1.12: $y(n) = nx(n)$. It has a time-varying coefficient.

Example 3.1.13: $y(n) = \frac{1}{n}\big[x(0)+x(1)+\cdots+x(n-1)\big]$. Cumulative average of n numbers. It can also be expressed recursively as in the following example.

Example 3.1.14: $y(n+1) = a_n y(n)+b_n x(n)$, where $a_n = n/(n+1)$, $b_n = 1-a_n = 1/(n+1)$. It corresponds to a first-order difference equation with time-varying coefficients a_n, b_n.

Example 3.1.15: $y(n) = x(2n)$. It acts as a rate compressor or *downsampler*, keeping every other sample of $x(n)$, thus, resulting in half of the input samples. That is, the input and output sequences are:

$$\{x_0, x_1, x_2, x_3, x_4, x_5, x_6, \dots\} \xrightarrow{H} \{x_0, x_2, x_4, x_6, \dots\}$$

Example 3.1.16: $y(n) = \begin{cases} x(n/2), & \text{if } n \text{ is even} \\ 0, & \text{if } n \text{ is odd} \end{cases}$. It acts as a rate expander or *upsampler*, inserting a zero sample between the samples of $x(n)$, thus, doubling the number of input samples. That is, the input and output sequences are:

$$\{x_0, x_1, x_2, x_3, x_4, \dots\} \xrightarrow{H} \{x_0, 0, x_1, 0, x_2, 0, x_3, 0, x_4, \dots\}$$

Examples 3.1.1–3.1.7 represent LTI systems; Examples 3.1.8–3.1.11 are nonlinear but time-invariant; and Examples 3.1.12–3.1.16 are linear but time-varying systems.

Although most applications of DSP use linear time-invariant systems, nonlinear and time-varying systems are used increasingly in a variety of applications. For example, median filters are used successfully in image processing because of their excellent edge-preserving and noise removal properties; time-varying filters are used in adaptive filtering applications, such as channel equalization and echo cancellation in digital data or voice channels; downsamplers and upsamplers are part of multirate signal processing systems, such as those used for interpolation, decimation, oversampling, and sample rate conversion.

3.2 Linearity and Time Invariance

A *linear system* has the property that the output signal due to a linear combination of two or more input signals can be obtained by forming the same linear combination of the individual outputs. That is, if $y_1(n)$ and $y_2(n)$ are the outputs due to the inputs $x_1(n)$ and $x_2(n)$, then the output due to the linear combination of inputs

$$x(n) = a_1 x_1(n) + a_2 x_2(n) \tag{3.2.1}$$

is given by the linear combination of outputs

$$y(n) = a_1 y_1(n) + a_2 y_2(n) \tag{3.2.2}$$

To test linearity one must determine separately the three outputs $y(n)$, $y_1(n)$, and $y_2(n)$ and then show that they satisfy Eq. (3.2.2). The required operations are shown in Fig. 3.2.1.

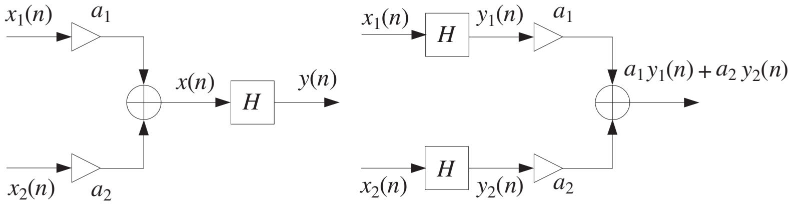


Fig. 3.2.1 Testing linearity.

Example 3.2.1: Test the linearity of the discrete-time systems defined in Examples 3.1.8 and 3.1.9, that is, defined by $y(n) = 2x(n)+3$ and $y(n) = x^2(n)$.

Solution: The I/O equation $y(n) = 2x(n)+3$ is a linear equation, but it does not represent a linear *system*. Indeed, the output due to the linear combination of Eq. (3.2.1) will be

$$y(n) = 2x(n)+3 = 2[a_1 x_1(n) + a_2 x_2(n)] + 3$$

and is not equal to the linear combination in the right-hand side of Eq. (3.2.2), namely,

$$a_1 y_1(n) + a_2 y_2(n) = a_1(2x_1(n)+3) + a_2(2x_2(n)+3)$$

Similarly, for the quadratic system $y(n)=x^2(n)$ of Example 3.1.9, we have

$$a_1x_1^2(n)+a_2x_2^2(n)\neq\big(a_1x_1(n)+a_2x_2(n)\big)^2$$

More simply, if a system is nonlinear, one can use a counterexample to show violation of linearity. For example, if the above quadratic system were linear, doubling of the input would cause doubling of the output. But in this case, doubling of the input quadruples the output. □

A *time-invariant* system is a system that remains unchanged over time. This implies that if an input is applied to the system today causing a certain output to be produced, then the same output will also be produced tomorrow if the same input is applied. The operation of waiting or delaying a signal by a time delay of, say, D units of time is shown in Fig. 3.2.2. It represents the *right translation* of $x(n)$ as a whole by D samples. A *time advance* would have negative D and correspond to the left translation of $x(n)$.

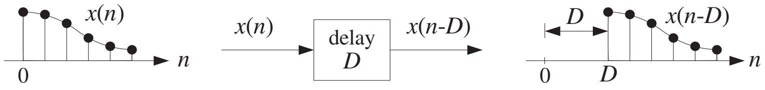


Fig. 3.2.2 Time delay by D samples.

The mathematical formulation of time invariance can be stated with the aid of Fig. 3.2.3. The upper diagram in this figure shows an input $x(n)$ being applied to the system producing the output $y(n)$. The lower diagram shows the same input delayed by D units of time, that is, the signal:

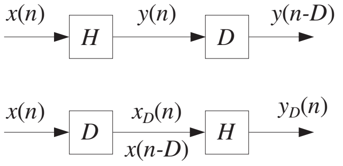


Fig. 3.2.3 Testing time invariance.

$$x_D(n)=x(n-D) \tag{3.2.3}$$

which is then applied to the system producing an output, say, $y_D(n)$.

To test whether the system will produce the same output later as the one it is producing now, we must take the output $y(n)$ produced now and save it until later, that is, delay it by D time units, as shown in the upper diagram of Fig. 3.2.3. Then, it can be compared with the output $y_D(n)$ that will be produced later. Thus, if

$$y_D(n)=y(n-D) \tag{3.2.4}$$

the system will be time-invariant. In other words, delaying the input, Eq. (3.2.3), causes the output to be delayed by the same amount, Eq. (3.2.4). Equivalently, in terms of the samples, if

$$\{x_0,x_1,x_2,\dots\}\xrightarrow{H}\{y_0,y_1,y_2,\dots\}$$

then

$$\{\underbrace{0,0,\dots,0}_{D\text{ zeros}},x_0,x_1,x_2,\dots\}\xrightarrow{H}\{\underbrace{0,0,\dots,0}_{D\text{ zeros}},y_0,y_1,y_2,\dots\}$$

Example 3.2.2: Test the time invariance of the discrete-time systems defined in Examples (3.1.12) and (3.1.15), that is, defined by $y(n)=nx(n)$ and $y(n)=x(2n)$.

Solution: Because the system $y(n)=nx(n)$ has a time-varying coefficient, we expect it not to be time-invariant. According to the given I/O rule, the signal $x_D(n)$ applied to the system will cause the output $y_D(n)=nx_D(n)$. But $x_D(n)$ is the delayed version of $x(n)$, $x_D(n)=x(n-D)$. Therefore,

$$y_D(n)=nx_D(n)=nx(n-D)$$

On the other hand, delaying the output signal $y(n)=nx(n)$ by D units gives, replacing n by $n-D$:

$$y(n-D)=(n-D)x(n-D)\neq nx(n-D)=y_D(n)$$

Thus, the system is not time-invariant. Example 3.1.15 described by $y(n)=x(2n)$ is a little more subtle. According to the I/O rule, the signal $x_D(n)$ will cause the output $y_D(n)=x_D(2n)$. But, $x_D(n)=x(n-D)$ and therefore, replacing the argument n by $2n$ gives $x_D(2n)=x(2n-D)$, or,

$$y_D(n)=x_D(2n)=x(2n-D)$$

On the other hand, replacing n by $n-D$ in $y(n)=x(2n)$ gives

$$y(n-D)=x\big(2(n-D)\big)=x(2n-2D)\neq x(2n-D)=y_D(n)$$

Thus, the downsampler is not time-invariant. This can also be seen more intuitively by considering the effect of the system on the original input sequence and its delay by one time unit. Noting that the output sequence is obtained by dropping every other input sample, we find:

$$\begin{aligned}\{x_0,x_1,x_2,x_3,x_4,x_5,x_6,\dots\}&\xrightarrow{H}\{x_0,x_2,x_4,x_6,\dots\}\\\{0,x_0,x_1,x_2,x_3,x_4,x_5,x_6,\dots\}&\xrightarrow{H}\{0,x_1,x_3,x_5,\dots\}\end{aligned}$$

We see that the lower output is not the upper output delayed by one time unit. □

3.3 Impulse Response

Linear time-invariant systems are characterized uniquely by their *impulse response* sequence $h(n)$, which is defined as the response of the system to a *unit impulse* $\delta(n)$, as shown in Fig. 3.3.1. The unit impulse is the discrete-time analog of the Dirac delta function $\delta(t)$ and is defined as

$$\delta(n)=\begin{cases}1 & \text{if } n=0\\ 0 & \text{if } n\neq 0\end{cases}$$

Thus, we have by definition,

$$\delta(n)\xrightarrow{H}h(n)$$

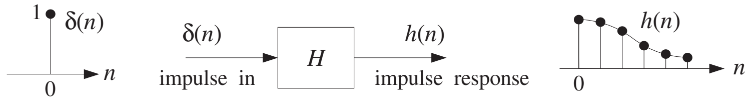


Fig. 3.3.1 Impulse response of an LTI system.

or, in terms of the sample values:[†]

$$\{1,0,0,0,\dots\}\xrightarrow{H}\{h_0,h_1,h_2,h_3,\dots\}$$

Time invariance implies that if the unit impulse is delayed or time shifted by a certain amount, say D units of time, then it will cause the impulse response to be delayed by the same amount, that is, $h(n-D)$. Thus,

$$\delta(n-D)\xrightarrow{H}h(n-D)$$

for any positive or negative delay D. Figure Fig. 3.3.2 shows this property for $D=0,1,2$. On the other hand, linearity implies that any linear combination of inputs causes the same linear combination of outputs, so that, for example, the sum of the three impulses of Fig. 3.3.2 will cause the sum of the three outputs, that is,

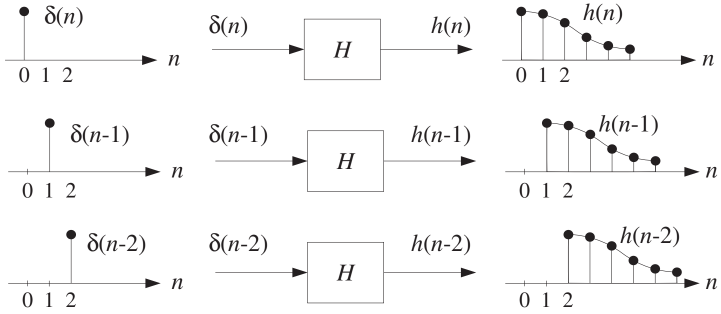


Fig. 3.3.2 Delayed impulse responses of an LTI system.

$$\delta(n)+\delta(n-1)+\delta(n-2)\xrightarrow{H}h(n)+h(n-1)+h(n-2)$$

or, more generally the weighted linear combination of the three inputs:

$$x(0)\delta(n)+x(1)\delta(n-1)+x(2)\delta(n-2)$$

will cause the same weighted combination of the three outputs:

$$x(0)h(n)+x(1)h(n-1)+x(2)h(n-2)$$

[†] Again, we denote $\{h(0),h(1),h(2),\dots\}$ by $\{h_0,h_1,h_2,\dots\}$.

as shown in Fig. 3.3.3. In general, an arbitrary input sequence $\{x(0),x(1),x(2),\dots\}$ can be thought of as the linear combination of shifted and weighted unit impulses:

$$x(n)=x(0)\delta(n)+x(1)\delta(n-1)+x(2)\delta(n-2)+x(3)\delta(n-3)+\cdots$$

This follows because each term of the right-hand side is nonzero only at the corresponding delay time, for example, at $n=0$ only the first term is nonzero, at $n=1$ only the second term is nonzero, and so on. Linearity and time invariance imply then that the corresponding output sequence will be obtained by replacing each delayed unit impulse by the corresponding delayed impulse response, that is,

$$y(n)=x(0)h(n)+x(1)h(n-1)+x(2)h(n-2)+x(3)h(n-3)+\cdots \tag{3.3.1}$$

or written more compactly:

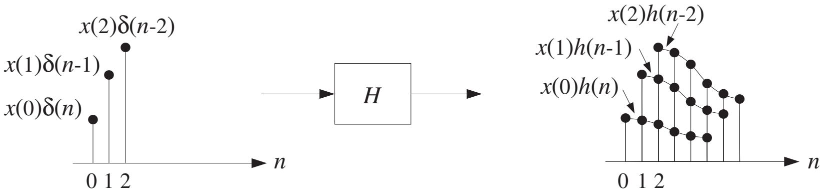


Fig. 3.3.3 Response to linear combination of inputs.

$$\boxed{y(n)=\sum_m x(m)h(n-m)}\qquad\text{(LTI form)}\tag{3.3.2}$$

This is the discrete-time *convolution* of the input sequence $x(n)$ with the filter sequence $h(n)$. Thus, LTI systems are convolvers.

In general, the summation could extend also over negative values of m, depending on the input signal. Because it was derived using the LTI properties of the system, Eq. (3.3.2) will be referred to as the *LTI form* of convolution. Changing the index of summation, it can also be written in the alternative form:

$$\boxed{y(n)=\sum_m h(m)x(n-m)}\qquad\text{(direct form)}\tag{3.3.3}$$

For reasons that will become clear later, Eq. (3.3.3) will be referred to as the *direct form* of convolution. The computational aspects of Eqs. (3.3.2) and (3.3.3) and their realization in terms of block or sample processing methods will be discussed in detail in the next chapter.

3.4 FIR and IIR Filters

Discrete-time LTI systems can be classified into FIR or IIR systems, that is, having finite or infinite impulse response $h(n)$, as depicted in Fig. 3.4.1.

An FIR filter has impulse response $h(n)$ that extends only over a finite time interval, say $0 \le n \le M$, and is identically zero beyond that:

$$\{h_0, h_1, h_2, \dots, h_M, 0, 0, 0, \dots\}$$

M is referred to as the *filter order*. The *length* of the impulse response vector $\mathbf{h} = [h_0, h_1, \dots, h_M]$ is:

$$\boxed{L_h = M + 1}$$

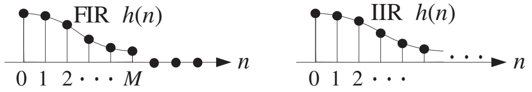


Fig. 3.4.1 FIR and IIR impulse responses.

The impulse response coefficients $\{h_0, h_1, \dots, h_M\}$ are referred to by various names, such as *filter coefficients, filter weights*, or *filter taps*, depending on the context. In the direct form of convolution of Eq. (3.3.3), all the terms for $m > M$ and $m < 0$ will be absent because by definition $h(m)$ vanishes for these values of m; only the terms $0 \le m \le M$ are present. Therefore, Eq. (3.3.3) is simplified to the finite-sum form:

$$\boxed{y(n) = \sum_{m=0}^{M} h(m)x(n-m)} \qquad \text{(FIR filtering equation)} \tag{3.4.1}$$

or, explicitly

$$y(n) = h_0 x(n) + h_1 x(n-1) + h_2 x(n-2) + \cdots + h_M x(n-M) \tag{3.4.2}$$

Thus, the I/O equation is obtained as a weighted sum of the *present* input sample $x(n)$ and the *past* M samples $x(n-1), x(n-2), \dots, x(n-M)$.

Example 3.4.1: Second-order FIR filters are characterized by three impulse response coefficients $\mathbf{h} = [h_0, h_1, h_2]$ and have I/O equation:

$$y(n) = h_0 x(n) + h_1 x(n-1) + h_2 x(n-2)$$

Such was the case of Example 3.1.2, which had $\mathbf{h} = [2, 3, 4]$.

Example 3.4.2: Similarly, third-order FIR filters are characterized by four weights $\mathbf{h} = [h_0, h_1, h_2, h_3]$ and have I/O equation:

$$y(n) = h_0 x(n) + h_1 x(n-1) + h_2 x(n-2) + h_3 x(n-3)$$

Example 3.4.3: Determine the impulse response $\mathbf{h}$ of the following FIR filters:

(a) $y(n) = 2x(n) + 3x(n-1) + 5x(n-2) + 2x(n-3)$

(b) $y(n) = x(n) - x(n-4)$

Solution: Comparing the given I/O equations with Eq. (3.4.2), we identify the impulse response coefficients:

(a) $\mathbf{h} = [h_0, h_1, h_2, h_3] = [2, 3, 5, 2]$

(b) $\mathbf{h} = [h_0, h_1, h_2, h_3, h_4] = [1, 0, 0, 0, -1]$

Alternatively, sending a unit impulse as input, $x(n) = \delta(n)$, will produce the impulse response sequence as output, $y(n) = h(n)$:

(a) $h(n) = 2\delta(n) + 3\delta(n-1) + 5\delta(n-2) + 2\delta(n-3)$

(b) $h(n) = \delta(n) - \delta(n-4)$

The expressions for $h(n)$ and $\mathbf{h}$ are equivalent. □

An IIR filter, on the other hand, has an impulse response $h(n)$ of infinite duration, defined over the infinite interval $0 \le n < \infty$. Eq. (3.3.3) now has an infinite number of terms:

$$\boxed{y(n) = \sum_{m=0}^{\infty} h(m)x(n-m)} \qquad \text{(IIR filtering equation)} \tag{3.4.3}$$

This I/O equation is not *computationally* feasible because we cannot deal with an infinite number of terms. Therefore, we must restrict our attention to a subclass of IIR filters, namely, those for which the infinite number of filter coefficients $\{h_0, h_1, h_2, \dots\}$ are not chosen arbitrarily, but rather they are coupled to each other through *constant-coefficient linear difference equations*.

For this subclass of IIR filters, Eq. (3.4.3) can be *rearranged* as a *difference equation* allowing the efficient *recursive* computation of the output $y(n)$. Some examples will make this point clear.

Example 3.4.4: Determine the I/O difference equation of an IIR filter whose impulse response coefficients $h(n)$ are coupled to each other by the difference equation:

$$h(n) = h(n-1) + \delta(n)$$

Solution: Setting $n = 0$, we have $h(0) = h(-1) + \delta(0) = h(-1) + 1$. Assuming causal initial conditions, $h(-1) = 0$, we find $h(0) = 1$. For $n > 0$, the delta function vanishes, $\delta(n) = 0$, and therefore, the difference equation reads $h(n) = h(n-1)$. In particular $h(1) = h(0) = 1$, $h(2) = h(1) = 1$, and so on. Thus, all of the samples $h(n)$ are equal to each other. In summary, we have the (causal) solution:

$$h(n) = u(n) = \begin{cases} 1 & \text{if } n \ge 0 \\ 0 & \text{if } n \le -1 \end{cases}$$

where $u(n)$ is the discrete-time unit-step function. Now, putting this solution into the convolutional I/O equation (3.4.3), we have

$$y(n) = \sum_{m=0}^{\infty} h(m)x(n-m) = \sum_{m=0}^{\infty} x(n-m)$$

where we set $h(m)= 1$. Writing it explicitly we have

$$y(n)= x(n)+x(n-1)+x(n-2)+x(n-3)+\cdots$$

Replacing n by $n-1$ gives the previous output

$$y(n-1)= x(n-1)+x(n-2)+x(n-3)+\cdots$$

Subtracting it from $y(n)$, we have

$$y(n)-y(n-1)= x(n)$$

Therefore, the I/O convolutional equation is equivalent to the recursive difference equation

$$y(n)= y(n-1)+x(n)$$

It represents an *accumulator*, or discrete-time integrator. Note that this is the *same* difference equation as that of $h(n)$, because by definition $h(n)$ is the output when the input is an impulse; that is, $y(n)= h(n)$ if $x(n)= \delta(n)$. □

Example 3.4.5: Suppose the filter coefficients $h(n)$ satisfy the difference equation

$$h(n)= ah(n-1)+\delta(n)$$

where a is a constant. Determine the I/O difference equation relating a general input signal $x(n)$ to the corresponding output $y(n)$.

Solution: Arguing as in the previous example, we have

$$\begin{aligned}
h(0) &= ah(-1)+\delta(0)= a\cdot 0+1 = 1\\
h(1) &= ah(0)+\delta(1)= a\cdot 1+0 = a\\
h(2) &= ah(1)+\delta(2)= a\cdot a+0 = a^2\\
h(3) &= ah(2)+\delta(3)= a\cdot a^2+0 = a^3
\end{aligned}$$

and so on. Therefore, we find the solution

$$h(n)= a^n u(n)= \begin{cases} a^n, & \text{if } n\geq 0\\ 0, & \text{if } n\leq -1\end{cases}$$

Inserting this solution into Eq. (3.4.3), we have

$$\begin{aligned}
y(n) &= x(n)+ax(n-1)+a^2x(n-2)+a^3x(n-3)+\cdots\\
&= x(n)+a\big[x(n-1)+ax(n-2)+a^2x(n-3)+\cdots\big]
\end{aligned}$$

The sum in the brackets is recognized now as the previous output $y(n-1)$. Therefore, we obtain the I/O difference equation:

$$y(n)= ay(n-1)+x(n)$$

As expected, it is the same as the difference equation satisfied by $h(n)$. □

Example 3.4.6: Determine the convolutional form and the (causal) impulse response of the IIR filter described by the following difference equation:

$$y(n)= -0.8y(n-1)+x(n)$$

Solution: This is the same example as above, with $a = -0.8$. Setting $x(n)= \delta(n)$ and $y(n)= h(n)$, we obtain the difference equation for $h(n)$:

$$h(n)= -0.8h(n-1)+\delta(n)$$

Assuming causal initial conditions, $h(-1)= 0$, and iterating a few values of n as we did in the previous example, we find the solution:

$$h(n)= (-0.8)^n u(n)= \begin{cases} (-0.8)^n, & \text{if } n\geq 0\\ 0, & \text{if } n\leq -1\end{cases}$$

Inserting the values for $h(n)$ into the convolutional equation (3.4.3), we find

$$y(n)= x(n)+(-0.8)x(n-1)+(-0.8)^2x(n-2)+(-0.8)^3x(n-3)+\cdots$$

which, in general, has an infinite number of terms. □

Example 3.4.7: In this example, start with an expression for $h(n)$ and work backwards to obtain the I/O difference equation satisfied by $y(n)$ in terms of $x(n)$, and also determine the difference equation satisfied by $h(n)$. Assume the IIR filter has a causal $h(n)$ defined by

$$h(n)= \begin{cases} 2, & \text{for } n=0\\ 4(0.5)^{n-1}, & \text{for } n\geq 1\end{cases}$$

Solution: The first two values $h(0)$ and $h(1)$ are chosen arbitrarily, but for $n\geq 2$ the values are recursively related to one another; for example, starting with $h(1)= 4$, we have $h(2)= 0.5h(1)$, $h(3)= 0.5h(2)$, $h(4)= 0.5h(3)$, and so on. Therefore, we expect that these recursions can be used to reassemble the I/O convolutional equation into a difference equation for $y(n)$.

Inserting the numerical values of $h(n)$ into Eq. (3.4.3), we find for the I/O equation

$$\begin{aligned}
y_n &= h_0x_n + h_1x_{n-1} + h_2x_{n-2} + h_3x_{n-3} + h_4x_{n-4} + \cdots\\
&= 2x_n + 4x_{n-1} + 2\big[x_{n-2} + 0.5x_{n-3} + 0.5^2x_{n-4} + \cdots\big]
\end{aligned}$$

and for the previous output

$$y_{n-1} = 2x_{n-1} + 4x_{n-2} + 2\big[x_{n-3} + 0.5x_{n-4} + \cdots\big]$$

Multiplying by 0.5, we have

$$0.5y_{n-1} = x_{n-1} + 2\big[x_{n-2} + 0.5x_{n-3} + 0.5^2x_{n-4} + \cdots\big]$$

Subtracting it from y_n, we find the I/O difference equation

$$y_n - 0.5y_{n-1} = 2x_n + 3x_{n-1}$$

and solving for $y(n)$

$$y(n)= 0.5y(n-1)+2x(n)+3x(n-1)$$

which is recognized as the difference equation of Example 3.1.5. Setting $x(n)=\delta(n)$ and $y(n)=h(n)$ gives the difference equation for $h(n)$:

$$h(n)= 0.5h(n-1)+2\delta(n)+3\delta(n-1)$$

Starting with the initial value $h(-1)=0$ and iterating for a few values of n, one can easily verify that this difference equation generates the sequence $h(n)$ we started out with. □

Example 3.4.8: Determine the convolutional form and the (causal) impulse response of the IIR filter described by the following difference equation:

$$y(n)= 0.25y(n-2)+x(n)$$

Solution: The impulse response $h(n)$ will satisfy the difference equation:

$$h(n)= 0.25h(n-2)+\delta(n)$$

to be iterated with zero initial conditions: $h(-1)=h(-2)=0$. A few iterations give:

$$\begin{aligned}
h(0) &= 0.25h(-2)+\delta(0)= 0.25\cdot 0+1 = 1\\
h(1) &= 0.25h(-1)+\delta(1)= 0.25\cdot 0+0 = 0\\
h(2) &= 0.25h(0)+\delta(2)= 0.25\cdot 1+0 = 0.25 = (0.5)^2\\
h(3) &= 0.25h(1)+\delta(3)= 0.25\cdot 0+0 = 0\\
h(4) &= 0.25h(2)+\delta(4)= 0.25\cdot 0.25+0 = (0.25)^2= (0.5)^4
\end{aligned}$$

And, in general, for $n\ge 0$

$$h(n)=\begin{cases}(0.5)^n, & \text{if } n = \text{even}\\ 0, & \text{if } n = \text{odd}\end{cases}$$

Equivalently, we can write:

$$\mathbf{h} = [1,\ 0,\ (0.5)^2,\ 0,\ (0.5)^4,\ 0,\ (0.5)^6,\ 0,\ (0.5)^8,\ 0,\ \dots\ ]$$

And, Eq. (3.4.3) becomes:

$$y_n = x_n + 0.5^2x_{n-2} + 0.5^4x_{n-4} + 0.5^6x_{n-6} + \cdots$$

which is the solution of the given difference equation in terms of $x(n)$. □

Example 3.4.9: Determine the I/O difference equation of the IIR filter that has the following causal periodic impulse response:

$$h(n)= \{2,3,4,5,2,3,4,5,2,3,4,5,\dots\}$$

where the dots denote the periodic repetition of the four samples $\{2,3,4,5\}$.

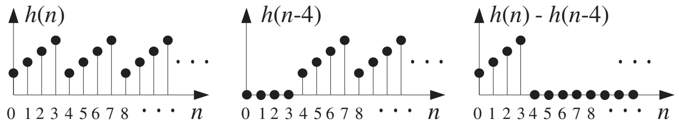
Solution: If we delay the given response by one period, that is, 4 samples, we get

$$h(n-4)= \{0,0,0,0,2,3,4,5,2,3,4,5,\dots\}$$

Subtracting it from $h(n)$, we get

$$h(n)-h(n-4)= \{2,3,4,5,0,0,0,0,0,0,0,0,\dots\}$$

with all samples beyond $n=4$ canceling to zero. These operations are depicted below.



Thus, the right-hand side is nonzero only for $n = 0,1,2,3$, and we can rewrite it as the difference equation

$$h(n)-h(n-4)= 2\delta(n)+3\delta(n-1)+4\delta(n-2)+5\delta(n-3)$$

or, solving for $h(n)$

$$h(n)= h(n-4)+2\delta(n)+3\delta(n-1)+4\delta(n-2)+5\delta(n-3)$$

Using the method of the previous example, we can show that $y(n)$ satisfies the same difference equation:

$$y_n = y_{n-4} + 2x_n + 3x_{n-1} + 4x_{n-2} + 5x_{n-3}$$

This example shows how to construct a *digital periodic waveform generator*: Think of the waveform to be generated as the impulse response of an LTI system, determine the difference equation for that system, and then hit it with an impulse, and it will generate its impulse response, that is, the desired waveform. See Section 16.1.2. □

More generally, the IIR filters that we will be concerned with have impulse responses $h(n)$ that satisfy constant-coefficient difference equations of the general type:

$$h(n)=\sum_{i=1}^{M}a_ih(n-i)+\sum_{i=0}^{L}b_i\delta(n-i)$$

or, written explicitly

$$h_n = a_1h_{n-1} + a_2h_{n-2} + \cdots + a_Mh_{n-M} + b_0\delta_n + b_1\delta_{n-1} + \cdots + b_L\delta_{n-L}$$

Using the methods of Example 3.4.7, it can be shown that the corresponding convolutional equation (3.4.3) can be reassembled into the same difference equation for $y(n)$ in terms of $x(n)$, that is,

$$y(n)=\sum_{i=1}^{M}a_iy(n-i)+\sum_{i=0}^{L}b_ix(n-i)$$

or, explicitly

$$y_n = a_1 y_{n-1} + a_2 y_{n-2} + \cdots + a_M y_{n-M} + b_0 x_n + b_1 x_{n-1} + \cdots + b_L x_{n-L}$$

We will explore the properties of IIR filters after we discuss z-transforms. Note also that FIR filters can be thought of as special cases of the IIR difference equations when the recursive terms are absent, that is, when the recursive coefficients are zero, $a_1 = a_2 = \cdots = a_M = 0$.

Eventually, we aim to develop several mathematically *equivalent* descriptions of FIR and IIR filters, such as,

- I/O difference equation
- Convolutional equation
- Impulse response $h(n)$
- Transfer function $H(z)$
- Frequency response $H(\omega)$
- Pole/zero pattern
- Block diagram realization and sample processing algorithm

The above examples show how to go back and forth between the first three of these descriptions—from the *difference equation* to the corresponding *impulse response* to the *convolutional form* of the filtering equation. We will see later that all of the tedious *time-domain* manipulations of the above examples can be avoided by working with z-transforms.

Each description serves a different *purpose* and provides a different *insight* into the properties of the filter. For example, in a typical application, we would provide desired frequency-domain specifications for the filter, that is, specify the desired shape of $H(\omega)$. Using a filter design technique, we would design a filter whose frequency response closely approximates the desired response. The output of the filter design technique is typically the transfer function $H(z)$ for IIR filters or the impulse response $h(n)$ for FIR filters. From $H(z)$ or $h(n)$, we would obtain an appropriate block diagram realization that can be used to implement the filter in real time.

3.5 Causality and Stability

Like analog signals, discrete-time signals can be classified into causal, anticausal, or mixed signals, as shown in Fig. 3.5.1.

A *causal* or right-sided signal $x(n)$ exists only for $n \geq 0$ and vanishes for all negative times $n \leq -1$. Causal signals are the most commonly encountered signals, because they are the type that we generate in our labs — for example, when we turn on a signal generator or signal source.

An *anticausal* or left-sided signal exists only for $n \leq -1$ and vanishes for all $n \geq 0$. A *mixed* or double-sided signal has both a left-sided and a right-sided part.

The placement of the time origin, $n = 0$, along the time axis is entirely a matter of convention. Typically, it is taken to be the time when we turn on our signal generators or the time when we begin our processing operations. Therefore, a signal that is double-sided with respect to a chosen time origin is simply a signal that *has already been in existence* when we start our processing.

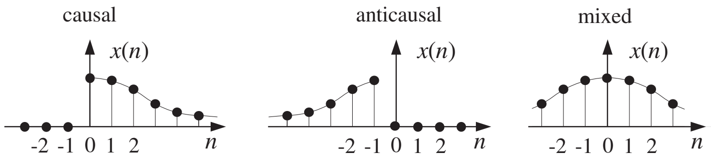


Fig. 3.5.1 Causal, anticausal, and mixed signals.

LTI systems can also be classified in terms of their causality properties depending on whether their impulse response $h(n)$ is causal, anticausal, or mixed. For a general double-sided $h(n)$, which can extend over $-\infty < n < \infty$, the I/O convolutional equation becomes

$$y(n) = \sum_{m=-\infty}^{\infty} h(m)x(n-m) \tag{3.5.1}$$

Such systems cannot be implemented in real time, as can be seen by writing a few of the positive and negative m terms:

$$y_n = \cdots + h_{-2}x_{n+2} + h_{-1}x_{n+1} + h_0 x_n + h_1 x_{n-1} + h_2 x_{n-2} + \cdots$$

which shows that to compute the output $y(n)$ at the current time n, one needs to know the *future* input samples $x(n+1), x(n+2), \dots$, which are not yet available for processing.

Anticausal and double-sided systems are very counter-intuitive, violating our sense of causality. For example, in response to a unit impulse $\delta(n)$, which is applied to the system at $n = 0$, the system will generate its impulse response output $h(n)$. But if $h(-1) \neq 0$, this means that the system had already produced an output sample at time $n = -1$, even before the input impulse was applied at $n = 0$!

Should we, therefore, be concerned with non-causal filters? Are they relevant, useful, or necessary in DSP? The answer to all of these questions is yes. Some typical applications where double-sided filters crop up are the design of FIR *smoothing* filters, the design of FIR *interpolation* filters used in oversampling DSP systems, and the design of *inverse* filters.

FIR smoothing and interpolation filters belong to a class of double-sided filters that are only finitely anticausal, that is, their anticausal part has *finite duration*, say over the period $-D \leq n \leq -1$. Such filters are shown in Fig. 3.5.2. In general, the causal part of $h(n)$ may be finite or infinite. The I/O equation (3.5.1) becomes for this class of filters:

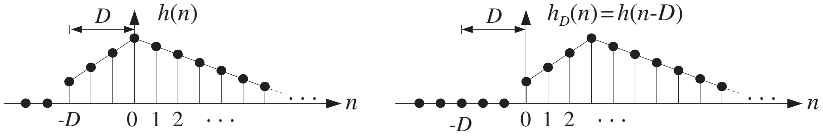


Fig. 3.5.2 Finitely anticausal filter and its causal version.

$$y(n)=\sum_{m=-D}^{\infty} h(m)x(n-m) \tag{3.5.2}$$

A standard technique for dealing with such filters is to make them causal by replacing $h(n)$ with its *delayed* version by D time units, that is,

$$h_D(n)=h(n-D)$$

As shown in Fig. 3.5.2, this operation translates $h(n)$ to the right by D units, making it causal. The I/O filtering equation for the causal filter $h_D(n)$ will be

$$y_D(n)=\sum_{m=0}^{\infty} h_D(m)x(n-m) \tag{3.5.3}$$

and will be implementable in real time. It is easily seen that the resulting sequence $y_D(n)$ is simply the delayed version of $y(n)$ as computed by Eq. (3.5.2):

$$y_D(n)=y(n-D)$$

Thus, the output samples are computed correctly, but come out with a time delay.

Example 3.5.1: Consider the typical 5-tap smoothing filter of Example 3.1.7 having filter coefficients $h(n)=1/5$ for $-2\le n\le 2$. The corresponding I/O convolutional equation (3.5.2) becomes

$$\begin{aligned} y(n) &= \sum_{m=-2}^{2} h(m)x(n-m)=\frac{1}{5}\sum_{m=-2}^{2} x(n-m) \\ &= \frac{1}{5}\big[x(n+2)+x(n+1)+x(n)+x(n-1)+x(n-2)\big] \end{aligned}$$

It is called a smoother or averager because at each n it replaces the current sample $x(n)$ by its average with the two samples ahead and two samples behind it, and therefore, it tends to average out rapid fluctuations from sample to sample.

Its anticausal part has duration $D=2$ and can be made causal with a time delay of two units, resulting in

$$y_2(n)=y(n-2)=\frac{1}{5}\big[x(n)+x(n-1)+x(n-2)+x(n-3)+x(n-4)\big]$$

This filtering equation must be thought of as smoothing the middle sample $x(n-2)$ and not the current sample $x(n)$. □

When real-time processing is not an issue, as in block processing methods, and the input data to be processed have already been collected and saved as a block of samples on some medium such as memory or tape, one may use the non-causal form of Eq. (3.5.2) directly. This is one of the advantages of DSP that does not have a parallel in analog signal processing. An example of this may be the processing of a still picture, where all the pixel information has been gathered into a block of samples.

In addition to their causality properties, LTI systems can be classified in terms of their stability properties. A *stable* LTI system is one whose impulse response $h(n)$

goes to zero sufficiently fast as $n\to\pm\infty$, so that the output of the system $y(n)$ never diverges, that is, it remains *bounded* by some bound $|y(n)|\le B$ if its input is *bounded*, say $|x(n)|\le A$. That is, a system is stable if bounded inputs always generate bounded outputs.

It can be shown that a *necessary and sufficient* condition for an LTI system to be stable in the above bounded-input/bounded-output sense is that its impulse response $h(n)$ be *absolutely summable*:

$$\boxed{\sum_{n=-\infty}^{\infty}|h(n)|<\infty} \qquad \text{(stability condition)} \tag{3.5.4}$$

Example 3.5.2: Consider the following four examples of $h(n)$:

$$\begin{aligned} h(n) &= (0.5)^n u(n) && \text{(stable and causal)} \\ h(n) &= -(0.5)^n u(-n-1) && \text{(unstable and anticausal)} \\ h(n) &= 2^n u(n) && \text{(unstable and causal)} \\ h(n) &= -2^n u(-n-1) && \text{(stable and anticausal)} \end{aligned}$$

In the two causal cases, the presence of the unit step $u(n)$ causes $h(n)$ to be nonzero only for $n\ge 0$, whereas in the anticausal cases, the presence of $u(-n-1)$ makes $h(n)$ nonzero only for $-n-1\ge 0$ or $n+1\le 0$ or $n\le -1$. The first example tends to zero exponentially for $n\to\infty$; the second diverges as $n\to-\infty$; indeed, because n is negative, one can write $n=-|n|$ and

$$h(n)=-(0.5)^n u(-n-1)=-(0.5)^{-|n|}u(-n-1)=-2^{|n|}u(-n-1)$$

and therefore it blows up exponentially for large negative n. The third example blows up for $n\to\infty$ and the fourth tends to zero exponentially for $n\to-\infty$, as can be seen from

$$h(n)=-2^n u(-n-1)=-2^{-|n|}u(-n-1)=-(0.5)^{|n|}u(-n-1)$$

Thus, cases one and four are stable and cases two and three unstable. The same conclusion can also be reached by the stability criterion of Eq. (3.5.4). We have in the four cases:

$$\begin{aligned} \sum_{n=-\infty}^{\infty}|h(n)| &= \sum_{n=0}^{\infty}(0.5)^n=\frac{1}{1-0.5}<\infty \\ \sum_{n=-\infty}^{\infty}|h(n)| &= \sum_{n=-1}^{-\infty}(0.5)^n=\sum_{m=1}^{\infty}2^m=\infty \\ \sum_{n=-\infty}^{\infty}|h(n)| &= \sum_{n=0}^{\infty}2^n=\infty \\ \sum_{n=-\infty}^{\infty}|h(n)| &= \sum_{n=-1}^{-\infty}2^n=\sum_{m=1}^{\infty}(0.5)^m=\frac{0.5}{1-0.5}<\infty \end{aligned}$$

where in the first and fourth cases, we used the infinite geometric series formulas:

$$\sum_{m=0}^{\infty}x^m=\frac{1}{1-x} \qquad \text{and} \qquad \sum_{m=1}^{\infty}x^m=\frac{x}{1-x}$$

valid for $|x| < 1$. For the second and third cases the geometric series have $x > 1$ and diverge. We will see later that cases one and two have the *same* transfer function, namely, $H(z)= \dfrac{1}{1-0.5z^{-1}}$, and therefore, cannot be distinguished on the basis of just the transfer function. Similarly, cases three and four have the common transfer function $H(z)= \dfrac{1}{1-2z^{-1}}$. □

Stability is absolutely essential in hardware or software implementations of LTI systems because it guarantees that the *numerical* operations required for computing the I/O convolution sums or the equivalent difference equations remain well behaved and never grow beyond bounds. In hardware implementations, such instabilities would quickly saturate the hardware registers and in software implementations they would exceed the numerical ranges of most computers resulting in numerical nonsense.

The concepts of stability and causality are logically independent, but are *not always compatible* with each other, that is, it may not be possible to satisfy simultaneously the conditions of stability and causality, as was the case of the last two systems of Example 3.5.2. However, because of the practical numerical considerations mentioned above, we must *always* prefer stability over causality.

If the anticausal part of a stable system $h(n)$ has finite duration, then it can be handled as above, making it causal by a time delay. If, on the other hand, the anticausal part is infinite, then $h(n)$ can only be handled *approximately* by the following procedure. Because $h(n)$ is stable, it will tend to zero for large negative n. Therefore, one may pick a *sufficiently large* negative integer $n = -D$ and *clip* the left tail of $h(n)$ for $n < -D$. That is, one can replace the true $h(n)$ by its clipped approximation:

$$\tilde{h}(n)= \begin{cases} h(n), & \text{for } n \geq -D \\ 0, & \text{for } n < -D \end{cases} \tag{3.5.5}$$

This clipped response will be of the finitely anticausal type shown in Fig. 3.5.2, and therefore, it can be made causal by a delay of D units of time, that is, $\tilde{h}_D(n)= \tilde{h}(n-D)$. The *approximation error* can be made as small as desired by increasing the value of D. To see this, let $\tilde{y}(n)$ be the output of the approximate system $\tilde{h}(n)$ for a bounded input, $|x(n)| \leq A$, and let $y(n)$ be the output of the exact system $h(n)$. It is easily shown that the error in the output is bounded from above by

$$|y(n)-\tilde{y}(n)| \leq A \sum_{m=-\infty}^{-D-1} |h(m)| \tag{3.5.6}$$

for all n. The above sum, being a partial sum of Eq. (3.5.4), is finite and tends to zero as D increases. For example, in case four of Example 3.5.2 we find

$$\sum_{m=-\infty}^{-D-1} |h(m)| = \sum_{m=D+1}^{\infty} (0.5)^m = (0.5)^{D+1}\frac{1}{1-0.5} = (0.5)^D$$

which can be made as small as desired by increasing D.

This type of stable but non-causal filter can often arise in the design of *inverse* filters. The inverse of a filter with transfer function $H(z)$ has transfer function

$$H_{\text{inv}}(z)= \frac{1}{H(z)}$$

Such inverse filters are used in various equalization applications, such as *channel equalization* for digital data transmission where $H(z)$ may represent the channel's transfer function, or the equalizer filters that we discussed in Chapter 1.

The corresponding impulse response of the inverse filter $h_{\text{inv}}(n)$ must be chosen to be stable. But, then it may be not be causal.[†] Therefore, in this case we must work with the approximate *clipped/delayed* inverse filter response

$$\tilde{h}_{\text{inv},D}(n)= \tilde{h}_{\text{inv}}(n-D)$$

We will consider examples of such designs later, after we discuss z-transforms.

3.6 Problems

3.1 Determine whether the discrete-time systems described by the following I/O equations are linear and/or time-invariant:

a. $y(n)= 3x(n)+5$

b. $y(n)= x^2(n-1)+x(2n)$,

c. $y(n)= e^{x(n)}$

d. $y(n)= nx(n-3)+3x(n)$.

e. $y(n)= n+3x(n)$

3.2 Determine the causal impulse response $h(n)$ for $n \geq 0$ of the LTI systems described by the following I/O difference equations:

a. $y(n)= 3x(n)-2x(n-1)+4x(n-3)$

b. $y(n)= 4x(n)+x(n-1)-3x(n-3)$

c. $y(n)= x(n)-x(n-3)$

3.3 Determine the causal impulse response $h(n)$ for $n \geq 0$ of the LTI systems described by the following I/O difference equations:

a. $y(n)= -0.9y(n-1)+x(n)$

b. $y(n)= 0.9y(n-1)+x(n)$

c. $y(n)= 0.64y(n-2)+x(n)$

d. $y(n)= -0.81y(n-2)+x(n)$

e. $y(n)= 0.5y(n-1)+4x(n)+x(n-1)$

3.4 Determine the I/O difference equations relating $x(n)$ and $y(n)$ for the LTI systems having the following impulse responses:

a. $h(n)= (0.9)^n u(n)$

b. $h(n)= (-0.6)^n u(n)$

c. $h(n)= (0.9)^n u(n)+(-0.9)^n u(n)$

d. $h(n)= (0.9j)^n u(n)+(-0.9j)^n u(n)$

[†] We will see later that this circumstance can arise if some of the *zeros* of the transfer function $H(z)$ lie outside the unit circle in the z-plane.

3.5 A causal IIR filter has impulse response $h(n)=4\delta(n)+3(0.5)^{n-1}u(n-1)$. Working with the convolutional equation $y(n)=\sum_m h(m)x(n-m)$, derive the *difference equation* satisfied by $y(n)$.

3.6 A causal IIR filter has impulse response:

$$h(n)=\begin{cases} 5, & \text{if } n=0 \\ 6(0.8)^{n-1}, & \text{if } n\ge 1\end{cases}$$

Working with the convolutional filtering equation, derive the *difference equation* satisfied by $y(n)$.

3.7 To understand the role played the first two values $h(0)$ and $h(1)$, redo Problem 3.6 starting with the more general expression for $h(n)$:

$$h(n)=\begin{cases} c_0 & \text{for } n=0 \\ c_1 a^{n-1} & \text{for } n\ge 1\end{cases}$$

which has $h(0)=c_0$ and $h(1)=c_1$. First, determine the difference equation satisfied by $h(n)$ for all $n\ge 0$. Then, using the I/O convolutional equation (3.3.3), determine the difference equation relating $y(n)$ to $x(n)$. How are $\{c_0, c_1\}$ related to the coefficients of the difference equation?

3.8 A causal linear time-invariant filter has impulse response:

$$h_n=[C_1p_1^n+C_2p_2^n+\cdots+C_Mp_M^n]u(n)$$

Without using any z-transforms and working entirely in the time domain, show that h_n satisfies the order-M difference equation:

$$h_n+a_1h_{n-1}+a_2h_{n-2}+\cdots+a_Mh_{n-M}=0,\qquad \text{for } n\ge M$$

where $\{1, a_1, a_2, \dots, a_M\}$ are the coefficients of the polynomial whose roots are the (complex) numbers $\{p_1, p_2, \dots, p_M\}$, that is,

$$1+a_1z^{-1}+a_2z^{-2}+\cdots+a_Mz^{-M}=(1-p_1z^{-1})(1-p_2z^{-1})\cdots(1-p_Mz^{-1})$$

Note that C_i are arbitrary and the restriction $n\ge M$ necessary.

3.9 A causal linear time-invariant filter has impulse response:

$$h_n=C_0\delta(n)+C_1p_1^n+C_2p_2^n+\cdots+C_Mp_M^n,\qquad n\ge 0$$

Show that it satisfies the same difference equation as in the previous problem, but with the restriction $n\ge M+1$.

3.10 A causal linear time-invariant filter has impulse response:

$$h_n=C_1p_1^n+C_2p_2^n,\qquad n\ge 0$$

Working in the time domain, show that the difference equation satisfied by h_n for all $n\ge 0$ and the difference equation relating the input and output signals are of the form:

$$h_n+a_1h_{n-1}+a_2h_{n-2}=b_0\delta(n)+b_1\delta(n-1)$$

$$y_n+a_1y_{n-1}+a_2y_{n-2}=b_0x_n+b_1x_{n-1}$$

Determine $\{a_1, a_2, b_0, b_1\}$ in terms of $\{C_1, C_2, p_1, p_2\}$.

3.11 A causal linear time-invariant filter has impulse response:

$$h_n=C_0\delta(n)+C_1p_1^n+C_2p_2^n,\qquad n\ge 0$$

Show that the difference equation satisfied by h_n for all $n\ge 0$ and the difference equation relating the input and output signals are of the form:

$$h_n+a_1h_{n-1}+a_2h_{n-2}=b_0\delta(n)+b_1\delta(n-1)+b_2\delta(n-2)$$

$$y_n+a_1y_{n-1}+a_2y_{n-2}=b_0x_n+b_1x_{n-1}+b_2x_{n-2}$$

Determine $\{a_1, a_2, b_0, b_1, b_2\}$ in terms of $\{C_0, C_1, C_2, p_1, p_2\}$.

3.12 A causal linear time-invariant filter has impulse response:

$$h_n=C_1p_1^n+C_2p_2^n+C_3p_3^n,\qquad n\ge 0$$

Working in the time domain, show that the difference equation satisfied by h_n for all $n\ge 0$ and the difference equation relating the input and output signals are of the form:

$$h_n+a_1h_{n-1}+a_2h_{n-2}+a_3h_{n-3}=b_0\delta(n)+b_1\delta(n-1)+b_2\delta(n-2)$$

$$y_n+a_1y_{n-1}+a_2y_{n-2}+a_3y_{n-3}=b_0x_n+b_1x_{n-1}+b_2x_{n-2}$$

Determine $\{a_1, a_2, a_3, b_0, b_1\}$ in terms of $\{C_1, C_2, C_3, p_1, p_2, p_3\}$.

3.13 Using the results of the previous two problems, determine and verify the difference equations satisfied by the impulse responses:

a. $h(n)=5(0.5)^nu(n)+4(0.8)^nu(n)$.

b. $h(n)=(0.5j)^nu(n)+(-0.5j)^nu(n)$.

c. $h(n)=\left[3(0.4)^n+4(0.5)^n-7(-0.5)^n\right]u(n)$.

3.14 The condition of Eq. (3.5.4) is *sufficient* for *bounded-input/bounded-output* (BIBO) stability. Assume $A=\sum_m|h(m)|<\infty$. Show that if the input is bounded, $|x(n)|\le B$, then the output is bounded by $|y(n)|\le AB$.

3.15 The condition of Eq. (3.5.4) is also *necessary* for BIBO stability. Assume that every bounded input results in a bounded output and consider the particular bounded input $x(n)=\text{sign}\big(h(-n)\big)$ defined to be the algebraic sign of $h(-n)$. Then, the corresponding output $y(n)$ will be bounded. By considering the particular output sample $y(0)$, prove that Eq. (3.5.4) must hold. What happens if when $h(-n)=0$ for some n?

3.16 The standard method for making an anticausal (but stable) system into a causal system is to clip off its anticausal tail at some large negative time $n=-D$ and then delay the impulse response by D time units to make it causal, that is, $h_D(n)=h(n-D)$. Let $y(n)$ be the output of $h(n)$ with input $x(n)$, and let $y_D(n)$ be the output of the delayed system $h_D(n)$ also with input $x(n)$. Working in the time domain, show that $y_D(n)$ is the delayed version of $y(n)$, that is, $y_D(n)=y(n-D)$.

3.17 In certain applications, such as data smoothing and FIR interpolation, the desired output $y(n)$ must be computed from a partly anticausal filter, that is, a filter $h(n)$ with anticausal duration of D time units. This filter can be made causal by a delay D, but this would cause the output to be delayed as we saw in the previous problem.

In order to get the correct *undelayed* output from the delayed causal filter $h_D(n)$, the input must be *time-advanced* by D time units, that is, $x_A(n)= x(n+D)$. Using time-domain convolution, show that $y(n)$ can be computed in the following two ways:

$$y(n)=\sum_m h(m)x(n-m)=\sum_m h_D(m)x_A(n-m)$$

In the z-domain, this is obvious:

$$Y(z)=H(z)X(z)=\big[z^{-D}H(z)\big]\big[z^{D}X(z)\big]$$

3.18 Prove the inequality (3.5.6). Is the right-hand side finite? Does it get smaller as D gets larger?

4

FIR Filtering and Convolution

Practical DSP methods fall in two basic classes:

- Block processing methods.
- Sample processing methods.

In block processing methods, the data are collected and processed in blocks. Some typical applications include, FIR filtering of finite-duration signals by convolution, fast convolution of long signals which are broken up in short segments, DFT/FFT spectrum computations, speech analysis and synthesis, and image processing.

In sample processing methods, the data are processed one at a time—with each input sample being subjected to a *DSP algorithm* which transforms it into an output sample. Sample processing methods are used primarily in *real-time* applications, such as real-time filtering of long signals, digital audio effects processing, digital control systems, and adaptive signal processing. Sample processing algorithms are essentially the *state-space* realizations of LTI filters.

In this chapter, we consider block processing and sample processing methods for FIR filtering applications. We discuss the *computational* aspects of the convolution equations (3.3.2) or (3.3.3) as they apply to FIR filters and finite-duration inputs and present various *equivalent* forms of convolution, namely,

- Direct form
- Convolution table
- LTI form
- Matrix form
- Flip-and-slide form
- Overlap-add block convolution form.

Each form has its own distinct advantages. For example, the LTI form is of fundamental importance because it incorporates the consequences of linearity and time invariance; the direct form leads directly to block diagram realizations of the filter and the corresponding sample-by-sample processing algorithms; the convolution table is convenient for quick computations by hand; the flip-and-slide form shows clearly the input-on and input-off transient and steady-state behavior of a filter; the matrix form provides a compact vectorial representation of the filtering operation and is widely used in some applications such as image processing; and the overlap-add form is used whenever the input is extremely long or infinite in duration.

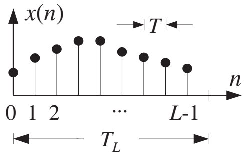
Then, we go on to discuss sample-by-sample processing methods for FIR filtering and discuss block diagram realizations which provide a *mechanization* of the sample processing algorithms. We develop the so-called *direct form* realizations of FIR filters and discuss some hardware issues for DSP chips. We develop also the concept of *circular addressing*, which is the "modern" way to implement delay lines, FIR, and IIR filters in both hardware and software.

4.1 Block Processing Methods

4.1.1 Convolution

In many practical applications, we sample our analog input signal (in accordance with the sampling theorem requirements) and collect a *finite* set of samples, say L samples, representing a finite time record of the input signal. The *duration* of the data record in seconds will be:[†]

$$\boxed{T_L = LT} \tag{4.1.1}$$



where T is the sampling time interval, related to the sampling rate by $f_s = 1/T$. Conversely, we can solve for the number of time samples L contained in a record of duration T_L seconds:

$$\boxed{L = T_L f_s} \tag{4.1.2}$$

The L collected signal samples, say $x(n)$, $n = 0, 1, \dots, L-1$, can be thought of as a block:

$$\mathbf{x} = [x_0, x_1, \dots, x_{L-1}] \tag{4.1.3}$$

which may then be processed further by a digital filter. The direct and LTI forms of convolution given by Eqs. (3.3.3) and (3.3.2)

$$y(n) = \sum_m h(m)x(n-m) = \sum_m x(m)h(n-m) \tag{4.1.4}$$

describe the filtering equation of an LTI system in general. An alternative way of writing these equations, called the *convolution table* form, is obtained by noting that the sum of the indices of $h(m)$ and $x(n-m)$ is $m + (n-m) = n$. Therefore, Eqs. (4.1.4) can be written in the form:

$$y(n) = \sum_{\substack{i,j \\ i+j=n}} h(i)x(j) \qquad \text{(convolution table form)} \tag{4.1.5}$$

[†] More correctly, $T_L = (L-1)T$, but for large L Eq. (4.1.1) is simpler. See also Section 9.1.

That is, the sum of all possible products $h(i)x(j)$ with $i+j = n$. The precise range of summation with respect to m in Eqs. (4.1.4) or i, j in Eq. (4.1.5) depends on the *particular* nature of the filter and input sequences, $h(n)$ and $x(n)$.

4.1.2 Direct Form

Consider a causal FIR filter of order M with impulse response $h(n)$, $n = 0, 1, \dots, M$. It may be represented as a block:

$$\mathbf{h} = [h_0, h_1, \dots, h_M] \tag{4.1.6}$$

Its length (i.e., the number of filter coefficients) is one more than its order:

$$\boxed{L_h = M + 1} \tag{4.1.7}$$

The convolution of the length-L input $\mathbf{x}$ of Eq. (4.1.3) with the order-M filter $\mathbf{h}$ will result in an output sequence $y(n)$. We must determine: (i) the range of values of the output index n, and (ii) the precise range of summation in m. For the direct form, we have

$$y(n) = \sum_m h(m)x(n-m)$$

The index of $h(m)$ must be within the range of indices in Eq. (4.1.6), that is, it must be restricted to the interval:

$$0 \le m \le M \tag{4.1.8}$$

Similarly, the index of $x(n-m)$ must lie within the legal range of indices in Eq. (4.1.3), that is,

$$0 \le n - m \le L - 1 \tag{4.1.9}$$

To determine the range of values of the output index n, we rewrite (4.1.9) in the form:

$$m \le n \le L - 1 + m$$

and use (4.1.8) to extend the limits to:

$$0 \le m \le n \le L - 1 + m \le L - 1 + M, \qquad \text{or,}$$

$$\boxed{0 \le n \le L - 1 + M} \tag{4.1.10}$$

This is the index range of the output sequence $y(n)$. Therefore, it is represented by a block

$$\mathbf{y} = [y_0, y_1, \dots, y_{L-1+M}] \tag{4.1.11}$$

with length

$$\boxed{L_y = L + M} \tag{4.1.12}$$

Thus, $\mathbf{y}$ is longer than the input $\mathbf{x}$ by M samples. As we will see later, this property follows from the fact that a filter of order M has *memory* M and keeps each input sample inside it for M time units. Setting $L_x = L$, and $L_h = M + 1$, we can rewrite Eq. (4.1.12) in the more familiar form:

$$\boxed{L_y = L_x + L_h - 1} \tag{4.1.13}$$

The relative block lengths are shown in Fig. 4.1.1. For any value of the output index n in the range (4.1.10), we must determine the summation range over m in the convolution equation. For fixed n, the inequalities (4.1.8) and (4.1.9) must be satisfied simultaneously by m. Changing the sign of (4.1.9), we obtain

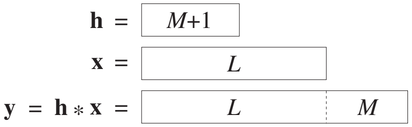


Fig. 4.1.1 Relative lengths of filter, input, and output blocks.

$$-(L-1) \le m - n \le 0$$

and adding n to all sides

$$n - L + 1 \le m \le n \tag{4.1.14}$$

Thus, m must satisfy simultaneously the inequalities:

$$0 \le m \le M$$
$$n - L + 1 \le m \le n$$

It follows that m must be greater than the maximum of the two left-hand sides and less than the minimum of the two right-hand sides, that is,

$$\boxed{\max(0, n - L + 1) \le m \le \min(n, M)} \tag{4.1.15}$$

Therefore, in the case of an order-M FIR filter and a length-L input, the direct form of convolution is given as follows:

$$\boxed{y(n) = \sum_{m=\max(0,n-L+1)}^{\min(n,M)} h(m)x(n-m)} \qquad \text{(direct form)} \tag{4.1.16}$$

for $n = 0, 1, \dots, L + M - 1$. Sometimes, we will indicate this convolutional operation by the compact notation:

$$\mathbf{y} = \mathbf{h} * \mathbf{x}$$

As an example, consider the case of an order-3 filter and a length-5 input signal. The filter, input, and output blocks are

$$\mathbf{h} = [h_0, h_1, h_2, h_3]$$
$$\mathbf{x} = [x_0, x_1, x_2, x_3, x_4]$$
$$\mathbf{y} = \mathbf{h} * \mathbf{x} = [y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7]$$

The output block has length $L_y = L + M = 5 + 3 = 8$ and is indexed as $0 \le n \le 7$. The convolutional equation (4.1.16) becomes:

$$y_n = \sum_{m=\max(0,n-4)}^{\min(n,3)} h_m x_{n-m}, \qquad n = 0, 1, \dots, 7$$

For $n = 0, 1, 2, \dots, 7$, the summation index m takes on the values:

$$\begin{aligned}
\max(0, 0-4) \le m \le \min(0, 3) \quad &\Rightarrow \quad m = 0 \\
\max(0, 1-4) \le m \le \min(1, 3) \quad &\Rightarrow \quad m = 0, 1 \\
\max(0, 2-4) \le m \le \min(2, 3) \quad &\Rightarrow \quad m = 0, 1, 2 \\
\max(0, 3-4) \le m \le \min(3, 3) \quad &\Rightarrow \quad m = 0, 1, 2, 3 \\
\max(0, 4-4) \le m \le \min(4, 3) \quad &\Rightarrow \quad m = 0, 1, 2, 3 \\
\max(0, 5-4) \le m \le \min(5, 3) \quad &\Rightarrow \quad m = 1, 2, 3 \\
\max(0, 6-4) \le m \le \min(6, 3) \quad &\Rightarrow \quad m = 2, 3 \\
\max(0, 7-4) \le m \le \min(7, 3) \quad &\Rightarrow \quad m = 3
\end{aligned} \tag{4.1.17}$$

So, for example, at $n = 5$ the output y_5 will be given by

$$y_5 = \sum_{m=1,2,3} h_m x_{5-m} = h_1 x_4 + h_2 x_3 + h_3 x_2$$

Using the values in Eq. (4.1.17), we find all the output samples:

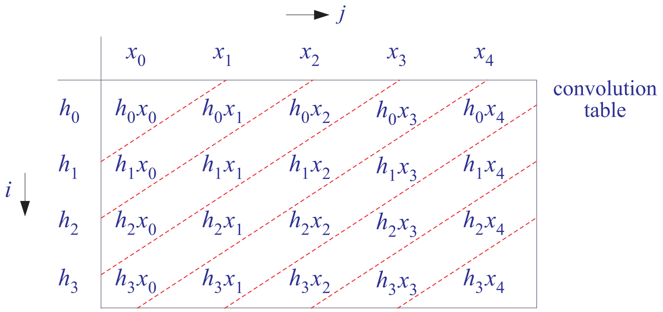
$$\begin{aligned}
y_0 &= h_0 x_0 \\
y_1 &= h_0 x_1 + h_1 x_0 \\
y_2 &= h_0 x_2 + h_1 x_1 + h_2 x_0 \\
y_3 &= h_0 x_3 + h_1 x_2 + h_2 x_1 + h_3 x_0 \\
y_4 &= h_0 x_4 + h_1 x_3 + h_2 x_2 + h_3 x_1 \\
y_5 &= h_1 x_4 + h_2 x_3 + h_3 x_2 \\
y_6 &= h_2 x_4 + h_3 x_3 \\
y_7 &= h_3 x_4
\end{aligned} \tag{4.1.18}$$

4.1.3 Convolution Table

Note how each output y_n in Eq. (4.1.18) is the sum of all possible products $h_i x_j$ with $i + j = n$. This leads directly to the convolution table of Eq. (4.1.5). For example, y_5 is obtained as

$$y_5 = \sum_{\substack{i,j \\ i+j=5}} h_i x_j = h_1 x_4 + h_2 x_3 + h_3 x_2$$

The required computations can be arranged in a table [24] as shown in Fig. 4.1.2 with the filter $\mathbf{h}$ written vertically and the input block $\mathbf{x}$ horizontally.[†]



	x_0	x_1	x_2	x_3	x_4
h_0	h_0x_0	h_0x_1	h_0x_2	h_0x_3	h_0x_4
h_1	h_1x_0	h_1x_1	h_1x_2	h_1x_3	h_1x_4
h_2	h_2x_0	h_2x_1	h_2x_2	h_2x_3	h_2x_4
h_3	h_3x_0	h_3x_1	h_3x_2	h_3x_3	h_3x_4

Fig. 4.1.2 Convolution table.

The nth row of the table is filled by multiplying the $\mathbf{x}$ samples by the corresponding h_n sample for that row. Then, the table is "folded" along its antidiagonal lines. In the ij-plane, the condition $i+j=n$ represents the nth antidiagonal straight line. Therefore, the entries within each antidiagonal strip are *summed* together to form the corresponding output value. There are as many antidiagonal strips as output samples y_n. For example, the $n=0$ strip contains only h_0x_0, which is y_0; the $n=1$ strip contains h_0x_1 and h_1x_0, whose sum is y_1, and so on; finally the $n=7$ strip contains only h_3x_4, which is y_7.

The convolution table is convenient for quick calculation by hand because it displays all required operations compactly.

Example 4.1.1: Calculate the convolution of the following filter and input signals:

$$\mathbf{h}=[1,2,-1,1],\qquad \mathbf{x}=[1,1,2,1,2,2,1,1]$$

Solution: The convolution table, with $\mathbf{h}$ arranged vertically and $\mathbf{x}$ horizontally, is

h\x	1	1	2	1	2	2	1	1
1	1	1	2	1	2	2	1	1
2	2	2	4	2	4	4	2	2
-1	-1	-1	-2	-1	-2	-2	-1	-1
1	1	1	2	1	2	2	1	1

Folding the table, we get

$$\mathbf{y}=[1,3,3,5,3,7,4,3,3,0,1]$$

Note that there are $L_y=L+M=8+3=11$ output samples. □

[†] It can also be arranged the other way, with the filter horizontally and the input vertically.

4.1.4 LTI Form

Next, we discuss the LTI form of convolution. A more intuitive way to understand it is in terms of the linearity and time-invariance properties of the filter. Consider again the filter $\mathbf{h}=[h_0,h_1,h_2,h_3]$ in the example of Eq. (4.1.18). The input signal

$$\mathbf{x}=[x_0,x_1,x_2,x_3,x_4]$$

can be written as a linear combination of delayed impulses:

$$\begin{aligned}\mathbf{x}&=x_0[1,0,0,0,0]\\&+x_1[0,1,0,0,0]\\&+x_2[0,0,1,0,0]\\&+x_3[0,0,0,1,0]\\&+x_4[0,0,0,0,1]\end{aligned}$$

It can also be written analytically for all n as a sum of delta functions:

$$x(n)=x_0\delta(n)+x_1\delta(n-1)+x_2\delta(n-2)+x_3\delta(n-3)+x_4\delta(n-4)$$

The effect of the filter is to replace each delayed impulse by the corresponding delayed impulse response, that is,

$$y(n)=x_0h(n)+x_1h(n-1)+x_2h(n-2)+x_3h(n-3)+x_4h(n-4)$$

We can represent the input and output signals as blocks:

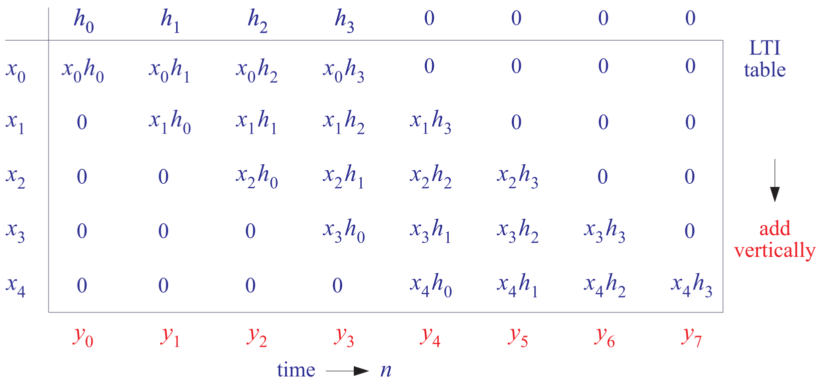
$$\begin{array}{rlcrl}\mathbf{x}=&x_0[1,0,0,0,0] & & \mathbf{y}=&x_0[h_0,h_1,h_2,h_3,0,0,0,0]\\ +&x_1[0,1,0,0,0] & & +&x_1[0,h_0,h_1,h_2,h_3,0,0,0]\\ +&x_2[0,0,1,0,0] & \xrightarrow{H} & +&x_2[0,0,h_0,h_1,h_2,h_3,0,0]\\ +&x_3[0,0,0,1,0] & & +&x_3[0,0,0,h_0,h_1,h_2,h_3,0]\\ +&x_4[0,0,0,0,1] & & +&x_4[0,0,0,0,h_0,h_1,h_2,h_3]\end{array}$$

The result is the same as Eq. (4.1.18). Indeed, the indicated linear combinations in the right-hand side give:

$$\begin{aligned}\mathbf{y}&=[h_0x_0,\ x_0h_1+x_1h_0,\ x_0h_2+x_1h_1+x_2h_0,\ \dots,\ x_4h_3]\\&=[y_0,y_1,y_2,\dots,y_7]\end{aligned}$$

For computational purposes, the LTI form can be represented pictorially in a table form, as shown in Fig. 4.1.3. The impulse response $\mathbf{h}$ is written horizontally, and the input $\mathbf{x}$ vertically.

The rows of the table correspond to the successive delays (right shifts) of the $\mathbf{h}$ sequence—the mth row corresponds to delay by m units. Each row is scaled by the



	h_0	h_1	h_2	h_3	0	0	0	0
x_0	x_0h_0	x_0h_1	x_0h_2	x_0h_3	0	0	0	0
x_1	0	x_1h_0	x_1h_1	x_1h_2	x_1h_3	0	0	0
x_2	0	0	x_2h_0	x_2h_1	x_2h_2	x_2h_3	0	0
x_3	0	0	0	x_3h_0	x_3h_1	x_3h_2	x_3h_3	0
x_4	0	0	0	0	x_4h_0	x_4h_1	x_4h_2	x_4h_3
	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7

Fig. 4.1.3 LTI form of convolution.

corresponding input sample, that is, the mth row represents the term $x_m h_{n-m}$ in the LTI form. After the table is filled, the table entries are summed *column-wise* to obtain the output samples $y(n)$, which is equivalent to forming the sum:

$$y(n)=\sum_m x(m)h(n-m)$$

Example 4.1.2: Calculate the convolution of Example 4.1.1 using the LTI form.

Solution: The corresponding LTI table is in this case:

n	0	1	2	3	4	5	6	7	8	9	10	
x\h	1	2	-1	1								partial output
1	1	2	-1	1								x_0h_n
1		1	2	-1	1							x_1h_{n-1}
2			2	4	-2	2						x_2h_{n-2}
1				1	2	-1	1					x_3h_{n-3}
2					2	4	-2	2				x_4h_{n-4}
2						2	4	-2	2			x_5h_{n-5}
1							1	2	-1	1		x_6h_{n-6}
1								1	2	-1	1	x_7h_{n-7}
y_n	1	3	3	5	3	7	4	3	3	0	1	$\sum_m x_m h_{n-m}$

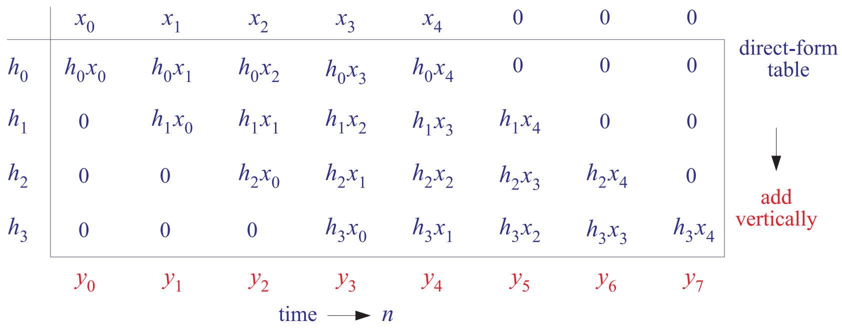
The output samples are obtained by summing the entries in each column. The result agrees with Example 4.1.1. □

The LTI form can also be written in a form similar to Eq. (4.1.16) by determining the proper limits of summation. Arguing as in the direct form, or interchanging the roles of $h(n)$ and $x(n)$ and correspondingly, the length quantities M and $L-1$, we obtain the following form:

$$\boxed{y(n)=\sum_{m=\max(0,n-M)}^{\min(n,L-1)} x(m)h(n-m)} \qquad \text{(LTI form)} \qquad (4.1.19)$$

for $n=0,1,\dots,L+M-1$.

In the *direct form*, which is equivalent to exchanging the roles of h_n and x_n in the LTI form, we form a linear combination of delayed replicas of the input signal, with coefficients being the impulse response. We display the delayed/scaled replicas horizontally, and add them vertically,



	x_0	x_1	x_2	x_3	x_4	0	0	0
h_0	h_0x_0	h_0x_1	h_0x_2	h_0x_3	h_0x_4	0	0	0
h_1	0	h_1x_0	h_1x_1	h_1x_2	h_1x_3	h_1x_4	0	0
h_2	0	0	h_2x_0	h_2x_1	h_2x_2	h_2x_3	h_2x_4	0
h_3	0	0	0	h_3x_0	h_3x_1	h_3x_2	h_3x_3	h_3x_4
	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7

4.1.5 Matrix Forms

The convolutional equations (4.1.16) or (4.1.19) can also be written in the linear matrix form:

$$\boxed{\mathbf{y}=H\mathbf{x}} \qquad (4.1.20)$$

where H is built out of the filter's impulse response $\mathbf{h}$. Because the output vector $\mathbf{y}$ has length $L+M$ and the input vector $\mathbf{x}$ length L, the filter matrix H must be rectangular with dimensions

$$L_y\times L_x=(L+M)\times L$$

To see this, consider again the example of Eq. (4.1.18). The output samples can be arranged in the matrix form

$$\mathbf{y}=\begin{bmatrix} y_0\\ y_1\\ y_2\\ y_3\\ y_4\\ y_5\\ y_6\\ y_7 \end{bmatrix}=\begin{bmatrix} h_0 & 0 & 0 & 0 & 0\\ h_1 & h_0 & 0 & 0 & 0\\ h_2 & h_1 & h_0 & 0 & 0\\ h_3 & h_2 & h_1 & h_0 & 0\\ 0 & h_3 & h_2 & h_1 & h_0\\ 0 & 0 & h_3 & h_2 & h_1\\ 0 & 0 & 0 & h_3 & h_2\\ 0 & 0 & 0 & 0 & h_3 \end{bmatrix}\begin{bmatrix} x_0\\ x_1\\ x_2\\ x_3\\ x_4 \end{bmatrix}=H\mathbf{x}$$

Note that the *columns* of H are the successively delayed replicas of the impulse response vector $\mathbf{h}$. There are as many columns as input samples. Note also that H is a so-called *Toeplitz matrix*, in the sense that it has the same entry along each diagonal. The Toeplitz property is a direct consequence of the time invariance of the filter. Note also that Eq. (4.1.20) is equivalent to the LTI table, transposed column-wise instead of row-wise.

Example 4.1.3: Calculate the convolution of Example 4.1.1 using the matrix form.

Solution: Because $L_y = 11$ and $L_x = 8$, the filter matrix will be 11×8 dimensional. We have,

$$
H\mathbf{x} = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
2 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\
-1 & 2 & 1 & 0 & 0 & 0 & 0 & 0\\
1 & -1 & 2 & 1 & 0 & 0 & 0 & 0\\
0 & 1 & -1 & 2 & 1 & 0 & 0 & 0\\
0 & 0 & 1 & -1 & 2 & 1 & 0 & 0\\
0 & 0 & 0 & 1 & -1 & 2 & 1 & 0\\
0 & 0 & 0 & 0 & 1 & -1 & 2 & 1\\
0 & 0 & 0 & 0 & 0 & 1 & -1 & 2\\
0 & 0 & 0 & 0 & 0 & 0 & 1 & -1\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix} 1\\ 1\\ 2\\ 1\\ 2\\ 2\\ 1\\ 1 \end{bmatrix}
=
\begin{bmatrix} 1\\ 3\\ 3\\ 5\\ 3\\ 7\\ 4\\ 3\\ 3\\ 0\\ 1 \end{bmatrix}
$$

which agrees with Example 4.1.1. □

There is also an alternative matrix form written as follows:

$$
\begin{bmatrix} y_0\\ y_1\\ y_2\\ y_3\\ y_4\\ y_5\\ y_6\\ y_7 \end{bmatrix}
=
\begin{bmatrix}
x_0 & 0 & 0 & 0\\
x_1 & x_0 & 0 & 0\\
x_2 & x_1 & x_0 & 0\\
x_3 & x_2 & x_1 & x_0\\
x_4 & x_3 & x_2 & x_1\\
0 & x_4 & x_3 & x_2\\
0 & 0 & x_4 & x_3\\
0 & 0 & 0 & x_4
\end{bmatrix}
\begin{bmatrix} h_0\\ h_1\\ h_2\\ h_3 \end{bmatrix}
\qquad (4.1.21)
$$

Instead of a filter matrix H acting on the input data vector $\mathbf{x}$, it has a data matrix acting on the filter vector $\mathbf{h}$. It can be written compactly in the form:

$$
\boxed{\mathbf{y} = X\mathbf{h}} \qquad (4.1.22)
$$

where the data matrix X has dimension:

$$
L_y \times L_h = (L+M)\times(M+1)
$$

The *first* column of X is the given input, padded with M zeros at the end to account for the input-off transients. The remaining columns are the successively delayed (down-shifted) versions of the first one. We will see in Section 4.3.2 that this form is essentially equivalent to the sample-by-sample processing algorithm of the direct form realization of the filter—with the nth row of X representing the filter's internal states at time n.

Example 4.1.4: Calculate the convolution of Example 4.1.1 using the matrix form (4.1.22).

Solution: The X matrix will have dimension $L_y \times L_h = 11\times 4$. Its first column is the input signal padded with 3 zeros at the end:

$$
X\mathbf{h} = \begin{bmatrix}
1 & 0 & 0 & 0\\
1 & 1 & 0 & 0\\
2 & 1 & 1 & 0\\
1 & 2 & 1 & 1\\
2 & 1 & 2 & 1\\
2 & 2 & 1 & 2\\
1 & 2 & 2 & 1\\
1 & 1 & 2 & 2\\
0 & 1 & 1 & 2\\
0 & 0 & 1 & 1\\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix} 1\\ 2\\ -1\\ 1 \end{bmatrix}
=
\begin{bmatrix} 1\\ 3\\ 3\\ 5\\ 3\\ 7\\ 4\\ 3\\ 3\\ 0\\ 1 \end{bmatrix}
$$

which agrees with Example 4.1.1. □

Matrix representations of convolution are very useful in some applications, such as image processing, and in more advanced DSP methods such as parametric spectrum estimation and adaptive filtering.

4.1.6 Flip-and-Slide Form

The LTI form is also closely related to the popular flip-and-slide form of convolution, in which the filter $h(n)$ is flipped around or reversed and then slid over the input data sequence. At each time instant, the output sample is obtained by computing the *dot* product of the flipped filter vector $\mathbf{h}$ with the $M+1$ input samples aligned below it, as shown in Fig. 4.1.4.

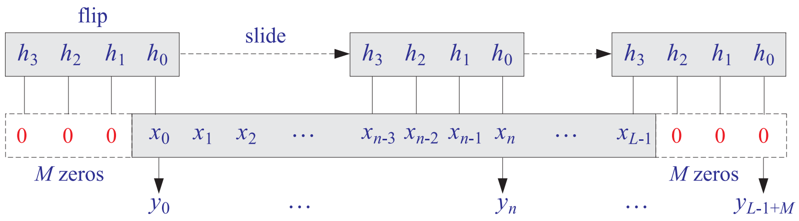


Fig. 4.1.4 Flip-and-slide form of convolution.

The input sequence is assumed to have been extended by padding M zeros to its left and to its right. At time $n = 0$, the only nonzero contribution to the dot product comes from h_0 and x_0 which are time aligned. It takes the filter M time units before it is completely over the nonzero portion of the input sequence. The first M outputs correspond to the *input-on transient* behavior of the filter. Then, for a period of time $M \le n \le L-1$, the filter remains completely over the nonzero portion of the input data, and the outputs are given by the form

$$
y_n = h_0x_n + h_1x_{n-1} + \cdots + h_Mx_{n-M}
$$

This period corresponds to the *steady-state* behavior of the filter. Finally, the *last* M outputs beyond the end of the input data are the *input-off transients*, that is, they are the outputs after the input has been turned off. They correspond to the time period $L \leq n \leq L-1+M$. During this period the filter slides over the last M zeros padded at the end of the input. The very last output is obtained when h_M is aligned over x_{L-1}, which gives $y_{L-1+M} = h_M x_{L-1}$.

One can also think of the filter block $\mathbf{h}$ as being stationary and the input block $\mathbf{x}$ sliding underneath it in the opposite direction. This view leads to the sample-by-sample processing algorithms for FIR filtering.

4.1.7 Transient and Steady-State Behavior

The transient and steady-state behavior of an FIR filter can also be understood using the direct form of convolution, Eq. (4.1.16). For a length-L input and order-M filter, the output time index n will be in the range:

$$0 \leq n \leq L-1+M$$

It can be divided into *three subranges*, depicted in Fig. 4.1.5, corresponding to the input-on transients, steady state, and input-off transients:

$$\begin{array}{ll} 0 \leq n < M & \text{(input-on transients)} \\ M \leq n \leq L-1 & \text{(steady state)} \\ L-1 < n \leq L-1+M & \text{(input-off transients)} \end{array}$$

These subranges affect differently the limits of the convolution summation equation (4.1.16). As implied in Fig. 4.1.5, we assumed that the filter length is much shorter than the length of the input, that is, $M+1 < L$ or $M < L-1$, otherwise the steady-state range defined above does not exist—the input is too short to exhibit any steady behavior.

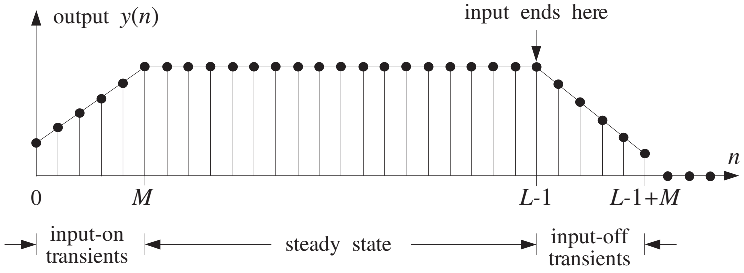


Fig. 4.1.5 Transient and steady-state filter outputs.

For the input-on transients, the restriction $0 \leq n < M < L-1$ implies the following summation limits:

$$\max(0, n-L+1) = 0, \qquad \min(n, M) = n$$

For the steady-state range, $M \leq n \leq L-1$, we have:

$$\max(0, n-L+1) = 0, \qquad \min(n, M) = M$$

And, for the input-off transients, $M < L-1 < n \leq L-1+M$, we have:

$$\max(0, n-L+1) = n-L+1, \qquad \min(n, M) = M$$

Therefore, Eq. (4.1.16) takes the following different forms depending on the value of the output index n:

$$y_n = \begin{cases} \displaystyle\sum_{m=0}^{n} h_m x_{n-m}, & \text{if } 0 \leq n < M, \qquad \text{(input-on)} \\[2ex] \displaystyle\sum_{m=0}^{M} h_m x_{n-m}, & \text{if } M \leq n \leq L-1, \qquad \text{(steady state)} \\[2ex] \displaystyle\sum_{m=n-L+1}^{M} h_m x_{n-m}, & \text{if } L-1 < n \leq L-1+M, \qquad \text{(input-off)} \end{cases}$$

During the input-on transient period, the number of terms in the sum is $n+1$ and is increasing. During the steady-state period, the number of terms is equal to the number of filter weights, $M+1$, and remains fixed. And during the input-off period, the number of terms in the sum keeps decreasing down to one because the lower summation limit is increasing.

In Eq. (4.1.18), the first three outputs $\{y_0, y_1, y_2\}$ are the input-on transients and the number of terms keeps increasing to 4. The next two outputs $\{y_3, y_4\}$ are the steady-state outputs, and the last three outputs $\{y_5, y_6, y_7\}$ are the input-off transients having a decreasing number of terms.

The I/O equation in the steady state was also discussed earlier in Eq. (3.4.1). It has a fixed number of terms:

$$\boxed{y(n) = \sum_{m=0}^{M} h(m)x(n-m)} \qquad \text{(steady state)} \tag{4.1.23}$$

In a certain sense, it also incorporates the input-on and input-off transients and is quoted often as the generic I/O equation for FIR filters. To understand why, consider again the example of Eq. (4.1.18). In this case, we write

$$y_n = h_0 x_n + h_1 x_{n-1} + h_2 x_{n-2} + h_3 x_{n-3}$$

If n is in the input-on range, that is, $0 \leq n \leq 2$, not every term in the sum will contribute, because x_n is assumed causal. For example, if $n = 1$, we have

$$y_1 = h_0 x_1 + h_1 x_{1-1} + h_2 x_{1-2} + h_3 x_{1-3} = h_0 x_1 + h_1 x_0$$

When n is in the steady range, then all terms are contributing. And, when n is in the input-off range, not all terms contribute. For example, if $n = 6$ we have

$$y_6 = h_0 x_6 + h_1 x_{6-1} + h_2 x_{6-2} + h_3 x_{6-3} = h_2 x_4 + h_3 x_3$$

because x_n was assumed to have length 5, and therefore $x_6 = x_5 = 0$. With these caveats in mind, it is correct to write Eq. (4.1.23) as the generic I/O filtering equation of an order-M FIR filter. For programming purposes, one must of course work with Eq. (4.1.16) which does not let the indices exceed the array bounds.

4.1.8 Convolution of Infinite Sequences

By taking appropriate limits of the direct form of convolution

$$y_n = \sum_{m=\max(0,n-L+1)}^{\min(n,M)} h_m x_{n-m}$$

we can obtain the correct summation limits for the following three cases:

1. Infinite filter, finite input; i.e., $M = \infty$, $L < \infty$.
2. Finite filter, infinite input; i.e., $M < \infty$, $L = \infty$.
3. Infinite filter, infinite input; i.e., $M = \infty$, $L = \infty$.

In all three cases, the range of the output index Eq. (4.1.10) is infinite, $0 \le n < \infty$, that is, the output $y(n)$ has infinite duration. When $M \to \infty$, the upper limit in the convolution sum becomes

$$\min(n, M) = n$$

and when $L \to \infty$, the lower limit becomes

$$\max(0, n - L + 1) = 0$$

Therefore, we find in the three cases:

$$y_n = \sum_{m=\max(0,n-L+1)}^{n} h_m x_{n-m}, \qquad \text{if } M = \infty, L < \infty$$

$$y_n = \sum_{m=0}^{\min(n,M)} h_m x_{n-m}, \qquad \text{if } M < \infty, L = \infty$$

$$y_n = \sum_{m=0}^{n} h_m x_{n-m}, \qquad \text{if } M = \infty, L = \infty$$

When the filter is infinite, we define steady state as the limit of $y(n)$ for large n.

Example 4.1.5: An IIR filter has impulse response $h(n) = (0.75)^n u(n)$. Using convolution, derive closed-form expressions for the output signal $y(n)$ when the input is:

(a) A unit step, $x(n) = u(n)$.

(b) An alternating step, $x(n) = (-1)^n u(n)$.

(c) A square pulse of duration $L = 25$ samples, $x(n) = u(n) - u(n-25)$.

In each case, determine the steady-state response of the filter.

Solution: In case (a), because both the input and filter are causal and have infinite duration, we use the formula:

$$y(n) = \sum_{m=0}^{n} h(m)x(n-m) = \sum_{m=0}^{n} (0.75)^m u(m)u(n-m)$$

or, using the finite geometric series:

$$y(n) = \sum_{m=0}^{n} (0.75)^m = \frac{1-(0.75)^{n+1}}{1-0.75} = 4 - 3(0.75)^n$$

The steady-state response is the large-n limit of this formula, that is, as $n \to \infty$

$$y(n) \to \frac{1}{1-0.75} = 4$$

For case (b), we have

$$\begin{aligned} y(n) &= \sum_{m=0}^{n} (0.75)^m (-1)^{n-m} = (-1)^n \sum_{m=0}^{n} (-0.75)^m \\ &= (-1)^n \frac{1-(-0.75)^{n+1}}{1+0.75} = \frac{4}{7}(-1)^n + \frac{3}{7}(0.75)^n \end{aligned}$$

where in the second term, we wrote $(-1)^n(-0.75)^n = (0.75)^n$. In the large-$n$ limit, $n \to \infty$, we have

$$y(n) \to (-1)^n \frac{1}{1+0.75} = \frac{4}{7}(-1)^n$$

We will see later that these steady-state responses correspond to special cases of the sinusoidal response of the filter (at frequencies $\omega = 0$ and $\omega = \pi$), and can be obtained very simply in terms of the transfer function $H(z)$ of the filter evaluated at $z = 1$ for part (a), and $z = -1$ for part (b), that is,

$$y(n) \to H(1) \qquad \text{and} \qquad y(n) \to (-1)^n H(-1)$$

where in this example,

$$H(z) = \frac{1}{1-0.75z^{-1}} \quad \Rightarrow \quad H(1) = \frac{1}{1-0.75} = 4, \quad H(-1) = \frac{1}{1+0.75} = \frac{4}{7}$$

In part (c), the input is finite with length $L = 25$. Therefore,

$$y_n = \sum_{m=\max(0,n-L+1)}^{n} h_m x_{n-m} = \sum_{m=\max(0,n-24)}^{n} (0.75)^m$$

We must distinguish two subranges in the output index: For $0 \le n \le 24$, we have

$$y_n = \sum_{m=0}^{n} (0.75)^m = \frac{1-(0.75)^{n+1}}{1-0.75} = 4 - 3(0.75)^n$$

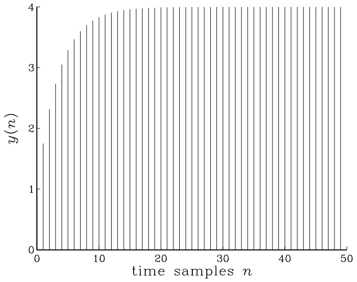
and for $25 \le n < \infty$,

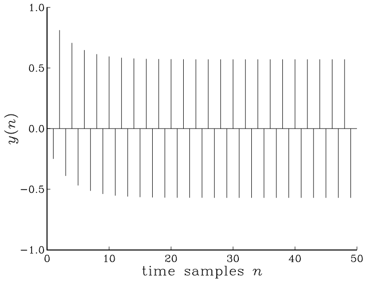
$$y_n = \sum_{m=n-24}^{n} (0.75)^m = (0.75)^{n-24}\frac{1-(0.75)^{n-(n-24)+1}}{1-0.75}$$

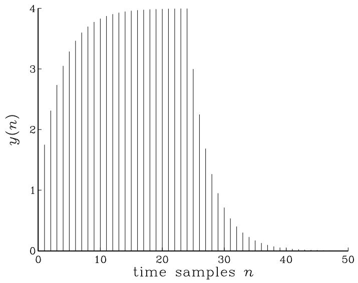
$$= (0.75)^{n-24}\frac{1-(0.75)^{25}}{1-0.75}$$

which corresponds to the input-off transient behavior. Because of the exponentially decaying nature of the impulse response, this filter acts like an RC-type integrator. During the input-on period $0 \le n \le 24$, the output “charges” up and during the input-off period $n \ge 25$, it “discharges” down. See Example 4.1.8 for a similar, but not quite identical, example.

The output signals $y(n)$ of the three cases (a-c) are shown below:







Notice the steady-state behavior of the first two cases, and the input-off transients of the third. □

Example 4.1.6: We saw in Example 3.4.5 that a filter of the form $h_n = (0.75)^n u(n)$ satisfies the difference equation:

$$y(n) = 0.75y(n-1) + x(n)$$

Verify that the expressions for $y(n)$ obtained in the three cases (a-c) in Example 4.1.5 are solutions of this difference equation, with causal initial conditions.

Solution: In case (a), we have $x(n) = u(n)$ and the difference equation becomes, for $n \ge 0$:

$$y(n) = 0.75y(n-1) + 1$$

For $n = 0$, it gives $y(0) = 0.75y(-1) + 1 = 0.75\cdot 0 + 1 = 1$, which agrees with the expression $y(n) = 4 - 3(0.75)^n$ evaluated at $n = 0$. For $n \ge 1$, we have, starting with the right-hand side:

$$0.75y(n-1) + 1 = 0.75\big[4 - 3(0.75)^{n-1}\big] + 1 = 4 - 3(0.75)^n = y(n)$$

In case (b), we have $x(n) = (-1)^n u(n)$ and the difference equation becomes for $n \ge 1$:

$$0.75y(n-1) + x(n) = 0.75\left[\frac{4}{7}(-1)^{n-1} + \frac{3}{7}(0.75)^{n-1}\right] + (-1)^n$$

$$= -0.75\frac{4}{7}(-1)^n + \frac{3}{7}(0.75)^n + (-1)^n = \frac{4}{7}(-1)^n + \frac{3}{7}(0.75)^n = y(n)$$

In case (c), we have the difference equations

$$y(n) = 0.75y(n-1) + 1, \qquad \text{for} \quad 0 \le n \le 24$$

and

$$y(n) = 0.75y(n-1), \qquad \text{for} \quad n \ge 25$$

The first value at $n = 25$ will be $y(25) = 0.75y(24)$, and therefore, it requires knowledge of the “initial” value $y(24)$. If that is known, then the solution of the homogeneous equation will be

$$y(n) = (0.75)^{n-24} y(24), \qquad \text{for} \quad n \ge 25$$

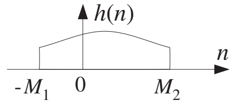
But, $y(24)$ is

$$y(24) = \frac{1-(0.75)^{25}}{1-0.75} = 4 - 3(0.75)^{24}$$

as obtained from the solution of the first difference equation evaluated at the endpoint $n = 24$. □

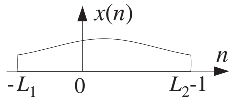
The most *general case* that covers any type of filter and input signal— finite or infinite, causal or non-causal—can be defined as follows. Assume the filter’s impulse response $h(n)$ is defined over the interval:

$$-M_1 \le n \le M_2$$



and the input signal $x(n)$ over the interval:

$$-L_1 \le n \le L_2 - 1$$



Any desired case can be obtained by taking appropriate limits in the quantities M_1, M_2, L_1, L_2. We wish to determine the range of the output index n and the limits of summation in the convolutional equation

$$y(n) = \sum_m h(m)x(n-m)$$

The index m of $h(m)$ and $n-m$ of $x(n-m)$ must lie within the given index ranges, that is,

$$-M_1 \le m \le M_2 \quad \text{and} \quad -L_1 \le n-m \le L_2 - 1$$

From these it follows that the output index n must vary over the range:

$$-M_1 - L_1 \le n \le L_2 + M_2 - 1$$

Note that the endpoints of the output index are the *sum* of the corresponding endpoints for $h(n)$ and $x(n)$. For each n in this output range, the summation index must be within the limits:

$$\max(-M_1, n-L_2+1) \le m \le \min(n+L_1, M_2)$$

Therefore, the I/O equation is in this case:

$$y(n) = \sum_{m=\max(-M_1,n-L_2+1)}^{\min(n+L_1,M_2)} h(m)x(n-m)$$

The results of Eq. (4.1.16) can be recovered as the special case corresponding to $M_1 = 0$, $M_2 = M$, $L_1 = 0$, $L_2 = L$.

4.1.9 Programming Considerations

The following C routine `conv.c` implements the direct form of convolution of Eq. (4.1.16):

```
/* conv.c - convolution of x[n] with h[n], resulting in y[n] */

#include <stdlib.h>                           defines max( ) and min( )

void conv(M, h, L, x, y)
double *h, *x, *y;                            h, x, y = filter, input, output arrays
int M, L;                                     M = filter order, L = input length
{
       int n, m;

       for (n = 0; n < L+M; n++)
              for (y[n] = 0, m = max(0, n-L+1); m <= min(n, M); m++)
                     y[n] += h[m] * x[n-m];
}
```

The quantities `h`, `x`, `y` are *arrays* and must be declared or allocated to proper dimension in the *main* program; for example, using `calloc`:

```
double *h, *x, *y;
h = (double *) calloc(M+1, sizeof(double));         (M+1)-dimensional
x = (double *) calloc(L,   sizeof(double));         L-dimensional
y = (double *) calloc(L+M, sizeof(double));         (L+M)-dimensional
```

In some C implementations,[†] the include-file `stdlib.h` contains the definitions of the two macros `max` and `min`; otherwise they must be added in the above routine:

[†] For example, Microsoft and Borland C.

```
#define max(a,b)  (((a) > (b)) ? (a) : (b))
#define min(a,b)  (((a) < (b)) ? (a) : (b))
```

Next, we present a few simulation examples illustrating some of the ideas we discussed, such as input-on and input-off transients, steady state, linearity, and time invariance. A quantity of interest in these examples will be the *DC gain* of a (stable) filter, that is, the steady-state value of its output when the input remains constant for a long period of time. For a unity input, it is given by the sum of the impulse response coefficients:

$$y_{\text{dc}} = \sum_m h(m) \tag{4.1.24}$$

It will be derived later on.

Example 4.1.7: Consider an integrator-like FIR filter of order $M = 14$ defined by the following I/O convolutional equation:

$$y(n) = G\big[x(n) + x(n-1) + x(n-2) + \cdots + x(n-14)\big]$$

where G is a convenient scale factor, taken to be $G = 0.1$. Such a filter accumulates (integrates) the present and past 14 samples of the input signal. Comparing with Eq. (4.1.23), we identify the impulse response of this filter:

$$h_n = \begin{cases} G, & \text{for} \quad 0 \le n \le 14 \\ 0, & \text{otherwise} \end{cases}$$

The DC gain will be:

$$y_{\text{dc}} = \sum_{m=0}^{14} h(m) = \sum_{m=0}^{14} G = 15G = 1.5$$

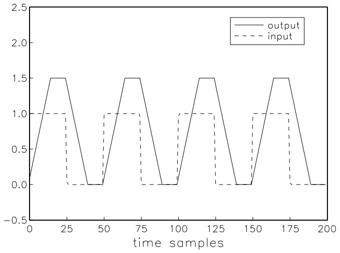
To observe the steady-state response of this filter, as well as the input-on and input-off transients, we consider a *square-wave* input signal x_n of length $L = 200$ and period of $K = 50$ samples. Such a signal may be generated by the simple for-loop:

```
for (n=0; n<L; n++)
    if (n%K < K/2)                      n % K is the MOD operation
        x[n] = 1;
    else
        x[n] = 0;
```

The output signal y_n will have length $L_y = L + M = 200 + 14 = 214$ samples. It can be obtained by a single call to the routine `conv`:

```
conv(M, h, L, x, y);
```

The figure below shows the output signal y_n plotted together with the periodic input.



As the square wave periodically goes on and off, we can observe the input-on transient, steady-state, and input-off transient behavior of the filter.

During each on-period of the square wave lasting for 25 samples, the filter exhibits an input-on transient behavior which lasts for 14 samples; then it reaches steady state (equal to its DC gain) lasting only $25-14 = 11$ samples, and then the square wave goes off causing the filter to undergo its input-off transient behavior which lasts another 14 samples. The filter's output settles down to zero only after $25+14 = 39$ samples and remains zero until the onset of the next on-period of the square wave, and the whole process repeats. □

Example 4.1.8: Consider the following two FIR filters, one defined in terms of its impulse response and the other in terms of its transfer function:

(a) $h_n = \begin{cases} ba^n, & \text{for } 0 \le n \le M \\ 0, & \text{otherwise} \end{cases}$

Take $M = 14$, $a = 0.75$, and $b = 1 - a = 0.25$. Its DC gain is almost unity:

$$y_{\rm dc} = \sum_{m=0}^{M} h(m) = b\sum_{m=0}^{M} a^m = (1-a)\cdot\frac{1-a^{M+1}}{1-a} = 1 - a^{M+1}$$

or, $y_{\rm dc} = 1 - (0.75)^{15} = 0.987$.

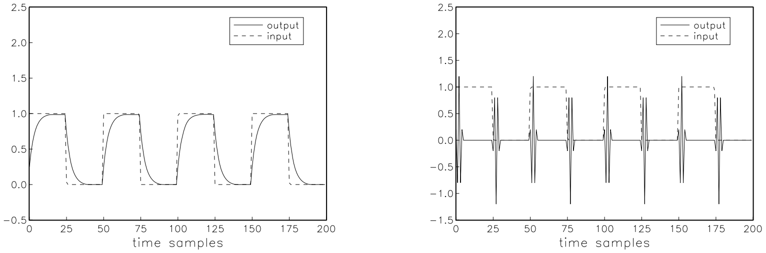
(b) $H(z) = \frac{1}{5}(1-z^{-1})^5 = 0.2 - z^{-1} + 2z^{-2} - 2z^{-3} + z^{-4} - 0.2z^{-5}$

This filter has $M = 5$ and acts as a 5-fold differentiator. Its impulse response can be extracted from $H(z)$:

$$\mathbf{h} = [0.2,\ -1,\ 2,\ -2,\ 1,\ -0.2] = \frac{1}{5}[1,\ -5,\ 10,\ -10,\ 5,\ -1]$$

The factor $1/5$ serves only as a convenient scale. Its DC gain is zero.

The square wave input of the previous example is fed into the two filters. The resulting output signals, computed by two calls to conv, are shown in the figure below:



Filter (a) acts more like an RC-type integrator than an accumulator. The exponentially decaying nature of the impulse response causes the *charging/discharging* type of output as the input goes on and off.

Filter (b) acts as a differentiator, differentiating the constant portions (i.e., the on portions) of the input to zero. The input-on and off transients have duration of $M = 5$, but the rest of the output is zero. □

Example 4.1.9: To demonstrate the concepts of impulse response, linearity, and time invariance, consider an FIR filter with impulse response

$$h(n) = (0.95)^n, \qquad \text{for } 0 \le n \le 24$$

and an input signal

$$x(n) = \delta(n) + 2\delta(n-40) + 2\delta(n-70) + \delta(n-80), \qquad n = 0, 1, \dots, 120$$

consisting of four impulses of the indicated strengths occurring at the indicated time instants. Note that the first two impulses are separated by more than the duration of the filter, whereas the last two are separated by less.

Using the LTI form of convolution, we obtain the filter output by replacing each delayed impulse by the delayed impulse response, that is,

$$y(n) = h(n) + 2h(n-40) + 2h(n-70) + h(n-80) \tag{4.1.25}$$

The input signal can be generated with the aid of the following C routine that implements a delta function $\delta(n)$:

```
/* delta.c - delta function */

double delta(n)
int n;
{
       if (n == 0)
              return 1;
       else
              return 0;
}
```

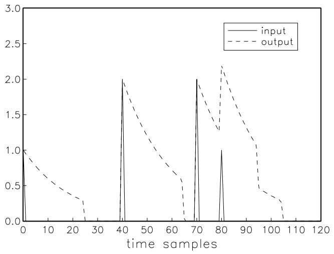
The input signal can be generated by a for-loop of the form

```
for (n=0; n<=120; n++)
    x[n] = delta(n) + 2*delta(n-40) + 2*delta(n-70) + delta(n-80);
```

The corresponding output signal will have length $L_y = L + M = 121 + 24 = 145$, and can be generated by single call to conv:

```
conv(24, h, 121, x, y);
```

The output is shown below, together with the impulsive input.



Each impulse of the input generates a copy of the impulse response at the output. The outputs due to the first and second terms of Eq. (4.1.25) do not overlap, but the outputs of the last two terms do. □

4.1.10 *Overlap-Add Block Convolution Method*

In the above examples, the entire input signal was passed to conv as a *single* block of samples. This is not feasible in those applications where the input is infinite or extremely long. A practical approach is to divide the long input into *contiguous* non-overlapping blocks of manageable length, say L samples, then filter each block and piece the output blocks together to obtain the overall output, as shown in Fig. 4.1.6. Thus, processing is carried out block by block.

This is known as the *overlap-add* method of block convolution. Each of the input sub-blocks $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \dots,$ is convolved with the order-M filter $\mathbf{h}$ producing the outputs blocks:

$$\begin{aligned} \mathbf{y}_0 &= \mathbf{h} * \mathbf{x}_0 \\ \mathbf{y}_1 &= \mathbf{h} * \mathbf{x}_1 \\ \mathbf{y}_2 &= \mathbf{h} * \mathbf{x}_2 \end{aligned} \tag{4.1.26}$$

and so on. The resulting blocks are pieced together according to their absolute timing. Block $\mathbf{y}_0$ starts at absolute time $n = 0$; block $\mathbf{y}_1$ starts at $n = L$ because the corresponding input block $\mathbf{x}_1$ starts then; block $\mathbf{y}_2$ starts at $n = 2L$, and so forth.

Because each output block is longer than the corresponding input block by M samples, the *last* M samples of each output block will *overlap* with the first M outputs of the *next* block. Note that only the next sub-block will be involved if we assume that $2L > L + M$, or, $L > M$. To get the correct output points, the overlapped portions must be added together (hence the name, overlap-add).

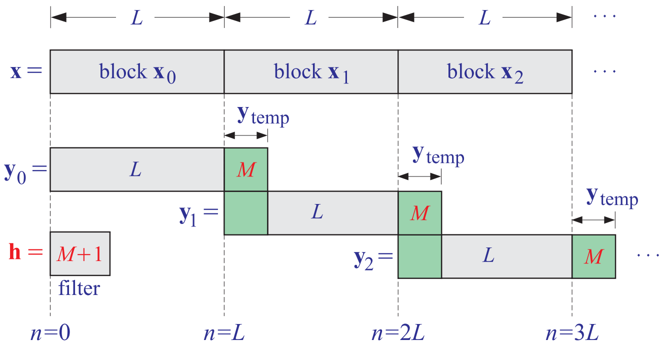


Fig. 4.1.6 Overlap-add block convolution method.

Example 4.1.10: Compute the output of Example 4.1.1 using the overlap-add method of block convolution. Use input blocks of length $L = 3$. Perform the required individual convolutions of Eq. (4.1.26) using the convolution table.

Solution: The input is divided into the following three contiguous blocks

$$\mathbf{x} = [\,\underbrace{1,1,2}_{\mathbf{x}_0},\underbrace{1,2,2}_{\mathbf{x}_1},\underbrace{1,1,0}_{\mathbf{x}_2}\,]$$

where we padded an extra zero at the end of $\mathbf{x}_2$ to get a length-3 block. Convolving each block separately with $\mathbf{h} = [1, 2, -1, 1]$ gives:

$$\begin{aligned} \mathbf{y}_0 &= \mathbf{h} * \mathbf{x}_0 = [1, 3, 3, 4, -1, 2] \\ \mathbf{y}_1 &= \mathbf{h} * \mathbf{x}_1 = [1, 4, 5, 3, 0, 2] \\ \mathbf{y}_2 &= \mathbf{h} * \mathbf{x}_2 = [1, 3, 1, 0, 1, 0] \end{aligned}$$

These convolutions can be done by separately folding the three convolution subtables:

	block 0			block 1			block 2		
h\x	1	1	2	1	2	2	1	1	0
1	1	1	2	1	2	2	1	1	0
2	2	2	4	2	4	4	2	2	0
-1	-1	-1	-2	-1	-2	-2	-1	-1	0
1	1	1	2	1	2	2	1	1	0

The three sub-blocks begin at the absolute times $n = 0, 3, 6$, respectively. It follows from time invariance that the corresponding output blocks will also begin at the same absolute times. Thus, aligning the output blocks according to their absolute timings and adding them up gives the final result:

n	0	1	2	3	4	5	6	7	8	9	10
$\mathbf{y}_0$	1	3	3	4	-1	2					
$\mathbf{y}_1$				1	4	5	3	0	2		
$\mathbf{y}_2$							1	3	1	0	1
$\mathbf{y}$	1	3	3	5	3	7	4	3	3	0	1

which agrees with Example 4.1.1. □

The method can be implemented by the following algorithm, which reads the input data in blocks $\mathbf{x}$ of length L and outputs the result also in blocks of length L:

```
for each length-$L$ input block $\mathbf{x}$ do:
    1. compute length-$(L+M)$ output:   $\mathbf{y} = \mathbf{h} * \mathbf{x}$
    2. for $i = 0, 1, \dots, M-1$:
           $y(i) = y(i) + y_{\text{temp}}(i)$      (overlap)
           $y_{\text{temp}}(i) = y(i+L)$           (save tail)
    3. for $i = 0, 1, \dots, L-1$:
           output $y(i)$
```

It uses a temporary M-dimensional vector $\mathbf{y}_{\text{temp}}$ to store the last M samples of each *previous* block. Before processing the first block, $\mathbf{y}_{\text{temp}}$ must be initialized to zero.

After computing the length-$(L+M)$ filter output $\mathbf{y} = \mathbf{h} * \mathbf{x}$, the first M samples of $\mathbf{y}$ are added to the last M samples of the previous block held in $\mathbf{y}_{\text{temp}}$. Then, the last M samples of the currently computed block $\mathbf{y}$ are saved in $\mathbf{y}_{\text{temp}}$ for use in the next iteration. Only the first L corrected output samples of $\mathbf{y}$ are sent to the output.

In practice this method is implemented efficiently by computing the individual block convolutions using the FFT instead of time-domain convolution. For an FIR filter of order M and an FFT of length N (which is a power of two), the length of each input block $\mathbf{x}$ is chosen to be $L = N - M$. We will see later that the computational gain of this *fast convolution* method versus the conventional time-domain "slow" method is approximately

$$\frac{\text{fast}}{\text{slow}} = \frac{\log_2 N}{M}$$

For example, for the values $N = 1024 = 2^{10}$ and $M = 100$, we have $\log_2 N/M = 10/100 = 1/10$, gaining a factor of 10 in computational speed. There is also an alternative fast convolution method called the *overlap-save* method that has comparable performance to the overlap-add method. We will also discuss it later.

The following routine `blockcon.c` is an implementation of the above algorithm. It calls the routine `conv` to perform the convolution of each input block. In using this routine, some care must be exercised in handling the very last input block, which in general will have length less than L.

```
/* blockcon.c - block convolution by overlap-add method */

void conv();

void blockcon(M, h, L, x, y, ytemp)
double *h, *x, *y, *ytemp;                    ytemp is tail of previous block
int M, L;                                     M = filter order, L = block size
{
    int i;

    conv(M, h, L, x, y);                      compute output block y

    for (i=0; i<M; i++) {
```

```
        y[i] += ytemp[i];                     add tail of previous block
        ytemp[i] = y[i+L];                    update tail for next call
        }
}
```

The quantities `h`, `x`, `y`, `ytemp` are arrays and must be allocated in the main program, for example, using `calloc`:

```
double *h, *x, *y, *ytemp;
h = (double *) calloc(M+1, sizeof(double));        (M+1)-dimensional
x = (double *) calloc(L,   sizeof(double));        L-dimensional
y = (double *) calloc(L+M, sizeof(double));        (L+M)-dimensional
ytemp = (double *) calloc(M, sizeof(double));      M-dimensional
```

To illustrate the usage of such a routine, suppose the input samples to be filtered have been stored sequentially[†] in a data file `x.dat`. Suppose also that the computed samples will be stored in the output file `y.dat`.

The following program segment keeps reading samples from the file `x.dat` in *blocks* of length L. For each such input block, it calls the routine `blockcon` to compute and save the output block in the file `y.dat`. When the end-of-file of `x.dat` is encountered, it determines the length of the last input block and calls `blockcon` one more time to process the last block.

```
for (;;) {                                             keep reading input blocks
       for (N=0; N<L; N++)
           if (fscanf(fpx, "%lf", x+N) == EOF) goto last;

       blockcon(M, h, L, x, y, ytemp);                 process input block

       for (i=0; i<L; i++)                             write output block
           fprintf(fpy, "%lf\n", y[i]);
       }
last:
       blockcon(M, h, N, x, y, ytemp);                 last block has N ≤ L

       for (i=0; i<N+M; i++)                           last output block
           fprintf(fpy, "%lf\n", y[i]);
```

Note that `x+N` stands for the address of `x[N]`, that is, `&x[N]`. The function `fscanf` returns EOF upon encountering the end of file `x.dat`. The last processed block has length $N \le L$. The entire last output block of length $(N+M)$ is written into the output file. The last M output samples represent the input-off transients. The file pointers, `fpx`, `fpy`, must be declared and defined in the main program by:

```
FILE *fpx, *fpy;                 file pointers
fpx = fopen("x.dat", "r");       open for read
fpy = fopen("y.dat", "w");       open for write
```

[†] Separated by white space, such as blanks, tabs, or newlines.

4.2 Numerical Evaluation of Continuous-Time Convolution

In continuous-time (CT), convolution integrals can be computed exactly only for simple functions $h(t)$ and $x(t)$ that lead to closed-form expressions for those integrals,

$$y(t)=\int_{-\infty}^{\infty}h(t')x(t-t')\,dt' = \int_{-\infty}^{\infty}h(t-t')x(t')\,dt' \tag{4.2.1}$$

For arbitrary signals, such integrals may be evaluated numerically. For example, a numerical approximation to the integral of the LTI form is obtained by considering the discrete time instants $t_n = nT$, where T is a small time step, and writing the integral as the limit of a sum:

$$y(t)=\int h(t-\tau)x(\tau)\,d\tau = \lim_{T\to 0}\Big[T\sum_m h(t-t_m)x(t_m)\Big] \tag{4.2.2}$$

where the extra factor T represents the $d\tau$ infinitesimal. Eq. (4.2.2) follows from the definition of integrals as limits. For small enough T, we may drop the limiting instruction and use the approximation,

$$y(t)=\int h(t-\tau)x(\tau)\,d\tau \approx T\sum_m h(t-t_m)x(t_m) \tag{4.2.3}$$

Other, more refined, approximations are possible, such as using the trapezoidal rule, however, Eq. (4.2.3) is adequate for our purposes. Replacing t by the sampled time instant, $t_n = nT$, we finally obtain,

$$\boxed{y(t_n)=\int h(t_n-\tau)x(\tau)\,d\tau \approx T\sum_m h(t_n-t_m)\,x(t_m)} \tag{4.2.4}$$

This can be implemented by the MATLAB code:

```
y = T * conv(h,x);                    (4.2.5)
```

where h_n, x_n are the arrays $h(t_n), x(t_n)$, and, of course, one must truncate $x(t_n), h(t_n)$ to finite-duration arrays for the purpose of computation.

The above numerical approximation method is equivalent to the so-called *impulse invariance* discretization method, discussed in Chap. 21.

4.2.1 Computer Experiment – Numerical Approximation

This example demonstrates the *linearity and time-invariance properties* of LTI systems and also looks at the issue of the *numerical approximation* of convolution, that is, (i) truncating the signals and (ii) replacing CT convolution by finite discrete-time (DT) convolution as in Eq. (4.2.5). We will observe that such approximation gets better as the sampling interval T gets smaller.

Consider a system described by the following differential equation, with corresponding impulse response and transfer function:

$$\frac{dy(t)}{dt}+ay(t)=ax(t) \quad\Rightarrow\quad h(t)=ae^{-at}u(t)\,,\quad H(s)=\frac{a}{s+a} \tag{4.2.6}$$

Let the input signal $x(t)$ be a square pulse of duration of t_d seconds, starting at $t=0$, that is,

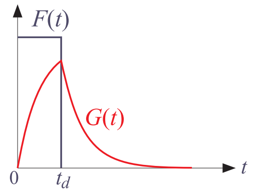
$$x(t)=F(t)\equiv u(t)-u(t-t_d)\,,\quad t_d>0 \tag{4.2.7}$$

The exact convolutional output due to the input $F(t)$ was found in Example 1 of this set:

$$y_{\text{exact}}(t)=G(t)\equiv e^{-at}\big[e^{a\min(t,t_d)}-1\big]u(t)=\begin{cases}1-e^{-at}\,, & 0\le t\le t_d\\ e^{-a(t-t_d)}\big[1-e^{-at_d}\big]\,, & t\ge t_d\\ 0\,, & t<0\end{cases} \tag{4.2.8}$$

These expressions define the functions $F(t), G(t)$ that are used later. They can be implemented in MATLAB as anonymous functions (assuming that t_d and a have already been defined previously),

```
% define td,a here
F = @(t) (t>=0) - (t>=td);
G = @(t) exp(-a*t) .* (exp(a*min(t,td)) - 1) .* (t>=0);
```



From the linearity and time-invariance of the system, we know that if one takes as input a linear combination of shifted copies of $F(t)$, then, the output would be the same linear combination of shifted copies of $G(t)$, for example,

$$\begin{aligned}x(t) &= c_1F(t-t_1)+c_2F(t-t_2)+c_3F(t-t_3)\\ y(t) &= c_1G(t-t_1)+c_2G(t-t_2)+c_3G(t-t_3)\end{aligned} \tag{4.2.9}$$

(a) Define the signals, $h(t), x(t)$, of Eqs. (4.2.6) and (4.2.9), for the following choice of parameters over a maximum time interval of $T_{\max}=25$, and pulse duration, $t_d=1$,

$$a=0.9\,,\quad T=0.05\,,\quad [c_1,c_2,c_3]=[1,\,2,\,1.5]\,,\quad [t_1,t_2,t_3]=[0,\,10,\,15]$$

with t defined to span the $[0, T_{\max}]$ interval in steps of T, that is, $t=0{:}T{:}T_{\max}$. Using the approximation of Eq. (4.2.5), calculate the output $y(t)$, as well as the exact output using Eq. (4.2.9), and plot both of them on the same graph, together with the input signal $x(t)$.

For plotting purposes you may wish to keep only the first, $N=\text{length}(t)$, convolutional outputs. This can be accomplished by redefining the computed output vector by:

```
    y = y(1:length(t));
```

(b) Observe in part (a) that because of the very short duration of the input pulses, the outputs due to the individual pulses can be clearly discerned, each being scaled and delayed by the proper amounts. If the pulse duration is increased, these outputs will begin to overlap.

Repeat part (a) for the pulse duration value, $t_d = 3$. The input pulses do not overlap, but the output ones overlap more strongly because the transient portions are closer to each other.

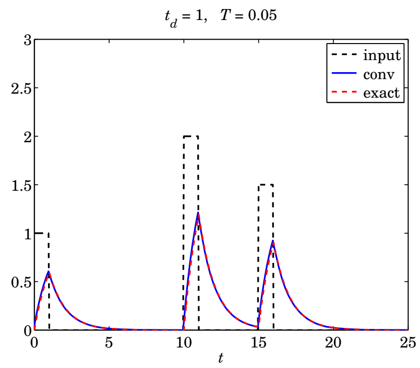
Repeat part (a) for the pulse duration value, $t_d = 5$, corresponding to the case when the last two pulses are just adjacent without overlap.

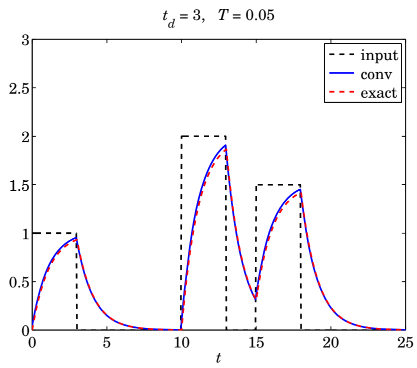
(c) To assess the nature of the approximation of Eq. (4.2.5), repeat parts (a,b) for the smaller value of the time step $T = 0.01$, and for the larger value $T = 0.1$.

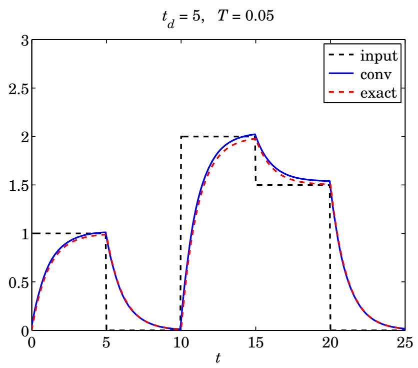
Solution

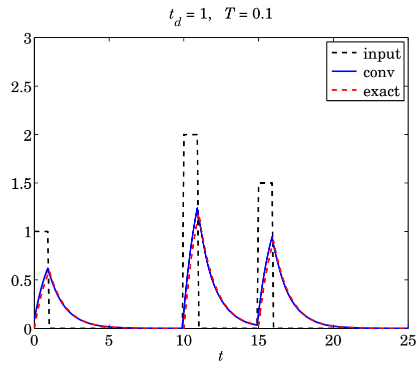
The typical MATLAB code for generating parts (a,b,c) is as follows, with some of the graphs shown at the end.

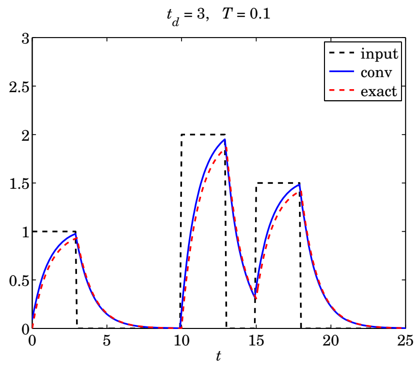
```
a = 0.9; td = 1;                 % run also with td=3 and td=5

F = @(t) (t>=0) - (t>=td);

G = @(t) exp(-a*t) .* (exp(a*min(t,td)) - 1) .* (t>=0);

t1 =  0; c1 = 1;                 % input parameters
t2 = 10; c2 = 2;
t3 = 15; c3 = 1.5;

Tmax = 25; T = 0.05;             % run also with T=0.01 and T=0.1

t = 0:T:Tmax;

h = a*exp(-a*t);                                % system

x  = c1*F(t-t1) + c2*F(t-t2) + c3*F(t-t3);      % input

y = T * conv(h,x);                              % conv output

y = y(1:length(t));                             % truncate y

ye = c1*G(t-t1) + c2*G(t-t2) + c3*G(t-t3);      % exact output

figure; plot(t,x,'k--', t,y,'b-', t,ye, 'r--');
xlabel('\itt');
```

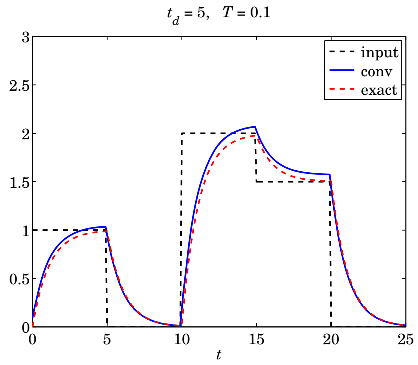

$t_d = 1,\ \ T = 0.05$


$t_d = 3,\ \ T = 0.05$


$t_d = 5,\ \ T = 0.05$


$t_d = 1,\ \ T = 0.1$


$t_d = 3,\ \ T = 0.1$



The graphs for the case $T = 0.01$ are not shown since the approximate outputs computed by Eq. (4.2.5) are virtually indistinguishable from the exact ones.

4.2.2 Computer Experiment – Transient and Steady-State Behavior

This example illustrates the *transient and steady-state* sinusoidal responses of linear systems, as well as the analytical and numerical computation of convolution. Consider a signal consisting of three consecutive sinusoidal bursts,

$$x(t) = \begin{cases} \sin(\omega_1 t), & 0 \le t < 30 \\ \sin(\omega_0 t), & 30 \le t < 70 \\ \sin(\omega_1 t), & 70 \le t < 100 \end{cases}$$

where $\omega_0 = 2$ and $\omega_1 = 3$. It can be generated over the time interval, $0 \le t \le 100$, by the following MATLAB code segment:

```
w0=2; w1=3; Tmax=100; T=Tmax/2000;    % T = 0.05 can be changed

t = 0:T:Tmax;

x = sin(w1*t) .* F(t,30) + ...
    sin(w0*t) .* F(t-30,40) + ...
    sin(w1*t) .* F(t-70,30);
```

where the pulse function $F(t, t_d)$ is defined as in the previous example, however, here way may think of it as function of two variables, t, t_d,

```
F = @(t,td) (t>=0) - (t>=td);
```

It is desired to eliminate the middle burst by means of a notch filter:

$$H(s) = \frac{s^2 + \omega_0^2}{s^2 + \alpha s + \omega_0^2} \tag{4.2.10}$$

where $\omega_0 = 2$ is the notch frequency coinciding with the frequency of the middle burst, and $\alpha = 0.3$ is a parameter that represents the 3-dB width of the notch, $\alpha = \Delta\omega$, (see graph below), thus, the filter Q is, $Q = \omega_0/\Delta\omega = \omega_0/\alpha$. It can be verified that the impulse response of this filter is:

$$h(t) = \delta(t) - g(t)\,, \quad g(t) = \alpha\, e^{-\alpha t/2}\Big[\cos(\omega_r t) - \frac{\alpha}{2\omega_r}\sin(\omega_r t)\Big]u(t)$$

$$\omega_r = \sqrt{\omega_0^2 - \frac{\alpha^2}{4}}$$

where it is recognized as an “underdamped” case since $\omega_0 > \alpha/2$.[†] It follows that the output signal will be given by the convolution integral (starting from $t = 0^-$ so that the $\delta(t)$ term will be included),

$$y(t) = \int_{0^-}^{\infty} h(t')\,x(t-t')\,dt' = \int_{0^-}^{\infty}\big[\delta(t') - g(t')\big]x(t-t')\,dt' = x(t) - \int_{0^-}^{\infty} g(t')\,x(t-t')\,dt'$$

which can be implemented approximately by the MATLAB code:

```
y = x - T * conv(g,x);                                    (4.2.11)
```

(a) Compute the output signal $y(t)$ using Eq. (4.2.11) and plot it versus t. On a separate graph, but using the same vertical and horizontal scales, plot the input signal $x(t)$. Note the removal of the middle burst after the transients have decayed.

The first and third bursts have also been attenuated by a slight amount, with a new amplitude equal approximately to $|H(\omega_1)|$ — this is only an approximation because steady-state is not yet reached for these bursts.

Calculate the numerical value of $|H(\omega_1)|$ and, on the graph for $y(t)$, add horizontal lines at that level for the first and third bursts (see the blue line segments in the example graphs below).

(b) Calculate the 40-dB time constant of this filter given in general by,

$$\tau = -\frac{\ln(10^{-40/20})}{|\,\mathrm{Re}(p)\,|} = \frac{\ln(100)}{|\,\mathrm{Re}(p)\,|} \tag{4.2.12}$$

where p is the stable pole closest to the $j\omega$ axis. Is the value of τ consistent with the transients that you observe in the plot of $y(t)$?

(c) Calculate the output signal $y(t)$ by the alternative method of using the function, **lsim**, and make a plot of $y(t)$ using the same scales as in part (a). Compare the outputs from the **lsim** and **conv** methods by computing the percentage error as the ratio,

$$\text{Error} = 100\cdot\frac{\|y_{\text{conv}} - y_{\text{lsim}}\|}{\|y_{\text{conv}}\|}$$

where the norm $\|y\|$ can be computed with the built-in function **norm**.

[†] the filter poles are at, $p = -\alpha/2 \pm j\omega_r$, in the left-hand s-plane

(d) The frequency and magnitude responses of the transfer function $H(s)$ of Eq. (4.2.10) are,

$$H(\omega)=\frac{\omega_0^2-\omega^2}{\omega_0^2-\omega^2+\alpha j\omega} \quad\Rightarrow\quad |H(\omega)|^2=\frac{(\omega^2-\omega_0^2)^2}{(\omega^2-\omega_0^2)^2+\alpha^2\omega^2} \tag{4.2.13}$$

The (positive) left and right 3-dB frequencies $\omega_\pm$, that is, the solutions of the 3-dB half-power condition, $|H(\omega)|^2 = 1/2$, are given by,

$$\omega_\pm = \pm\frac{\alpha}{2}+\sqrt{\omega_0^2+\frac{\alpha^2}{4}} \tag{4.2.14}$$

Note that these frequencies satisfy the following useful relationships,[†]

$$\boxed{\begin{aligned}\omega_+-\omega_- &= \alpha\\ \omega_+\omega_- &= \omega_0^2\end{aligned}} \tag{4.2.15}$$

where, $\Delta\omega = \omega_+ - \omega_- = \alpha$, represents the 3-dB width of the notch. Make a plot of the magnitude-square response $|H(\omega)|^2$ over the interval, $0 \le \omega \le 5$, and add to it (with dots) the points at $\omega = \omega_0$ (the notch), and at $\omega = \omega_1$. Also, add the horizontal line between the two 3-dB frequencies $\omega_\pm$ in order to indicate the width of the notch.

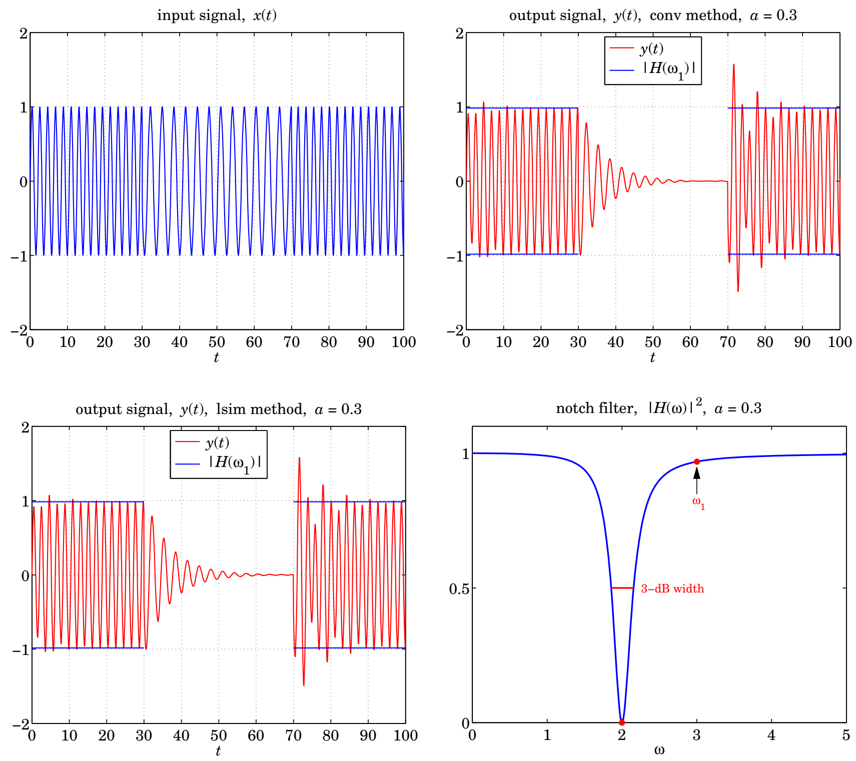
(e) Repeat parts (a–d) for the case $\alpha = 0.9$. Now the time constant and the transients will be shorter, but the notch width will be wider, and the first and last bursts at ω_1 will be more distorted in amplitude.

By combining Eqs. (4.2.12) and (4.2.15), show that the 40-dB time constant and the 3-dB width satisfy the following inverse relationship, which is a form of the uncertainty principle and captures the tradeoff between time constant and narrowness of the notch,

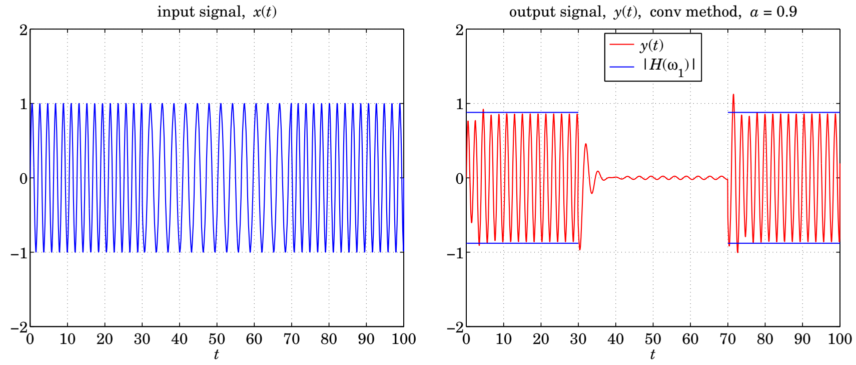
$$\tau = \frac{2\ln(100)}{\Delta\omega}$$

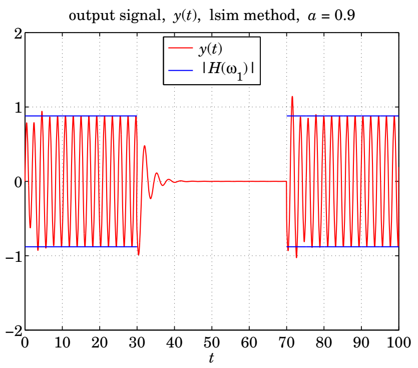
More generally, this tradeoff states that *the more stringent the filter specifications* in the frequency domain, *the longer* the filter time constants in the time domain.

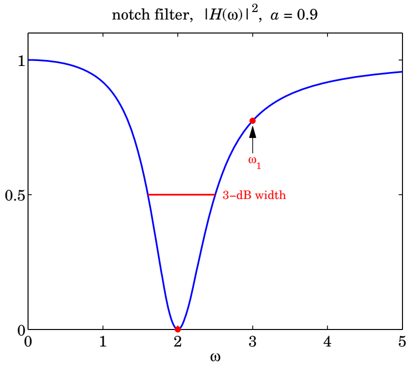
[†] First prove, $\omega_+^2\omega_-^2 = \omega_0^4$, then, use the identity, $(\omega_+-\omega_-)^2 = \omega_+^2+\omega_-^2-2\omega_+\omega_-$



The graphs for the case, $a = 0.9$, are shown below, where we again the basic tradeoff between time constant and narrowness of the notch is observed.







The typical MATLAB code for generating the above graphs is as follows:

```
F = @(t,td) (t>=0) - (t>=td);                       % pulse function

Tmax = 100; T = Tmax/2000;                          % try also, T=0.01

t = 0:T:Tmax;                                       % sampled in steps of T

w0 = 2; w1 = 3; a = 0.3; wr = sqrt(w0^2 - a^2/4);

tau = 2*log(100)/a;                                 % 40-dB time constant, tau=30.7

wa = -a/2 + sqrt(a^2/4 + w0^2);                     % left/right 3-dB frequencies
wb = +a/2 + sqrt(a^2/4 + w0^2);

x = sin(w1*t).*F(t,30) + sin(w0*t).*F(t-30,40) + sin(w1*t).*F(t-70,30);
g = a*exp(-a*t/2) .* (cos(wr*t) - a/2/wr * sin(wr*t));

yg = T*conv(g,x);  yg = yg(1:length(t));            % truncate to length(t)
y = x - yg;                                         % convolutional output

H1 = abs((w0^2 - w1^2)/(w0^2 - w1^2 + j*a*w1));     % |H(w1)| = 0.9686 when a=0.3
y1 = NaN(size(t)); y1(t<=30 | t>=70) = H1;          % constant envelope

s = tf('s');                                        % transfer function class
H = (s^2 + w0^2)/(s^2 + a*s + w0^2);
yf = lsim(H,x,t)';                                  % LSIM output

percent_error = 100 * norm(y-yf)/norm(y)            % Error = 0.9725 for a = 0.3
                                                    % redo with T=0.01 for improvement

figure; plot(t,x,'b-');                             % plot input
figure; plot(t,y,'r-',  t,y1,'b-', t,-y1,'b-');     % plot convolutional output
figure; plot(t,yf,'r-', t,y1,'b-', t,-y1,'b-');     % plot LSIM output
xlabel('\itt');

w = linspace(0,5,501); s = j*w;                     % frequency axis
H = abs((s.^2+w0^2)./(s.^2+a*s+w0^2)).^2;           % magnitude-response square |H(w)|^2

figure; plot(w,H,'b-', w0,0,'r.', w1,H1^2,'r.');
hold on; plot([wa,wb], [1,1]/2, 'r-')               % add 3-dB width
```

```
xlabel('\omega');
```

4.3 Sample Processing Methods

Convolution methods process the input signal on a *block-by-block* basis. Here, we discuss alternative formulations of FIR filters that operate on a *sample-by-sample* basis. As we mentioned earlier, such methods are convenient for *real-time applications* that require the continuous processing of the incoming input.

Sample processing algorithms are closely related to *block diagram* realizations of the I/O filtering equations. A block diagram is a *mechanization* of the I/O equation in terms of the three basic *building blocks*: adders, multipliers, and delays, shown in Fig. 4.3.1.

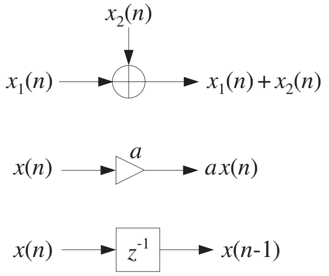


Fig. 4.3.1 Basic building blocks of DSP systems.

In general, a filter may have several *equivalent* block diagram realizations depending on how its I/O equation is organized. Each realization gives rise to its own sample processing algorithm. Some standard filter realizations are the *direct, canonical*, and *cascade* forms and the corresponding *transposed* versions. We will discuss them systematically in a later chapter. In this chapter, we consider only the *direct form* for FIR filters; its transpose is discussed in the Problems.

4.3.1 Pure Delays

As an introduction to the concept of a sample processing algorithm, consider the case of a single delay, shown in Fig. 4.3.2. It is an LTI system with I/O relationship:

$$y(n)=x(n-1)$$

It can be thought of as a *register* holding the *previous* input sample $x(n-1)$. At each time instant n, two steps must be carried out: (a) the current content $x(n-1)$ is clocked out to the output and (b) the current input $x(n)$ gets stored in the register, where it will be held for one sampling instant and become the output at the *next* time $n+1$.

We can think of the content of the delay register at time n as the *internal state* of the filter. Let us denote it by

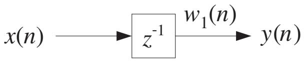


Fig. 4.3.2 Single delay.

$$w_1(n)= x(n-1) \qquad \text{(internal state at time } n\text{)}$$

Thus, the output is $y(n)= w_1(n)$. Replacing n by n+1, we obtain the content of the register at the next time instant,

$$w_1(n+1)= x(n) \qquad \text{(internal state at time } n\text{+1)}$$

The two processing steps (a) and (b) can be expressed then as follows:

$$\boxed{\begin{aligned} &y(n)= w_1(n) \\ &w_1(n+1)= x(n) \end{aligned}} \tag{4.3.1}$$

In words, at time n the content of the register $w_1(n)$ becomes the output and the input $x(n)$ is saved and becomes the new content. At time n+1, the two steps are repeated:

$$\begin{aligned} y(n+1)&= w_1(n+1) \\ w_1(n+2)&= x(n+1) \end{aligned}$$

The internal state $w_1(n+1)$ is available from the *previous* time step when it was saved; the current input $x(n+1)$ is also available and gets saved for the next time step. Before processing the first input sample, the delay register is typically *initialized* to zero, that is, at time $n=0$ it contains

$$w_1(0)= 0$$

The following table shows the values of the input $x(n)$, the internal state $w_1(n)$, and the output $y(n)$ at different time instants:

n	$x(n)$	$w_1(n)$	$y(n)$
0	x_0	0	0
1	x_1	x_0	x_0
2	x_2	x_1	x_1
3	x_3	x_2	x_2
4	x_4	x_3	x_3
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Thus, the input sequence gets delayed as a whole by one time unit:

$$[x_0, x_1, x_2, x_3, \dots\,] \xrightarrow{H} [0, x_0, x_1, x_2, x_3, \dots\,]$$

The two steps of Eq. (4.3.1), representing the output and the state updating, can be expressed in the following algorithmic form, which applies repetitively to every input sample:

```
for each input sample $x$ do:
    $y := w_1$
    $w_1 := x$
```

This is the sample-by-sample processing algorithm implementing a single delay. Consider next a double delay, depicted in Fig. 4.3.3. Its I/O equation is

$$y(n)= x(n-2)$$

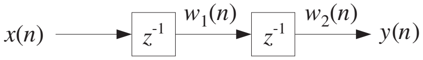


Fig. 4.3.3 Double delay.

Now, there are two registers holding the *previous two* input samples. Denoting the contents of the two registers by $w_1(n)$ and $w_2(n)$, we note that $w_1(n)$ is the delayed version of the input $x(n)$, and $w_2(n)$ the delayed version of $w_1(n)$:

$$\begin{aligned} w_2(n)&= w_1(n-1) \\ w_1(n)&= x(n-1) \end{aligned} \tag{4.3.2}$$

Therefore, $w_2(n)$ is the doubly delayed version of $x(n)$:

$$w_2(n)= w_1(n-1)= x\big((n-1)-1\big)= x(n-2)$$

At time n, $w_2(n)$ becomes the output, $y(n)= w_2(n)$, and the contents of the two registers are updated in preparation for the next time step, that is,

$$\begin{aligned} w_2(n+1)&= w_1(n) \\ w_1(n+1)&= x(n) \end{aligned}$$

In words, the *next* contents of the two registers are obtained by shifting w_1 into w_2, and x into w_1. In summary, the I/O equations describing the double delay are:

$$\boxed{\begin{aligned} &y(n)= w_2(n) \\ &w_2(n+1)= w_1(n) \\ &w_1(n+1)= x(n) \end{aligned}}$$

The repetitive sample processing algorithm describing these equations is:

```
for each input sample $x$ do:
    $y := w_2$
    $w_2 := w_1$
    $w_1 := x$
```

Note, that the *order of updating* the internal states is important: the *last* delay must always be updated first. Once the current value of w_1 has been shifted into w_2, the value of w_1 may be overwritten by x.

The following table shows the values of $x(n)$, the contents of the two registers $w_1(n)$, $w_2(n)$, and the output $y(n)$ at different times (with zero initial values $w_1(0)=w_2(0)=0$):

n	$x(n)$	$w_1(n)$	$w_2(n)$	$y(n)$
0	x_0	0	0	0
1	x_1	x_0	0	0
2	x_2	x_1	x_0	x_0
3	x_3	x_2	x_1	x_1
4	x_4	x_3	x_2	x_2
5	x_5	x_4	x_3	x_3
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

A triple delay, shown in Fig. 4.3.4, can be described in a similar fashion. Let $w_1(n)$, $w_2(n)$, and $w_3(n)$ denote the contents of the three registers at time n. They are successive delays of each other, that is,

$$\begin{aligned} w_3(n) &= w_2(n-1) \\ w_2(n) &= w_1(n-1) \\ w_1(n) &= x(n-1) \end{aligned}$$

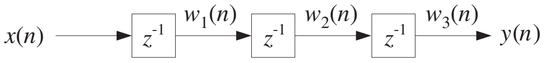


Fig. 4.3.4 Triple delay.

Thus, $w_3(n)=w_2(n-1)=w_1(n-2)=x(n-3)$. Their updates to time n+1 are:

$$\begin{aligned} w_3(n+1) &= w_2(n) \\ w_2(n+1) &= w_1(n) \\ w_1(n+1) &= x(n) \end{aligned} \tag{4.3.3}$$

Therefore, the I/O equations for a triple delay will be:

$$\boxed{\begin{aligned} y(n) &= w_3(n) \\ w_3(n+1) &= w_2(n) \\ w_2(n+1) &= w_1(n) \\ w_1(n+1) &= x(n) \end{aligned}}$$

And the corresponding sample processing algorithm:

```
for each input sample $x$ do:
    $y := w_3$
    $w_3 := w_2$
    $w_2 := w_1$
    $w_1 := x$
```

In general, for a delay by D units of time, shown in Fig. 4.3.5, the contents of the D registers are denoted by $w_i(n)$, $i=1,2,\dots,D$. For convenience, the input is denoted by $w_0(n)$. The output of each register is the delayed version of its input:

$$\boxed{w_i(n)=w_{i-1}(n-1), \qquad \text{for} \quad i=1,2,\dots,D} \tag{4.3.4}$$

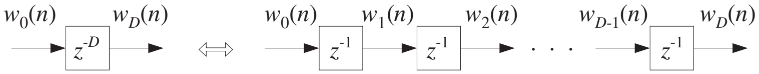


Fig. 4.3.5 D-unit delay.

At time n, the content of the Dth register is output, $y(n)=w_D(n)$. Then, in preparation for the next time step, the content of the w_{D-1} register is shifted into w_D, the content of w_{D-2} is shifted into w_{D-1}, and so on, and finally the current input w_0 is shifted into w_1. These updates may be expressed by replacing n by n+1 in Eq. (4.3.4) and reversing the order of the equations. The complete set of I/O equations describing a D-delay becomes:

$$\boxed{\begin{aligned} & y(n)=w_D(n) \\ & w_0(n)=x(n) \\ & w_i(n+1)=w_{i-1}(n), \qquad i=D,D{-}1,\dots,2,1 \end{aligned}}$$

The corresponding sample processing algorithm will be:

```
for each input sample $x$ do:
    $y := w_D$
    $w_0 := x$
    for $i = D, D-1, \dots, 1$ do:
        $w_i := w_{i-1}$
```

or, more simply:

```
for each input sample $w_0$ do:
    for $i = D, D-1, \dots, 1$ do:
        $w_i := w_{i-1}$
```

The following C routine `delay.c` is an implementation of this algorithm:

```
/* delay.c - delay by D time samples */

void delay(D, w)                                w[0] = input, w[D] = output
int D;
double *w;
{
       int i;

       for (i=D; i>=1; i--)                     reverse-order updating
              w[i] = w[i-1];

}
```

The array w has dimension $D+1$ and must be allocated in the main program by a statement of the form:

```
double *w;
w = (double *) calloc(D+1, sizeof(double));        (D+1)-dimensional
```

The array w serves both as input and output of the routine. Upon exit, w is the shifted version of itself. Prior to the first call of this routine, the array w must be initialized to zero. This is indirectly accomplished by calloc. The usage of the routine is illustrated by the following program segment, which implements the I/O equation $y(n)=x(n-D)$, for $n=0,1,\dots,N_{\text{tot}}-1$:

```
for (n = 0; n < Ntot; n++) {
       y[n] = w[D];                 write output
       w[0] = x[n];                 read input
       delay(D, w);                 update delay line
       }
```

We will use this routine to implement FIR and IIR filters and also, in cascaded and feedback arrangements, to implement several digital audio effects, such as digital reverb, and to implement periodic waveform generators.

When used in feedback configurations, we note that its output w[D] is available even *before* the input w[0]. This is illustrated to some degree by the above program segment, where the output is returned before the input is read into w[0]. However, the current input w[0] must be known before the delay line can be updated by the call to delay.

4.3.2 FIR Filtering in Direct Form

We saw in Eq. (4.1.23) that the direct form I/O convolutional equation for an FIR filter of order M is given by

$$y(n)=h_0x(n)+h_1x(n-1)+\cdots+h_Mx(n-M) \tag{4.3.5}$$

with impulse response $\mathbf{h}=[h_0,h_1,\dots,h_M]$. For example, a third-order filter

$$\mathbf{h}=[h_0,h_1,h_2,h_3]$$

will have I/O equation:

$$y(n)=h_0x(n)+h_1x(n-1)+h_2x(n-2)+h_3x(n-3) \tag{4.3.6}$$

In order to mechanize this equation, we need to use an *adder* to accumulate the sum of products in the right-hand side; we need *multipliers* to implement the multiplications by the filter weights; and, we need *delays* to implement the delayed terms $x(n-1)$, $x(n-2)$, $x(n-3)$.

Fig. 4.3.6 shows a mechanization of Eq. (4.3.6). It is called a *direct form* realization because it directly realizes all the terms in the right-hand side. The four inputs to the adder are the four terms of the right-hand side of Eq. (4.3.6), and the output of the adder is the left-hand side.

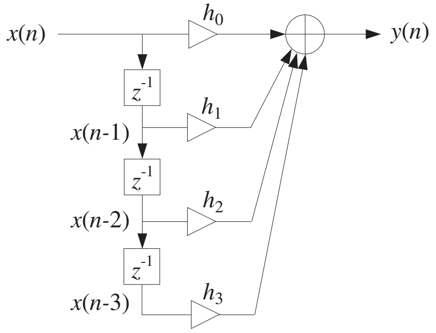


Fig. 4.3.6 Direct form realization of third-order filter.

The three delays are equivalent to the triple delay of Fig. 4.3.4, and therefore, we can introduce the same set of three *internal states* $w_1(n)$, $w_2(n)$, $w_3(n)$ to describe the contents of the three registers. Thus, we define:

$$\boxed{\begin{aligned} w_0(n)&=x(n)\\ w_1(n)&=x(n-1)=w_0(n-1)\\ w_2(n)&=x(n-2)=w_1(n-1)\\ w_3(n)&=x(n-3)=w_2(n-1)\end{aligned}} \tag{4.3.7}$$

so that each is a delayed version of the previous one. With these definitions, we can rewrite Eq. (4.3.6) in the form:

$$y(n)=h_0w_0(n)+h_1w_1(n)+h_2w_2(n)+h_3w_3(n) \tag{4.3.8}$$

Fig. 4.3.7 shows the realization in this case. The advantage of this equation is that all the terms in the right-hand side refer to the *same* time instant n. All are available for processing at time n; that is, $w_0(n)$ is the current input $x(n)$, and $w_i(n)$, $i=1,2,3$ are the current contents of the delay registers.

The set of delays is sometimes called a *tapped delay line* because the individual outputs of each delay are tapped out and diverted into the filter multipliers.

Once the current output is computed, the delay registers may be updated to hold the values that will be needed at the next time instant $n+1$. The updating is implemented

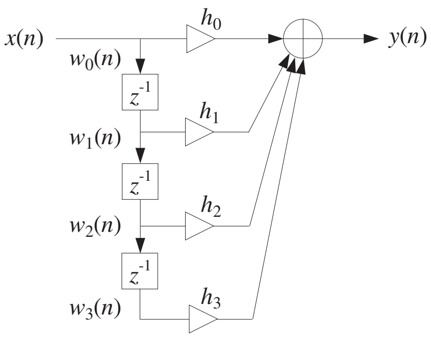


Fig. 4.3.7 Direct form with internal states.

via Eq. (4.3.4), that is, by shifting from the *bottom up*, w_2 into w_3, w_1 into w_2, and w_0 into w_1. Thus, the I/O equation (4.3.6) is equivalent to the system:

$$\boxed{\begin{aligned} &w_0(n)= x(n)\\ &y(n)= h_0w_0(n)+h_1w_1(n)+h_2w_2(n)+h_3w_3(n)\\ &w_3(n+1)= w_2(n)\\ &w_2(n+1)= w_1(n)\\ &w_1(n+1)= w_0(n)\end{aligned}} \tag{4.3.9}$$

It can be mechanized by the following sample-by-sample processing algorithm:[†]

```
for each input sample x do:
    $w_0 = x$
    $y = h_0w_0 + h_1w_1 + h_2w_2 + h_3w_3$
    $w_3 = w_2$
    $w_2 = w_1$
    $w_1 = w_0$
```
(4.3.10)

It is shown in Fig. 4.3.8. Thus, each input sample x is subjected to this algorithm and transformed to the output sample y. Before processing the first input sample, the internal states w_1, w_2, and w_3 must be initialized to *zero*.

The input-off and input-on transient behavior of an FIR filter can be understood in terms of the block diagram realization. Initially, the delay registers are cleared to zero. During the input-on transients, the three delays gradually fill up with input samples. It takes $M = 3$ time units for that to happen. Similarly when the input turns off, it takes the last input sample M time units to propagate through the delays, that is, it takes M time units for the delays to empty out their contents and be filled with zeros again.

[†] For notational simplicity, we used = instead of :=.

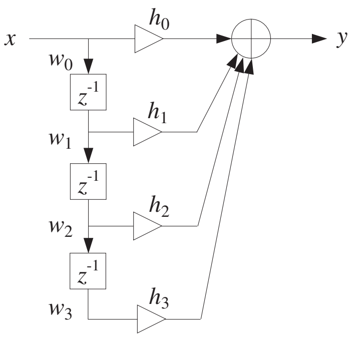


Fig. 4.3.8 Block diagram form of sample processing algorithm.

The following table shows the contents of the delays at different times and the corresponding outputs for the length-5 input given in Eq. (4.1.18):

n	x	w_0	w_1	w_2	w_3	$y = h_0w_0 + h_1w_1 + h_2w_2 + h_3w_3$
0	x_0	x_0	0	0	0	h_0x_0
1	x_1	x_1	x_0	0	0	$h_0x_1 + h_1x_0$
2	x_2	x_2	x_1	x_0	0	$h_0x_2 + h_1x_1 + h_2x_0$
3	x_3	x_3	x_2	x_1	x_0	$h_0x_3 + h_1x_2 + h_2x_1 + h_3x_0$
4	x_4	x_4	x_3	x_2	x_1	$h_0x_4 + h_1x_3 + h_2x_2 + h_3x_1$
5	0	0	x_4	x_3	x_2	$h_1x_4 + h_2x_3 + h_3x_2$
6	0	0	0	x_4	x_3	$h_2x_4 + h_3x_3$
7	0	0	0	0	x_4	h_3x_4

(4.3.11)

Each column of w's is the delayed (down-shifted) version of the previous one. Each row of w's is the delayed (right-shifted) version of the previous row. The computed outputs agree with Eq. (4.1.18). The three zeros padded at the end of the input correspond to the input-off transients. Note also that the four w columns are essentially the data matrix X of Eq. (4.1.21).

More generally, for an Mth order filter, we may define $w_0(n)= x(n)$ and for $i = 1,2,\dots,M$

$$w_i(n)= x(n-i) \tag{4.3.12}$$

They satisfy

$$w_i(n)= w_{i-1}(n-1), \qquad i = 1,2,\dots,M \tag{4.3.13}$$

Indeed, $w_{i-1}(n-1)= x\big((n-1)-(i-1)\big) = x(n-i)= w_i(n)$. Therefore, at the next time instant:

$$w_i(n+1)= w_{i-1}(n), \qquad i = 1,2,\dots,M$$

It follows that Eq. (4.3.5) can be written as

$$y(n)= h_0w_0(n)+h_1w_1(n)+\cdots+h_Mw_M(n)$$

Thus, the FIR filter Eq. (4.3.5) is described by the following system:

$$\boxed{\begin{aligned}
&w_0(n)=x(n)\\
&y(n)=h_0w_0(n)+h_1w_1(n)+\cdots+h_Mw_M(n)\\
&w_i(n+1)=w_{i-1}(n),\qquad\text{for}\quad i=M,M-1,\dots,1
\end{aligned}}\tag{4.3.14}$$

with corresponding sample processing algorithm:

```
for each input sample x do:
    $w_0 = x$
    $y = h_0w_0 + h_1w_1 + \cdots + h_Mw_M$
    for $i = M, M-1, \dots, 1$ do:
        $w_i = w_{i-1}$
```
(4.3.15)

Fig. 4.3.9 shows the corresponding direct form realization.

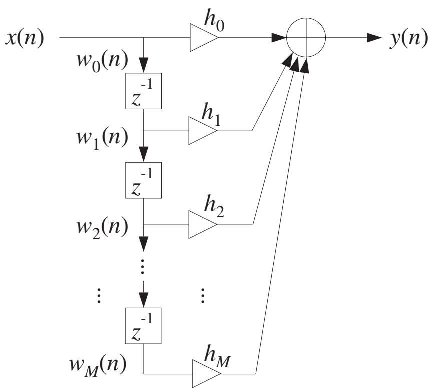


Fig. 4.3.9 Direct form realization of Mth order filter.

Example 4.3.1: Determine the sample processing algorithm of Example 4.1.1, which had filter and input

$$\mathbf{h}=[1,2,-1,1],\qquad \mathbf{x}=[1,1,2,1,2,2,1,1]$$

Then, using the algorithm compute the corresponding output, including the input-off transients.

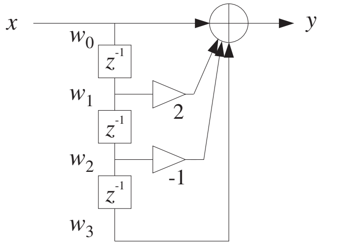
Solution: The I/O equation of this filter is

$$y(n)=x(n)+2x(n-1)-x(n-2)+x(n-3)$$

Introducing the internal states $w_i(n)=x(n-i)$, $i=1,2,3$, and setting $w_0(n)=x(n)$, we obtain the following system describing the output equation and the state updating:

$$\begin{aligned}
&w_0(n)=x(n)\\
&y(n)=w_0(n)+2w_1(n)-w_2(n)+w_3(n)\\
&w_3(n+1)=w_2(n)\\
&w_2(n+1)=w_1(n)\\
&w_1(n+1)=w_0(n)
\end{aligned}$$

The corresponding block diagram realization and sample processing algorithm are shown below:



```
for each input sample x do:
    $w_0 = x$
    $y = w_0 + 2w_1 - w_2 + w_3$
    $w_3 = w_2$
    $w_2 = w_1$
    $w_1 = w_0$
```

The sample processing algorithm generates the following output samples:

n	x	w_0	w_1	w_2	w_3	$y=w_0+2w_1-w_2+w_3$
0	1	1	0	0	0	1
1	1	1	1	0	0	3
2	2	2	1	1	0	3
3	1	1	2	1	1	5
4	2	2	1	2	1	3
5	2	2	2	1	2	7
6	1	1	2	2	1	4
7	1	1	1	2	2	3
8	0	0	1	1	2	3
9	0	0	0	1	1	0
10	0	0	0	0	1	1

The first three outputs correspond to the input-on transients (the internal delay registers are still filling up). The period $3\le n\le 7$ corresponds to steady state (all delays are filled). The last three outputs—in general, the last M outputs for an Mth order FIR filter—are the input-off ($x=0$) transients (the delays gradually empty out their contents). □

Example 4.3.2: To illustrate the repetitive nature of the sample processing algorithm, we present a small C program that implements the previous example.

```
/* firexmpl.c - Example of FIR sample processing algorithm */

#include <stdio.h>
#include <stdlib.h>                                        declares calloc
```

```
double x[8] = {1,1,2,1,2,2,1,1};                    input signal

double filter();

void main()
{
       int n;
       double y, *w;

       w = (double *) calloc(4, sizeof(double));    allocate/initialize w

       for (n=0; n<8; n++) {                    on-transients & steady state
              y = filter(x[n], w);              nth output sample
              printf("%lf\n", y);
              }

       for (n=8; n<11; n++) {                   input-off transients
              y = filter(0.0, w);               called with x = 0
              printf("%lf\n", y);
              }
}                                               end of main

double filter(x, w)                             Usage: y = filter(x, w);
double x, *w;
{
       double y;

       w[0] = x;                                read input sample

       y = w[0] + 2 * w[1] - w[2] + w[3];       compute output sample

       w[3] = w[2];                             update internal states
       w[2] = w[1];
       w[1] = w[0];

       return y;
}
```

The sample processing algorithm is implemented by the routine `filter` whose input is the current input sample and the internal states w. At each call, it returns the computed output sample and the updated state vector w. The routine `filter` is called 8 times (i.e., the length of the input) producing the first 8 outputs. Then, it is called 3 more times ($M = 3$), to generate the input-off transients. The total number of output samples is $L_y = L + M = 11$. □

Example 4.3.3: Draw the direct form realization and write the corresponding sample processing algorithm of the FIR filter defined by the I/O equation:

$$y(n)= x(n)-x(n-4)$$

For the input $\mathbf{x} = [1,1,2,1,2,2,1,1]$, compute the output using the sample processing algorithm.

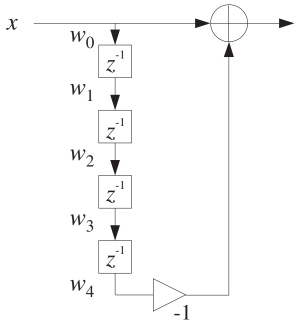
Solution: Because the filter has order $M = 4$, we define the following internal states:

$$\begin{aligned}
w_0(n) &= x(n)\\
w_1(n) &= x(n-1)= w_0(n-1)\\
w_2(n) &= x(n-2)= w_1(n-1)\\
w_3(n) &= x(n-3)= w_2(n-1)\\
w_4(n) &= x(n-4)= w_3(n-1)
\end{aligned}$$

Then, the given I/O equation together with the state-updating equations will read:

$$\begin{aligned}
w_0(n)&= x(n)\\
y(n)&= w_0(n)-w_4(n)
\end{aligned}
\quad\text{and}\quad
\begin{aligned}
w_4(n+1) &= w_3(n)\\
w_3(n+1) &= w_2(n)\\
w_2(n+1) &= w_1(n)\\
w_1(n+1) &= w_0(n)
\end{aligned}$$

This leads to the following sample processing algorithm and block diagram realization:



```
for each input sample x do:
    $w_0 = x$
    $y = w_0 - w_4$
    $w_4 = w_3$
    $w_3 = w_2$
    $w_2 = w_1$
    $w_1 = w_0$
```

The following table shows the computation of the output.

n	x	w_0	w_1	w_2	w_3	w_4	$y = w_0 - w_4$
0	1	1	0	0	0	0	1
1	1	1	1	0	0	0	1
2	2	2	1	1	0	0	2
3	1	1	2	1	1	0	1
4	2	2	1	2	1	1	1
5	2	2	2	1	2	1	1
6	1	1	2	2	1	2	−1
7	1	1	1	2	2	1	0
8	0	0	1	1	2	2	−2
9	0	0	0	1	1	2	−2
10	0	0	0	0	1	1	−1
11	0	0	0	0	0	1	−1

The 4 zeros padded at the end of the input correspond to the input-off transients, during which the contents of the delays gradually become empty. A similar program as the above `firexmpl.c` can be used to implement this example. The function `filter` will be in this case:

```
double filter(x, w)                         Usage: y = filter(x, w);
double x, *w;
{
       double y;

       w[0] = x;                            read input sample

       y = w[0] - w[4];                     compute output sample

       w[4] = w[3];                         update internal states
       w[3] = w[2];
       w[2] = w[1];
       w[1] = w[0];

       return y;
}
```

where `w` must be allocated as a 5-dimensional array in the main program. □

4.3.3 Programming Considerations

The following C routine `fir.c` is an implementation of the sample processing algorithm Eq. (4.3.15):

```
/* fir.c - FIR filter in direct form */

double fir(M, h, w, x)                      Usage: y = fir(M, h, w, x);
double *h, *w, x;                           h = filter, w = state, x = input sample
int M;                                      M = filter order
{
       int i;
       double y;                            output sample

       w[0] = x;                            read current input sample x

       for (y=0, i=0; i<=M; i++)
              y += h[i] * w[i];             compute current output sample y

       for (i=M; i>=1; i--)                 update states for next call
              w[i] = w[i-1];                done in reverse order

       return y;
}
```

It is the generalization of the example routine `filter` to the Mth order case. It is patterned after the Fortran and C routines in [45]. The routine returns the computed output sample into a `double`, so that its typical usage will be of the form:

```
y = fir(M, h, w, x);
```

The filter vector `h` and internal state `w` are $(M+1)$-dimensional arrays, which must be defined and allocated in the main program by the statements:

```
double *h, *w;
h = (double *) calloc(M+1, sizeof(double));         (M+1)-dimensional
w = (double *) calloc(M+1, sizeof(double));         (M+1)-dimensional
```

Note that `calloc` initializes `w` to zero. The following program segment illustrates the usage of the routine. The input samples are read one at a time from an input file `x.dat`, and the output samples are written one at a time into the output file `y.dat`, as they are computed.

```
FILE *fpx, *fpy;
fpx = fopen("x.dat", "r");                     input file
fpy = fopen("y.dat", "w");                     output file

while(fscanf(fpx, "%lf", &x) != EOF) {         read x from x.dat
       y = fir(M, h, w, x);                    process x to get y
       fprintf(fpy, "%lf\n", y);               write y into y.dat
       }

for (i=0; i<M; i++) {                          M input-off transients
       y = fir(M, h, w, 0.0);                  with x = 0
       fprintf(fpy, "%lf\n", y);
       }
```

Filtering stops as soon as the end of file of `x.dat` is detected and then the input-off transients are computed by making M additional calls to `fir` with zero input.

The `fir` routine performs three basic operations: (i) reading the current input sample, (ii) computing the current output by the *dot product* of the filter vector with the state vector, and (iii) updating the delay line containing the states. The dot product operation is defined by

$$y = h_0w_0 + h_1w_1 + \cdots + h_Mw_M = [h_0, h_1, \dots, h_M]\begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_M \end{bmatrix} = \mathbf{h}^T\mathbf{w}$$

and can be implemented by the following routine `dot.c`:

```
/* dot.c - dot product of two length-(M+1) vectors */

double dot(M, h, w)                            Usage: y = dot(M, h, w);
double *h, *w;                                 h = filter vector, w = state vector
int M;                                         M = filter order
{
       int i;
       double y;

       for (y=0, i=0; i<=M; i++)               compute dot product
              y += h[i] * w[i];

       return y;
}
```

The updating of the delay line can be implemented by the routine `delay` given earlier. Therefore, a second version of `fir`, which separates these three conceptual parts, is as follows:

```
/* fir2.c - FIR filter in direct form */

double dot();
void delay();

double fir2(M, h, w, x)                   Usage: y = fir2(M, h, w, x);
double *h, *w, x;                          h = filter, w = state, x = input
int M;                                     M = filter order
{
       double y;

       w[0] = x;                           read input

       y = dot(M, h, w);                   compute output

       delay(M, w);                        update states

       return y;
}
```

It has the same usage as `fir`. (See Appendix C for the MATLAB version `fir.m`.) The sample processing algorithm Eq. (4.3.15) reads in this case:

```
for each input sample x do:
    $w_0 = x$
    $y = \text{dot}(M, \mathbf{h}, \mathbf{w})$
    $\text{delay}(M, \mathbf{w})$
```
(4.3.16)

4.3.4 Hardware Realizations and Circular Buffers

The FIR filtering algorithms of Eqs. (4.3.15) or (4.3.16) can be realized in hardware using DSP chips or special purpose dedicated hardware.

Modern programmable DSP chips, such as the Texas Instruments TMS320C25, C30, or C50, the Motorola DSP56001, or DSP96002, the AT&T DSP16A or DSP32C, and the Analog Devices ADSP-2101 or ADSP-21020, have architectures that are *optimized* for the specialized repetitive nature of sample-by-sample processing algorithms. They excel at performing the multiplications and accumulations required in computing the dot product $y = \mathbf{h}^T\mathbf{w}$, and at performing the memory moves required in updating the contents of the delay-line registers.

A generic DSP chip is shown in Fig. 4.3.10. The tapped delay-line registers w_i are sequential RAM locations on board the chip and the filter weights h_i reside either in RAM or ROM. In addition, there is program RAM or ROM on board (not shown in the figure) to hold the instructions for the filtering algorithm. A typical DSP chip may have data wordlengths of 16–32 bits, several double-precision accumulators, and on-board RAM and ROM of 512 to 4k words.

The workhorse of the DSP chip is an on-board multiplier accumulator (MAC), which implements the dot product multiplications/accumulations, that is, the operations:

$$y := y + h_i w_i$$

State-of-the-art DSP chips can perform this type of MAC operation in one instruction cycle in about:

$$T_{\text{instr}} = 30\text{–}80 \text{ nanoseconds} \tag{4.3.17}$$

Assuming a MAC operation counts for two floating point operations (one multiplication and one addition), this corresponds to a numerical computation speed of 25–67 million floating point operations per second (MFLOPS).

For an order-M filter having $M+1$ taps, one would require about $(M+1)T_{\text{instr}}$ sec to calculate the required dot product. To this time, one must add the overhead required for shifting the input sample from the input port to the register w_0, the time required to update the delay-line registers, and the time it takes to send y to the output port.

The goal of modern DSP architectures has been to try to minimize this overhead as much as possible. To see the type of computational efficiencies built into DSP chips, let us rewrite the sample processing algorithm for a third-order filter given in Eq. (4.3.10) in the following equivalent form:

```
for each input sample x do:
    $w_0 := x$
    $y := h_3 w_3$
    $w_3 := w_2$
    $y := y + h_2 w_2$
    $w_2 := w_1$
    $y := y + h_1 w_1$
    $w_1 := w_0$
    $y := y + h_0 w_0$
```

This works because once the multiplication h_3w_3 is performed, the current content of w_3 is no longer needed and can be updated to the next time instant. Similarly, once h_2w_2 has been accumulated into y, w_2 may be updated, and so on. In general, for a filter of order M, we can rewrite Eq. (4.3.15) in the form:

```
for each input sample x do:
    $w_0 := x$
    $y := h_M w_M$
    for $i = M-1, \dots, 1, 0$ do:
        $w_{i+1} := w_i$
        $y := y + h_i w_i$
```
(4.3.18)

In earlier generations of DSP chips, the two operations:

$$w_{i+1} := w_i$$

$$y := y + h_i w_i$$

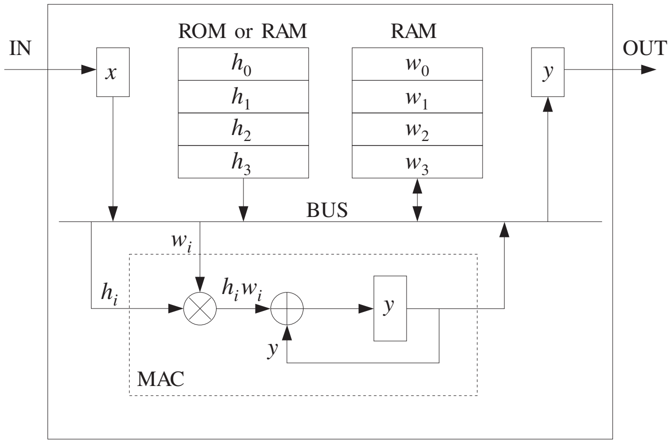


Fig. 4.3.10 Typical DSP chip.

were carried out with two instructions, one for the data shifting and the other for the MAC operation. In modern DSP chips, the two operations can be carried out with a *single* instruction, such as MACD of the TMS320C25.

Therefore, the total processing time for each input sample is about T_{instr} per filter tap, or, for an Mth order filter:

$$\boxed{T_{\text{proc}} = (M+1)T_{\text{instr}}} \tag{4.3.19}$$

As discussed in Chapter 1, this imposes a maximum limit on the allowed sampling rate for the application:

$$T \geq T_{\text{proc}} \quad \Rightarrow \quad f_s \leq \frac{1}{T_{\text{proc}}} \tag{4.3.20}$$

Example 4.3.4: What is the longest FIR filter that can be implemented with a 50 nsec per instruction DSP chip for digital audio applications?

Solution: We have from Eq. (4.3.19)

$$T = (M+1)T_{\text{instr}} \quad \Rightarrow \quad M+1 = \frac{T}{T_{\text{instr}}} = \frac{1}{f_s T_{\text{instr}}} = \frac{f_{\text{instr}}}{f_s}$$

where the *instruction rate* is $f_{\text{instr}} = 1/T_{\text{instr}} = 20$ million instructions per second (MIPS). For digital audio at $f_s = 44.1$ kHz, we find

$$M+1 = \frac{f_{\text{instr}}}{f_s} = \frac{20\cdot 10^6}{44.1\cdot 10^3} = 453 \text{ taps}$$

This filter length is quite sufficient to implement several digital audio algorithms. □

The following C routine `fir3.c` is yet a third version of the sample-by-sample processing algorithm implementing Eq. (4.3.18). Its usage is the same as `fir`'s:

```
/* fir3.c - FIR filter emulating a DSP chip */

double fir3(M, h, w, x)
double *h, *w, x;
int M;
{
       int i;
       double y;

       w[0] = x;                                    read input

       for (y=h[M]*w[M], i=M-1; i>=0; i--) {
              w[i+1] = w[i];                        data shift instruction
              y += h[i] * w[i];                     MAC instruction
              }

       return y;
}
```

The sample processing algorithm (4.3.18) and the routine `fir3` assume a *linear* delay-line memory buffer for holding the internal states $\{w_0, w_1, \dots, w_M\}$. At each time instant, the data in the delay line are shifted one memory location ahead. This arrangement is used in some DSP processors, such as the TMS32020.

An alternative way to update the internal states is to use a *circular delay-line buffer*. This is used, for example, by the Motorola DSP56001/96002, the Texas Instruments TMS320C30–C50, and the Analog Devices ADSP2101–21020 processors. Instead of shifting the data forward while holding the buffer addresses fixed, the data are kept fixed and the addresses are shifted backwards in the circular buffer. The relative movement of data versus addresses remains the same.

To understand this, consider first the conventional linear delay-line buffer case, but wrap it around in a circle, as shown in Fig. 4.3.11 for the case $M = 3$.

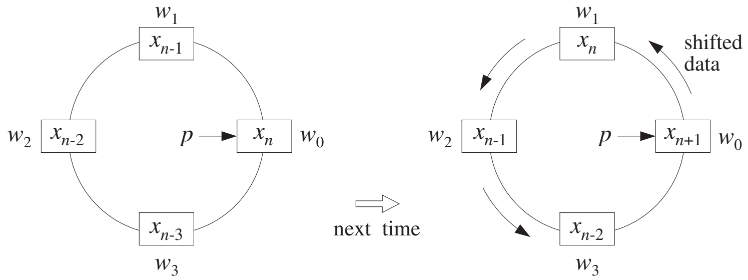


Fig. 4.3.11 Wrapped linear delay-line buffer.

Going from time n to $n+1$ involves shifting the content of each register counterclockwise into the next register. The addresses of the four registers $\{w_0, w_1, w_2, w_3\}$ remain the same, but now they hold the shifted data values, and the first register w_0 receives the next input sample x_{n+1}.

By contrast, in the circular buffer arrangement shown in Fig. 4.3.12, instead of shifting the data counterclockwise, the buffer addresses are decremented, or shifted clockwise once, so that w_3 becomes the new beginning of the circular buffer and will hold the next input x_{n+1}.

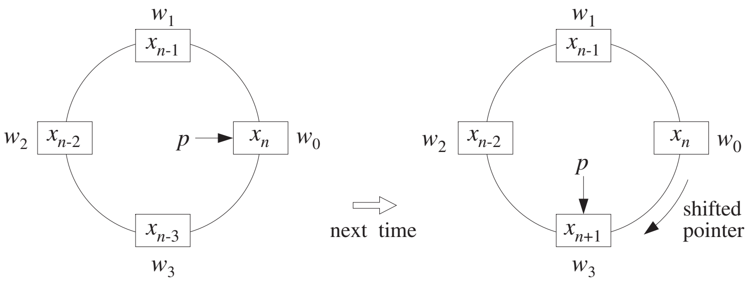


Fig. 4.3.12 Modulo-$(M+1)$ circular delay-line buffer.

The internal state vectors at times n and $n+1$ are the *same* in both the linear and circular buffer implementations, namely,

$$\mathbf{s}(n)=\begin{bmatrix} x_n \\ x_{n-1} \\ x_{n-2} \\ x_{n-3}\end{bmatrix}, \qquad \mathbf{s}(n+1)=\begin{bmatrix} x_{n+1} \\ x_n \\ x_{n-1} \\ x_{n-2}\end{bmatrix} \tag{4.3.21}$$

In both the linear and circular implementations, the starting address of the state vector is the current input sample, and from there, the addresses pointing to the rest of the state vector are incremented counterclockwise.

But, whereas in the linear case the starting address is always *fixed* and pointing to w_0, as shown in Fig. 4.3.11, the starting address in the circular case is back-shifted from one time instant to the next. To keep track of this changing address, we introduce a *pointer* variable p which always points to the current input, as shown in Fig. 4.3.12.

At each time instant, the w-register pointed to by p gets loaded with the current input sample, that is, $*p = x$, or $p[0] = x$. After computing the current output, the pointer p is decremented circularly. Figure 4.3.13 shows the position of the pointer p at successive sampling instants.

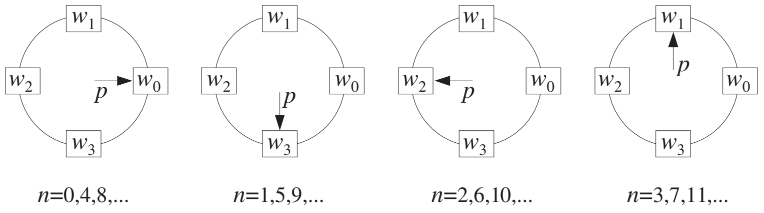


Fig. 4.3.13 Successive positions of address pointer p, repeating modulo-$(M+1)$.

The pointer p is restricted to lie within the pointer range of the linear buffer w, that is, in C notation:

$$w \le p \le w + M \tag{4.3.22}$$

and therefore, it will always point at some w-register, say $w[q]$,

$$p = w + q \quad \Rightarrow \quad *p = p[0] = w[q] \tag{4.3.23}$$

where q is an integer that gives the offset of p with respect to the fixed beginning of the w-buffer. The restriction of Eq. (4.3.22) translates to the restriction on the range of q:

$$0 \le q \le M \tag{4.3.24}$$

Inspecting the indices of the w's pointed to by the successive p's in Fig. 4.3.13, we may identify the periodically repeating sequence of values of q:

$$q = 0, 3, 2, 1, 0, 3, 2, 1, 0, 3, 2, 1, \dots$$

and in general, q cycles over the values: $q = 0, M, M-1, \dots, 1$.

At each time instant, the pointer p, or equivalently the offset q, defines the vector $\mathbf{s}$ of internal states. The sequence of pointers $p, p+1, \dots, p+M$, point at the components of the state vector, that is,

$$s_i = p[i] = *(p+i) = *(w+q+i) = w[q+i], \qquad i = 0, 1, \dots, M$$

This definition is correct as long as the shifted pointer $p+i$ does not exceed the array bounds of w, or equivalently, as long as $q+i \le M$. If $q+i > M$, it must be *reduced modulo-*$(M+1)$. Therefore, the correct definition of the internal state vector defined by the pointer p is, in C notation:

$$\boxed{s_i = w[\,(q+i)\%(M+1)\,] = w[\,(p-w+i)\%(M+1)\,]} \tag{4.3.25}$$

for $i = 0, 1, \dots, M$, where we solved Eq. (4.3.23) for $q = p - w$.[†] In particular, note that the first component of the state vector is:

$$s_0 = w[q] = p[0] = *p = *(w+q) \tag{4.3.26}$$

and the last component, corresponding to $i = M$:

$$s_M = w[\,(q+M)\%(M+1)\,] = w[\,(p-w+M)\%(M+1)\,] \tag{4.3.27}$$

Note that $s_M = w[M] = p[M]$ if $q = 0$, and $s_M = w[q-1] = p[-1]$ otherwise. Therefore, s_M is always in the w-register that circularly *precedes* $w[q]$.

Assuming that the current input sample x has been read into $s_0 = w[q] = *p$, that is, $*p = x$, the corresponding output sample y will be computed by the dot product:

$$y = \sum_{i=0}^{M} h_i s_i = [h_0, h_1, \dots, h_M]\begin{bmatrix} s_0 \\ s_1 \\ \vdots \\ s_M \end{bmatrix} = \mathbf{h}^T\mathbf{s}$$

[†] Some C compilers may require the cast: `q = (int) (p-w)`.

As an example illustrating Eq. (4.3.25), Fig. 4.3.14 shows p at the time when it points to w_2, so that $p = w+2$ and $q = 2$. We assume that w_2 has been loaded with the current input sample. In this case, the indices $q + i = 2 + i$ and their mod-4 reductions are, for $i = 0, 1, 2, 3$:

$$q + i = 2 + i = 2, 3, 4, 5 \quad \xrightarrow{\text{mod-4}} \quad 2, 3, 0, 1$$

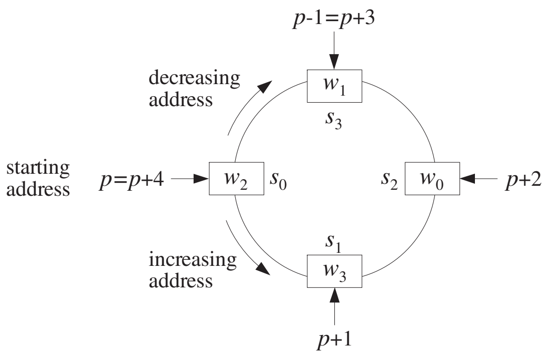


Fig. 4.3.14 Internal states defined by circular pointers $p + i$, $i = 0, 1, \dots, M$.

Therefore, the state vector defined by this value of p will be

$$\mathbf{s} = \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} w_2 \\ w_3 \\ w_0 \\ w_1 \end{bmatrix}$$

The following table shows the succession of “rotating” state vectors of Fig. 4.3.13 and their contents, as they fill up with input samples x_n at the successive time instants $n = 0, 1, \dots, 7$. An equivalent view is shown in Fig. 4.3.15. The w_i columns show the contents of the array $\mathbf{w} = [w_0, w_1, w_2, w_3]$ over which the pointer p circulates. At each time instant, *only one entry* in each row changes as it receives the new input sample, namely, the entry w_q. By contrast, in the linear buffer case given in Eq. (4.3.11), all entries of $\mathbf{w}$ shift. The first four s_i columns show the w-registers selected by p or q. The last four s_i columns show the actual contents of these w-registers:

n	q	w_0	w_1	w_2	w_3	s_0	s_1	s_2	s_3	s_0	s_1	s_2	s_3
0	0	x_0	0	0	0	w_0	w_1	w_2	w_3	x_0	0	0	0
1	3	x_0	0	0	x_1	w_3	w_0	w_1	w_2	x_1	x_0	0	0
2	2	x_0	0	x_2	x_1	w_2	w_3	w_0	w_1	x_2	x_1	x_0	0
3	1	x_0	x_3	x_2	x_1	w_1	w_2	w_3	w_0	x_3	x_2	x_1	x_0
4	0	x_4	x_3	x_2	x_1	w_0	w_1	w_2	w_3	x_4	x_3	x_2	x_1
5	3	x_4	x_3	x_2	x_5	w_3	w_0	w_1	w_2	x_5	x_4	x_3	x_2
6	2	x_4	x_3	x_6	x_5	w_2	w_3	w_0	w_1	x_6	x_5	x_4	x_3
7	1	x_4	x_7	x_6	x_5	w_1	w_2	w_3	w_0	x_7	x_6	x_5	x_4

(4.3.28)

It is evident that the contents of the columns s_0, s_1, s_2, s_3 are the successively delayed signal samples $x_n, x_{n-1}, x_{n-2}, x_{n-3}$. Therefore, the state vector $\mathbf{s}(n)$ at each time instant n is generated correctly according to Eq. (4.3.21).

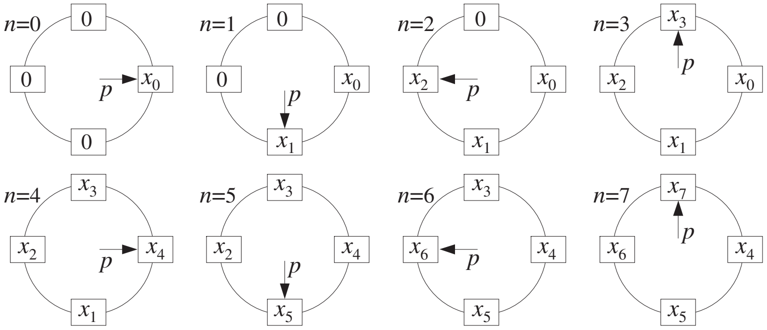


Fig. 4.3.15 Contents of circular buffer at successive time instants.

To make sure that p and its shifts $p + i$ always stay within the address space of w given by Eq. (4.3.22), they must be *wrapped* modulo-$(M+1)$. This is accomplished by the following routine `wrap.c`, which adjusts the bounds of p, such that if $p = w - 1$, it wraps it around to $p = (w - 1) + (M + 1) = w + M$, and if $p = w + M + 1$, it wraps it to $p = (w + M + 1) - (M + 1) = w$.

```
/* wrap.c - circular wrap of pointer p, relative to array w */

void wrap(M, w, p)
double *w, **p;
int M;
{
       if (*p > w + M)
              *p -= M + 1;           when ∗p = w + M + 1, it wraps around to ∗p = w

       if (*p < w)
              *p += M + 1;           when ∗p = w − 1, it wraps around to ∗p = w + M
}
```

Note that p is modified at each call and serves both as an input *and* output of the routine; therefore, it must be passed *by reference*, that is, as a pointer to pointer. If in the main program w and p are declared as ordinary pointers:

```
double *w, *p;
```

then p must be passed into `wrap` by its address, that is,

```
wrap(M, w, &p);
```

With the help of the routine `wrap`, an FIR filter can be implemented using a circular delay-line buffer by the following routine `cfir.c`, which replaces `fir`:

```
/* cfir.c - FIR filter implemented with circular delay-line buffer */

void wrap();

double cfir(M, h, w, p, x)
double *h, *w, **p, x;                                 p = circular pointer to w
int M;                                                 M = filter order
{
        int i;
        double y;

        **p = x;                                       read input sample x

        for (y=0, i=0; i<=M; i++) {                    compute output sample y
                y += (*h++) * (*(*p)++);
                wrap(M, w, p);
                }

        (*p)--;                                        update circular delay line
        wrap(M, w, p);

        return y;
}
```

The following three operations are carried out by the routine:

- The current input sample x is read into the w-register pointed to by the current value of the pointer p.
- The for-loop computes the filter's output sample y by accumulating the terms $h_i s_i$ of products of the filter coefficients with the components of the internal state vector defined by the pointer p.

  Each pass through the loop post-increments the h and p pointers and wraps p, if necessary. This loop could as well have been replaced by the following more obscure loop, which uses Eq. (4.3.25) for the states:

  ```
  for (y=0, i=0; i<=M; i++)
          y += h[i] * w[(*p-w+i)%(M+1)];          that is, y = y + h_i s_i
  ```

  Upon exit from the loop, the pointer p has been circularly incremented $M+1$ times and therefore, it has wrapped around to its original value, that is, pointing again at the current input sample.

  The filter pointer h is also incremented $M+1$ times and, after the loop, it points beyond its allowed range, but this does not matter because h will be reset at the next call of `cfir`. In hardware, h is also stored in a circular buffer, and therefore it wraps back to $h[0]$.
- Finally, the circular delay line is updated by simply decrementing the pointer p and wrapping it modulo $M+1$ if necessary. The pointer p is left pointing at the w-register containing the *last* component s_M of the state vector. This component will be overwritten by the next input sample.

In DSP chips that support circular or modulo addressing, each pass through the above for-loop requires only *one* instruction—the calls to `wrap` are not necessary because the incrementing pointer wraps around automatically. Therefore, the total number of instructions per call is essentially $M+1$. The total processing time per sample will be $T_{\text{proc}} = (M+1)T_{\text{instr}}$.

Each call of `cfir` changes the value of the pointer p, and therefore, p must be passed by reference, as in the routine `wrap`. The arrays h and w must be declared and allocated in the main program in the usual way, and w must be initialized to zero. The pointer p is initialized to point to $w[0]$, that is, $p = w$. The following program segment illustrates the proper initialization and usage of `cfir`.

```
double *h, *w, *p;
h = (double *) calloc(M+1, sizeof(double));
w = (double *) calloc(M+1, sizeof(double));         also, initializes w to zero

p = w;                                              initialize p

for (n = 0; n < Ntot; n++)
        y[n] = cfir(M, h, w, &p, x[n]);             p passed by address
```

The routine `cfir` imitates the hardware implementation of FIR filtering on the Motorola DSP56K chip, as illustrated in Example 4.3.5. A slightly different version of `cfir`, which essentially emulates the TMS320C30, is given below:

```
/* cfir1.c - FIR filter implemented with circular delay-line buffer */

void wrap();

double cfir1(M, h, w, p, x)
double *h, *w, **p, x;
int M;
{
        int i;
        double y;

        *(*p)-- = x;
        wrap(M, w, p);                                 p now points to s_M

        for (y=0, h+=M, i=M; i>=0; i--) {              h starts at h_M
                y += (*h--) * (*(*p)--);
                wrap(M, w, p);
                }

        return y;
}
```

After loading the current input sample into the w-register pointed to by p, it post-decrements p circularly, so that p becomes the wrapped $p+M$ and points to the last component s_M of the state vector.

The for-loop shifts the h pointer to point to h_M and then runs backwards from $i = M$ down to $i = 0$, accumulating the terms $h_i s_i$ and post-decrementing h and p at each pass.

Upon exit from the loop, p has wrapped around to point back to $p + M$ and is left pointing there upon exit from `cfir1`, but, that is where it should be pointing for processing the next input sample.

Example 4.3.5: It is beyond the scope of this book to discuss architectures and instruction sets for particular DSP chips. However, in order to illustrate the way `fir3`, `cfir`, and `cfir1` emulate assembly code for DSP chips, we present some code examples for the TMS32020, DSP32C, DSP56K, and TMS320C30; see Refs. [95–104] for details.

The following TMS32020 code segment implements the algorithm (4.3.18) for the case $M = 3$:

```
NEWX   IN W0, PA2             read new $x$ into $w_0$
       ZAC                    zero accumulator, $y = 0$

       LT  W3                 load $w_3$
       MPY H3                 multiply by $h_3$, $y = h_3 w_3$

       LTD W2                 $w_3 = w_2$, load and shift
       MPY H2                 $y = y + h_2 w_2$

       LTD W1                 $w_2 = w_1$
       MPY H1                 $y = y + h_1 w_1$

       LTD W0                 $w_1 = w_0$
       MPY H0                 $y = y + h_0 w_0$

       APAC                   accumulate final sum
       SACH Y, 1              store accumulator in register Y
       OUT Y, PA2             output Y from output port

       B NEWX                 branch back to NEWX to get next input sample
```

The shift/MAC pairs of instructions LTD/MPY can be replaced by single MACD instructions.

The same filter would be implemented on the AT&T DSP32C floating point processor by the program segment:

```
a1 = *r4++ * *r2++                        $y = h_3 w_3$
a1 = a1 + (*r3++ = *r4++) * *r2++         $y = y + h_2 w_2,\ w_3 = w_2$
a1 = a1 + (*r3++ = *r4++) * *r2++         $y = y + h_1 w_1,\ w_2 = w_1$
a0 = a1 + (*r3 = *r5) * *r2++             $y = y + h_0 w_0,\ w_1 = w_0$
*r6 = a0 = a0
```

The address pointer `r4` points to the internal states w_i and `r2` to the filter weights h_i. The first line puts the product $h_3 w_3$ into accumulator `a1` and increments the pointers to point to w_2, h_2. The second line accumulates the product $h_2 w_2$ into `a1` and simultaneously shifts the value of w_2 pointed to by `r4` into w_3 pointed to by `r3`, then post-increments the pointers.

In the fourth line, `r5` points to the input data, that is, $w_0 = x$. The sum of the previous value of `a1` and product $h_0 w_0$ are put into accumulator `a0` and simultaneously w_0 is moved into w_1 pointed to by `r3`. In the last line, `r6` points to the computed output sample y.

The following code implements the same filter on the Motorola DSP56K using a circular buffer. It is essentially equivalent to `cfir`.

```
clr    a          x0,x:(r0)+    y:(r4)+,y0
rep    #M
mac    x0,y0,a    x:(r0)+,x0    y:(r4)+,y0
macr   x0,y0,a    (r0)-
```

Here, the circular buffer resides in the chip's X-memory and is pointed to by the modulo pointer `r0`. The filter coefficients reside in Y-memory and are pointed to by the modulo pointer `r4`.

The `clr` instruction clears the accumulator register `a`, loads the temporary registers `x0`, `y0` with the values w_0 and h_0, and increments the pointers `r0`, `r4`.

The `rep` instruction repeats the `mac` instruction M times. During the ith repetition, the registers `x0`, `y0` hold the values of w_{i-1} and h_{i-1}; these values get multiplied and accumulated into the accumulator `a`, and then `x0`, `y0` are loaded with w_i, h_i, and the modulo pointers `r0`, `r4` are incremented.

Upon exit from this loop, the pointer `r0` has been incremented $M+1$ times, and therefore it has wrapped around to point to w_0 again. The last `macr` instruction performs the final accumulation of $w_M h_M$, and then it *decrements* the pointer `r0`, so that it now points to w_{-1} which is the same as w_M. The value in this register will be overwritten by the next input sample. The total number of instruction cycles for this example is $(M+1)+3$, instead of the nominal $M+1$.

The floating point TMS320C30/C40 processor implements circular addressing in a similar fashion, as illustrated by the following code:

```
NEWX   LDF    IN,  R3                read new input sample $x$
       STF    R3,  *AR1++%           put $x$ in $w_0$ and increment AR1
       LDF    0.0, R0                initialize accumulators
       LDF    0.0, R2

       RPTS   M                      repeat for $i$=$M$ down to $i$=0
       MPYF3  *AR0++%, *AR1++%, R0   $h_i w_i \to R_0$, and in parallel
    || ADDF3  R0, R2, R2             accumulate previous $R_0$ into $R_2$

       ADDF   R0, R2                 accumulate last product
       STF    R2, Y                  store $R_2$ into output register Y
       B      NEWX                   branch to NEWX and repeat
```

Here, the filter coefficients h_i and internal states w_i are stored in reverse order, so that h_M, w_M are in the lowest and h_0, w_0 at the highest address of the modulo-$(M+1)$ circular buffers. Therefore, pointers are incremented, instead of decremented as in `cfir1`.

Upon entry, the address pointer `AR0` points to the beginning of the h-buffer, that is, to h_M, and the pointer `AR1` points to the bottom of the w-buffer, that is, to w_0 which receives the current input sample x_n by the first `STF` instruction. The `AR1` pointer is then post-incremented and wraps around to point to the beginning of the w-buffer, that is, to w_M.

The `RPTS` loop repeats the following instruction $M+1$ times. The multiply instruction `MPYF3` and accumulate instruction `ADDF3` are done in parallel. The loop accumulates the terms $h_i w_i$, for $i = M, \dots, 1, 0$. Each repetition post-increments the pointers `AR0` and `AR1`. Therefore, after $M+1$ repetitions, `AR0` and `AR1` will wrap around and point to the beginning of the circular buffers.

Thus, `AR1` is circularly incremented a total of $(M+1)+1$ times and will be left pointing to w_M, which will receive the next input sample x_{n+1}, as shown in Fig. 4.3.12. □

The final part of `cfir`, which updates the circular delay line by modulo decrementing p, can be put by itself into a routine that implements the *circular* version of the delay routine `delay.c` of Section 4.3.1. Denoting M by D in this definition, we have:

```
/* cdelay.c - circular buffer implementation of D-fold delay */

void wrap();

void cdelay(D, w, p)
int D;
double *w, **p;
{
       (*p)--;                        decrement pointer and wrap modulo-(D + 1)
       wrap(D, w, p);                 when *p = w − 1, it wraps around to *p = w + D
}
```

Note that because p is decreasing, only the second half of wrap that tests the lower bound $w \le p$ is effective.

As in the case of the routine delay, the output of the delay line is available even before its input. This output is the last component of the internal state vector and is obtained from Eq. (4.3.27) with $M = D$:

$$s_D = w[(q+D)\%(D+1)] = w[(p-w+D)\%(D+1)]$$

Again, p must be passed by address into cdelay. The usage of the routine is illustrated by the following program segment, which implements the delay equation $y(n) = x(n-D)$:

```
p = w;                                    initialize p

for (n = 0; n < Ntot; n++) {
       y[n] = w[(p-w+D)%(D+1)];           write output
       *p = x[n];                         read input; equivalently, p[0]= x[n]
       cdelay(D, w, &p);                  update delay line
       }
```

The table in Eq. (4.3.28) illustrates this delay operation, with $D = 3$.

In the linear buffer implementations of fir and delay, the state vector is w itself, that is, $\mathbf{s} = \mathbf{w}$, and its components are directly accessible as $s_i = w[i]$, for $i = 0, 1, \dots, D$. In the circular buffer case, the state vector components are given by Eq. (4.3.25). To avoid having to write the complicated expressions of Eq. (4.3.25), we find it convenient to define a routine that returns the ith component s_i, or ith tap, of the circular tapped delay-line state vector:

```
/* tap.c - i-th tap of circular delay-line buffer */

double tap(D, w, p, i)                      usage: si = tap(D, w, p, i);
double *w, *p;                              p passed by value
int D, i;                                   i = 0, 1, . . . , D
{
       return w[(p - w + i) % (D + 1)];
}
```

Note that p is not changed by this routine, and therefore, it is passed by value. With the help of this routine, the above example of the D-fold delay would read as follows:

```
p = w;                                     initialize p

for (n = 0; n < Ntot; n++) {
       y[n] = tap(D, w, p, D);             Dth component of state vector
       *p = x[n];                          read input; equivalently, p[0]= x[n]
       cdelay(D, w, &p);                   update delay line
       }
```

The circular buffer implementation of a delay line is very efficient, consisting of just decrementing an address pointer *without* shifting any data (except for the input read into $p[0]$). It is especially useful in implementing digital audio effects, such as reverb, because D can be fairly large in these applications. For example, a 100 msec delay at 44.1 kHz sampling rate corresponds to $D = 100 \times 44.1 = 4410$ samples. It is also used in *wavetable sound synthesis*, where a stored waveform can be generated periodically by cycling over the circular buffer.

Because p is determined uniquely by the offset index q, via $p = w + q$, it is possible to rewrite all of the above routines so that they manipulate the index q instead of the pointer p. These versions can be translated easily into other languages, such as Fortran or MATLAB, that do not support pointer manipulation (see Appendix C).

The following routine wrap2.c replaces wrap. It simply keeps q within its allowed range, $0 \le q \le M$, by wrapping it modulo-$(M+1)$.

```
/* wrap2.c - circular wrap of pointer offset q, relative to array w */

void wrap2(M, q)
int M, *q;
{
       if (*q > M)
              *q -= M + 1;             when *q = M + 1, it wraps around to *q = 0

       if (*q < 0)
              *q += M + 1;             when *q = −1, it wraps around to *q = M
}
```

Because q is modified by wrap2, it must be passed by reference, that is, as a pointer to integer. The following routine cfir2.c replaces cfir. Note that the current input sample is placed in $p[0] = w[q]$, that is, $w[q] = x$.

```
/* cfir2.c - FIR filter implemented with circular delay-line buffer */

void wrap2();

double cfir2(M, h, w, q, x)
double *h, *w, x;                                q = circular offset index
int M, *q;                                       M = filter order
{
       int i;
       double y;

       w[*q] = x;                                read input sample x

       for (y=0, i=0; i<=M; i++) {               compute output sample y
              y += (*h++) * w[(*q)++];
              wrap2(M, q);
```

```
                }

        (*q)--;                                  update circular delay line
        wrap2(M, q);

        return y;
}
```

If so desired, the for-loop in this routine can be replaced by the following version, which accesses the *i*th state via Eq. (4.3.25):

```
for (y=0, i=0; i<=M; i++)
        y += h[i] * w[(*q+i)%(M+1)];             used by cfir2.m of Appendix C
```

The index q must be initialized to $q = 0$, which is equivalent to $p = w$. The usage of `cfir2` is illustrated by the following program segment:

```
double *h, *w;
int q;
h = (double *) calloc(M+1, sizeof(double));
w = (double *) calloc(M+1, sizeof(double));       also, initializes w to zero

q = 0;                                            initialize q

for (n = 0; n < Ntot; n++)
        y[n] = cfir2(M, h, w, &q, x[n]);          q passed by address
```

The implementation of the circular delay line is completely trivial. The following routine `cdelay2.c` replaces `cdelay`, and consists simply of decrementing q modulo $D+1$, assuming that the current input has been read into $w[q]$, namely, $w[q]= x$.

```
/* cdelay2.c - circular buffer implementation of D-fold delay */

void wrap2();

void cdelay2(D, q)
int D, *q;
{
        (*q)--;                       decrement offset and wrap modulo-(D + 1)
        wrap2(D, q);                  when *q = −1, it wraps around to *q = D
}
```

Its usage is illustrated by the following program segment. Note, again, that its output, namely, the Dth component of the internal state, is available even before its input:

```
q = 0;                                 initialize q

for (n = 0; n < Ntot; n++) {
        y[n] = w[(q+D)%(D+1)];         alternatively, y[n] = tap2(D, w, q, D);
        w[q] = x[n];                   read input
        cdelay2(D, &q);                update delay line
        }
```

Finally, the components of the internal state vector given by Eq. (4.3.25) are returned by the following routine `tap2.c` which replaces `tap`:

```
/* tap2.c - i-th tap of circular delay-line buffer */

double tap2(D, w, q, i)                   usage: si = tap2(D, w, q, i);
double *w;
int D, q, i;                              i = 0, 1, . . . , D
{
        return w[(q + i) % (D + 1)];
}
```

In summary, the circular buffer implementation of the FIR sample processing algorithm can be stated in the following form (initialized to $p = w$):

```
for each input sample x do:
    $s_0 = *p = x$
    for $i = 1, 2, \dots, M$ determine states:
        $s_i = \mathrm{tap}(M, \mathbf{w}, p, i)$
    $y = h_0 s_0 + h_1 s_1 + \cdots + h_M s_M$
    $\mathrm{cdelay}(M, \mathbf{w}, \&p)$
```

where for convenience, we used the routine `tap` to get the current states. In terms of the offset index q (initialized to $q = 0$):

```
for each input sample x do:
    $s_0 = w[q] = x$
    for $i = 1, 2, \dots, M$ determine states:
        $s_i = \mathrm{tap2}(M, \mathbf{w}, q, i)$
    $y = h_0 s_0 + h_1 s_1 + \cdots + h_M s_M$
    $\mathrm{cdelay2}(M, \&q)$
```

Example 4.3.6: The circular buffer implementation of Example 4.3.1 is as follows:

```
for each input sample x do:
    $s_0 = *p = x$
    $s_1 = \mathrm{tap}(3, \mathbf{w}, p, 1)$
    $s_2 = \mathrm{tap}(3, \mathbf{w}, p, 2)$
    $s_3 = \mathrm{tap}(3, \mathbf{w}, p, 3)$
    $y = s_0 + 2s_1 - s_2 + s_3$
    $\mathrm{cdelay}(3, \mathbf{w}, \&p)$
```

where w is to be declared as a 4-dimensional array and initialized to zero. In terms of the variable q, we have

```
for each input sample x do:
    $s_0 = w[q] = x$
    $s_1 = \mathrm{tap2}(3, \mathbf{w}, q, 1)$
    $s_2 = \mathrm{tap2}(3, \mathbf{w}, q, 2)$
    $s_3 = \mathrm{tap2}(3, \mathbf{w}, q, 3)$
    $y = s_0 + 2s_1 - s_2 + s_3$
    $\mathrm{cdelay2}(3, \&q)$
```

For the same input, the output signal samples and internal states computed by either of the above algorithms are exactly those given in the table of Example 4.3.1.

The linear buffer version discussed in Example 4.3.1 can be obtained from the above by freezing the pointer p to always point to w, that is, $p = w$. Then, we have for $i = 0, 1, 2, 3$:

$$s_i = \text{tap}(3, \mathbf{w}, p, i) = w[(p - w + i)\%4] = w[i\%4] = w_i$$

and the algorithm becomes the conventional one:

```
for each input sample x do:
    $w_0 = x$
    $y = w_0 + 2w_1 - w_2 + w_3$
    delay(3, w)
```

where `cdelay` was replaced by `delay`. □

4.4 Problems

4.1 Compute the convolution, $\mathbf{y} = \mathbf{h} * \mathbf{x}$, of the filter and input,

$$\mathbf{h} = [1, 1, 2, 1], \qquad \mathbf{x} = [1, 2, 1, 1, 2, 1, 1, 1]$$

using the following three methods: (a) The convolution table. (b) The LTI form of convolution, arranging the computations in a table form. (c) The overlap-add method of block convolution with length-3 input blocks. Repeat using length-5 input blocks.

4.2 Repeat Problem 4.1 for the filter and input:

$$\mathbf{h} = [2, -2, -1, 1], \qquad \mathbf{x} = [2, 2, 0, 1, -1, 0, 1, 2],$$

4.3 The impulse response $h(n)$ of a filter is nonzero over the index range $3 \le n \le 6$. The input signal $x(n)$ to this filter is nonzero over the index range $10 \le n \le 20$. Consider the direct and LTI forms of convolution:

$$y(n) = \sum_m h(m)x(n-m) = \sum_m x(m)h(n-m)$$

a. Determine the overall index range n for the output $y(n)$. For each n, determine the corresponding summation range over m, for both the direct and LTI forms.

b. Assume $h(n) = 1$ and $x(n) = 1$ over their respective index ranges. Calculate and sketch the output $y(n)$. Identify (with an explanation) the input on/off transient and steady state parts of $y(n)$.

4.4 An LTI filter has infinite impulse response $h(n) = a^n u(n)$, where $|a| < 1$. Using the convolution summation formula $y(n) = \sum_m h(m)x(n-m)$, derive closed-form expressions for the output signal $y(n)$ when the input is:

a. A unit step, $x(n) = u(n)$

b. An alternating step, $x(n) = (-1)^n u(n)$.

In each case, determine the steady state and transient response of the filter.

4.5 Consider the IIR filter $h(n) = a^n u(n)$, where $0 < a < 1$. The square pulse $x(n) = u(n) - u(n-L)$ of duration L is applied as input.

Using the time-domain convolution formula, determine a closed-form expression for the output signal $y(n)$ for the two time ranges: $0 \le n \le L - 1$ and $n \ge L$.

4.6 The filter of Problem 4.5 satisfies the difference equation: $y(n) = ay(n-1) + x(n)$. Verify that the solution $y(n)$ that you obtained above satisfies this difference equation for all n.

4.7 *Computer Experiment: Convolution Routines.* Write C or MATLAB routines that implement convolution in: (a) the convolution table form and (b) the LTI form; that is,

$$y_n = \sum_{\substack{i,j \\ i+j=n}} h_i x_j = \sum_m x_m h_{n-m}$$

The routines must have the same input/output variables as `conv.c` of the text. Write a small main program that tests your routines.

4.8 *Computer Experiment: Filtering by Convolution.* Write small C or MATLAB routines to reproduce all the results and graphs of Examples 4.1.7, 4.1.8, and 4.1.9. The inputs must be treated as single blocks and passed into the routine `conv`.

4.9 *Computer Experiment: Block-by-Block Processing.* Write a stand-alone C program, say `blkfilt.c`, that implements the overlap-add block convolution method. The program must have usage:

```
blkfilt h.dat L  < x.dat  > y.dat
```

It must have as command-line inputs a file of impulse response coefficients `h.dat` (stored one coefficient per line) and the desired input block length L. It must read the input signal samples from `stdin` or a file `x.dat` and write the computed output samples to `stdout` or a file `y.dat`. It must have the following features built-in: (a) it must allocate storage dynamically for the impulse $h(n)$ read from `h.dat` (the program should abort with an error message if $L < M$); (b) it must read the input signal in length-L blocks, and call the routine `blockcon.c` of the text to process each block; (c) it must write the output also in blocks; (d) it must compute correctly both the input-on and input-off transients.

Note that the essence of such a program was already given in Section 4.1.10. Test your program on Example 4.1.10, with blocksizes $L = 3, 4, 5, 6$.

4.10 *Computer Experiment: Sample-by-Sample Processing.* Write a stand-alone C program, say `firfilt.c`, that implements the FIR sample processing algorithm of Eq. (4.3.15). The program must have usage:

```
firfilt h.dat  < x.dat  > y.dat
```

It must read and dynamically allocate the impulse response vector $\mathbf{h}$ from an input file, say `h.dat`, and must allocate the internal state vector $\mathbf{w}$. Using the routine `fir.c`, it must keep processing input samples, reading them one at a time from `stdin` or from a file `x.dat`, and writing the computed output samples to `stdout` or a file `y.dat`.

It must correctly account for the input-on and input-off transients. Thus, it must produce identical results with convolution or block convolution if applied to a finite input block of samples.

The essence of such a program was given in Section 4.3.3. Test your program on Examples 4.1.7–4.1.10.

Such filters can be cascaded together by piping the output of one into the input to another. For example, the filtering operation by the combined filter $\mathbf{h} = \mathbf{h}_1 * \mathbf{h}_2$ can be implemented by:

```
firfilt h1.dat | firfilt h2.dat  < x.dat  > y.dat
```

Alternatively or additionally, write a MATLAB version, say `firfilt.m`, with usage:

```
y = firfilt(h, x);
```

It must read the filter and input vectors **h**, **x**, and compute the output vector **y**. The input-off transients must also be part of the output vector. You may use the MATLAB functions `delay.m` and `fir.m` of Appendix C.

4.11 *Computer Experiment: Sample Processing with Circular Buffers.* Rewrite the above program, say `cfirfilt.c`, such that the basic sample-by-sample filtering operation is implemented by the circular buffer routine `cfir.c` instead of `fir.c`. The usage of the program will be the same as above:

```
cfirfilt h.dat  < x.dat  > y.dat
```

Test your program on Examples 4.1.7–4.1.10. It must produce identical results as the `firfilt` program. Rewrite versions of this program that use the alternative circular FIR routines `cfir1.c` and `cfir2.c`. You may also write a MATLAB version using `cfir2.m`.

4.12 *Computer Experiment: Delay Using Circular Buffers.* Write a stand-alone C program, say `cdelfilt.c`, that implements a plain delay by up to D samples, that is, $y(n)= x(n-i)$, $i = 0, 1, \dots, D$, and has usage:

```
cdelfilt i D  < x.dat  > y.dat
```

It must read the input samples one at a time from `stdin` and write the delayed samples into `stdout`. It must make use of the circular buffer routines `cdelay.c` and `tap.c`. The delay-line buffer must have length $D+1$.

Test your program on a length-20 input using the values $D = 5$, and $i = 0, 1, \dots, 5$. Then, write another version of this program that uses the routines `cdelay2` and `tap2` and test it.

4.13 *Computer Experiment: Alternative Sample Processing Algorithms.* The FIR sample processing algorithm of Eq. (4.3.15) proceeds by (a) reading the current input, (b) processing it, and (c) updating the delay line.

In some texts, the algorithm is structured in a slightly different way, such that the update of the delay line is done *before* the current input is read. Derive the difference equations describing this version of the algorithm. [*Hint:* Use Eq. (4.3.13).]

Translate the difference equations into a sample processing algorithm like Eq. (4.3.15) and then, write a C routine that implements it. Discuss the proper initialization of the internal states in this case. Test your routine to make sure it produces identical results with `fir.c`.

4.14 Consider the filter and input of Problem 4.1. Draw a block diagram realization of the filter, introduce internal states, write the corresponding sample processing algorithm, and convert it into a C routine.

Then, using the sample processing algorithm, compute the full output signal, including the input-off transients (for these, apply $x = 0$ to the algorithm). Display your computations in a table form. Identify on the table which outputs correspond to the input-on transients, to the steady state, and to the input-off transients. [*Note*: The computed output signal should agree exactly with that computed by the convolution method of Problem 4.1.]

4.15 Consider the filter with I/O equation: $y(n)= x(n)-x(n-3)$.

a. Determine the impulse response sequence $h(n)$, for all $n \geq 0$.

b. Draw a *block diagram* realization, introduce appropriate *internal states*, and write the corresponding *sample processing* algorithm. Moreover, implement the algorithm by a C routine. Test your routine on the results of parts (c) and (d).

c. Send as input the sequence $\mathbf{x} = [1, 1, 2, 2, 4, \dots]$. Using *convolution*, compute the first five output samples, $y(n)$, $n = 0, 1, \dots, 4$.

d. Compute the same outputs using the *sample processing* algorithm. Display your computations in a table that, at each sampling instant n, shows the corresponding input sample $x(n)$, the values of the internal states, and the computed output sample $y(n)$.

4.16 Repeat Problem 4.15 for the filter: $y(n)= 0.8y(n-1)+x(n)$.

4.17 Repeat Problem 4.15 for the filter: $y(n)= 0.25y(n-2)+x(n)$.

4.18 Let $\mathbf{x} = [1, 1, 2, 2, 2, 2, 1, 1]$ be an input to the filter described by the I/O equation:

$$y(n)= x(n)-x(n-2)+2x(n-3)$$

a. Determine the impulse response $h(n)$ of this filter.

b. Compute the corresponding output signal $y(n)$ using the *LTI form* of convolution. Show your computations in table form.

c. Compute the same output using the overlap-add method of block convolution by partitioning the input signal into length-4 blocks.

d. Draw a block diagram realization of this filter. Then, introduce appropriate internal states and write the corresponding sample processing algorithm.

4.19 The length-8 input signal $x(n)= \{8, 7, 6, 5, 4, 3, 2, 1\}$ is applied to the input of a 2-fold delay described by the I/O equation $y(n)= x(n-2)$.

The circular buffer version of the sample processing implementation of the delay operation requires the use of a 3-dimensional *linear buffer array* of internal states $\mathbf{w} = [w_0, w_1, w_2]$ and a pointer p circulating over $\mathbf{w}$.

Make a table of the numerical values of the contents of the array $\mathbf{w}$ for the successive time instants $0 \leq n \leq 10$. In each row, indicate that array element, w_i, which represents the current output $y(n)$ of the delay line. Explain your reasoning in filling this table. Compare with the linear buffer case.

4.20 Figure 4.4.1 shows the transposed realization of the third-order filter with impulse response $\mathbf{h} = [h_0, h_1, h_2, h_3]$. Write the difference equations describing the I/O operation of this realization. Show that the combined effect of these difference equations is equivalent to the standard direct form operation of Eq. (4.3.6).

Write the transposed difference equations as a sample processing algorithm and apply it to the filter and input of Example 4.3.1. Make a table of the values of x, y, and all the variables v_i, for each time instant $0 \leq n \leq 10$.

4.21 *Computer Experiment: Transposed FIR Realization.* Generalize the block diagram of Fig. 4.4.1 to an arbitrary Mth order filter with impulse response $\mathbf{h} = [h_0, h_1, \dots, h_M]$. Write the corresponding sample processing algorithm and translate it into a C routine `firtr.c` that has usage:

```
y = firtr(M, h, v, x);
```

where **h** and **v** are $(M+1)$-dimensional vectors, and x, y are the current input and output samples.

To test your routine, write a version of the program `firfilt.c` of Problem 4.10 that uses the routine `firtr.c` instead of `fir.c`. Test your program on Example 4.3.1.

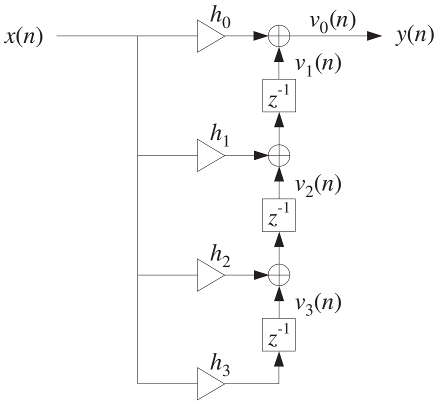


Fig. 4.4.1 Transposed realization of third-order filter.

5

z-Transforms

5.1 Basic Properties

Here, we review briefly z-transforms and their properties. We assume that the reader already has a basic familiarity with the subject. Our usage of z-transforms in this book is basically as a *tool* for the analysis, design, and implementation of digital filters.

Given a discrete-time signal $x(n)$, its z-transform is defined as the following series:

$$\boxed{X(z)=\sum_{n=-\infty}^{\infty}x(n)z^{-n}} \qquad \text{(z-transform)} \tag{5.1.1}$$

or, writing explicitly a few of the terms:

$$X(z)=\cdots+x(-2)z^2+x(-1)z+x(0)+x(1)z^{-1}+x(2)z^{-2}+\cdots$$

There are as many terms as nonzero signal values $x(n)$. The terms z^{-n} can be thought of as place holders for the values $x(n)$. If the signal $x(n)$ is *causal*, only *negative* powers z^{-n}, $n\geq 0$ appear in the expansion. If $x(n)$ is strictly *anticausal*, being nonzero for $n\leq -1$, only *positive* powers will appear in the expansion, that is, $z^{-n}=z^{|n|}$, for $n\leq -1$. And if $x(n)$ is mixed with both causal and anticausal parts, then both negative and positive powers of z will appear.

The definition (5.1.1) can also be applied to the impulse response sequence $h(n)$ of a digital filter. The z-transform of $h(n)$ is called the *transfer function* of the filter and is defined by:

$$\boxed{H(z)=\sum_{n=-\infty}^{\infty}h(n)z^{-n}} \qquad \text{(transfer function)} \tag{5.1.2}$$

Example 5.1.1: Determine the transfer function $H(z)$ of the two causal filters of Example 3.4.3, namely,

(a) $\mathbf{h}=[h_0,h_1,h_2,h_3]=[2,3,5,2]$

(b) $\mathbf{h}=[h_0,h_1,h_2,h_3,h_4]=[1,0,0,0,-1]$

Solution: Using the definition (5.1.2), we find:

$$H(z)=h_0+h_1z^{-1}+h_2z^{-2}+h_3z^{-3}=2+3z^{-1}+5z^{-2}+2z^{-3}$$

for case (a), and

$$H(z)=h_0+h_1z^{-1}+h_2z^{-2}+h_3z^{-3}+h_4z^{-4}=1-z^{-4}$$

for case (b). □

The three most important properties of z-transforms that facilitate the analysis and synthesis of linear systems are:

- linearity property
- delay property
- convolution property

The linearity property simply states that the z-transform of a linear combination of signals is equal to the linear combination of z-transforms, that is, if $X_1(z)$ and $X_2(z)$ are the z transforms of the signals $x_1(n)$ and $x_2(n)$, then the z-transform of the linear combination $a_1x_1(n)+a_2x_2(n)$ is

$$\boxed{a_1x_1(n)+a_2x_2(n)\xrightarrow{z}a_1X_1(z)+a_2X_2(z)} \qquad \text{(linearity)} \qquad (5.1.3)$$

The delay property states that the effect of delaying a signal by D sampling units is equivalent to multiplying its z-transform by a factor z^{-D}, namely,

$$\boxed{x(n)\xrightarrow{z}X(z) \quad\Rightarrow\quad x(n-D)\xrightarrow{z}z^{-D}X(z)} \qquad \text{(delay)} \qquad (5.1.4)$$

Note that D can also be negative, representing a time advance. Finally, the convolution property states that convolution in the time domain becomes multiplication in the z-domain:

$$\boxed{y(n)=h(n)*x(n) \quad\Rightarrow\quad Y(z)=H(z)X(z)} \qquad \text{(convolution)} \qquad (5.1.5)$$

that is, the z-transform of the convolution of two sequences is equal to the product of the z-transforms of the sequences.

Example 5.1.2: The two filters of the above example and of Example 3.4.3 can also be written in the following "closed" forms, valid for all n:

$$\text{(a)}\quad h(n)=2\delta(n)+3\delta(n-1)+5\delta(n-2)+2\delta(n-3)\,, \qquad \text{(b)}\quad h(n)=\delta(n)-\delta(n-4)$$

Their transfer functions can be obtained using the linearity and delay properties as follows. First, note that the z-transform of $\delta(n)$ is unity:

$$\delta(n)\xrightarrow{z}\sum_{n=-\infty}^{\infty}\delta(n)z^{-n}=\delta(0)z^{-0}=1$$

Then, from the delay property, we have

$$\delta(n-1)\xrightarrow{z}z^{-1}\cdot 1=z^{-1}, \qquad \delta(n-2)\xrightarrow{z}z^{-2}, \qquad \delta(n-3)\xrightarrow{z}z^{-3}, \quad \text{etc.}$$

Using linearity, we obtain

$$2\delta(n)+3\delta(n-1)+5\delta(n-2)+2\delta(n-3)\xrightarrow{z}2+3z^{-1}+5z^{-2}+2z^{-3}$$

for case (a), and

$$h(n)=\delta(n)-\delta(n-4)\xrightarrow{z}H(z)=1-z^{-4}$$

for case (b). □

Example 5.1.3: Using the unit-step identity $u(n)-u(n-1)=\delta(n)$, valid for all n, and the z-transform properties, determine the z-transforms of the two signals:

$$\text{(a)}\quad x(n)=u(n)\,, \qquad \text{(b)}\quad x(n)=-u(-n-1)$$

Solution: For case (a), we have the difference equation:

$$x(n)-x(n-1)=u(n)-u(n-1)=\delta(n)$$

Taking z-transforms of both sides and using the linearity and delay properties, we obtain

$$x(n)-x(n-1)=\delta(n)\xrightarrow{z}X(z)-z^{-1}X(z)=1 \qquad\Rightarrow\qquad X(z)=\frac{1}{1-z^{-1}}$$

Similarly, for case (b) we have the difference equation:

$$x(n)-x(n-1)=-u(-n-1)+u\big(-(n-1)-1\big)=u(-n)-u(-n-1)=\delta(-n)$$

where in the last equation we used the given identity with n replaced by $-n$. Noting that $\delta(-n)=\delta(n)$, and taking z-transforms of both sides, we find

$$x(n)-x(n-1)=\delta(-n)\xrightarrow{z}X(z)-z^{-1}X(z)=1 \qquad\Rightarrow\qquad X(z)=\frac{1}{1-z^{-1}}$$

Thus, even though the two signals $u(n)$ and $-u(-n-1)$ are completely different in the time domain (one is causal, the other anticausal), their z-transforms are the *same*. We will see in the next section that they can be distinguished in terms of their region of convergence. □

Example 5.1.4: Compute the output of Example 4.1.1 by carrying out the convolution operation as multiplication in the z-domain.

Solution: The two sequences

$$\mathbf{h}=[1,2,-1,1]\,, \qquad \mathbf{x}=[1,1,2,1,2,2,1,1]$$

have z-transforms:

$$\begin{aligned}H(z)&=1+2z^{-1}-z^{-2}+z^{-3}\\X(z)&=1+z^{-1}+2z^{-2}+z^{-3}+2z^{-4}+2z^{-5}+z^{-6}+z^{-7}\end{aligned}$$

Multiplying these polynomials, we find for the product $Y(z)=H(z)X(z)$:

$$Y(z)=1+3z^{-1}+3z^{-2}+5z^{-3}+3z^{-4}+7z^{-5}+4z^{-6}+3z^{-7}+3z^{-8}+z^{-10}$$

The coefficients of the powers of z are the convolution output samples:

$$\mathbf{y}=\mathbf{h}*\mathbf{x}=[1,3,3,5,3,7,4,3,3,0,1]$$

Note that the term z^{-9} is absent, which means that its coefficient is zero. □

5.2 Region of Convergence

If $x(n)$ has infinite duration, Eq. (5.1.1) becomes an infinite series, and it is possible that certain values of the complex variable z might render it divergent.

The *region of convergence* (ROC) of the z-transform $X(z)$ is defined to be that *subset* of the complex z-plane $\mathbb{C}$ for which the series (5.1.1) *converges*, that is,

$$\text{Region of Convergence}=\Big\{z\in\mathbb{C}\;\Big|\;X(z)=\sum_{n=-\infty}^{\infty}x(n)z^{-n}\neq\infty\Big\} \tag{5.2.1}$$

The ROC is an important concept in many respects: It allows the unique inversion of the z-transform and provides convenient characterizations of the causality and stability properties of a signal or system.

The ROC depends on the signal $x(n)$ being transformed. As an example, consider the following causal signal:

$$x(n)=(0.5)^n u(n)=\{1,\,0.5,\,0.5^2,\,0.5^3,\,\dots\}$$

Its z-transform will be:

$$X(z)=\sum_{n=-\infty}^{\infty}(0.5)^n u(n)z^{-n}=\sum_{n=0}^{\infty}(0.5)^n z^{-n}=\sum_{n=0}^{\infty}(0.5z^{-1})^n$$

where the summation was restricted over $n\geq 0$ because of the causality of $x(n)$. This infinite sum can be done with the help of the *infinite geometric series* formula:

$$\boxed{1+x+x^2+x^3+\cdots=\sum_{n=0}^{\infty}x^n=\frac{1}{1-x}} \tag{5.2.2}$$

which is valid only for $|x|<1$ and diverges otherwise. Setting $x=0.5z^{-1}$ we find the sum:

$$X(z)=\sum_{n=0}^{\infty}(0.5z^{-1})^n=\sum_{n=0}^{\infty}x^n=\frac{1}{1-x}\,,\qquad\text{or,}$$

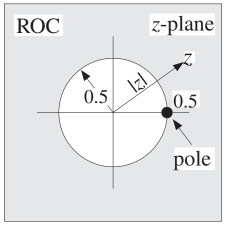
$$X(z)=\frac{1}{1-0.5z^{-1}}=\frac{z}{z-0.5}$$

where the convergence of the geometric series requires

$$|x|=|0.5z^{-1}|<1\quad\Rightarrow\quad|z|>0.5$$

Thus, the ROC is the set of z's in the z-plane that lie strictly *outside* the circle of radius 0.5, as shown below:

$$\text{ROC}=\{z\in\mathbb{C}\;\big|\;|z|>0.5\}$$



Note, that the z-transform has a *pole* at $z=0.5$. In summary, we have

$$(0.5)^n u(n)\xrightarrow{z}\frac{1}{1-0.5z^{-1}}\,,\qquad\text{with}\quad|z|>0.5$$

A z-transform and its ROC are *uniquely* determined by the time signal $x(n)$. However, it is possible for two different time signals $x(n)$ to have the *same* z-transform, as was the case in Example 5.1.3. Such signals can only be distinguished in the z-domain by their region of convergence. Consider, for example, the anticausal signal

$$x(n)=-(0.5)^n u(-n-1)$$

The presence of the anti-unit step $u(-n-1)$ restricts n to be $-n-1\geq 0$ or, $n\leq -1$. Its z-transform will be:

$$X(z)=-\sum_{n=-\infty}^{-1}(0.5)^n z^{-n}=-\sum_{n=-\infty}^{-1}\big((0.5)^{-1}z\big)^{-n}=-\sum_{m=1}^{\infty}\big((0.5)^{-1}z\big)^m$$

where we changed summation variables from n to $m=-n$. To sum it, we use the following variant of the infinite geometric series:

$$\boxed{x+x^2+x^3+\cdots=\sum_{m=1}^{\infty}x^m=\frac{x}{1-x}}$$

which is valid only for $|x|<1$ and diverges otherwise. Setting $x=(0.5)^{-1}z$, we have

$$X(z)=-\sum_{m=1}^{\infty}\big((0.5)^{-1}z\big)^m=-\sum_{m=1}^{\infty}x^m=-\frac{x}{1-x}=-\frac{0.5^{-1}z}{1-0.5^{-1}z}\,,\qquad\text{or,}$$

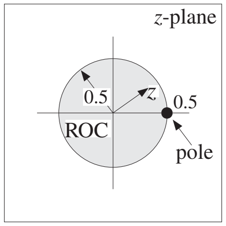
$$X(z)=\frac{z}{z-0.5}=\frac{1}{1-0.5z^{-1}}$$

which is the same as the causal example above. However, the ROC in this case is different. It is determined by the geometric series convergence condition

$$|x| = |0.5^{-1}z| < 1 \qquad \Rightarrow \qquad |z| < 0.5$$

which is the set of z's that lie strictly *inside* the circle of radius 0.5, as shown below:

$$\text{ROC} = \{z \in \mathbb{C} \;\big|\; |z| < 0.5\}$$



To summarize, we have determined the z-transforms:

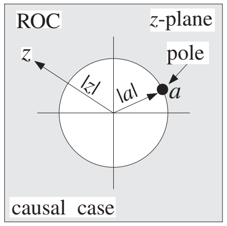
$$(0.5)^n u(n) \xrightarrow{z} \frac{1}{1-0.5z^{-1}}, \qquad \text{with } |z| > 0.5$$

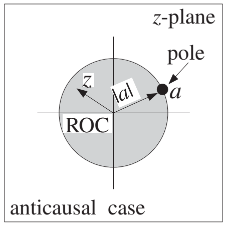
$$-(0.5)^n u(-n-1) \xrightarrow{z} \frac{1}{1-0.5z^{-1}}, \qquad \text{with } |z| < 0.5$$

The two signals have the same z-transform but completely disjoint ROCs. More generally, we have the result:

$$\boxed{\begin{aligned} a^n u(n) &\xrightarrow{z} \frac{1}{1-az^{-1}}, \qquad \text{with } |z| > |a| \\ -a^n u(-n-1) &\xrightarrow{z} \frac{1}{1-az^{-1}}, \qquad \text{with } |z| < |a| \end{aligned}} \tag{5.2.3}$$

where a is any *complex* number. Their ROCs are shown below.





The z-transforms (5.2.3), together with the linearity and delay properties, can be used to construct more complicated transforms.

Example 5.2.1: Setting $a = \pm 1$ in Eq. (5.2.3), we obtain the z-transforms of the causal and anticausal unit-steps and alternating unit-steps:

$$u(n) \xrightarrow{z} \frac{1}{1-z^{-1}}, \qquad \text{with } |z| > 1$$

$$-u(-n-1) \xrightarrow{z} \frac{1}{1-z^{-1}}, \qquad \text{with } |z| < 1$$

$$(-1)^n u(n) \xrightarrow{z} \frac{1}{1+z^{-1}}, \qquad \text{with } |z| > 1$$

$$-(-1)^n u(-n-1) \xrightarrow{z} \frac{1}{1+z^{-1}}, \qquad \text{with } |z| < 1$$

which agree with Example 5.1.3. □

Example 5.2.2: Determine the z-transform and corresponding region of convergence of the following signals:

1. $x(n) = u(n-10)$
2. $x(n) = (-0.8)^n u(n)$
3. $x(n) = (-0.8)^n [u(n) - u(n-10)]$
4. $x(n) = \frac{1}{2}[u(n) + (-1)^n u(n)] = \{1, 0, 1, 0, 1, 0, 1, 0, \dots\}$
5. $x(n) = \frac{1}{2}[(0.8)^n u(n) + (-0.8)^n u(n)]$
6. $x(n) = \cos(\frac{\pi n}{2}) u(n) = \{1, 0, -1, 0, 1, 0, -1, 0, 1, 0, -1, 0, \dots\}$
7. $x(n) = (0.8)^n \cos(\frac{\pi n}{2}) u(n)$
8. $x(n) = \frac{1}{2}[(0.8j)^n u(n) + (-0.8j)^n u(n)]$
9. $x(n) = \cos(\omega_0 n) u(n)$ and $x(n) = \sin(\omega_0 n) u(n)$
10. $x(n) = \{1, 2, 3, 1, 2, 3, 1, 2, 3, \dots\}$, periodically repeating $\{1, 2, 3\}$

Solution: Using the delay property, we have in case (1):

$$X(z) = z^{-10} U(z) = \frac{z^{-10}}{1-z^{-1}}$$

with ROC $|z| > 1$. In case (2), we apply Eq. (5.2.3) with $a = -0.8$ to get

$$X(z) = \frac{1}{1+0.8z^{-1}}, \qquad \text{with ROC: } |z| > |-0.8| = 0.8$$

For case (3), we write

$$x(n) = (-0.8)^n u(n) - (-0.8)^{10}(-0.8)^{n-10} u(n-10)$$

where in the second term we multiplied and divided by the factor $(-0.8)^{10}$ in order to reproduce the delayed (by 10 units) version of the first term. Thus, using the linearity and delay properties and the results of case (2), we get

$$X(z) = \frac{1}{1+0.8z^{-1}} - (-0.8)^{10}\frac{z^{-10}}{1+0.8z^{-1}} = \frac{1-(-0.8)^{10}z^{-10}}{1+0.8z^{-1}}$$

Here, the ROC is not $|z| > 0.8$ as might appear at first glance. Rather it is the set of all nonzero z's, $z \neq 0$. This follows by recognizing $x(n)$ to be a length-10 finite sequence. Indeed, setting $a = -0.8$, we have

$$x(n) = a^n [u(n) - u(n-10)] = \{1, a, a^2, a^3, a^4, a^5, a^6, a^7, a^8, a^9, 0, 0, 0, \dots\}$$

and therefore, its z-transform can be computed by the finite sum

$$X(z)= 1 + az^{-1} + a^2z^{-2} + \cdots + a^9z^{-9}$$

which exists for any $z \neq 0$. Using the finite geometric series

$$1 + x + x^2 + \cdots + x^{N-1} = \frac{1 - x^N}{1 - x}$$

we may sum the above series to

$$X(z)= 1 + az^{-1} + a^2z^{-2} + \cdots + a^9z^{-9} = \frac{1 - a^{10}z^{-10}}{1 - az^{-1}} = \frac{1 - (-0.8)^{10}z^{-10}}{1 + 0.8z^{-1}}$$

For case (4), we have, using linearity and Eq. (5.2.3) with $a = 1$ and $a = -1$:

$$X(z)= \frac{1}{2}\left[\frac{1}{1 - z^{-1}} + \frac{1}{1 + z^{-1}}\right] = \frac{1}{1 - z^{-2}}$$

with ROC $|z| > 1$. The same result can be obtained using the definition (5.1.1) and summing the series:

$$X(z)= 1 + 0z^{-1} + z^{-2} + 0z^{-3} + z^{-4} + \cdots = 1 + z^{-2} + z^{-4} + z^{-6} + \cdots$$

which is an infinite geometric series of the type of Eq. (5.2.2) with $x = z^{-2}$. Therefore,

$$X(z)= \left.\frac{1}{1 - x}\right|_{x=z^{-2}} = \frac{1}{1 - z^{-2}}$$

The convergence of the series requires $|x| = |z^{-2}| < 1$ or equivalently, $|z| > 1$. In case (5), we find again using linearity and Eq. (5.2.3):

$$X(z)= \frac{1}{2}\left[\frac{1}{1 - 0.8z^{-1}} + \frac{1}{1 + 0.8z^{-1}}\right] = \frac{1}{1 - 0.64z^{-2}}$$

with ROC $|z| > 0.8$. Case (6) can be handled directly by the definition (5.1.1):

$$X(z)= 1 - z^{-2} + z^{-4} - z^{-6} + z^{-8} + \cdots = 1 + x + x^2 + x^3 + x^4 + \cdots$$

where $x = -z^{-2}$. The series will converge to

$$X(z)= \frac{1}{1 - x} = \frac{1}{1 + z^{-2}}$$

provided $|x| = |-z^{-2}| < 1$, or equivalently, $|z| > 1$. The same result can be obtained using Euler's formula to split the cosine into exponential signals of the type (5.2.3):

$$x(n)= \cos(\frac{\pi n}{2})u(n)= \frac{1}{2}\big[e^{j\pi n/2}u(n)+e^{-j\pi n/2}u(n)\big] = \frac{1}{2}\big[a^n u(n)+a^{*n}u(n)\big]$$

where $a = e^{j\pi/2} = j$ and $a^* = e^{-j\pi/2} = -j$. Thus, we find

$$X(z)= \frac{1}{2}\left[\frac{1}{1 - jz^{-1}} + \frac{1}{1 + jz^{-1}}\right] = \frac{1}{1 + z^{-2}}$$

In case (7), using Euler's formula as above, we find

$$x(n)= (0.8)^n\cos(\frac{\pi n}{2})u(n)= \frac{1}{2}\big[(0.8)^n e^{j\pi n/2}u(n)+(0.8)^n e^{-j\pi n/2}u(n)\big]$$

which can be written as the signal of case (8):

$$x(n)= \frac{1}{2}\big[(0.8j)^n u(n)+(-0.8j)^n u(n)\big]$$

Thus, cases (7) and (8) are the same. Their z-transform is obtained using $a = \pm 0.8j$ in Eq. (5.2.3):

$$X(z)= \frac{1}{2}\left[\frac{1}{1 - 0.8jz^{-1}} + \frac{1}{1 + 0.8jz^{-1}}\right] = \frac{1}{1 + 0.64z^{-2}}$$

with ROC $|z| > |0.8j| = 0.8$. The cosinewave in case (9) can be handled in a similar fashion. We write

$$\cos(\omega_0 n)u(n)= \frac{1}{2}\big[e^{j\omega_0 n} + e^{-j\omega_0 n}\big]u(n) \xrightarrow{z} \frac{1}{2}\left[\frac{1}{1 - e^{j\omega_0}z^{-1}} + \frac{1}{1 - e^{-j\omega_0}z^{-1}}\right]$$

which combines to give:

$$X(z)= \frac{1 - \cos(\omega_0)z^{-1}}{1 - 2\cos(\omega_0)z^{-1} + z^{-2}}$$

Setting $\omega_0 = \pi/2$, we recover case (6). Similarly, for a sinewave we have

$$\sin(\omega_0 n)u(n)= \frac{1}{2j}\big[e^{j\omega_0 n} - e^{-j\omega_0 n}\big]u(n) \xrightarrow{z} \frac{1}{2j}\left[\frac{1}{1 - e^{j\omega_0}z^{-1}} - \frac{1}{1 - e^{-j\omega_0}z^{-1}}\right]$$

which combines to give:

$$X(z)= \frac{\sin(\omega_0)z^{-1}}{1 - 2\cos(\omega_0)z^{-1} + z^{-2}}$$

Finally, we consider case (10). Using the definition (5.1.1) and grouping the terms in groups of 3, we obtain:

$$\begin{aligned} X(z) &= (1 + 2z^{-1} + 3z^{-2}) + (1 + 2z^{-1} + 3z^{-2})z^{-3} + \\ &\quad + (1 + 2z^{-1} + 3z^{-2})z^{-6} + (1 + 2z^{-1} + 3z^{-2})z^{-9} + \cdots \\ &= (1 + 2z^{-1} + 3z^{-2})(1 + z^{-3} + z^{-6} + z^{-9} + \cdots) \\ &= \frac{1 + 2z^{-1} + 3z^{-2}}{1 - z^{-3}} \end{aligned}$$

The infinite geometric series converges for $|z^{-3}| < 1$ or $|z| > 1$. An alternative method is to delay $x(n)$ by one period, that is, 3 time units

$$x(n - 3)= \{0, 0, 0, 1, 2, 3, 1, 2, 3, \dots\}$$

and subtract it from $x(n)$ to get the difference equation

$$x(n)-x(n - 3)= \{1, 2, 3, 0, 0, 0, 0, 0, 0, \dots\} = \delta(n)+2\delta(n - 1)+3\delta(n - 2)$$

Then, taking z-transforms of both sides, we get

$$X(z)-z^{-3}X(z)= 1 + 2z^{-1} + 3z^{-2} \quad \Rightarrow \quad X(z)= \frac{1 + 2z^{-1} + 3z^{-2}}{1 - z^{-3}}$$

This technique can be generalized to any periodic sequence. It will be used later to implement digital periodic waveform generators. □

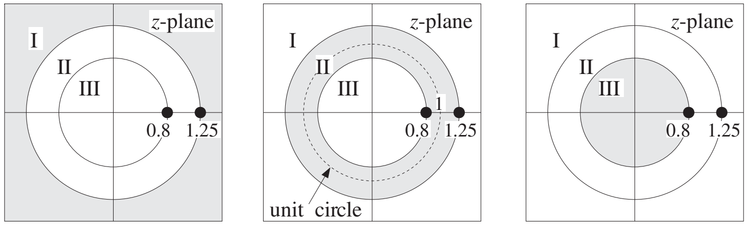
Example 5.2.3: Determine the z-transform and corresponding region of convergence of the following signals:

1. $x(n)= (0.8)^n u(n)+(1.25)^n u(n)$
2. $x(n)= (0.8)^n u(n)-(1.25)^n u(-n-1)$
3. $x(n)= -(0.8)^n u(-n-1)-(1.25)^n u(-n-1)$
4. $x(n)= -(0.8)^n u(-n-1)+(1.25)^n u(n)$

Solution: Using Eq. (5.2.3) with $a = 0.8$ and $a = 1.25$, we note that the first three cases have exactly the *same* z-transform, namely,

$$X(z)= \frac{1}{1-0.8z^{-1}} + \frac{1}{1-1.25z^{-1}} = \frac{2-2.05z^{-1}}{1-2.05z^{-1}+z^{-2}}$$

The three cases differ only in their ROCs. In case (1), both terms are causal, and therefore, we must have simultaneously $|z| > 0.8$ and $|z| > 1.25$. Thus, the ROC is $|z| > 1.25$. In case (2), the second term is anticausal and therefore we must have the simultaneous inequalities: $|z| > 0.8$ and $|z| < 1.25$. Thus, the ROC is $0.8 < |z| < 1.25$. In case (3), both terms are anticausal requiring $|z| < 0.8$ and $|z| < 1.25$. Therefore, the ROC is $|z| < 0.8$. The three ROCs are shown below:



The two poles of $X(z)$ at $z = 0.8$ and $z = 1.25$ divide the z-plane into three non-overlapping regions which are the three possible ROCs.

Note that case (1) is causal but unstable because the term $(1.25)^n$ diverges for large positive n. Case (2) is stable, because the term $(0.8)^n$ converges to zero exponentially for large positive n, and the term $(1.25)^n$ converges to zero exponentially for large negative n. And, case (3) is anticausal and unstable because the term $(0.8)^n$ diverges for large negative n.

The unit circle is contained entirely within the ROC of case (2), in accordance with the general criterion of stability of Section 5.3.

The fourth case, which is unstable both for $n \to \infty$ and $n \to -\infty$, does not have a z-transform because convergence requires $|z| < 0.8$ for the anticausal term and $|z| > 1.25$ for the causal term. Thus, there is no z for which $X(z)$ is converges. The ROC is the empty set. □

5.3 Causality and Stability

The z-domain characterizations of causality and stability can be obtained with the help of the basic result (5.2.3). A *causal* signal of the form

$$x(n)= A_1 p_1^n u(n)+A_2 p_2^n u(n)+\cdots \tag{5.3.1}$$

will have z-transform

$$X(z)= \frac{A_1}{1-p_1z^{-1}} + \frac{A_2}{1-p_2z^{-1}} + \cdots \tag{5.3.2}$$

with the restrictions $|z| > |p_1|$, $|z| > |p_2|$, and so forth. Therefore, the common ROC of all the terms will be

$$|z| > \max_i |p_i| \tag{5.3.3}$$

that is, the *outside* of the circle defined by the pole of *maximum magnitude*. Similarly, if the signal is completely *anticausal*

$$x(n)= -A_1 p_1^n u(-n-1)-A_2 p_2^n u(-n-1)-\cdots \tag{5.3.4}$$

its z-transform will be the same as Eq. (5.3.2), but the ROC restrictions on z will be $|z| < |p_1|$, $|z| < |p_2|$, and so forth. Thus, the ROC is in this case:

$$|z| < \min_i |p_i| \tag{5.3.5}$$

that is, the *inside* of the circle defined by the pole of *minimum magnitude*. The ROCs of these two cases are shown in Fig. 5.3.1.

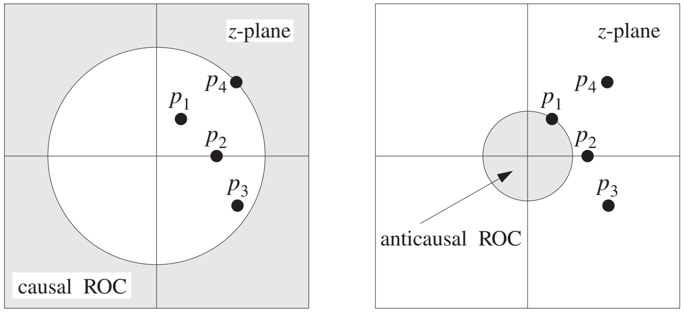


Fig. 5.3.1 Causal and anticausal ROCs.

In summary, causal signals are characterized by ROCs that are outside the maximum pole circle. Anticausal signals have ROCs that are inside the minimum pole circle. Mixed signals have ROCs that are the annular region between two circles—with the poles that lie inside the inner circle contributing causally and the poles that lie outside the outer circle contributing anticausally.

Stability can also be characterized in the z-domain in terms of the choice of the ROC. It can be shown that a *necessary and sufficient* condition for the stability of a signal $x(n)$ is that the ROC of the corresponding z-transform contain the unit circle. For a system $h(n)$, it can be shown that this condition is equivalent to the condition (3.5.4) discussed in Chapter 3.

Stability is not necessarily compatible with causality. For a signal or system to be simultaneously *stable and causal*, it is necessary that *all* its poles lie strictly *inside* the unit circle in the z-plane. This follows from Eq. (5.3.3) which is required for a causal ROC. If this ROC is to also correspond to a stable signal, then it must contain the unit circle. In other words, we may set $|z| = 1$ in Eq. (5.3.3):

$$1 > \max_i |p_i|$$

which implies that all poles must have magnitude less than one. A signal or system can also be simultaneously stable and anticausal, but in this case all its poles must lie strictly *outside* the unit circle. Indeed, the anticausality condition Eq. (5.3.5), together with the stability condition that the ROC contain the points $|z| = 1$, imply

$$1 < \min_i |p_i|$$

which means that all poles must have magnitude greater than one. If some of the poles have magnitude less than one and some greater than one, then it is possible to have a stable signal but it will be of the mixed kind. Those poles that lie inside the unit circle will contribute causally and those that lie outside will contribute anticausally.

Figure 5.3.2 illustrates three such possible stable cases. In all cases, the z-transform has the same form, namely,

$$X(z)= \frac{A_1}{1-p_1z^{-1}} + \frac{A_2}{1-p_2z^{-1}} + \frac{A_3}{1-p_3z^{-1}} + \frac{A_4}{1-p_4z^{-1}}$$

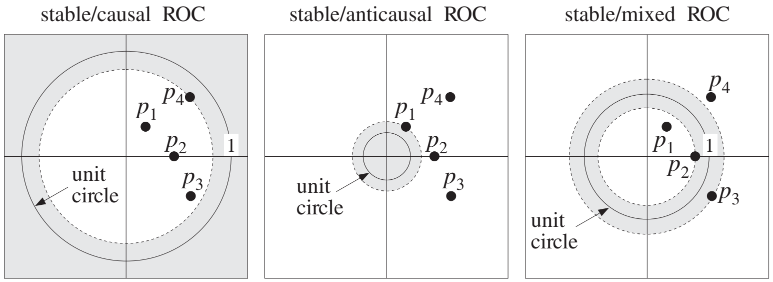


Fig. 5.3.2 Stable ROCs.

In the stable and causal case, all poles must have magnitude less than one, that is, $|p_i| < 1$, $i = 1,2,3,4$ and the signal $x(n)$ will be

$$x(n)= \big[A_1p_1^n + A_2p_2^n + A_3p_3^n + A_4p_4^n\big]u(n)$$

with all terms converging to zero exponentially for large positive n. In the stable/anticausal case, all poles have magnitude greater than one, $|p_i| > 1$, $i = 1,2,3,4$, and $x(n)$ will be:

$$x(n)= -\big[A_1p_1^n + A_2p_2^n + A_3p_3^n + A_4p_4^n\big]u(-n-1)$$

where because n is negative, each term will tend to zero exponentially for large negative n. This can be seen more clearly by writing a typical term as

$$-A_1p_1^nu(-n-1)= -A_1p_1^{-|n|}u(-n-1)= -A_1\left(\frac{1}{p_1}\right)^{|n|}u(-n-1)$$

where we set $n = -|n|$ for negative n. Because $|p_1| > 1$ it follows that $|1/p_1| < 1$ and its successive powers will tend to zero exponentially. In the mixed case, we have $|p_1| < |p_2| < 1$ and $|p_4| > |p_3| > 1$. Therefore, the stable signal will be

$$x(n)= \big[A_1p_1^n + A_2p_2^n\big]u(n) - \big[A_3p_3^n + A_4p_4^n\big]u(-n-1)$$

with p_1, p_2 contributing causally, and p_3, p_4 anticausally. An example of such a stable but mixed signal was given in the second case of Example 5.2.3, namely,

$$x(n)= (0.8)^nu(n) - (1.25)^nu(-n-1)$$

As we emphasized in Chapter 3, stability is more important in DSP than causality in order to avoid numerically divergent computations. Causality can be reconciled exactly if all the poles are inside the unit circle, but only approximately if some of the poles are outside. We will discuss this issue later.

An important class of signals are the so-called *marginally stable* signals, which neither diverge nor converge to zero for large n. Rather, they remain *bounded*. The unit-step, alternating unit-step, and more general sinusoidal signals fall in this class. Such signals have poles that lie *on* the unit circle.

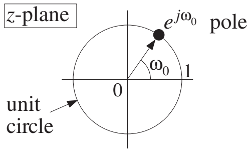
Some examples were cases (1,4,6,9,10) of Example 5.2.2. A simpler example is the case of a complex sinusoid of frequency ω_0

$$\text{(causal)} \qquad x(n)= e^{j\omega_0 n}u(n)$$

$$\text{(anticausal)} \qquad x(n)= -e^{j\omega_0 n}u(-n-1)$$

which is a special case of Eq. (5.2.3) with $a = e^{j\omega_0}$. Note that the plain unit-step $u(n)$ and alternating step $(-1)^nu(n)$ are special cases of this with $\omega_0 = 0$ and $\omega_0 = \pi$. The corresponding z-transform follows from Eq. (5.2.3):

$$X(z)= \frac{1}{1-e^{j\omega_0}z^{-1}}$$



with ROC being either $|z| > 1$ for the causal case, or $|z| < 1$ for the anticausal one.

5.4 Frequency Spectrum

The *frequency spectrum*, frequency content, or *discrete-time Fourier transform* (DTFT) of a signal $x(n)$ is defined by

$$\boxed{X(\omega)= \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n}} \qquad \text{(DTFT)} \qquad (5.4.1)$$

It is recognized as the *evaluation* of the z-transform *on the unit circle*, that is, at the z points:

$$\boxed{z = e^{j\omega}} \qquad (5.4.2)$$

Indeed, we have:[†]

$$X(z)\big|_{z=e^{j\omega}} = \sum_{n=-\infty}^{\infty} x(n)z^{-n}\Big|_{z=e^{j\omega}} = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n} = X(\omega)$$

The *frequency response* $H(\omega)$ of a linear system $h(n)$ with transfer function $H(z)$ is defined in the same way, namely,

$$\boxed{H(\omega)=\sum_{n=-\infty}^{\infty} h(n)e^{-j\omega n}} \qquad \text{(frequency response)} \tag{5.4.3}$$

and it is also the evaluation of $H(z)$ on the unit circle:

$$H(\omega)=H(z)\big|_{z=e^{j\omega}}$$

As discussed in Chapter 1, the *digital frequency* ω is in units of [radians/sample] and is related to the physical frequency f in Hz by

$$\boxed{\omega = \frac{2\pi f}{f_s}} \qquad \text{(digital frequency)} \tag{5.4.4}$$

The Nyquist interval $[-f_s/2, f_s/2]$ is the following interval in units of ω:

$$\boxed{-\pi \le \omega \le \pi} \qquad \text{(Nyquist interval)} \tag{5.4.5}$$

In Chapter 1, the quantity $X(\omega)$ was denoted by

$$\hat{X}(f)=\sum_{n=-\infty}^{\infty} x(nT)e^{-2\pi j f n/f_s}$$

It was the Fourier spectrum of the sampled signal $x(nT)$ and was given by the periodic replication of the original analog spectrum at multiples of f_s.

In units of ω, periodicity in f with period f_s becomes periodicity in ω with period 2π. Therefore, $X(\omega)$ may be considered only over one period, such as the Nyquist interval (5.4.5).

The *inverse DTFT* recovers the time sequence $x(n)$ from its spectrum $X(\omega)$ over the Nyquist interval:

$$\boxed{x(n)=\frac{1}{2\pi}\int_{-\pi}^{\pi} X(\omega)e^{j\omega n}\,d\omega} \qquad \text{(inverse DTFT)} \tag{5.4.6}$$

It expresses $x(n)$ as a linear combination of discrete-time sinusoids $e^{j\omega n}$ of different frequencies. The relative amplitudes and phases of these sinusoidal components are given by the DTFT $X(\omega)$. One quick way to prove Eq. (5.4.6) is to think of Eq. (5.4.1) as the Fourier series expansion of the periodic function $X(\omega)$. Then, Eq. (5.4.6) gives simply the Fourier series expansion coefficients. In terms of the physical frequency f in Hertz, the inverse DTFT reads as

$$\boxed{x(n)=\frac{1}{f_s}\int_{-f_s/2}^{f_s/2} X(f)e^{2\pi j f n/f_s}\,df}$$

As an example, consider a (double-sided) complex sinusoid of frequency ω_0:

$$\boxed{x(n)=e^{j\omega_0 n}, \quad -\infty < n < \infty}$$

Then, its DTFT will be given by

$$X(\omega)=2\pi\delta(\omega-\omega_0)+\text{(Nyquist replicas)}$$

where the term “Nyquist replicas” refers to the periodic replication of the first term at intervals of 2π. This is needed in order to make $X(\omega)$ periodic with period 2π. More precisely, the full expression will be

$$X(\omega)=2\pi\sum_{m=-\infty}^{\infty}\delta(\omega-\omega_0-2\pi m)$$

To verify it, we insert it into the inverse DTFT equation (5.4.6) and recover the given sinusoid. It was also discussed in Chapter 1, Example 1.5.1. Assuming that ω_0 lies in the Nyquist interval $[-\pi,\pi]$, then the restriction of $X(\omega)$ within it will be given only by the $m=0$ term, that is:

$$\boxed{X(\omega)=2\pi\delta(\omega-\omega_0), \quad -\pi \le \omega \le \pi}$$

Therefore, Eq. (5.4.6) gives

$$x(n)=\frac{1}{2\pi}\int_{-\pi}^{\pi} X(\omega)e^{j\omega n}\,d\omega = \frac{1}{2\pi}\int_{-\pi}^{\pi} 2\pi\delta(\omega-\omega_0)e^{j\omega n}\,d\omega = e^{j\omega_0 n}$$

Similarly, for a linear combination of two sinusoids we have:

$$x(n)=A_1e^{j\omega_1 n}+A_2e^{j\omega_2 n} \longrightarrow X(\omega)=2\pi A_1\delta(\omega-\omega_1)+2\pi A_2\delta(\omega-\omega_2)$$

This can be verified in the same way, if we assume that both ω_1 and ω_2 lie in the Nyquist interval. In particular, for real-valued cosine and sine signals, we have:

$$\boxed{\begin{aligned}\cos(\omega_0 n) &\longrightarrow \pi\delta(\omega-\omega_0)+\pi\delta(\omega+\omega_0)\\ \sin(\omega_0 n) &\longrightarrow -j\pi\delta(\omega-\omega_0)+j\pi\delta(\omega+\omega_0)\end{aligned}}$$

Another useful relationship is *Parseval's equation*, which relates the *total energy* of a sequence to its spectrum:

$$\boxed{\sum_{n=-\infty}^{\infty}|x(n)|^2 = \frac{1}{2\pi}\int_{-\pi}^{\pi}|X(\omega)|^2\,d\omega} \qquad \text{(Parseval)} \tag{5.4.7}$$

[†] Here, we abuse the notation and write $X(\omega)$ instead of $X(e^{j\omega})$.

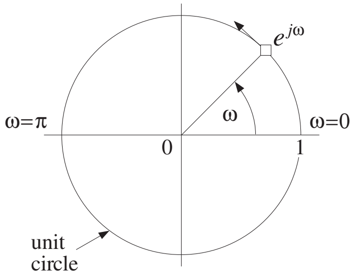


Fig. 5.4.1 Evaluation of z-transform on the unit circle.

The DTFT can be given a geometric interpretation by recognizing that the points $z = e^{j\omega}$ lie on the unit circle on the z-plane. As ω varies over the Nyquist interval $[-\pi, \pi]$, the complex point $z = e^{j\omega}$ moves around the unit circle, as shown in Fig. 5.4.1. The phase angle of z is ω.

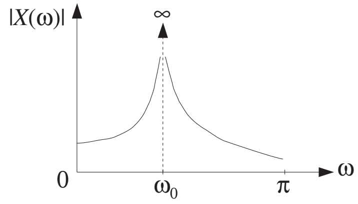
In order for the spectrum $X(\omega)$ to exist,[†] the ROC of the z-transform $X(z)$ must contain the unit circle; otherwise the z-transform will diverge at the unit circle points $z = e^{j\omega}$. But if the ROC contains the unit circle, the signal $x(n)$ must be stable. Thus, the Fourier transform $X(\omega)$ exists only for *stable* signals.

Marginally stable signals, such as sinusoids, strictly speaking do not have a spectrum because their poles lie on the unit circle and therefore the evaluation of $X(z)$ on the unit circle will cause $X(z)$ to diverge at certain z's. However, it is intuitively useful to consider their spectra. For example, for the causal complex sinusoid of the previous section we have:

$$x(n) = e^{j\omega_0 n}u(n) \xrightarrow{z} X(z) = \frac{1}{1 - e^{j\omega_0}z^{-1}}$$

and therefore the *formal* replacement of z by $e^{j\omega}$ will yield

$$X(\omega) = \frac{1}{1 - e^{j\omega_0}e^{-j\omega}} = \frac{1}{1 - e^{j(\omega_0 - \omega)}}$$



which diverges at $\omega = \omega_0$. However, this is to be expected because if the signal were a pure sinusoid $x(n) = e^{j\omega_0 n}$, its spectrum would be a single spectral line concentrated at $\omega = \omega_0$, that is, $X(\omega) = 2\pi\delta(\omega - \omega_0)$ (plus its Nyquist replicas). Here, the signal is not a pure sinusoid; it is a causal, truncated version of a pure sinusoid and therefore additional frequencies are present. However, the dominant frequency is still ω_0.

The shape of the spectrum $X(\omega)$ or $H(\omega)$ is affected by the *pole/zero pattern* of the z-transform $X(z)$ or $H(z)$, that is, by the relative geometric locations of the poles and zeros on the z-plane. To see this, consider a simple z-transform having a single pole at $z = p_1$ and a single zero at $z = z_1$.

[†] That is, to be finite $X(\omega) \neq \infty$ for all ω.

$$X(z) = \frac{1 - z_1 z^{-1}}{1 - p_1 z^{-1}} = \frac{z - z_1}{z - p_1}$$

The corresponding spectrum and its magnitude are obtained by replacing z by $e^{j\omega}$:

$$X(\omega) = \frac{e^{j\omega} - z_1}{e^{j\omega} - p_1} \quad \Rightarrow \quad |X(\omega)| = \frac{|e^{j\omega} - z_1|}{|e^{j\omega} - p_1|}$$

Figure 5.4.2 shows the relative locations of the fixed points z_1, p_1 and the moving point $z = e^{j\omega}$. A rough plot of $|X(\omega)|$ based on this pole/zero pattern is also shown. The magnitude spectrum $|X(\omega)|$ is the ratio of the distance of the point $e^{j\omega}$ to the zero z_1, namely, $|e^{j\omega} - z_1|$ divided by the distance of $e^{j\omega}$ to the pole p_1, namely, $|e^{j\omega} - p_1|$.

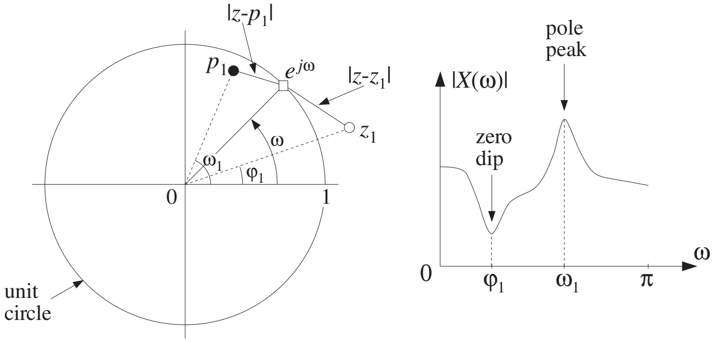


Fig. 5.4.2 Geometric interpretation of frequency spectrum.

As $e^{j\omega}$ moves around the unit circle, these distances will vary. As $e^{j\omega}$ passes near the pole, the denominator distance will become small causing the value of $|X(\omega)|$ to increase. If ω_1 is the phase angle of the pole p_1, then the point of closest approach to p_1 will occur at $\omega = \omega_1$ causing a *peak* in $|X(\omega)|$ there. The closer the pole is to the unit circle, the smaller the denominator distance will become at $\omega = \omega_1$, and the sharper the peak of $|X(\omega)|$.

Similarly, as $e^{j\omega}$ passes near the zero z_1, the numerator distance will become small, causing $|X(\omega)|$ to decrease. At the zero's phase angle, say $\omega = \phi_1$, this distance will be smallest, causing a *dip* in $|X(\omega)|$ there. The closer the zero to the unit circle, the sharper the dip. The zero z_1 can also lie *on* the unit circle, in which case $|X(\omega)|$ will vanish exactly at $\omega = \phi_1$.

In summary, we can draw a rough sketch of the spectrum $|X(\omega)|$ by letting $e^{j\omega}$ trace the unit circle and draw peaks as $e^{j\omega}$ passes near poles, and dips as it passes near zeros. By proper location of the zeros and poles of $X(z)$ or $H(z)$, one can design any desired shape for $X(\omega)$ or $H(\omega)$.

It is convenient to divide the unit circle into low-, medium-, and high-frequency wedge regions, as shown in Fig. 5.4.3. This subdivision is somewhat arbitrary because what is “low” or “high” frequency depends on the application. It aids, however, in the placement of poles and zeros. For example, to make a lowpass filter that emphasizes low frequencies and attenuates high ones, one would place poles inside the circle somewhere within the low-frequency wedge and/or zeros within the high-frequency wedge.

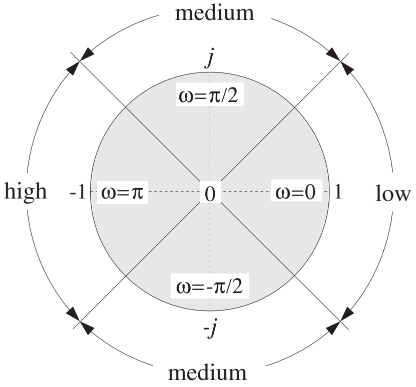


Fig. 5.4.3 Low-, medium-, and high-frequency parts of the unit circle.

Such filter design methods are somewhat crude and are used in practice only for the design of simple and/or specialized filters, such as resonator or notch filters or biquadratic filter sections for digital audio graphic and parametric equalizers. Such design examples will be considered later on.

The DTFT $X(\omega)$ of a signal $x(n)$ is a complex-valued quantity and therefore, it can be characterized also by its real and imaginary parts $\mathrm{Re}\,X(\omega)$, $\mathrm{Im}\,X(\omega)$ or, in its polar form, by its magnitude and phase responses $|X(\omega)|$, $\arg X(\omega)$. Thus,

$$X(\omega)=\mathrm{Re}\,X(\omega)+j\,\mathrm{Im}\,X(\omega)=|X(\omega)|\,e^{j\arg X(\omega)}$$

For *real-valued* signals $x(n)$, the quantity $X(\omega)$ satisfies the following so-called *hermitian* property:

$$\boxed{X(\omega)^*=X(-\omega)} \tag{5.4.8}$$

which translates to the following relationships for the magnitude and phase responses:

$$\boxed{\begin{aligned}|X(\omega)| &= |X(-\omega)|\\ \arg X(\omega) &= -\arg X(-\omega)\end{aligned}} \tag{5.4.9}$$

that is, the magnitude response is *even* in ω and the phase response *odd.* Similar definitions and results apply to the frequency response $H(\omega)$ of a real-valued system $h(n)$.

We note finally that the multiplicative filtering property $Y(z)=H(z)X(z)$ evaluated on the unit circle takes the following frequency-domain form:

$$\boxed{Y(\omega)=H(\omega)X(\omega)} \qquad \text{(filtering in frequency domain)} \tag{5.4.10}$$

Its consequences will be explored later on.

5.5 Inverse z-Transforms

The problem of inverting a given z-transform $X(z)$ is to find the time signal $x(n)$ whose z-transform is $X(z)$. As we saw already, the answer for $x(n)$ is not necessarily unique. But it can be made unique by specifying the corresponding ROC.

In inverting a z-transform, it is convenient to break it into its *partial fraction* (PF) expansion form, that is, into a sum of individual pole terms of the type (5.3.2).

Once $X(z)$ is written in the form (5.3.2), one still needs to know how to invert each term, that is, causally or anticausally. This depends on the choice of the ROC.

In general, the circles through the poles at $z=p_1$, $z=p_2$, and so on, divide the z-plane into non-overlapping regions, which are all possible candidates for ROCs. Any one of these ROC regions will result into a different $x(n)$. Among all possible $x(n)$, there will be a *unique* one that is *stable*, because the unit circle lies in exactly one of the possible ROCs.

Example 5.5.1: In Example (5.2.3), the first three signals had a common z-transform:

$$X(z)=\frac{1}{1-0.8z^{-1}}+\frac{1}{1-1.25z^{-1}}$$

The two circles through the poles at $z=0.8$ and $z=1.25$ divide the z-plane into the three regions I, II, III, shown in Example 5.2.3. There are therefore three possible inverse z-transforms, that is, three different signals $x(n)$ corresponding to the three ROC choices. But, only II is stable. □

The *partial fraction expansion method* can be applied to z-transforms that are ratios of two polynomials in z^{-1} of the form:

$$X(z)=\frac{N(z)}{D(z)}$$

The zeros of the denominator polynomial $D(z)$ are the poles of $X(z)$. Assuming $D(z)$ has degree M, there will be M denominator zeros, say at $p_1, p_2, \dots, p_M$, and $D(z)$ may be assumed to be in the factored form

$$D(z)=(1-p_1z^{-1})(1-p_2z^{-1})\cdots(1-p_Mz^{-1})$$

The partial fraction expansion of $X(z)$ is given by[†]

$$\begin{aligned}X(z) &= \frac{N(z)}{D(z)}=\frac{N(z)}{(1-p_1z^{-1})(1-p_2z^{-1})\cdots(1-p_Mz^{-1})}\\ &= \frac{A_1}{1-p_1z^{-1}}+\frac{A_2}{1-p_2z^{-1}}+\cdots+\frac{A_M}{1-p_Mz^{-1}}\end{aligned} \tag{5.5.1}$$

For this expansion to be possible as an identity in z^{-1}, the degree of the numerator polynomial $N(z)$ must be *strictly less* than the degree M of the denominator polynomial. The PF expansion coefficients A_i can be computed by the formulas:

$$A_i=\big[(1-p_iz^{-1})X(z)\big]_{z=p_i}=\left[\frac{N(z)}{\prod\limits_{j\neq i}(1-p_jz^{-1})}\right]_{z=p_i} \tag{5.5.2}$$

[†] We have assumed that all the poles are single poles.

for $i = 1, 2, \dots, M$. In words, the factor $(1 - p_i z^{-1})$ is deleted from the denominator and the remaining expression is evaluated at the pole $z = p_i$.

Example 5.5.2: In Example 5.2.3 the z-transform was written in the form

$$X(z) = \frac{2 - 2.05z^{-1}}{1 - 2.05z^{-1} + z^{-2}} = \frac{2 - 2.05z^{-1}}{(1 - 0.8z^{-1})(1 - 1.25z^{-1})}$$

Because the numerator polynomial has degree one in the variable z^{-1}, there is a PF expansion of the form:

$$X(z) = \frac{2 - 2.05z^{-1}}{(1 - 0.8z^{-1})(1 - 1.25z^{-1})} = \frac{A_1}{1 - 0.8z^{-1}} + \frac{A_2}{1 - 1.25z^{-1}}$$

The two coefficients are obtained by Eq. (5.5.2) as follows:

$$A_1 = \big[(1 - 0.8z^{-1})X(z)\big]_{z=0.8} = \left[\frac{2 - 2.05z^{-1}}{1 - 1.25z^{-1}}\right]_{z=0.8} = \frac{2 - 2.05/0.8}{1 - 1.25/0.8} = 1$$

$$A_2 = \big[(1 - 1.25z^{-1})X(z)\big]_{z=1.25} = \left[\frac{2 - 2.05z^{-1}}{1 - 0.8z^{-1}}\right]_{z=1.25} = 1$$

which are as expected. □

If the degree of the numerator polynomial $N(z)$ is exactly *equal* to the degree M of the denominator $D(z)$, then the PF expansion (5.5.1) must be modified by adding an extra term of the form:

$$\begin{aligned} X(z) &= \frac{N(z)}{D(z)} = \frac{N(z)}{(1 - p_1 z^{-1})(1 - p_2 z^{-1})\cdots(1 - p_M z^{-1})} \\ &= A_0 + \frac{A_1}{1 - p_1 z^{-1}} + \frac{A_2}{1 - p_2 z^{-1}} + \cdots + \frac{A_M}{1 - p_M z^{-1}} \end{aligned} \tag{5.5.3}$$

The coefficients A_i, $i = 1, 2, \dots, M$ are computed in exactly the same way by Eq. (5.5.2). The extra coefficient A_0 is computed by evaluating the z-transform at $z = 0$, that is,

$$A_0 = X(z)\big|_{z=0} \tag{5.5.4}$$

If the degree of $N(z)$ is strictly *greater* than M, one may divide the polynomial $D(z)$ into $N(z)$, finding the *quotient* and *remainder* polynomials, so that

$$N(z) = Q(z)D(z) + R(z)$$

and then writing

$$X(z) = \frac{N(z)}{D(z)} = \frac{Q(z)D(z) + R(z)}{D(z)} = Q(z) + \frac{R(z)}{D(z)}$$

where now the second term will admit an ordinary PF expansion of the form (5.5.1) because the degree of the remainder polynomial $R(z)$ is strictly less than M. Alternatively, one may simply *remove* the numerator polynomial $N(z)$ altogether, then carry out an ordinary PF expansion of the quantity

$$W(z) = \frac{1}{D(z)}$$

and finally *restore* the numerator by writing

$$X(z) = N(z)W(z)$$

We may refer to this method as the “remove/restore” method. Some examples will illustrate these techniques.

Example 5.5.3: We emphasize that a PF expansion may exist in one independent variable, say z^{-1}, but not in another, say z. For example, the z-transform

$$X(z) = \frac{2 - 2.05z^{-1}}{(1 - 0.8z^{-1})(1 - 1.25z^{-1})} = \frac{z(2z - 2.05)}{(z - 0.8)(z - 1.25)}$$

has numerator of degree one with respect to the variable z^{-1}, but degree two with respect to z. Thus, it admits an expansion of the form (5.5.1) with respect to z^{-1}, but not with respect to z.

Many texts prefer to work with z and therefore to make the PF expansion possible, a factor z is divided out to lower the degree of the numerator and then restored at the end, that is,

$$\frac{X(z)}{z} = \frac{(2z - 2.05)}{(z - 0.8)(z - 1.25)} = \frac{A_1}{z - 0.8} + \frac{A_2}{z - 1.25}$$

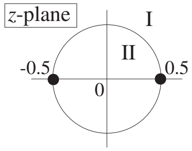
When z is restored, one gets

$$X(z) = \frac{zA_1}{z - 0.8} + \frac{zA_2}{z - 1.25} = \frac{A_1}{1 - 0.8z^{-1}} + \frac{A_2}{1 - 1.25z^{-1}}$$

It is easily verified that the PF expansion coefficients will be the same in the two approaches. In this book, we prefer to work directly with z^{-1} and avoid the extra algebraic steps required to write everything in terms of z, divide by z, restore z, and rewrite the final answer in terms of z^{-1}. □

Example 5.5.4: Compute all possible inverse z-transforms of

$$X(z) = \frac{6 + z^{-1}}{1 - 0.25z^{-2}}$$



Solution: Because the numerator has degree one in z^{-1}, we have the PF expansion:

$$X(z) = \frac{6 + z^{-1}}{1 - 0.25z^{-2}} = \frac{6 + z^{-1}}{(1 - 0.5z^{-1})(1 + 0.5z^{-1})} = \frac{A_1}{1 - 0.5z^{-1}} + \frac{A_2}{1 + 0.5z^{-1}}$$

where

$$A_1 = \left[\frac{6 + z^{-1}}{1 + 0.5z^{-1}}\right]_{z=0.5} = 4, \qquad A_2 = \left[\frac{6 + z^{-1}}{1 - 0.5z^{-1}}\right]_{z=-0.5} = 2$$

The two poles at ±0.5 have the same magnitude and therefore divide the z-plane into two ROC regions I and II: $|z| > 0.5$ and $|z| < 0.5$. For the first ROC, both terms in the PF expansion are inverted causally giving:

$$x(n) = A_1(0.5)^n u(n) + A_2(-0.5)^n u(n)$$

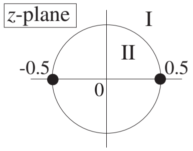
Because this ROC also contains the unit circle the signal $x(n)$ will be stable. For the second ROC, both PF expansion terms are inverted anticausally giving:

$$x(n) = -A_1(0.5)^n u(-n-1) - A_2(-0.5)^n u(-n-1)$$

This answer is unstable, because the ROC does not contain the unit circle. □

Example 5.5.5: Determine all inverse z-transforms of

$$X(z) = \frac{10 + z^{-1} - z^{-2}}{1 - 0.25z^{-2}}$$



Solution: Ordinary partial fraction expansion is not valid in this case because the degree of the numerator is the same as the degree of the denominator. However, we may still have an expansion of the form (5.5.3)

$$\begin{aligned} X(z) &= \frac{10 + z^{-1} - z^{-2}}{1 - 0.25z^{-2}} = \frac{10 + z^{-1} - z^{-2}}{(1 - 0.5z^{-1})(1 + 0.5z^{-1})} \\ &= A_0 + \frac{A_1}{1 - 0.5z^{-1}} + \frac{A_2}{1 + 0.5z^{-1}} \end{aligned}$$

where A_1 and A_2 are determined in the usual manner and A_0 is determined by evaluating $X(z)$ at $z = 0$:

$$A_0 = \left[\frac{10 + z^{-1} - z^{-2}}{1 - 0.25z^{-2}}\right]_{z=0} = \left[\frac{10z^2 + z - 1}{z^2 - 0.25}\right]_{z=0} = \frac{-1}{-0.25} = 4$$

$$A_1 = \left[\frac{10 + z^{-1} - z^{-2}}{1 + 0.5z^{-1}}\right]_{z=0.5} = 4, \qquad A_2 = \left[\frac{10 + z^{-1} - z^{-2}}{1 - 0.5z^{-1}}\right]_{z=-0.5} = 2$$

Again, there are only two ROCs I and II: $|z| > 0.5$ and $|z| < 0.5$. For the first ROC, the A_1 and A_2 terms are inverted causally, and the A_0 term inverts into a simple $\delta(n)$:

$$x(n) = A_0\delta(n) + A_1(0.5)^n u(n) + A_2(-0.5)^n u(n)$$

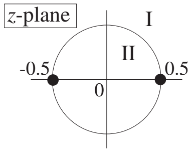
For the second ROC, we have:

$$x(n) = A_0\delta(n) - A_1(0.5)^n u(-n-1) - A_2(-0.5)^n u(-n-1)$$

Only the first inverse is stable because its ROC contains the unit circle. □

Example 5.5.6: Determine the causal inverse z-transform of

$$X(z) = \frac{6 + z^{-5}}{1 - 0.25z^{-2}}$$



Solution: Here, the degree of the numerator is strictly greater than that of the denominator. The first technique is to divide the denominator into the numerator, giving

$$(6 + z^{-5}) = (1 - 0.25z^{-2})(-16z^{-1} - 4z^{-3}) + (6 + 16z^{-1})$$

where $(6 + 16z^{-1})$ is the remainder polynomial and $(-16z^{-1} - 4z^{-3})$ the quotient. Then,

$$X(z) = \frac{6 + z^{-5}}{1 - 0.25z^{-2}} = -16z^{-1} - 4z^{-3} + \frac{6 + 16z^{-1}}{1 - 0.25z^{-2}}$$

and expanding the last term in PF expansion:

$$X(z) = -16z^{-1} - 4z^{-3} + \frac{19}{1 - 0.5z^{-1}} - \frac{13}{1 + 0.5z^{-1}}$$

The causal inverse, having ROC $|z| > 0.5$, will be:

$$x(n) = -16\delta(n-1) - 4\delta(n-3) + 19(0.5)^n u(n) - 13(-0.5)^n u(n)$$

The second technique is the "remove/restore" method. Ignoring the numerator we have

$$W(z) = \frac{1}{1 - 0.25z^{-2}} = \frac{0.5}{1 - 0.5z^{-1}} + \frac{0.5}{1 + 0.5z^{-1}}$$

which has the causal inverse

$$w(n) = 0.5(0.5)^n u(n) + 0.5(-0.5)^n u(n)$$

Once $w(n)$ is known, one can obtain $x(n)$ by restoring the numerator:

$$X(z) = (6 + z^{-5})W(z) = 6W(z) + z^{-5}W(z)$$

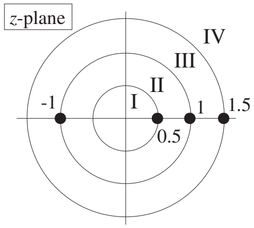
Taking inverse z-transforms of both sides and using the delay property, we find

$$\begin{aligned} x(n) &= 6w(n) + w(n-5) = 3(0.5)^n u(n) + 3(-0.5)^n u(n) \\ &+ 0.5(0.5)^{n-5} u(n-5) + 0.5(-0.5)^{n-5} u(n-5) \end{aligned}$$

The two expressions for $x(n)$ from the two techniques are equivalent. □

Example 5.5.7: Determine all possible inverse z-transforms of

$$X(z) = \frac{7 - 9.5z^{-1} - 3.5z^{-2} + 5.5z^{-3}}{(1 - z^{-2})(1 - 0.5z^{-1})(1 - 1.5z^{-1})}$$



Solution: $X(z)$ admits the PF expansion:

$$X(z)=\frac{1}{1-z^{-1}}+\frac{1}{1+z^{-1}}+\frac{3}{1-0.5z^{-1}}+\frac{2}{1-1.5z^{-1}}$$

where the PF expansion coefficients are easily found. The four poles at $z=0.5, 1, -1, 1.5$ divide the z-plane into the four ROC regions I, II, III, IV. Region I corresponds to the completely anticausal inverse and region IV to the completely causal one. For region II, the pole at $z=0.5$ will be inverted causally and the rest anticausally. For region III, $z=0.5$ and $z=\pm 1$ will be inverted causally and $z=1.5$ anticausally. Thus, the four possible inverse z-transforms are:

$$\begin{aligned}
x_1(n) &= -\big[1+(-1)^n+3(0.5)^n+2(1.5)^n\big]u(-n-1)\\
x_2(n) &= 3(0.5)^n u(n)-\big[1+(-1)^n+2(1.5)^n\big]u(-n-1)\\
x_3(n) &= \big[1+(-1)^n+3(0.5)^n\big]u(n)-2(1.5)^n u(-n-1)\\
x_4(n) &= \big[1+(-1)^n+3(0.5)^n+2(1.5)^n\big]u(n)
\end{aligned}$$

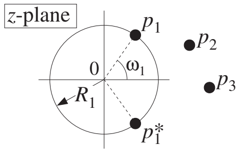
Strictly speaking there is no stable answer because two of the poles, $z=\pm 1$, lie on the unit circle. However, $x_2(n)$ and $x_3(n)$ are marginally stable, that is, neither diverging nor converging to zero for large n. In both cases, the anticausal term $(1.5)^n$ tends to zero for large negative n. Indeed, because n is negative, we write $n=-|n|$ and

$$(1.5)^n=(1.5)^{-|n|}\to 0 \quad \text{as} \quad n\to -\infty$$

The terms due to the poles $z=\pm 1$ are causal or anticausal in cases III and II, but they remain bounded. The other two signals $x_1(n)$ and $x_4(n)$ are unstable because the unit circle does not lie in their ROCs. □

The assumption that the numerator and denominator polynomials $N(z)$ and $D(z)$ have *real-valued* coefficients implies that the *complex-valued* poles of $X(z)$ come in *complex-conjugate pairs*. In that case, the PF expansion takes the form

$$X(z)=\frac{A_1}{1-p_1z^{-1}}+\frac{A_1^*}{1-p_1^*z^{-1}}+\frac{A_2}{1-p_2z^{-1}}+\cdots$$



where the PF expansion coefficients also come in *conjugate pairs*. Thus, it is necessary to determine only one of them, not both. The corresponding inverse z-transform will be real-valued; indeed, considering the causal case we have

$$x(n)=A_1p_1^n u(n)+A_1^*p_1^{*n}u(n)+A_2p_2^n u(n)+\cdots$$

Because the first two terms are complex conjugates of each other, we may use the result that $C+C^*=2\mathrm{Re}(C)$, for any complex number C, to write the first term as

$$A_1p_1^n+A_1^*p_1^{*n}=2\mathrm{Re}\big[A_1p_1^n\big]$$

Writing A_1 and p_1 in their polar form, say, $A_1=B_1e^{j\alpha_1}$ and $p_1=R_1e^{j\omega_1}$, with $B_1>0$ and $R_1>0$, we have

$$\mathrm{Re}\big[A_1p_1^n\big]=\mathrm{Re}\big[B_1e^{j\alpha_1}R_1^n e^{j\omega_1 n}\big]=B_1R_1^n\mathrm{Re}\big[e^{j\omega_1 n+j\alpha_1}\big]$$

and taking the real part of the exponential, we find

$$A_1p_1^n+A_1^*p_1^{*n}=2\mathrm{Re}\big[A_1p_1^n\big]=2B_1R_1^n\cos(\omega_1 n+\alpha_1)$$

and for $x(n)$

$$x(n)=2B_1R_1^n\cos(\omega_1 n+\alpha_1)u(n)+A_2p_2^n u(n)+\cdots$$

Thus, complex-valued poles correspond to *exponentially decaying sinusoids* (if $R_1<1$). The decay envelope R_1^n and the frequency ω_1 depend on the complex pole by $p_1=R_1e^{j\omega_1}$.

The first-order terms in the partial fraction expansion corresponding to complex conjugate poles can be reassembled into second-order terms with *real-valued* coefficients, as follows:

$$\frac{A_1}{1-p_1z^{-1}}+\frac{A_1^*}{1-p_1^*z^{-1}}=\frac{(A_1+A_1^*)-(A_1p_1^*+A_1^*p_1)z^{-1}}{(1-p_1z^{-1})(1-p_1^*z^{-1})}$$

Using the identities

$$\boxed{(1-p_1z^{-1})(1-p_1^*z^{-1})=1-2\mathrm{Re}(p_1)z^{-1}+|p_1|^2z^{-2}}$$

or,

$$\boxed{(1-R_1e^{j\omega_1}z^{-1})(1-R_1e^{-j\omega_1}z^{-1})=1-2R_1\cos(\omega_1)z^{-1}+R_1^2z^{-2}}$$

and writing

$$\begin{aligned}
A_1+A_1^* &= 2\mathrm{Re}(A_1)=2B_1\cos(\alpha_1)\\
A_1p_1^*+A_1^*p_1 &= 2\mathrm{Re}(A_1p_1^*)=2B_1R_1\cos(\alpha_1-\omega_1)
\end{aligned}$$

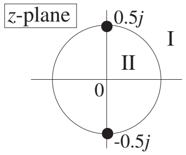
we find

$$\frac{A_1}{1-p_1z^{-1}}+\frac{A_1^*}{1-p_1^*z^{-1}}=\frac{2B_1\cos(\alpha_1)-2B_1R_1\cos(\alpha_1-\omega_1)z^{-1}}{1-2R_1\cos(\omega_1)z^{-1}+R_1^2z^{-2}}$$

having real-valued coefficients.

Example 5.5.8: Determine all possible inverse z-transforms of

$$X(z)=\frac{4-3z^{-1}+z^{-2}}{1+0.25z^{-2}}$$



Solution: We write

$$X(z) = \frac{4-3z^{-1}+z^{-2}}{1+0.25z^{-2}} = \frac{4-3z^{-1}+z^{-2}}{(1-0.5jz^{-1})(1+0.5jz^{-1})}$$

$$= A_0 + \frac{A_1}{1-0.5jz^{-1}} + \frac{A_1^*}{1+0.5jz^{-1}}$$

with the numerical values:

$$A_0 = \left[\frac{4-3z^{-1}+z^{-2}}{1+0.25z^{-2}}\right]_{z=0} = 4\,, \qquad A_1 = \left[\frac{4-3z^{-1}+z^{-2}}{1+0.5jz^{-1}}\right]_{z=0.5j} = 3j$$

Therefore,

$$X(z) = 4 + \frac{3j}{1-0.5jz^{-1}} - \frac{3j}{1+0.5jz^{-1}} = 4 - \frac{3z^{-1}}{1+0.25z^{-2}}$$

The causal ROC is $|z| > |0.5j| = 0.5$, resulting in

$$x(n) = 4\delta(n) + 3j(0.5j)^n u(n) - 3j(-0.5j)^n u(n)$$

Because the last two terms are complex conjugates of each other, we may write them as

$$x(n) = 4\delta(n) + 2\mathrm{Re}\big[3j(0.5j)^n u(n)\big] = 4\delta(n) + 6(0.5)^n u(n)\mathrm{Re}\big[j^{n+1}\big]$$

Writing $j^{n+1} = e^{j\pi(n+1)/2}$ and taking real parts we find

$$\mathrm{Re}\big[j^{n+1}\big] = \cos\big(\frac{\pi(n+1)}{2}\big) = -\sin\big(\frac{\pi n}{2}\big)$$

and

$$x(n) = 4\delta(n) - 6(0.5)^n \sin\big(\frac{\pi n}{2}\big)u(n)$$

Similarly, we find

$$x(n) = 4\delta(n) + 6(0.5)^n \sin\big(\frac{\pi n}{2}\big)u(-n-1)$$

for the anticausal version with ROC $|z| < 0.5$. Some additional examples with complex conjugate poles were cases (6-9) of Example 5.2.2. □

5.6 Unilateral z-Transform

Given a possibly double-sided sequence, x_n, its causal part, x_n^+, is defined as the causal signal corresponding to $n \ge 0$, that is,

$$\begin{array}{rcccccccccl}
n = & [\cdots & -2 & -1 & 0 & 1 & 2 & 3 & \cdots] \\
x_n = & [\cdots & x_{-2} & x_{-1} & x_0 & x_1 & x_2 & x_3 & \cdots] \\
x_n^+ = & [\cdots & 0 & 0 & x_0 & x_1 & x_2 & x_3 & \cdots]
\end{array} \tag{5.6.1}$$

Similarly, the causal parts of the left-shifted time-advanced versions, x_{n+1}^+, x_{n+2}^+, will be,

$$\begin{array}{rcccccccccl}
n = & [\cdots & -2 & -1 & 0 & 1 & 2 & 3 & \cdots] \\
x_{n+1} = & [\cdots & x_{-1} & x_0 & x_1 & x_2 & x_3 & x_4 & \cdots] \\
x_{n+1}^+ = & [\cdots & 0 & 0 & x_1 & x_2 & x_3 & x_4 & \cdots] \\
x_{n+2} = & [\cdots & x_0 & x_1 & x_2 & x_3 & x_4 & x_5 & \cdots] \\
x_{n+2}^+ = & [\cdots & 0 & 0 & x_2 & x_3 & x_4 & x_5 & \cdots]
\end{array} \tag{5.6.2}$$

And, the causal parts of the right-shifted time-delayed versions, x_{n-1}^+, x_{n-2}^+, will be,

$$\begin{array}{rcccccccccl}
n = & [\cdots & -2 & -1 & 0 & 1 & 2 & 3 & \cdots] \\
x_{n-1} = & [\cdots & x_{-3} & x_{-2} & x_{-1} & x_0 & x_1 & x_2 & \cdots] \\
x_{n-1}^+ = & [\cdots & 0 & 0 & x_{-1} & x_0 & x_1 & x_2 & \cdots] \\
x_{n-2} = & [\cdots & x_{-4} & x_{-3} & x_{-2} & x_{-1} & x_0 & x_1 & \cdots] \\
x_{n-2}^+ = & [\cdots & 0 & 0 & x_{-2} & x_{-1} & x_0 & x_1 & \cdots]
\end{array} \tag{5.6.3}$$

Denoting by $X^+(z)$, the z-transform of x_n^+,

$$\boxed{X^+(z) = \sum_{n=0}^{\infty} x_n z^{-n} = x_0 + x_1 z^{-1} + z_2 z^{-2} + \cdots} \quad \text{(unilateral } z\text{-transform)} \tag{5.6.4}$$

which is referred to as the *unilateral* z-transform of original signal x_n. It then follows from Eqs. (5.6.1)–(5.6.3) that the z-transforms of the time-advanced and time-delayed causal parts, $x_{n\pm1}^+$, $x_{n\pm2}^+$, will be related to $X^+(z)$ via,

$$\boxed{\begin{array}{l}
x_n^+ \xrightarrow{z} X^+(z) \\[2ex]
x_{n+1}^+ \xrightarrow{z} z\big[X^+(z) - x_0\big] \\[2ex]
x_{n+2}^+ \xrightarrow{z} z^2\big[X^+(z) - x_0 - z^{-1}x_1\big] \\[2ex]
x_{n-1}^+ \xrightarrow{z} z^{-1}\big[X^+(z) + zx_{-1}\big] \\[2ex]
x_{n-2}^+ \xrightarrow{z} z^{-2}\big[X^+(z) + z^2 x_{-2} + zx_{-1}\big]
\end{array}} \tag{5.6.5}$$

for example, we see from Eq. (5.6.2) that,

$$\begin{aligned}
x_{n+1}^+ \xrightarrow{z} \; & x_1 + z^{-1}x_2 + z^{-2}x_3 + \cdots = z\big[z^{-1}x_1 + z^{-2}x_2 + z^{-3}x_3 + \cdots\big] \\
& z\big[x_0 + z^{-1}x_1 + z^{-2}x_2 + z^{-3}x_3 + \cdots\big] - zx_0 = zX^+(z) - zx_0
\end{aligned}$$

Such relationships are useful in solving difference equations with given initial conditions. They are akin to the Laplace transform differentiation properties,

$$
\begin{aligned}
x(t) &\xrightarrow{\mathcal{L}} X(s) \\
\dot{x}(t) &\xrightarrow{\mathcal{L}} sX(s) - x(0^-) \\
\ddot{x}(t) &\xrightarrow{\mathcal{L}} s^2 X(s) - sx(0^-) - \dot{x}(0^-)
\end{aligned}
\tag{5.6.6}
$$

with the usual definition of the unilateral Laplace transform [597,598],

$$
x(t) \xrightarrow{\mathcal{L}} X(s) = \int_{0^-}^{\infty} x(t)\, e^{-st}\, dt
\tag{5.6.7}
$$

Example 5.6.1: *Loan / Mortgage Amortization.* In obtaining a loan or mortgage payable at a fixed rate in a fixed number of years, the payment amount (per payment period) can be calculated from the formula:

$$
P = \frac{r y_0 (1+r)^N}{(1+r)^N - 1}
\tag{5.6.8}
$$

where y_0 is the initial loan amount and, assuming monthly payments, $r = R/12$, where R is the annual interest rate, and N is the total number of months.

For example, for a 5-year loan at a 6% annual rate, we have, $R = 0.06$ and $r = R/12 = 0.005$, and N is the total number of payment periods, e.g., $N = 12 \times 5 = 60$ months.

Let y_n denote the loan balance at the end of the n-th payment period. At the next period, $n+1$, the payment of P dollars is subtracted from the present balance of y_n, but before this is done, the bank charges an interest, $i_n = r y_n$, therefore, only the amount, $b_n = P - r y_n$, is used to reduce the balance, so that the next balance will be:

$$
y_{n+1} = y_n - b_n = y_n - (P - r y_n) = (1+r) y_n - P
$$

or, defining the quantity, $a = 1 + r$, for convenience,

$$
y_{n+1} = a y_n - P, \quad n = 0, 1, 2, \dots
\tag{5.6.9}
$$

This recursion can be solved analytically, with solution,

$$
y_n = \left[y_0 - \frac{P}{r} \right] a^n + \frac{P}{r}, \quad n = 0, 1, 2, \dots
\tag{5.6.10}
$$

Requiring that $y_n = 0$ at $n = N$, and solving for P, we obtain, Eq. (5.6.8). Then, after using Eq. (5.6.8), we can rewrite the solution as,

$$
y_n = \frac{a^N - a^n}{a^N - 1}\, y_0, \quad n = 0, 1, 2, \dots, N
\tag{5.6.11}
$$

The solution correctly gives $y_n = y_0$ at $n = 0$, and $y_n = 0$ at $n = N$.

a. Derive the solution Eq. (5.6.10), and then, prove Eqs. (5.6.8) and (5.6.11).

b. Consider the numerical values $y_0 = 10000$, $R = 0.06$, $N = 60$. Compute the payment amount P and the values of the balance y_n for $n = 0, 1, \dots, N$, as well as the interest i_n paid at the n-th period, and the amount b_n used to reduce the balance, and make a table exactly like the one shown below (but with all 61 entries printed.)

```
  n     y(n)      i(n)     b(n)
-----------------------------
  0  10000.00    50.00   143.33
  1   9856.67    49.28   144.04
  2   9712.63    48.56   144.76
  3   9567.86    47.84   145.49
  4   9422.37    47.11   146.22
----------- etc ------------
 57    574.23     2.87   190.46
 58    383.78     1.92   191.41
 59    192.37     0.96   192.37
 60      0.00      -        -
```

Note that initially the interest payment is high but it gets smaller with time, while the amount that goes to repay the loan increases. Banks usually provide a payment schedule just like this table.

c. Compute the total amount that went to re-pay the loan, and the total amount of interest paid. Make a plot of y_n versus n and observe how it is driven to zero at $n = N$. Make also a plot of the cumulative interest.

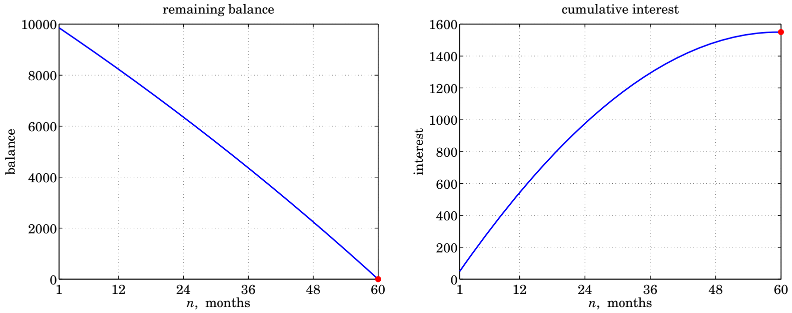


Fig. 5.6.1 Balance and interest versus time.

Solution: Derivation of Eq. (5.6.10). We may work with unilateral z-transforms and apply the time-advance identity of Eq. (5.6.2)

$$
\begin{aligned}
y_n^+ &\xrightarrow{Z} Y^+(z) \\
y_{n+1}^+ &\xrightarrow{Z} z\left[Y^+(z) - y_0\right]
\end{aligned}
$$

Then, Eq. (5.6.9) reads, in the time and z domains, where $u(n)$ is the unit-step,

$$
\begin{aligned}
y_{n+1}^+ &= a y_n^+ - P u(n) \\
z\left[Y^+(z) - y_0\right] &= a Y^+(z) - \frac{P}{1 - z^{-1}}
\end{aligned}
$$

which can be solved for $Y^+(z)$, followed by partial fraction expansion:

$$Y^+(z)=\frac{y_0}{1-az^{-1}}-\frac{Pz^{-1}}{(1-az^{-1})(1-z^{-1})}=\frac{y_0}{1-az^{-1}}-\frac{P}{r}\left[\frac{1}{1-az^{-1}}-\frac{1}{1-z^{-1}}\right]$$

whose causal inverse z-transform is Eq. (5.6.10):

$$y_n^+=y_0a^nu(n)-\frac{P}{r}\big[a^nu(n)-u(n)\big]=\left[y_0-\frac{P}{r}\right]a^n+\frac{P}{r}\,,\quad n=0,1,2,\dots$$

Derivation of Eq. (5.6.10) by iteration. A few iterations of Eq. (5.6.9) for, $n=0,1,2,3$, are listed below,

$$\begin{aligned}
y_1&=ay_0-P\\
y_2&=ay_1-P=a[ay_0-P]-P=a^2y_0-P(1+a)\\
y_3&=ay_2-P=a[a^2y_0-(1+a)P]-P=a^3y_0-P(1+a+a^2)\\
y_4&=ay_3-P=a[a^3y_0-(1+a+a^2)P]-P=a^4y_0-P(1+a+a^2+a^3)
\end{aligned}$$

thus, by induction, we conclude that,

$$y_n=a^ny_0-P(1+a+a^2+\cdots a^{n-1})=a^ny_0-P\,\frac{a^n-1}{a-1}=\left[y_0-\frac{P}{a-1}\right]a^n+\frac{P}{a-1}$$

which is the same as Eq. (5.6.10), since, $a-1=r$, and we used the finite geometric series,

$$1+a+a^2+\cdots a^{n-1}=\frac{a^n-1}{a-1}$$

Another derivation of Eq. (5.6.10). From the theory of linear difference equations with constant coefficients, it follows that the solution of,

$$y_{n+1}^+=ay_n^+-Pu(n) \tag{5.6.12}$$

can be expressed as the sum a particular solution and a homogeneous solution, the latter being the solution of the homogeneous equation,

$$y_{n+1}^+=ay_n^+\quad\Rightarrow\quad y_{\text{homog}}(n)=ca^n$$

with an arbitrary constant c to be fixed by the initial conditions. Since the input, $Pu(n)$, is a constant for $n\geq 0$, a particular solution of the full equation (5.6.12) could also be chosen to be a constant, say, $y_{\text{part}}(n)=y_c$, thus, we require,

$$y_c=ay_c-P\quad\Rightarrow\quad y_c=-\frac{P}{1-a}=\frac{P}{r}$$

It follows that the complete solution will be, for $n\geq 0$,

$$y_n=y_{\text{homog}}(n)+y_{\text{part}}(n)=ca^n+\frac{P}{r}$$

The constant c is fixed by requiring the initial condition, $y_n=y_0$ at $n=0$, which gives,

$$y_0=c+\frac{P}{r}\quad\Rightarrow\quad c=y_0-\frac{P}{r}\quad\Rightarrow\quad y_n=\left[y_0-\frac{P}{r}\right]a^n+\frac{P}{r}\,,\quad n\geq 0$$

Note also that the total interest paid can be calculated by summing up the individual interests,

$$I_{\text{tot}}=\sum_{n=0}^{N-1}ry_n$$

Using Eq. (5.6.11), we find

$$I_{\text{tot}}=r\sum_{n=0}^{N-1}y_0a^n-r\sum_{n=0}^{N-1}\frac{P}{r}\big[a^n-1\big]=NP-\left[\frac{P}{r}-y_0\right]r\sum_{n=0}^{N-1}a^n$$

which, after using the finite geometric series, simplifies into,

$$I_{\text{tot}}=NP-\left[\frac{P}{r}-y_0\right](a^N-1)$$

and using Eq. (5.6.8) for P, it simplifies further into,

$$I_{\text{tot}}=\sum_{n=0}^{N-1}ry_n=NP-y_0$$

which is as expected since, over N payment periods, the total amount paid is NP, part of which goes into paying up the initial loan of y_0, and the rest is the interest, that is, $NP=y_0+I_{\text{tot}}$.

For the given values of, $y_0=10000$ loan amount, $R=0.06$ annual rate, $N=60$ months, we find, $r=R/12=0.005$, $a=1+r=1.005$. The monthly payment amount is found from Eq. (5.6.8) to be, $P=193.33$ and the total interest, $I_{\text{tot}}=NP-y_0=1599.68$.

We note finally, that even though the solution Eq. (5.6.10) is unstable, since $a>1$, the solution is only used for a finite time period until it becomes zero, $y_n=0$. □

5.7 Problems

5.1 Prove the linearity, delay, and convolution properties of z-transforms given by Eqs. (5.1.3)–(5.1.5).

5.2 Compute the z-transform of the following sequences and determine the corresponding region of convergence:

a. $x(n)=\delta(n-5)$

b. $x(n)=\delta(n+5)$

c. $x(n)=u(n-5)$

d. $x(n)=u(-n+5)$

5.3 Compute the z-transform of the following sequences and determine the corresponding region of convergence:

a. $x(n)=(-0.5)^nu(n)$

b. $x(n)=(-0.5)^n\big[u(n)-u(n-10)\big]$

c. $x(n)=(0.5)^nu(n)+(-0.5)^nu(n)$

5.4 Compute the z-transform of the following sequences and determine the corresponding region of convergence:

a. $x(n)=2(0.8)^n\cos(\pi n/2)u(n)$

b. $x(n)=(0.8j)^n u(n)+(-0.8j)^n u(n)$

5.5 Compute the z-transform of the following sequences and determine the corresponding region of convergence:

a. $x(n)=(0.25)^n u(n)+4^n u(n)$

b. $x(n)=(0.25)^n u(n)-4^n u(-n-1)$

c. $x(n)=-(0.25)^n u(-n-1)-4^n u(-n-1)$

d. Explain why $x(n)=-(0.25)^n u(-n-1)+4^n u(n)$ does not have a z-transform.

5.6 Using the power series definition of z-transforms, derive the z-transform and its ROC of the signal $x(n)=\cos(\pi n/2)u(n)$.

5.7 Using partial fractions or power series expansions, compute the inverse z-transform of the following z-transforms and determine whether the answer is causal and/or stable:

a. $X(z)=(1-4z^{-2})(1+3z^{-1})$

b. $X(z)=5+3z^3+2z^{-2}$

5.8 Using partial fractions or power series expansions, determine all possible inverse z-transforms of the following z-transforms, sketch their ROCs, and discuss their stability and causality properties:

a. $X(z)=\dfrac{3(1+0.3z^{-1})}{1-0.81z^{-2}}$

b. $X(z)=\dfrac{6-3z^{-1}-2z^{-2}}{1-0.25z^{-2}}$

c. $X(z)=\dfrac{6+z^{-5}}{1-0.64z^{-2}}$

d. $X(z)=\dfrac{10+z^{-2}}{1+0.25z^{-2}}$

e. $X(z)=\dfrac{6-2z^{-1}-z^{-2}}{(1-z^{-1})(1-0.25z^{-2})}, \qquad \text{ROC } |z|>1$

f. $X(z)=-4+\dfrac{1}{1+4z^{-2}}$

g. $X(z)=\dfrac{4-0.6z^{-1}+0.2z^{-2}}{(1-0.5z^{-1})(1+0.4z^{-1})}$

5.9 Consider the z-transform pair:

$$x_a(n)=a^n u(n) \quad\Leftrightarrow\quad X_a(z)=\frac{1}{1-az^{-1}}$$

Applying the derivative operator $\partial/\partial a$ to the pair, derive the z-transform of the sequence $x(n)=na^n u(n)$.

5.10 Consider the differential operator $D=a\dfrac{\partial}{\partial a}$. First, show that its k-fold application gives $D^k a^n = n^k a^n$. Then, use this result to obtain the z-transform of $x(n)=n^k a^n u(n)$. Derive the explicit transforms for the cases $k=1,2,3,4$.

5.11 Show the z-transform of a triangular signal:

$$\sum_{n=-L}^{L}\left(1-\frac{|n|}{L}\right)z^{-n}=\frac{1}{L}\left[\frac{1-z^{-L}}{1-z^{-1}}\right]^2 z^{L-1}$$

5.12 Using Euler's formula and the z-transform pair of Problem 5.9, derive the z-transforms of the signals $x(n)=R^n\cos(\omega_0 n)u(n)$ and $x(n)=R^n\sin(\omega_0 n)u(n)$.

5.13 Consider the causal sequence $x(n)=\{a_0,a_1,a_2,a_3,a_0,a_1,a_2,a_3,\cdots\}$, where the dots indicate the periodic repetition of the four samples $\{a_0,a_1,a_2,a_3\}$. Determine the z-transform of $x(n)$ and the corresponding ROC.

5.14 Using partial fraction expansions, determine the inverse z-transform of the z-transform of Problem 5.13. Verify that the sum of the PFE terms generate the periodic sequence $x(n)$ of that problem.

5.15 Consider the z-transform for $|z|>1$:

$$X(z)=1-z^{-2}+z^{-4}-z^{-6}+z^{-8}-\cdots$$

Derive a rational expression for $X(z)$ in two ways: (a) by summing the above series, and (b) by showing that it satisfies the equation $X(z)=1-z^{-2}X(z)$.

Derive also the inverse z-transform $x(n)$ for all n.

5.16 Without using partial fractions, determine the causal inverse z-transforms of:

a. $X(z)=\dfrac{1}{1+z^{-4}}$

b. $X(z)=\dfrac{1}{1-z^{-4}}$

c. $X(z)=\dfrac{1}{1+z^{-8}}$

d. $X(z)=\dfrac{1}{1-z^{-8}}$

5.17 Using partial fraction expansions, determine the inverse z-transforms of Problem 5.16. Verify that you get the same answers as in that problem.

5.18 Consider a transfer function $H(z)=N(z)/D(z)$, where the numerator and denominator polynomials have real-valued coefficients and degrees L and M in z^{-1}, and assume $L>M$. Show that $H(z)$ can be written in the form:

$$H(z)=Q(z)+\sum_{i=1}^{K}\frac{b_{i0}+z^{-1}b_{i1}}{1+a_{i1}z^{-1}+a_{i2}z^{-2}}$$

where $Q(z)$ is a polynomial of degree $L-M$ in z^{-1} and the second-order sections have real coefficients. The number of sections K is related to M by $K=M/2$ if M is even and $K=(M-1)/2$ if M is odd. This result forms the basis of the *parallel realization form* of $H(z)$.

5.19 Determine the factorization into first-order zeros of:

$$1-z^{-D}=\prod_{k=0}^{D-1}(1-z_k z^{-1})$$

$$1+z^{-D}=\prod_{k=0}^{D-1}(1-z_k z^{-1})$$

where D is an integer. What are the zeros z_k in the two cases? For $D=4$ and $D=8$, place these zeros on the z-plane with respect to the unit circle.

5.20 Given $a > 0$ and integer D, repeat the previous problem for:

$$1 - az^{-D} = \prod_{k=0}^{D-1}(1 - z_k z^{-1})$$

$$1 + az^{-D} = \prod_{k=0}^{D-1}(1 - z_k z^{-1})$$

5.21 Prove the “modulation” property of z-transforms:

$$x(n) \xrightarrow{z} X(z) \quad \Rightarrow \quad a^n x(n) \xrightarrow{z} X(z/a)$$

For $a = e^{j\omega_0}$, show that in the frequency domain this property becomes:

$$x(n) \longrightarrow X(\omega) \quad \Rightarrow \quad e^{j\omega_0 n}x(n) \longrightarrow X(\omega - \omega_0)$$

5.22 Given the DTFT equation (5.4.1), prove the inverse DTFT, Eq. (5.4.6).

5.23 Prove the Parseval equation (5.4.7).

5.24 For real-valued signals, prove the hermitian properties (5.4.8) and (5.4.9). What are the hermitian properties satisfied by the real and imaginary parts of the DTFT spectrum?

6

Transfer Functions

6.1 Equivalent Descriptions of Digital Filters

In this chapter, with the aid of z-transforms, we develop several *mathematically equivalent* ways to describe and characterize FIR and IIR filters, namely, in terms of their:

- Transfer function $H(z)$
- Frequency response $H(\omega)$
- Block diagram realization and sample processing algorithm
- I/O difference equation
- Pole/zero pattern
- Impulse response $h(n)$
- I/O convolutional equation

The most important one is the *transfer function* description because from it we can easily obtain all the others. Figure 6.1.1 shows the relationships among these descriptions. The need for such multiple descriptions is that each provides a different insight into the nature of the filter and each serves a different purpose.

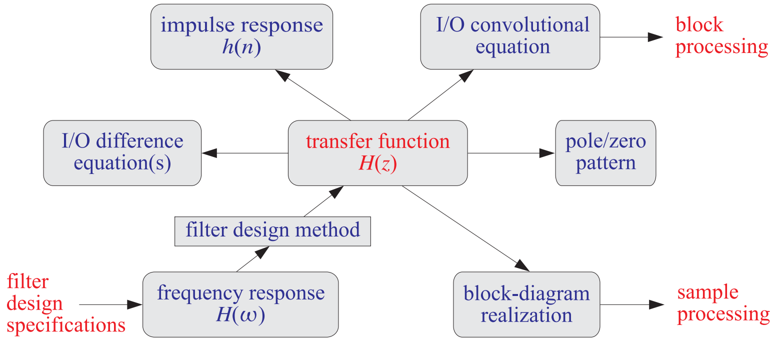


Fig. 6.1.1 Equivalent descriptions of digital filters.

In practice, a typical usage of these descriptions is to start by specifying a set of desired *frequency response specifications*, that is, the desired shape of $H(\omega)$ (lower left corner in Fig. 6.1.1). Then, through a *filter design method*, obtain a transfer function $H(z)$ that satisfies the given specifications. From $H(z)$ one can then derive a *block diagram* realization and the corresponding *sample-by-sample* processing algorithm that tells how to operate the designed filter in real time (lower right corner of Fig. 6.1.1). For an FIR filter, one can alternatively obtain the impulse response $h(n)$ and then use one of the convolution-based *block* processing methods to implement the operation of the filter (upper right corner of Fig. 6.1.1).

6.2 Transfer Functions

Here, we illustrate the central role played by the transfer function $H(z)$ of a filter by showing how to pass back and forth from one description to another.

Given a transfer function $H(z)$ one can obtain: (a) the impulse response $h(n)$, (b) the difference equation satisfied by the impulse response, (c) the I/O difference equation relating the output $y(n)$ to the input $x(n)$, (d) the block diagram realization of the filter, (e) the sample-by-sample processing algorithm, (f) the pole/zero pattern, (g) the frequency response $H(\omega)$. Conversely, given any of (a)–(g) as the starting point, one can obtain $H(z)$ and from it the rest of (a)–(g).

As an example, consider the transfer function:

$$\boxed{H(z)=\frac{5+2z^{-1}}{1-0.8z^{-1}}} \tag{6.2.1}$$

To obtain the *impulse response*, we use partial fraction expansion to write it in the form:

$$H(z)=\frac{5+2z^{-1}}{1-0.8z^{-1}}=A_0+\frac{A_1}{1-0.8z^{-1}}=-2.5+\frac{7.5}{1-0.8z^{-1}}$$

where A_0 and A_1 are obtained by:

$$\begin{aligned}A_0&=H(z)\big|_{z=0}=\frac{5+2z^{-1}}{1-0.8z^{-1}}\bigg|_{z=0}=\frac{5z+2}{z-0.8}\bigg|_{z=0}=\frac{2}{-0.8}=-2.5\\A_1&=(1-0.8z^{-1})H(z)\big|_{z=0.8}=(5+2z^{-1})\big|_{z=0.8}=5+2/0.8=7.5\end{aligned}$$

Assuming the filter is causal, we find:

$$\boxed{h(n)=-2.5\delta(n)+7.5(0.8)^n u(n)} \tag{6.2.2}$$

The *difference equation* satisfied by $h(n)$ can be obtained from $H(z)$. The standard approach is to eliminate the denominator polynomial of $H(z)$ and then transfer back to the time domain. Starting with Eq. (6.2.1) and multiplying both sides by the denominator, we find

$$(1-0.8z^{-1})H(z)=5+2z^{-1} \quad\Rightarrow\quad H(z)=0.8z^{-1}H(z)+5+2z^{-1}$$

Taking inverse z-transforms of both sides and using the linearity and delay properties, we obtain the difference equation for $h(n)$:

$$\boxed{h(n)=0.8h(n-1)+5\delta(n)+2\delta(n-1)} \tag{6.2.3}$$

It is easily verified that Eq. (6.2.2) is the causal solution, that is, the solution with the causal initial condition $h(-1)=0$. Given the impulse response $h(n)$, we can obtain the general I/O convolutional equation for the filter, that is,

$$\begin{aligned}y_n&=h_0x_n+h_1x_{n-1}+h_2x_{n-2}+h_3x_{n-3}+\cdots\\&=5x_n+7.5\big[(0.8)x_{n-1}+(0.8)^2x_{n-2}+(0.8)^3x_{n-3}+\cdots\big]\end{aligned}$$

It can be rearranged into a difference equation for $y(n)$ using the time-domain techniques of Chapter 3, as in Example 3.4.7. This difference equation can be determined very quickly using z-transforms with the aid of the z-domain equivalent of convolution:

$$Y(z)=H(z)X(z)$$

Again, the standard procedure is to eliminate denominators and go back to the time domain. For this example, we have:

$$Y(z)=H(z)X(z)=\frac{5+2z^{-1}}{1-0.8z^{-1}}X(z) \quad\Rightarrow\quad (1-0.8z^{-1})Y(z)=(5+2z^{-1})X(z)$$

which can be written as

$$Y(z)-0.8z^{-1}Y(z)=5X(z)+2z^{-1}X(z)$$

Taking inverse z-transforms of both sides, we have

$$y(n)-0.8y(n-1)=5x(n)+2x(n-1)$$

Therefore, the I/O difference equation is:

$$\boxed{y(n)=0.8y(n-1)+5x(n)+2x(n-1)} \tag{6.2.4}$$

Note that Eq. (6.2.3) is a special case of this, with $x(n)=\delta(n)$ and $y(n)=h(n)$. If the difference equation (6.2.4) was the starting point, we could obtain $H(z)$ by *reversing* all of the above steps, that is, taking z-transforms of both sides

$$\begin{aligned}&Y(z)=0.8z^{-1}Y(z)+5X(z)+2z^{-1}X(z) \quad\Rightarrow\\&(1-0.8z^{-1})Y(z)=(5+2z^{-1})X(z)\end{aligned}$$

and solving for the ratio

$$H(z)=\frac{Y(z)}{X(z)}=\frac{5+2z^{-1}}{1-0.8z^{-1}}$$

Once the I/O difference equation is determined, one can mechanize it by a *block diagram*. For example, Eq. (6.2.4) can be implemented as shown in Fig. 6.2.1. This is

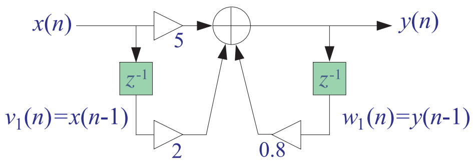


Fig. 6.2.1 Direct form realization of $H(z)$.

referred to as the *direct form realization* because it realizes directly the various terms in the right-hand side of Eq. (6.2.4).

As in the FIR case, the *sample processing algorithm* can be obtained by assigning *internal state variables* to *all* the delays that are present in the block diagram. That is, we may define

$$v_1(n)=x(n-1) \quad\Rightarrow\quad v_1(n+1)=x(n)$$

where $v_1(n)$ is the content of the x-delay at time n. Similarly, we define:

$$w_1(n)=y(n-1) \quad\Rightarrow\quad w_1(n+1)=y(n)$$

so that $w_1(n)$ is the content of the y-delay at time n. In terms of these definitions, we can replace Eq. (6.2.4) by the system of equations:

$$\begin{aligned} &\text{(compute output)} && y(n)=0.8w_1(n)+5x(n)+2v_1(n) \\ &\text{(update states)} && v_1(n+1)=x(n) \\ & && w_1(n+1)=y(n) \end{aligned}$$

It may be written as the repetitive sample processing algorithm:

```
for each input sample x do:
    $y = 0.8w_1 + 5x + 2v_1$
    $v_1 = x$
    $w_1 = y$
```

(direct form) (6.2.5)

The *frequency response* of this particular filter can be obtained by replacing z by $e^{j\omega}$ into $H(z)$. This substitution is valid here because the filter is stable and therefore its ROC, $|z|>0.8$, contains the unit circle. We find:

$$H(z)=\frac{5(1+0.4z^{-1})}{1-0.8z^{-1}} \quad\Rightarrow\quad \boxed{H(\omega)=\frac{5(1+0.4e^{-j\omega})}{1-0.8e^{-j\omega}}}$$

Using the identity

$$|1-ae^{-j\omega}|=\sqrt{1-2a\cos\omega+a^2}$$

which is valid for any real-valued a, we obtain an expression for the *magnitude response*:

$$|H(\omega)|=\frac{5\sqrt{1+0.8\cos\omega+0.16}}{\sqrt{1-1.6\cos\omega+0.64}}$$

This quantity may be plotted with the help of the pole/zero geometric pattern. The filter has a zero at $z=-0.4$ and a pole at $z=0.8$. Fig. 6.2.2 shows the pole/zero locations relative to the unit circle.

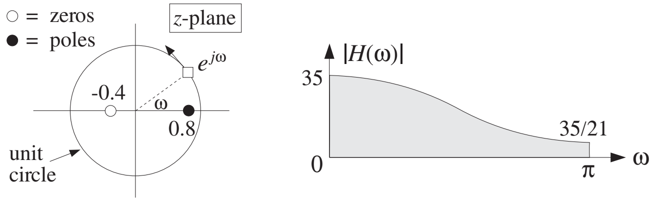


Fig. 6.2.2 Pole/zero pattern and magnitude response.

A quick *sketch* of the magnitude response $|H(\omega)|$ can be obtained by letting the point $e^{j\omega}$ trace the unit circle and drawing peaks when passing near poles and dips when passing near zeros.

The moving point $e^{j\omega}$ is nearest to the pole $z=0.8$ when $\omega=0$ and therefore there must be a peak there. Similarly, at $\omega=\pi$ there must be a dip because $e^{j\omega}$ is closest to the zero $z=-0.4$. In particular, setting $z=1$ or $\omega=0$, and $z=-1$ or $\omega=\pi$, we can calculate the actual frequency response values at the endpoints of the Nyquist interval:

$$\begin{aligned} H(\omega)\big|_{\omega=0} &= H(z)\big|_{z=1}=\frac{5+2}{1-0.8}=35 \\ H(\omega)\big|_{\omega=\pi} &= H(z)\big|_{z=-1}=\frac{5-2}{1+0.8}=\frac{5}{3}=\frac{35}{21} \end{aligned}$$

This filter acts like a *lowpass* filter because it emphasizes low frequencies and attenuates high frequencies. The highest frequency is attenuated by a factor of 21 relative to the lowest one:

$$\frac{|H(\pi)|}{|H(0)|}=\frac{1}{21}$$

or, in decibels:

$$20\log_{10}\left|\frac{H(\pi)}{H(0)}\right|=20\log_{10}\left(\frac{1}{21}\right)=-26.4\text{ dB}$$

The block diagram realization of a transfer function is not unique. Different but mathematically equivalent forms of the transfer function may lead to a different set of I/O difference equations which are implemented by a different block diagram and corresponding sample processing algorithm. For our example, the partial fraction expansion form of Eq. (6.2.1)

$$H(z)=\frac{5+2z^{-1}}{1-0.8z^{-1}}=-2.5+\frac{7.5}{1-0.8z^{-1}}$$

may be thought of as a *parallel* implementation, that is, the sum of two transfer functions

$$H(z)=H_1(z)+H_2(z)$$

where $H_1(z)=-2.5$ and $H_2(z)=7.5/(1-0.8z^{-1})$. Fig. 6.2.3 shows a block diagram implementation of this form. At first glance, it may not be obvious that the transfer function of this block diagram is the above $H(z)$.

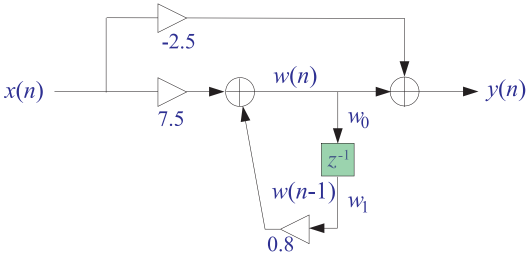


Fig. 6.2.3 Parallel form realization of $H(z)$.

To verify it, we follow the standard procedure of assigning labels, that is, names to all the signal lines that do not already have a label. The output adder has two inputs, one due to the direct connection of the input to the output through the multiplier -2.5, that is, the term $-2.5x(n)$. The other input is assigned a temporary name $w(n)$. Thus, the output adder equation becomes

$$y(n)=w(n)-2.5x(n) \tag{6.2.6}$$

The quantity $w(n)$ is recognized as the output of the filter $H_2(z)$ with input $x(n)$. The I/O difference equation of $H_2(z)$ is

$$w(n)=0.8w(n-1)+7.5x(n) \tag{6.2.7}$$

The two equations (6.2.6) and (6.2.7) together describe the operation of the block diagram in the time domain. Transforming both equations to the z-domain, we obtain

$$\begin{aligned} Y(z) &= W(z)-2.5X(z) \\ W(z) &= 0.8z^{-1}W(z)+7.5X(z) \quad\Rightarrow\quad W(z)=\frac{7.5X(z)}{1-0.8z^{-1}} \end{aligned}$$

and therefore,

$$Y(z)=W(z)-2.5X(z)=\frac{7.5X(z)}{1-0.8z^{-1}}-2.5X(z)$$

Solving for the ratio $Y(z)/X(z)$ gives the corresponding transfer function:

$$H(z)=\frac{Y(z)}{X(z)}=\frac{7.5}{1-0.8z^{-1}}-2.5$$

The sample processing algorithm corresponding to this block diagram is obtained by introducing an internal state holding the content of the delay. That is, we define

$$\begin{aligned} w_0(n) &= w(n) \\ w_1(n) &= w(n-1) \end{aligned} \quad\Rightarrow\quad w_1(n+1)=w_0(n)$$

Then, Eqs. (6.2.6) and (6.2.7) can be replaced by the system:

$$\begin{aligned} w_0(n) &= 0.8w_1(n)+7.5x(n) \\ y(n) &= w_0(n)-2.5x(n) \\ w_1(n+1) &= w_0(n) \end{aligned}$$

which can be written in the algorithmic form:

```
for each input sample x do:
    $w_0 = 0.8w_1 + 7.5x$
    $y = w_0 - 2.5x$
    $w_1 = w_0$
```

(parallel form) (6.2.8)

Other block diagram realizations can be derived by rearranging the I/O computations differently. A third realization is the so-called *canonical form realization* and is depicted in Fig. 6.2.4. It can be justified as follows. Starting with the z-domain filtering equation

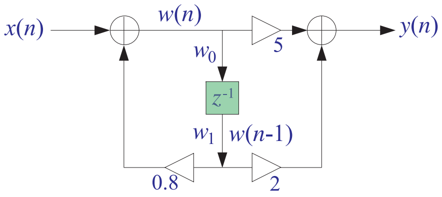


Fig. 6.2.4 Canonical form realization of $H(z)$.

$$Y(z)=H(z)X(z)=\frac{5+2z^{-1}}{1-0.8z^{-1}}X(z)$$

we separate out the effect of the filter's denominator by defining the temporary quantity

$$W(z)=\frac{1}{1-0.8z^{-1}}X(z)$$

then, the output z-transform can be computed by

$$Y(z)=(5+2z^{-1})W(z)$$

Writing these equations in the time domain, we obtain

$$W(z)=\frac{1}{1-0.8z^{-1}}X(z) \quad\Rightarrow\quad W(z)=0.8z^{-1}W(z)+X(z)$$

or,

$$w(n)=0.8w(n-1)+x(n)$$

Similarly,

$$Y(z)=5W(z)+2z^{-1}W(z) \quad\Rightarrow\quad y(n)=5w(n)+2w(n-1)$$

Thus, we obtain the system of I/O equations

$$\begin{aligned} w(n) &= 0.8w(n-1)+x(n) \\ y(n) &= 5w(n)+2w(n-1) \end{aligned}$$

which are mechanized in the block diagram of Fig. 6.2.4. Introducing internal states

$$\begin{aligned} w_0(n) &= w(n) \\ w_1(n) &= w(n-1) \end{aligned} \quad\Rightarrow\quad w_1(n+1)=w_0(n)$$

we rewrite the above system as:

$$\begin{aligned} w_0(n) &= 0.8w_1(n)+x(n) \\ y(n) &= 5w_0(n)+2w_1(n) \\ w_1(n+1) &= w_0(n) \end{aligned}$$

which can be written in the algorithmic form:

```
for each input sample x do:
    $w_0 = 0.8w_1 + x$
    $y = 5w_0 + 2w_1$
    $w_1 = w_0$
```

(canonical form) (6.2.9)

A fourth block diagram realization can be obtained by transposing the canonical realization following the transposition rules of replacing adders by nodes, nodes by adders, reversing all flows, and exchanging input with output. The resulting *transposed* realization is depicted in Fig. 6.2.5.

Again, we have assigned an internal state variable $w_1(n)$ to hold the contents of the delay register. The input to the delay is the sum $2x(n)+0.8y(n)$ which gets delayed and becomes $w_1(n)$. Thus,

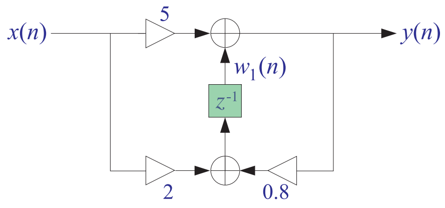


Fig. 6.2.5 Transposed realization of $H(z)$.

$$w_1(n)=2x(n-1)+0.8y(n-1)$$

The complete I/O description of this realization is then given by the system:

$$\begin{aligned} y(n) &= w_1(n)+5x(n) \\ w_1(n+1) &= 2x(n)+0.8y(n) \end{aligned}$$

which translates to the following sample processing algorithm:

```
for each input sample x do:
    $y = w_1 + 5x$
    $w_1 = 2x + 0.8y$
```

(transposed form) (6.2.10)

To verify that this realization describes the same transfer function, we transform the I/O equations to the z-domain:

$$\begin{aligned} Y(z) &= W_1(z)+5X(z) \\ zW_1(z) &= 2X(z)+0.8Y(z) \end{aligned}$$

Then, solve the second for $W_1(z)$, insert it in the first, and solve for the ratio $Y(z)/X(z)$. We have:

$$W_1(z)=0.8z^{-1}Y(z)+2z^{-1}X(z)$$

and

$$Y(z)=W_1(z)+5X(z)=0.8z^{-1}Y(z)+2z^{-1}X(z)+5X(z)$$

which gives

$$H(z)=\frac{Y(z)}{X(z)}=\frac{5+2z^{-1}}{1-0.8z^{-1}}$$

Given a particular block diagram implementation, one can easily translate the corresponding sample processing algorithm into a software or hardware routine. For example, the canonical form of Eq. (6.2.9) can be implemented by the following C routine `filter.c`:

```
/* filter.c - IIR example routine */

double filter(x, w)                         usage: y = filter(x, w);
double x, *w;
{
       double y;

       w[0] = 0.8 * w[1] + x;

       y = 5 * w[0] + 2 * w[1];             compute output

       w[1] = w[0];                         update internal state

       return y;
}
```

The array w must be declared to be a two-dimensional array in the main program. The following program segment illustrates the usage of this routine for processing N input samples:

```
w = (double *) calloc(2, sizeof(double));

for (n=0; n<N; n++)
       y[n] = filter(x[n], w);
```

The internal state array w must be initialized to zero prior to the first call of `filter`. This is indirectly accomplished by `calloc` during the allocation of w.

Our aim in this example was to show not only how to pass from one filter description to another using z-transforms, but also to illustrate how different block diagram realizations correspond to different but equivalent ways of arranging the required I/O filtering equations. A more systematic discussion of filter realizations will be presented in the next chapter.

In general, the transfer function of an IIR filter is given as the ratio of two polynomials of degrees, say L and M:

$$\boxed{H(z)=\frac{N(z)}{D(z)}=\frac{b_0+b_1z^{-1}+b_2z^{-2}+\cdots+b_Lz^{-L}}{1+a_1z^{-1}+a_2z^{-2}+\cdots+a_Mz^{-M}}} \qquad \text{(IIR)} \tag{6.2.11}$$

Note that by convention, the 0th coefficient of the denominator polynomial has been set to unity $a_0=1$. The filter $H(z)$ will have L zeros and M poles. Assuming that the numerator and denominator coefficients are real-valued, then if any of the zeros or poles are complex, they must come in conjugate pairs.

To determine the impulse response $h(n)$ of such a filter, we may use the inverse z-transform techniques of Chapter 5, such as partial fraction expansions. The relative locations of the poles on the z-plane will divide the plane into non-overlapping regions which may be taken as the possible ROCs for $h(n)$.

In particular, to get a stable impulse response, we must pick the ROC that contains the unit circle. Recall that in order for the stable $h(n)$ to also be causal, *all poles* of $H(z)$, that is, the zeros of $D(z)$, must lie strictly *inside* the unit circle. Then, the ROC for inverting $H(z)$ will be the outside of the unit circle.

As the above example showed, there are many different, but mathematically equivalent, I/O difference equations describing such a filter—each leading to a particular block diagram implementation and sample processing algorithm. The simplest one is the direct form obtained by writing

$$Y(z)=H(z)X(z)=\frac{b_0+b_1z^{-1}+b_2z^{-2}+\cdots+b_Lz^{-L}}{1+a_1z^{-1}+a_2z^{-2}+\cdots+a_Mz^{-M}}X(z)$$

then, multiplying by the denominator:

$$(1+a_1z^{-1}+\cdots+a_Mz^{-M})Y(z)=(b_0+b_1z^{-1}+\cdots+b_Lz^{-L})X(z)$$

and finally, transforming back to the time domain:

$$\boxed{y_n+a_1y_{n-1}+\cdots+a_My_{n-M}=b_0x_n+b_1x_{n-1}+\cdots+b_Lx_{n-L}} \tag{6.2.12}$$

It can also be written as:

$$y_n=-a_1y_{n-1}-\cdots-a_My_{n-M}+b_0x_n+b_1x_{n-1}+\cdots+b_Lx_{n-L}$$

Note also that if the denominator coefficients are zero, that is, $a_i=0,\ i=1,2,\dots,M$, the denominator polynomial is trivial $D(z)=1$ and $H(z)$ becomes equal to the numerator polynomial $H(z)=N(z)$, that is, an FIR filter:

$$\boxed{H(z)=N(z)=b_0+b_1z^{-1}+b_2z^{-2}+\cdots+b_Lz^{-L}} \qquad \text{(FIR)} \tag{6.2.13}$$

In this case, the difference equation (6.2.12) becomes the usual I/O convolutional equation for an FIR filter:

$$\boxed{y_n=b_0x_n+b_1x_{n-1}+\cdots+b_Lx_{n-L}} \qquad \text{(FIR I/O equation)} \tag{6.2.14}$$

Various implementations of the FIR case were discussed in Chapter 4. The implementations of the IIR case will be discussed in detail in Chapter 7.

Next, we present some further examples. In each case, we determine the transfer function, impulse response, frequency response, pole/zero pattern, block diagram realization and sample processing algorithm.

Example 6.2.1: Determine the transfer function of the following third-order FIR filter with impulse response:

$$\mathbf{h}=[1,6,11,6]$$

Solution: The filter's I/O equation is

$$y(n)=x(n)+6x(n-1)+11x(n-2)+6x(n-3)$$

The z-transform of the finite impulse response sequence is:

$$H(z)=h_0+h_1z^{-1}+h_2z^{-2}+h_3z^{-3}=1+6z^{-1}+11z^{-2}+6z^{-3}$$

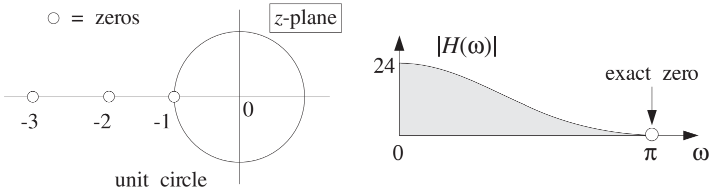
Noting that $H(z)$ has a zero at $z = -1$, we may factor it in the form

$$H(z)= (1 + z^{-1})(1 + 5z^{-1} + 6z^{-2})= (1 + z^{-1})(1 + 2z^{-1})(1 + 3z^{-1})$$

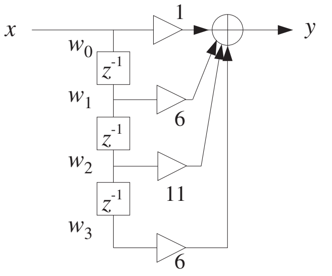
The corresponding frequency response is obtained by the substitution $z = e^{j\omega}$:

$$H(\omega)= 1 + 6e^{-j\omega} + 11e^{-2j\omega} + 6e^{-3j\omega} = (1 + e^{-j\omega})(1 + 2e^{-j\omega})(1 + 3e^{-j\omega})$$

The filter has zeros at $z = -1, -2, -3$. The pole/zero pattern is shown below together with a sketch of the magnitude response $|H(\omega)|$. (The multiple pole at the origin $z = 0$ is not shown.)



The filter tends to attenuate high frequencies, that is, it will act as a lowpass filter. The filter vanishes exactly at $z = -1$ or $\omega = \pi$. At $\omega = 0$ or $z = 1$, it is equal to $1 + 6 + 11 + 6 = 24$. The block diagram realization and the sample-by-sample processing algorithm are:



```
for each input sample x do:
    $w_0 = x$
    $y = w_0 + 6w_1 + 11w_2 + 6w_3$
    $w_3 = w_2$
    $w_2 = w_1$
    $w_1 = w_0$
```

The block diagram and sample processing algorithm correspond to the FIR direct form discussed in Chapter 4. □

Example 6.2.2: An FIR filter is described by the I/O equation:

$$y(n)= x(n)-x(n-4)$$

Determine its transfer function $H(z)$ and impulse response $h(n)$.

Solution: The I/O equation becomes in the z-domain:

$$Y(z)= X(z)-z^{-4}X(z) \quad\Rightarrow\quad H(z)= \frac{Y(z)}{X(z)} = 1 - z^{-4}$$

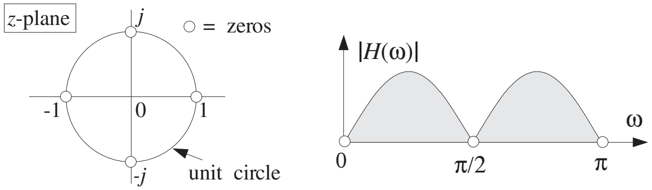
It follows that $\mathbf{h} = [1, 0, 0, 0, -1]$. The frequency response is obtained by setting $z = e^{j\omega}$:

$$H(\omega)= 1 - e^{-4j\omega} = (e^{2j\omega} - e^{-2j\omega})e^{-2j\omega} = 2j\sin(2\omega)e^{-2j\omega}$$

Thus, its magnitude response will be $|H(\omega)| = 2|\sin(2\omega)|$. The zeros of $H(z)$ are the fourth roots of unity, obtained by solving $1 - z^{-4} = 0$ or,

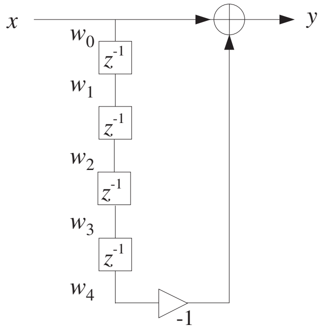
$$z^4 = 1 \quad\Rightarrow\quad z = e^{2\pi jk/4},\ k = 0, 1, 2, 3 \quad\Rightarrow\quad z = 1, j, -1, -j$$

Thus, the magnitude response $|H(\omega)|$ will vanish at $\omega = 2\pi k/4 = 0, \pi/2, \pi, 3\pi/2$, for $k = 0, 1, 2, 3$, as shown below:



The magnitude response $|H(\omega)|$ is plotted only over the right half of the Nyquist interval, $0 \le \omega \le \pi$, and therefore the zero at $\omega = 3\pi/2$ is not shown—it gets aliased to the negative side: $3\pi/2 - 2\pi = -\pi/2$.

The block diagram realization of the direct form and the corresponding sample processing algorithm are as shown:



```
for each input sample x do:
    $w_0 = x$
    $y = w_0 - w_4$
    $w_4 = w_3$
    $w_3 = w_2$
    $w_2 = w_1$
    $w_1 = w_0$
```

This is a special case of a comb filter with notches at the four frequencies $\omega = 2\pi k/4$, $k = 0, 1, 2, 3$. Comb filters and their applications will be discussed in Chapter 16. □

Example 6.2.3: Determine the transfer function and causal impulse response of the two filters described by the difference equations:

(a) $y(n)= 0.25y(n-2)+x(n)$

(b) $y(n)= -0.25y(n-2)+x(n)$

Solution: For case (a), we take z-transforms of both sides of the difference equation to get:

$$Y(z) = 0.25z^{-2}Y(z) + X(z)$$

Solving for $Y(z)/X(z)$ we find the transfer function:

$$H(z) = \frac{1}{1-0.25z^{-2}} = \frac{A_1}{1-0.5z^{-1}} + \frac{A_2}{1+0.5z^{-1}}$$

with $A_1 = A_2 = 0.5$. Thus, the causal impulse response will be:

$$h(n) = A_1(0.5)^n u(n) + A_2(-0.5)^n u(n)$$

The pole at $z = 0.5$ is in the low-frequency part of the unit circle, and the pole $z = -0.5$ is in the high-frequency part. Thus, the filter will tend to enhance both the low and high frequencies, that is, it will behave as a 2-band bandpass filter, or as a bandstop filter—attenuating the intermediate frequencies between low and high.

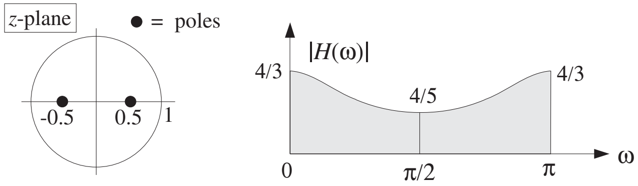
Indeed, the value of $H(z)$ at $\omega = 0, \pi$ or $z = \pm 1$ is

$$H(0) = H(\pi) = H(z)\big|_{z=\pm 1} = \frac{1}{1-0.25} = \frac{4}{3}$$

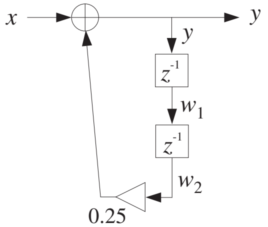
which is larger than the value of $H(z)$ at the intermediate frequency $\omega = \pi/2$ or $z = j$ or $z^2 = -1$, that is, the value

$$H(\pi/2) = H(z)\big|_{z=j} = \frac{1}{1-0.25(-1)} = \frac{4}{5}$$

The pole/zero pattern and magnitude spectra are shown below. The peaks at the high-/low-frequency ends are not too high because the poles are not too close to the unit circle.



The block diagram implementation of the given difference equation and corresponding sample processing algorithm are:



```
for each input sample x do:
    $y = 0.25w_2 + x$
    $w_2 = w_1$
    $w_1 = y$
```

For case (b), the difference equation becomes in the z-domain:

$$Y(z) = -0.25z^{-2}Y(z) + X(z)$$

which can be solved for $Y(z)/X(z)$ to give:

$$H(z) = \frac{1}{1+0.25z^{-2}} = \frac{A_1}{1-0.5jz^{-1}} + \frac{A_1^*}{1+0.5jz^{-1}}$$

with $A_1 = 0.5$. Notice the poles are conjugate pairs as are the PF expansion coefficients. The causal impulse response will be:

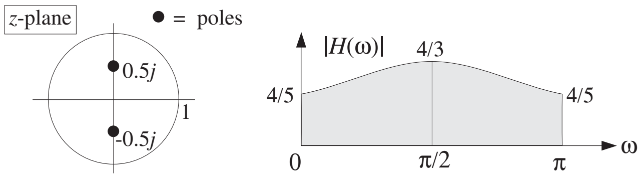
$$h(n) = A_1(0.5j)^n u(n) + A_1^*(-0.5j)^n u(n)$$

which can be written in the exponentially decaying form:

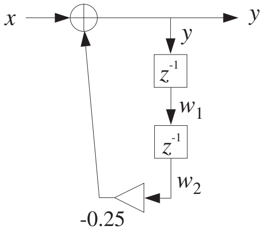
$$\begin{aligned} h(n) &= 2\mathrm{Re}[A_1(0.5)^n j^n]u(n) = 2\mathrm{Re}[0.5(0.5)^n e^{j\pi n/2}]u(n) \\ &= (0.5)^n \cos(\pi n/2)u(n) \end{aligned}$$

The two conjugate poles are in the "midfrequency" range, $z = \pm 0.5j = 0.5e^{\pm j\pi/2}$. Thus, the filter will emphasize the middle frequencies, that is, it will act as a bandpass filter.

Again, the value of the magnitude response at $\omega = \pi/2$ or $z = j$ or $z^2 = -1$ is $1/(1+0.25(-1)) = 4/3$, whereas the value at $\omega = 0, \pi$ or $z = \pm 1$ or $z^2 = 1$ is $1/(1+0.25) = 4/5$.



The block diagram and corresponding sample processing algorithm are:



```
for each input sample x do:
    $y = -0.25w_2 + x$
    $w_2 = w_1$
    $w_1 = y$
```

The two cases differ by a simple change of sign in the difference equation coefficient 0.25 which leads to drastically different pole locations and frequency responses. □

6.3 Sinusoidal Response

6.3.1 Steady-State Response

The response of a filter to an input sinusoidal signal is referred to as the *sinusoidal response*. Knowing what happens to sinusoids under filtering is important because they are the elementary building blocks of more complicated signals.

Consider an infinitely long, double-sided, complex sinusoid of frequency ω_0 which is applied to the input of a stable filter $h(n)$:

$$x(n)=e^{j\omega_0 n},\quad -\infty<n<\infty$$

The resulting output can be determined in two ways: (1) using convolution in the time domain, or (2) using multiplication in the frequency domain. Using the first method, we have

$$y(n)=\sum_m h(m)x(n-m)=\sum_m h(m)e^{j(n-m)\omega_0}=e^{j\omega_0 n}\sum_m h(m)e^{-j\omega_0 m},\qquad \text{or,}$$

$$y(n)=H(\omega_0)e^{j\omega_0 n} \tag{6.3.1}$$

where $H(\omega_0)$ is the frequency response of the filter evaluated at $\omega=\omega_0$:

$$H(\omega_0)=\sum_m h(m)e^{-j\omega_0 m}$$

Using the frequency-domain method, we start with the spectrum of the input signal, namely,

$$X(\omega)=2\pi\delta(\omega-\omega_0)+\text{(replicas)}$$

Then, using the frequency-domain multiplication formula (5.4.10), we obtain (the first replica of) the spectrum of the output:

$$Y(\omega)=H(\omega)X(\omega)=H(\omega)2\pi\delta(\omega-\omega_0)=2\pi H(\omega_0)\delta(\omega-\omega_0)$$

where ω was replaced by ω_0 in the argument of $H(\omega)$, because the delta function $\delta(\omega-\omega_0)$ forces $\omega=\omega_0$. Putting $Y(\omega)$ into the inverse DTFT formula (5.4.6), we find:

$$y(n)=\frac{1}{2\pi}\int_{-\pi}^{\pi}Y(\omega)e^{j\omega n}\,d\omega=\frac{1}{2\pi}\int_{-\pi}^{\pi}2\pi H(\omega_0)\delta(\omega-\omega_0)e^{j\omega n}\,d\omega$$

The presence of $\delta(\omega-\omega_0)$ causes the integrand to be evaluated at ω_0 resulting in Eq. (6.3.1). To summarize, an infinite double-sided input sinusoid of frequency ω_0 reappears at the output *unchanged* in frequency but modified by the frequency response factor $H(\omega_0)$:

$$\boxed{e^{j\omega_0 n}\xrightarrow{H}H(\omega_0)e^{j\omega_0 n}} \tag{6.3.2}$$

Because $H(\omega)$ is a complex-valued quantity, we can write it in terms of its magnitude and phase as:

$$H(\omega)=|H(\omega)|e^{j\arg H(\omega)}$$

Therefore, Eq. (6.3.2) can be written in the form:

$$\boxed{e^{j\omega_0 n}\xrightarrow{H}|H(\omega_0)|e^{j\omega_0 n+j\arg H(\omega_0)}} \tag{6.3.3}$$

which shows that the filter introduces both a *magnitude* modification by an amount $|H(\omega_0)|$, as well as a *relative phase shift* by an amount $\arg H(\omega_0)$. Taking real or imaginary parts of both sides of this result, we obtain the cosine and sine versions:

$$\boxed{\begin{aligned}\cos(\omega_0 n)&\xrightarrow{H}|H(\omega_0)|\cos\bigl(\omega_0 n+\arg H(\omega_0)\bigr)\\ \sin(\omega_0 n)&\xrightarrow{H}|H(\omega_0)|\sin\bigl(\omega_0 n+\arg H(\omega_0)\bigr)\end{aligned}} \tag{6.3.4}$$

Figure 6.3.1 illustrates this result. Note that the phase shift corresponds to the translation of the sinewave as a whole by an amount $\arg H(\omega_0)$ relative to the input sinewave. Typically, $\arg H(\omega_0)$ is negative and therefore it represents a time delay, that is, translation to the right.

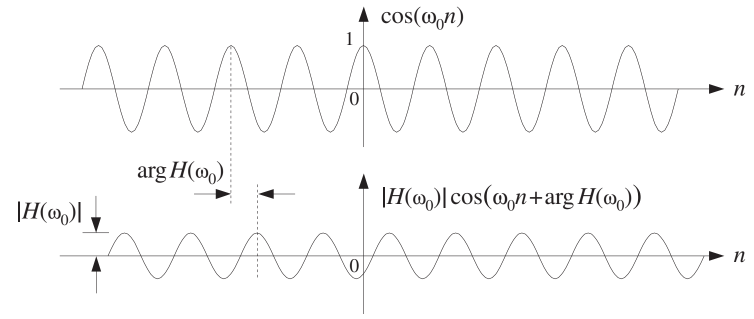


Fig. 6.3.1 Magnitude and phase-shift modification introduced by filtering.

The filtering result (6.3.2) is one of the most fundamental results in signal processing. It essentially justifies the use of LTI filters and explains their widespread application. By proper design of the frequency response shape $H(\omega)$, it allows complete control over the frequency content of the input signal.

Using the linearity property of the filter, we can apply Eq. (6.3.2) to a linear combination of two input sinusoids of frequencies ω_1 and ω_2, resulting in the same linear combination of the corresponding outputs, that is,

$$A_1e^{j\omega_1 n}+A_2e^{j\omega_2 n}\xrightarrow{H}A_1H(\omega_1)e^{j\omega_1 n}+A_2H(\omega_2)e^{j\omega_2 n}$$

which shows that the effect of filtering is to change the *relative amplitudes* and phases of the two sinusoids from the values $\{A_1,A_2\}$ to the values $\{A_1H(\omega_1),A_2H(\omega_2)\}$. In the frequency domain, we have for the spectra of the input and output signals:

$$A_1\delta(\omega-\omega_1)+A_2\delta(\omega-\omega_2)\xrightarrow{H}A_1H(\omega_1)\delta(\omega-\omega_1)+A_2H(\omega_2)\delta(\omega-\omega_2)$$

where for simplicity, we dropped a common factor of 2π. Figure 6.3.2 shows the input and output spectra and illustrates how the filter alters the relative balance by multiplication by the appropriate frequency response factors.

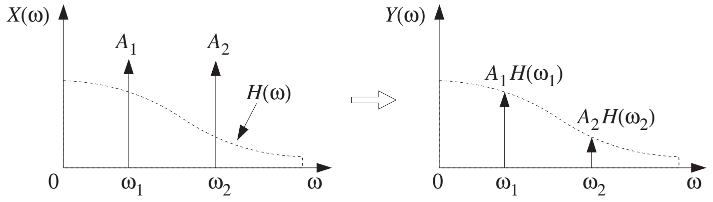


Fig. 6.3.2 Relative amplitudes before and after filtering.

If one of the sinusoids, say ω_1, were a *desired signal* and the other an unwanted *interference*, one could design a filter to remove the interference. For example, the choice:

$$H(\omega_1)=1,\quad H(\omega_2)=0$$

would leave the desired signal unaffected and remove the interference. The resulting output signal would be in this case:

$$y(n)=A_1H(\omega_1)e^{j\omega_1 n}+A_2H(\omega_2)e^{j\omega_2 n}=A_1e^{j\omega_1 n}$$

A more general input $x(n)$ with a more complicated spectrum $X(\omega)$ can be resolved into its sinusoidal components by the inverse DTFT formula:

$$x(n)=\frac{1}{2\pi}\int_{-\pi}^{\pi}X(\omega)e^{j\omega n}\,d\omega$$

The filter $H(\omega)$ reshapes the input spectrum $X(\omega)$ into the output spectrum by $Y(\omega)=H(\omega)X(\omega)$. It changes, in a controlled manner, the relative amplitudes and phases of the various frequency components of the input signal. The resulting output signal can be reconstructed from the inverse DTFT formula:

$$y(n)=\frac{1}{2\pi}\int_{-\pi}^{\pi}Y(\omega)e^{j\omega n}\,d\omega=\frac{1}{2\pi}\int_{-\pi}^{\pi}H(\omega)X(\omega)e^{j\omega n}\,d\omega \tag{6.3.5}$$

Another useful filtering concept is that of the *phase delay* defined in terms of the phase response $\arg H(\omega)$ as follows:

$$\boxed{d(\omega)=-\frac{\arg H(\omega)}{\omega}}\quad\Rightarrow\quad \arg H(\omega)=-\omega d(\omega) \tag{6.3.6}$$

Similarly, the *group delay* of a filter is defined as:

$$\boxed{d_g(\omega)=-\frac{d}{d\omega}\arg H(\omega)} \tag{6.3.7}$$

The sinusoidal response of Eqs. (6.3.2) or (6.3.3) can be expressed in terms of the phase delay as follows:

$$e^{j\omega n}\xrightarrow{H}|H(\omega)|e^{j\omega(n-d(\omega))} \tag{6.3.8}$$

which shows that different frequency components get delayed by different amounts, depending on the filter's phase delay.

Linear phase filters have the property that their phase delay $d(\omega)$ is independent of frequency, say $d(\omega)=D$, so that the phase response is linear in ω, $\arg H(\omega)=-\omega D$. Such filters cause every frequency component to be delayed by the *same* amount D, thus corresponding to an overall delay in the output:

$$e^{j\omega n}\xrightarrow{H}|H(\omega)|e^{j\omega(n-D)} \tag{6.3.9}$$

This overall delay can also be seen by the inverse DTFT formulas:

$$x(n)=\int_{-\pi}^{\pi}X(\omega)e^{j\omega n}\frac{d\omega}{2\pi}\quad\xrightarrow{H}\quad y(n)=\int_{-\pi}^{\pi}|H(\omega)|X(\omega)e^{j\omega(n-D)}\frac{d\omega}{2\pi}$$

The design of FIR linear phase filters will be discussed in Chapter 11. IIR filters that have linear phase over the entire Nyquist interval cannot be designed. However, they can be designed to have approximately linear phase over their *passband* (for example, Bessel filters).

6.3.2 Transient Response

In obtaining the result (6.3.2), we assumed that the input sinusoid had been on for a very long time (since $n=-\infty$), and therefore, Eq. (6.3.2) represents the *steady-state output* resulting after all the filter transients have died out.

In practice, we typically begin processing an input signal at some instant of time, say $n=0$, and therefore, we must deal with the input-on transients, as well as the input-off transients taking place after the input is turned off. Figure 6.3.3 shows the difference between a double-sided sinewave and a causal one that is turned on at $n=0$.

If we start generating and filtering an input sinewave at $n=0$, the filter will not "know" immediately that its input is sinusoidal. It takes the filter a certain period of time to settle into its sinusoidal behavior given by Eq. (6.3.2). The analysis of the filter's response in this case can be carried out using z-transforms. Consider the causal sinusoidal input and its z-transform:

$$\boxed{x(n)=e^{j\omega_0 n}u(n)\xrightarrow{Z}X(z)=\frac{1}{1-e^{j\omega_0}z^{-1}}}$$

having ROC $|z|>|e^{j\omega_0}|=1$. Assume a filter of the form:

$$H(z)=\frac{N(z)}{D(z)}=\frac{N(z)}{(1-p_1z^{-1})(1-p_2z^{-1})\cdots(1-p_Mz^{-1})}$$

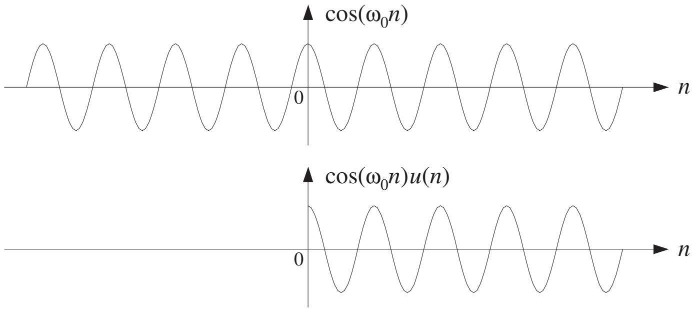


Fig. 6.3.3 Double-sided and one-sided sinewaves.

with M poles that lie strictly within the unit circle, so that the filter is stable and causal. The output z-transform will be:

$$Y(z)=H(z)X(z)=\frac{N(z)}{(1-e^{j\omega_0}z^{-1})(1-p_1z^{-1})(1-p_2z^{-1})\cdots(1-p_Mz^{-1})}$$

Assuming that the degree of the numerator polynomial $N(z)$ is strictly less than the degree $M+1$ of the denominator,[†] we can write the PF expansion:

$$Y(z)=\frac{C}{1-e^{j\omega_0}z^{-1}}+\frac{B_1}{1-p_1z^{-1}}+\frac{B_2}{1-p_2z^{-1}}+\cdots+\frac{B_M}{1-p_Mz^{-1}}$$

The PF expansion coefficients are obtained in the usual fashion, via Eq. (5.5.2). In particular, the coefficient of the first term will be:

$$C=(1-e^{j\omega_0}z^{-1})Y(z)\big|_{z=e^{j\omega_0}}=\left[(1-e^{j\omega_0}z^{-1})\frac{H(z)}{1-e^{j\omega_0}z^{-1}}\right]_{z=e^{j\omega_0}}$$

Canceling the $(1-e^{j\omega_0}z^{-1})$ factors, we find that C is none other than the frequency response $H(\omega)$ evaluated at $\omega=\omega_0$, that is,

$$\boxed{C=H(z)\big|_{z=e^{j\omega_0}}=H(\omega_0)} \tag{6.3.10}$$

Therefore, the PF expansion will read:

$$\boxed{Y(z)=\frac{H(\omega_0)}{1-e^{j\omega_0}z^{-1}}+\frac{B_1}{1-p_1z^{-1}}+\frac{B_2}{1-p_2z^{-1}}+\cdots+\frac{B_M}{1-p_Mz^{-1}}}$$

Taking the causal inverse z-transform (with ROC $|z|>1$), we find for $n\geq 0$:

$$\boxed{y(n)=H(\omega_0)e^{j\omega_0n}+B_1p_1^n+B_2p_2^n+\cdots+B_Mp_M^n} \tag{6.3.11}$$

Because the filter was assumed to have all its poles inside the unit circle, namely, $|p_i|<1$, it follows that in the limit of large n, the p_i^n terms will drop to zero exponentially giving the steady-state output:

[†] This assumption is not critical. The same conclusions can be drawn otherwise.

$$y(n)\to H(\omega_0)e^{j\omega_0n}\quad\text{as}\quad n\to\infty$$

For smaller values of n, Eq. (6.3.11) gives the transient response of the filter.

Example 6.3.1: Determine the full transient response of the filter

$$H(z)=\frac{5+2z^{-1}}{1-0.8z^{-1}}$$

for a causal complex sinusoidal input of frequency ω_0.

Solution: We have the partial fraction expansion for the output z-transform $Y(z)=H(z)X(z)$:

$$Y(z)=\frac{5+2z^{-1}}{(1-e^{j\omega_0}z^{-1})(1-0.8z^{-1})}=\frac{H(\omega_0)}{1-e^{j\omega_0}z^{-1}}+\frac{B_1}{1-0.8z^{-1}}$$

where the coefficient B_1 is found by

$$B_1=(1-0.8z^{-1})Y(z)\big|_{z=0.8}=\left[\frac{5+2z^{-1}}{1-e^{j\omega_0}z^{-1}}\right]_{z=0.8}=\frac{7.5}{1-1.25e^{j\omega_0}}$$

The causal inverse z-transform will be:

$$y(n)=H(\omega_0)e^{j\omega_0n}+B_1(0.8)^n,\qquad n\geq 0$$

For large n, the term $(0.8)^n$ drops to zero and the output settles into its steady-state sinusoidal response

$$y(n)\to H(\omega_0)e^{j\omega_0n}$$

where $H(\omega_0)=\dfrac{5+2e^{-j\omega_0}}{1-0.8e^{-j\omega_0}}$. □

There are four straightforward conclusions that can be drawn from Eq. (6.3.11). First, it shows clearly the requirement of *stability* for the filter. If any of the filter poles, say p_1, were outside the unit circle, such that $|p_1|>1$, the term p_1^n would be unstable, diverging as $n\to\infty$. This term would dominate completely the rest of terms of Eq. (6.3.11) and there would be no steady-state response. (Of course, we know that in this case the series definition, Eq. (5.4.3), of the frequency response $H(\omega)$ does not converge because the unit circle does not lie in the causal region of convergence $|z|>|p_1|>1$.)

Second, assuming the filter is strictly stable, all the transient terms p_i^n will drop to zero exponentially. But some of them will drop to zero faster than others. The effective *time constant* to reach the sinusoidal steady state is dictated by the *slowest* converging pole term, that is, the term with the *largest* magnitude, namely, max $|p_i|$. Equivalently, this is the pole that lies *closest* to the unit circle (from the inside). Denoting the maximum pole magnitude by

$$\rho=\max_i|p_i|$$

we may define the effective time constant to be the time n_{eff} at which the quantity ρ^n has dropped below a certain small value, for example, when it drops below 1% its initial value. We can make this definition more quantitative by defining n_{eff} such that:

$$\rho^{n_{\text{eff}}} = \epsilon$$

where ϵ is the desired level of smallness, for example, $\epsilon = 1\% = 0.01$. It follows that:

$$\boxed{n_{\text{eff}} = \frac{\ln\epsilon}{\ln\rho} = \frac{\ln(1/\epsilon)}{\ln(1/\rho)}} \qquad \text{(time constant)} \tag{6.3.12}$$

Because both ϵ and ρ are less than one, their logs are negative, but the ratio is positive. In the last expression, we have the ratio of two positive numbers. The effective time constant n_{eff} becomes larger if the slowest pole is pushed closer to the unit circle, that is, increasing ρ toward one, and also if we require a smaller threshold ϵ.

The value $\epsilon = 1\%$ corresponds to the amplitude of the filter's output falling by a factor of 10^{-2} or 40 dB. The time constant in seconds, $\tau = n_{\text{eff}}T$, is referred to as the 40-dB time constant. In the study of reverberation properties of concert halls, the 60-dB time constants are used, which correspond to $\epsilon = 0.1\% = 10^{-3}$.

In conclusion, the *speed of response* of a stable and causal IIR filter is controlled by the poles nearest to the unit circle. The filter is *slow* reaching steady state if its poles are *near* the unit circle, and fast if they are further away (toward the center).

Example 6.3.2: A sinusoid of frequency $\omega_0 = 0.1\pi$ and duration of 300 samples, that is, $x(n) = \sin(\omega_0 n)$, $0 \le n < 300$, is input to a (causal) filter with transfer function

$$H(z) = \frac{b}{1 - az^{-1}}$$

where $a = 0.97$. Determine the 1% time constant of this filter. Adjust the scale factor b such that the filter's gain at ω_0 is unity. Determine and plot the output of the filter $y(n)$ over the interval $0 \le n < 450$, by iterating the difference equation of the filter.

Solution: The 1% time constant of this filter is computed from Eq. (6.3.12),

$$n_{\text{eff}} = \frac{\ln\epsilon}{\ln a} = \frac{\ln(0.01)}{\ln(0.97)} = 151.2 \ \text{ samples}$$

The frequency and magnitude responses are

$$H(\omega) = \frac{b}{1 - ae^{-j\omega}} \quad \Rightarrow \quad |H(\omega)| = \frac{b}{\sqrt{1 - 2a\cos\omega + a^2}}$$

The requirement that $|H(\omega_0)| = 1$ leads to the condition on b:

$$|H(\omega_0)| = \frac{b}{\sqrt{1 - 2a\cos\omega_0 + a^2}} = 1 \quad \Rightarrow \quad b = \sqrt{1 - 2a\cos\omega_0 + a^2} = 0.3096$$

The value of the frequency response at ω_0 becomes then,

$$H(\omega_0) = \frac{b}{1 - ae^{-j\omega_0}} = 0.2502 - 0.9682j = 1 \cdot e^{-j1.3179}$$

so that its phase response will be $\arg H(\omega_0) = -1.3179$ radians. The resulting output $y(n)$, shown in Fig. 6.3.4, was computed by the following program segment, which implements the difference equation $y(n) = ay(n-1) + bx(n)$ and sample processing algorithm of this filter:

```
for (y1=0, n=0; n<450; n++) {
        if (n < 300)
                x = sin(w0 * n);
        else
                x = 0;
        y[n] = a * y1 + b * x;                 /* y1 = y[n-1] */
        y1 = y[n];
        }
```

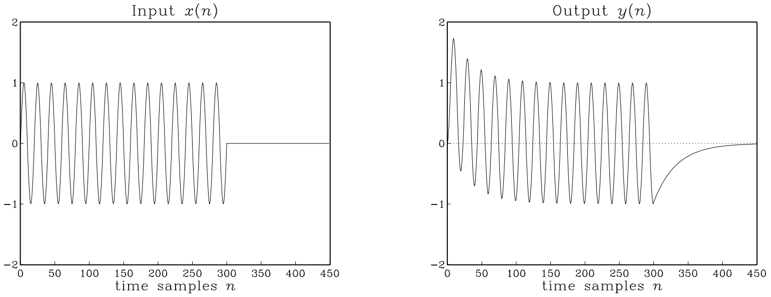


Fig. 6.3.4 Input and output of Example 6.3.2.

Notice the input-on and input-off transients, each lasting approximately $n_{\text{eff}} = 151$ time samples. The time interval $150 \le n \le 300$ corresponds to the steady-state period, during which the output settles to its sinusoidal behavior according to Eq. (6.3.4). The amplitude of the steady-state output is unity because $|H(\omega_0)| = 1$. There is a slight phase delay relative to the input, due to the negative value of the phase response $\arg H(\omega_0) = -1.3179$ radians. □

Example 6.3.3: Derive closed-form expressions for the output $y(n)$ of the previous example in two ways: (a) working with convolution in the time domain, and (b) working with z-transforms.

Solution: It proves convenient to work with the complex-valued version of the sinusoid, $x(n) = e^{j\omega_0 n}$, $0 \le n \le 299$, and take imaginary parts of the answer at the end. Using convolution, we obtain

$$y(n) = \sum_{m=\max(0,n-299)}^{n} h(m)x(n-m) = \sum_{m=\max(0,n-299)}^{n} ba^m e^{j\omega_0(n-m)}$$

where we used the impulse response $h(m) = ba^m u(m)$. The summation limits were obtained by the requirements that the indices of $h(m)$ and $x(n-m)$ do not exceed the array bounds, that is,

$$0 \le m < \infty, \qquad 0 \le n - m \le 299$$

which are equivalent to $\max(0, n-299) \le m \le n$. It follows that if $0 \le n \le 299$, then

$$y(n) = e^{j\omega_0 n}\sum_{m=0}^{n} b a^m e^{-j\omega_0 m} = e^{j\omega_0 n} b\,\frac{1 - a^{n+1}e^{-j\omega_0(n+1)}}{1 - ae^{-j\omega_0}}$$

Setting $H_0 = H(\omega_0) = b/(1 - ae^{-j\omega_0})$ and noting that $(1 - ae^{-j\omega_0})H_0 = b$ which gives $b - H_0 = -H_0 a e^{-j\omega_0}$, we obtain

$$y(n) = H_0 e^{j\omega_0 n} + (b - H_0)a^n, \qquad \text{for } \ 0 \le n \le 299$$

On the other hand, if $300 \le n \le 449$, then the limits of summation become:

$$y(n) = e^{j\omega_0 n}\sum_{m=n-299}^{n} b a^m e^{-j\omega_0 m}$$

and we must use the finite geometric series formula

$$\sum_{m=m_1}^{m_2} x^m = \frac{x^{m_1} - x^{m_2+1}}{1 - x} \tag{6.3.13}$$

with $x = ae^{-j\omega_0}$, to obtain:

$$y(n) = H_0 e^{j\omega_0 n}\big[a^{n-299}e^{-j\omega_0(n-299)} - a^{n+1}e^{-j\omega_0(n+1)}\big]$$

Noting that the factor $e^{j\omega_0 n}$ cancels, we obtain

$$y(n) = H_0\big[a^{n-299}e^{j299\omega_0} - a^{n+1}e^{-j\omega_0}\big] = H_0 a e^{-j\omega_0}(e^{j300\omega_0} - a^{300})a^{n-300}$$

At $n = 300$, we have $y(300) = H_0 a e^{-j\omega_0}(e^{j300\omega_0} - a^{300})$, Therefore, we can write

$$y(n) = y(300)a^{n-300}, \qquad \text{for } \ 300 \le n < \infty$$

Thus, the input-off transients are exponentially decaying as seen in Fig. 6.3.4. Note that the two expressions of $y(n)$ for $n \le 299$ and $n \ge 300$ join smoothly, in the following sense. The difference equation for $y(n)$ gives at $n = 300$, $y(300) = ay(299) + bx(300)$. But $x(300) = 0$ and thus $y(300) = ay(299)$. This condition can be verified for the above expressions. We have, using $b - H_0 = -aH_0 e^{-j\omega_0}$:

$$\begin{aligned} ay(299) &= a\big[H_0 e^{j299\omega_0} + (b - H_0)a^{299}\big] \\ &= aH_0 e^{j300\omega_0}e^{-j\omega_0} - a^2 H_0 e^{-j\omega_0}a^{299} \\ &= aH_0 e^{-j\omega_0}(e^{j300\omega_0} - a^{300}) = y(300) \end{aligned}$$

The real-valued versions can be obtained by taking imaginary parts of these answers. Writing $H_0 = |H_0|e^{j\phi_0}$, with $|H_0| = 1$ and $\phi_0 = \arg H_0 = -1.3179$ radians, we have for $n \le 299$

$$y(n) = H_0 e^{j\omega_0 n} + (b - H_0)a^n = e^{j(\omega_0 n + \phi_0)} + (b - e^{j\phi_0})a^n$$

Taking imaginary parts, we find

$$y(n) = \sin(\omega_0 n + \phi_0) - a^n \sin\phi_0, \qquad \text{for } \ 0 \le n \le 299$$

and

$$y(n) = y(300)a^{n-300}, \qquad \text{for } \ n \ge 300$$

where we calculate $y(300) = ay(299)$. The numerical values of $y(n)$ agree, of course, with the iterated values of Example 6.3.2.

All of the above expressions can be obtained much faster using z-transforms. First, we write the length-300 complex sinusoid in a form which is valid for all n:

$$x(n) = e^{j\omega_0 n}\big(u(n) - u(n-300)\big) = e^{j\omega_0 n}u(n) - e^{j300\omega_0}e^{j\omega_0(n-300)}u(n-300)$$

where, in the second term we have the delayed version of the first. Taking z-transforms and using the delay property, we find:

$$X(z) = \frac{1 - e^{j300\omega_0}z^{-300}}{1 - e^{j\omega_0}z^{-1}}$$

The output z-transform will be then

$$Y(z) = H(z)X(z) = \frac{b(1 - e^{j300\omega_0}z^{-300})}{(1 - az^{-1})(1 - e^{j\omega_0}z^{-1})}$$

The (causal) inverse z-transform can be found using the "remove/restore" method. Ignoring the numerator temporarily, we have

$$W(z) = \frac{b}{(1 - az^{-1})(1 - e^{j\omega_0}z^{-1})} = \frac{C}{1 - e^{j\omega_0}z^{-1}} + \frac{B}{1 - az^{-1}}$$

where, as we have seen $C = H_0$. Similarly, one can verify that $B = b - H_0$. Therefore, the causal inverse z-transform of $W(z)$ will be:

$$w(n) = H_0 e^{j\omega_0 n}u(n) + (b - H_0)a^n u(n)$$

Restoring the numerator of $Y(z)$, that is, $Y(z) = (1 - e^{j300\omega_0}z^{-300})W(z)$, we find

$$y(n) = w(n) - e^{j300\omega_0}w(n-300)$$

which gives rise to the following expression for $y(n)$:

$$\begin{aligned} y(n) &= H_0 e^{j\omega_0 n}u(n) + (b - H_0)a^n u(n) - \\ &\quad - e^{j300\omega_0}\big[H_0 e^{j\omega_0(n-300)}u(n-300) + (b - H_0)a^{n-300}u(n-300)\big] \end{aligned}$$

We leave it up to the reader to verify that this expression agrees with the separate expressions given above for $n \le 299$ and $n \ge 300$. □

A third consequence of Eq. (6.3.11) is its application to two important special cases, namely, the unit-step and the alternating unit-step responses. The *unit-step response* is the output $y(n)$ due to a unit-step input:

$$x(n)=u(n)$$

It is a special case of a sinusoid $e^{j\omega_0 n}u(n)$ with $\omega_0=0$. The value $\omega_0=0$ corresponds to the complex point $z=e^{j\omega_0}=1$ on the z-plane. Equation (6.3.11) becomes in this case:

$$y(n)=H(0)+B_1p_1^n+B_2p_2^n+\cdots+B_Mp_M^n,\qquad n\geq 0$$

Thus, in the limit of large n, the output will settle into a constant value given by

$$y(n)\to H(0)=H(z)\big|_{z=1}\qquad\text{as}\qquad n\to\infty$$

We may also refer to it as the DC response of the filter, that is, its response to a constant input. Setting $\omega=0$ into the definition of $H(\omega)$, Eq. (5.4.3), or $z=1$ into the definition of $H(z)$, Eq. (5.1.2), we can express $H(0)$ in the alternative form:

$$\boxed{H(0)=H(z)\big|_{z=1}=\sum_{n=0}^{\infty}h(n)}\qquad\text{(DC gain)}\tag{6.3.14}$$

In Chapter 4, we referred to it as the DC gain of the filter and used it in Eq. (4.1.24) and in the simulation examples. In a similar fashion, we can discuss the alternating step response, namely, the output due to the alternating input:

$$x(n)=(-1)^n u(n)$$

Writing $-1=e^{j\pi}$, we have $(-1)^n u(n)=e^{j\pi n}u(n)$, and therefore, we recognize this as a special case of Eq. (6.3.11) with $\omega_0=\pi$, which corresponds to the z-point $z=e^{j\omega_0}=-1$. Eq. (6.3.11) becomes:

$$y(n)=H(\pi)e^{j\pi n}+B_1p_1^n+B_2p_2^n+\cdots+B_Mp_M^n,\qquad n\geq 0$$

And, in the limit of large n, the output tends to

$$y(n)\to H(\pi)(-1)^n\qquad\text{as}\qquad n\to\infty$$

The quantity $H(\pi)$ may be called the *AC gain* and can be expressed in terms of $h(n)$ by setting $z=-1$ into the definition for $H(z)$:

$$\boxed{H(\pi)=H(z)\big|_{z=-1}=\sum_{n=0}^{\infty}(-1)^n h(n)}\qquad\text{(AC gain)}$$

Example 6.3.4: Determine the DC and alternating-step responses of the filter of Example 6.3.1. Determine also the effective time constant n_{eff} to reach steady state to within one percent.

Solution: Setting $n=0$ in the expression for $y(n)$ in Example 6.3.1 gives the relationship $y(0)=H(\omega_0)+B_1$. Inspecting the expression for $Y(z)$, the value of $y(0)$ is found to be $y(0)=5$. More systematically, $y(0)$ can be found by evaluating $Y(z)$ at $z=\infty$. We have therefore, $H(\omega_0)+B_1=5$, which can be solved for $B_1=5-H(\omega_0)$. Thus, for a general ω_0 we may write:

$$y(n)=H(\omega_0)e^{j\omega_0 n}+\big(5-H(\omega_0)\big)(0.8)^n,\quad n\geq 0$$

For the DC unit-step response, we have setting $\omega_0=0$:

$$H(0)=\left[\frac{5+2e^{-j\omega_0}}{1-0.8e^{-j\omega_0}}\right]_{\omega_0=0}=\frac{5+2}{1-0.8}=35$$

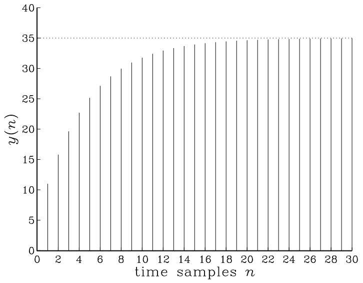
Therefore, $B_1=5-H(0)=5-35=-30$, and the response to a unit-step is:

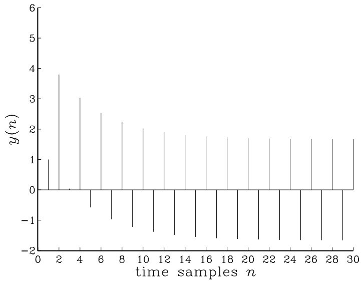
$$y(n)=35-30(0.8)^n,\quad n\geq 0$$

Similarly, we find $H(\pi)=5/3$, and $B_1=5-5/3=10/3$. Thus, the alternating unit-step response will be:

$$y(n)=\frac{5}{3}(-1)^n+\frac{10}{3}(0.8)^n,\quad n\geq 0$$

The output signals of the two cases are shown below:





This filter has only one pole; therefore the effective time constant is determined by the quantity $a=0.8$. At the 1% level, we have $\epsilon=1\%=0.01$, and we find

$$n_{\text{eff}}=\frac{\ln\epsilon}{\ln a}=\frac{\ln(0.01)}{\ln(0.8)}=20.6$$

Thus, the effective duration of the transient behavior is about 20 time steps, after which the transient term $(0.8)^n$ drops by more than 1% its initial value. □

The fourth consequence of Eq. (6.3.11) is its application to *marginally stable* filters. Many filters of practical interest are not strictly stable, but only marginally so, with poles *on* the unit circle. This includes for example, accumulators (integrators), periodic function generators, and others. It is therefore useful to know what happens to the sinusoidal response in such cases.

Suppose the filter $H(z)$ has a pole p_1 on the unit circle at some phase angle θ_1, that is, $p_1 = e^{j\theta_1}$. Of course, the conjugate pole $p_1^* = e^{-j\theta_1}$ is also present. For example, the filters:

$$H(z) = \frac{1}{1 - z^{-1}}, \qquad H(z) = \frac{1}{1 + z^{-1}}, \qquad H(z) = \frac{1}{1 + z^{-2}}$$

have poles at $p_1 = 1 = e^{j0}$, $p_1 = -1 = e^{j\pi}$, and $\pm j = e^{\pm j\pi/2}$, respectively.

Suppose also that all other poles lie inside the unit circle. Then the transient response Eq. (6.3.11) will be:

$$y(n) = H(\omega_0)e^{j\omega_0 n} + B_1 p_1^n + B_1^* p_1^{*n} + B_2 p_2^n + \cdots, \qquad \text{or,}$$

$$y(n) = H(\omega_0)e^{j\omega_0 n} + B_1 e^{j\theta_1 n} + B_1^* e^{-j\theta_1 n} + B_2 p_2^n + \cdots$$

In the limit of large n, the terms p_2^n, p_3^n, etc., will drop to zero exponentially, but the p_1^n being sinusoidal will not. Thus, the filter output tends to

$$y(n) \to H(\omega_0)e^{j\omega_0 n} + B_1 e^{j\theta_1 n} + B_1^* e^{-j\theta_1 n}$$

for large n. There is no true sinusoidal response even though the output is a sum of sinusoids. Once the sinusoidal pole terms $e^{j\theta_1 n}$ are excited, they will remain in the output forever.

This analysis of Eq. (6.3.11) applies only to the case when $\omega_0 \neq \pm\theta_1$. If $\omega_0 = \pm\theta_1$, one hits a “resonance” of the system and the output $y(n)$ becomes unstable diverging to infinity. In this case, the output z-transform $Y(z)$ has a *double pole* and the discussion must be modified. For example, if $\omega_0 = \theta_1$, then $e^{j\omega_0} = e^{j\theta_1} = p_1$ and $Y(z)$ becomes:

$$\begin{aligned} Y(z) = H(z)X(z) &= \frac{N(z)}{(1 - e^{j\omega_0}z^{-1})(1 - p_1 z^{-1})\cdots(1 - p_M z^{-1})} \\ &= \frac{N(z)}{(1 - p_1 z^{-1})^2(1 - p_2 z^{-1})\cdots(1 - p_M z^{-1})} \end{aligned}$$

The partial fraction expansion takes the form in this case:

$$Y(z) = \frac{B_1}{1 - p_1 z^{-1}} + \frac{B_1'}{(1 - p_1 z^{-1})^2} + \frac{B_2}{1 - p_2 z^{-1}} + \cdots + \frac{B_M}{1 - p_M z^{-1}}$$

Using the causal inverse z-transform (with ROC $|z| > |a|$):

$$\frac{1}{(1 - az^{-1})^2} \xrightarrow{Z^{-1}} (n+1)a^n u(n)$$

we find for the output signal:

$$y(n) = B_1 p_1^n + B_1'(n+1)p_1^n + B_2 p_2^n + \cdots + B_M p_M^n, \qquad \text{or,}$$

$$y(n) = B_1 e^{j\theta_1 n} + B_1'(n+1)e^{j\theta_1 n} + B_2 p_2^n + \cdots + B_M p_M^n$$

which diverges linearly in n.

Until now, the entire discussion of transient response was geared to IIR filters that have nontrivial poles. FIR filters do not have any poles (except at $z = 0$), and therefore, the analysis of their steady versus transient sinusoidal response must be carried out differently.

Consider an FIR filter of order M with impulse response $\mathbf{h} = [h_0, h_1, \cdots, h_M]$. For a causal sinusoidal input $x(n) = e^{j\omega_0 n}u(n)$, the output will be, as discussed in Chapter 4:

$$y(n) = \sum_{m=0}^{\min(n,M)} h(m)x(n-m) = \sum_{m=0}^{\min(n,M)} h(m)e^{j\omega_0(n-m)}$$

or, for any $n \geq 0$:

$$\boxed{y(n) = e^{j\omega_0 n}\sum_{m=0}^{\min(n,M)} h(m)e^{-j\omega_0 m}}$$

When $n \geq M$, the upper summation limit becomes M, giving

$$y(n) = e^{j\omega_0 n}\sum_{m=0}^{M} h(m)e^{-j\omega_0 m} = H(\omega_0)e^{j\omega_0 n}, \qquad n \geq M$$

Therefore, as we have already seen in Chapter 4, the input-on transients last only for the time period $0 \leq n \leq M$. After that period, steady state sets in.

6.4 Pole/Zero Designs

6.4.1 First-Order Filters

Pole/zero placement can be used to design simple filters, such as first-order smoothers, notch filters, and resonators. To illustrate the technique, we design the transfer function

$$H(z) = \frac{5 + 2z^{-1}}{1 - 0.8z^{-1}} = \frac{5(1 + 0.4z^{-1})}{1 - 0.8z^{-1}}$$

discussed in Section 6.2. We begin with the more general transfer function

$$H(z) = \frac{G(1 + bz^{-1})}{1 - az^{-1}} \tag{6.4.1}$$

where both a and b are positive and less than one. The gain factor G is arbitrary. The pole/zero pattern is shown in Fig. 6.4.1.

The filter zero at $z = -b$ lies in the left half (the high-frequency part) of the unit circle, and the filter pole at $z = a$ lies in the right half (the low-frequency part). Therefore,

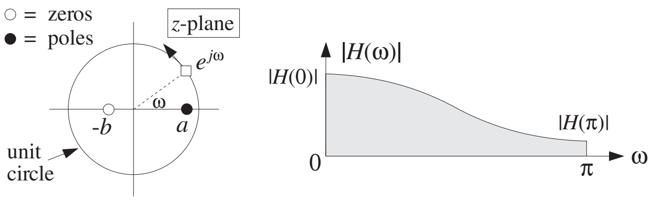


Fig. 6.4.1 Pole/zero pattern and frequency response.

the pole emphasizes low frequencies and the zero attenuates high frequencies; in other words, the filter acts as a lowpass filter.

The frequency response values at the lowest and highest frequencies $\omega = 0, \pi$ are found by setting $z = \pm 1$ in Eq. (6.4.1):

$$H(0) = \frac{G(1+b)}{1-a}, \qquad H(\pi) = \frac{G(1-b)}{1+a}$$

Therefore, the attenuation of the highest frequency relative to the lowest one is:

$$\frac{H(\pi)}{H(0)} = \frac{(1-b)(1-a)}{(1+b)(1+a)} \tag{6.4.2}$$

To determine the two unknown parameters a and b in Eq. (6.4.1), we need two *design equations*. One such equation can be Eq. (6.4.2). If a is known, then for a desired level of attenuation $H(\pi)/H(0)$, we can solve for b.

To determine a, we may impose a constraint on the *speed of response* of the filter, that is, we may specify the effective time constant n_{eff}, which is controlled by the value of a. For example, requiring that $n_{\text{eff}} = 20$ time samples and taking $\epsilon = 0.01$, we can solve Eq. (6.3.12) for a:

$$a = \epsilon^{1/n_{\text{eff}}} = (0.01)^{1/20} \simeq 0.8$$

With this value of a, requiring that $H(\pi)/H(0) = 1/21$, Eq. (6.4.2) would give:

$$\frac{(1-b)(1-0.8)}{(1+b)(1+0.8)} = \frac{1}{21} \quad \Rightarrow \quad b = 0.4$$

which gives the desired designed filter, up to the gain G:

$$H(z) = \frac{G(1+0.4z^{-1})}{1-0.8z^{-1}}$$

Because the parameter b is restricted to the interval $0 \le b \le 1$, we may look at the two extreme designs, namely, for $b = 0$ and $b = 1$. Setting $b = 0$ in Eqs. (6.4.1) and (6.4.2), gives:

$$H(z) = \frac{G}{1-0.8z^{-1}}, \qquad \frac{H(\pi)}{H(0)} = \frac{1}{9}$$

and setting $b = 1$,

$$H(z) = \frac{G(1+z^{-1})}{1-0.8z^{-1}}, \qquad \frac{H(\pi)}{H(0)} = 0$$

corresponding to $H(\pi) = 0$. The two design criteria that we used are not the only possible ones. In Section 15.1, we will replace the design equation (6.4.2) by an alternative criterion, which is better suited for the design of noise reduction filters.

6.4.2 Parametric Resonators and Equalizers

As another example, consider the design of a simple second-order "resonator" filter whose frequency response is dominated by a single narrow pole peak at some frequency ω_0. Such frequency response is shown in Fig. 6.4.2.

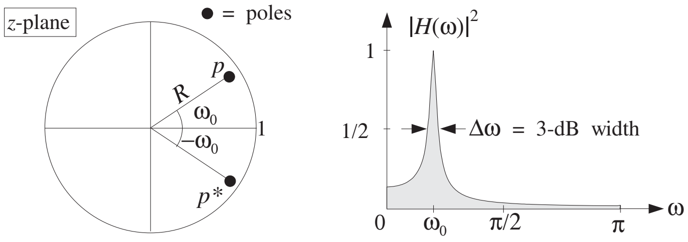


Fig. 6.4.2 Pole/zero pattern and frequency response of resonator filter.

To make a peak at $\omega = \omega_0$, we place a pole inside the unit circle along the ray with phase angle ω_0, that is, at the complex location:

$$p = Re^{j\omega_0}$$

where the pole magnitude is $0 < R < 1$. Together with the conjugate pole $p^* = Re^{-j\omega_0}$, we obtain the transfer function:

$$\boxed{H(z) = \frac{G}{(1-Re^{j\omega_0}z^{-1})(1-Re^{-j\omega_0}z^{-1})} = \frac{G}{1+a_1z^{-1}+a_2z^{-2}}} \tag{6.4.3}$$

where a_1 and a_2 are related to R and ω_0 by

$$\boxed{a_1 = -2R\cos\omega_0, \quad a_2 = R^2}$$

The gain G may be fixed so as to normalize the filter to unity at ω_0, that is, $|H(\omega_0)| = 1$. The frequency response of the filter is obtained by the substitution $z = e^{j\omega}$:

$$H(\omega) = \frac{G}{(1-Re^{j\omega_0}e^{-j\omega})(1-Re^{-j\omega_0}e^{-j\omega})} = \frac{G}{1+a_1e^{-j\omega}+a_2e^{-2j\omega}}$$

The normalization requirement $|H(\omega_0)| = 1$ gives the condition:

$$|H(\omega_0)| = \frac{G}{|(1 - Re^{j\omega_0}e^{-j\omega_0})(1 - Re^{-j\omega_0}e^{-j\omega_0})|} = 1$$

which can be solved for G:

$$G = (1 - R)\sqrt{1 - 2R\cos(2\omega_0) + R^2}$$

The magnitude response squared can also be expressed in the form:

$$|H(\omega)|^2 = \frac{G^2}{(1 - 2R\cos(\omega - \omega_0) + R^2)(1 - 2R\cos(\omega + \omega_0) + R^2)}$$

The 3-dB width $\Delta\omega$ of the peak is defined as the *full width at half maximum* of the magnitude squared response. It can be found by solving the equation

$$|H(\omega)|^2 = \frac{1}{2}|H(\omega_0)|^2 = \frac{1}{2}$$

In dB, this condition reads

$$20\log_{10}\left|\frac{H(\omega)}{H(\omega_0)}\right| = 10\log_{10}\left(\frac{1}{2}\right) = -3\text{ dB}$$

This equation has two solutions, say ω_1 and ω_2, the first to the left of ω_0 and the second to the right. The full width is defined as $\Delta\omega = \omega_2 - \omega_1$. These two frequencies are called the 3-dB frequencies. It can be shown that when p is near the unit circle, that is, $R \lesssim 1$, the full width is given approximately by

$$\boxed{\Delta\omega \simeq 2(1 - R)} \tag{6.4.4}$$

Thus, the closer R is to one, the sharper the peak, but also the slower the filter will be in reaching its steady-state response, as we discussed in the previous section.

Equation (6.4.4) can be shown geometrically, as follows [15]. In Fig. 6.4.3, the pole p is indicated by the point P whose distance from the origin is $|\text{OP}| = R$. Therefore, the distance $|\text{PQ}|$ to the unit circle will be $|\text{PQ}| = 1 - R$.

Assuming the pole P is very near the unit circle, the small 3-dB angle $\Delta\omega$ subtended about the direction OQ will intersect the circle at two points which may be taken to be approximately the points A and B that lie along the tangent to the circle at Q. Denoting by z_{A} and z_{Q} the complex numbers represented by the points A and Q, we have the values for the transfer function:

$$|H(z_{\text{A}})| = \frac{G}{|z_{\text{A}} - p||z_{\text{A}} - p^*|}, \quad |H(z_{\text{Q}})| = \frac{G}{|z_{\text{Q}} - p||z_{\text{Q}} - p^*|}$$

Assuming P is very near the circle, all four points P, Q, A, and B will be very closely clustered to each other. Therefore, their distances to the conjugate pole p^* will be approximately equal, that is, $|z_{\text{A}} - p^*| \simeq |z_{\text{Q}} - p^*|$. Thus, we have for the ratio:

$$\frac{|H(z_{\text{A}})|}{|H(z_{\text{Q}})|} = \frac{|z_{\text{Q}} - p|}{|z_{\text{A}} - p|} = \frac{|\text{PQ}|}{|\text{PA}|}$$

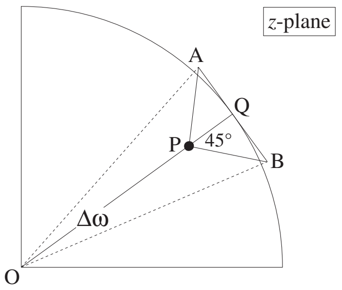


Fig. 6.4.3 Geometric interpretation of 3-dB width.

Then, the 3-dB condition that $|H(z_{\text{A}})|/|H(z_{\text{Q}})| = 1/\sqrt{2}$, implies $|\text{PQ}|/|\text{PA}| = 1/\sqrt{2}$, or, $|\text{PA}| = \sqrt{2}|\text{PQ}|$, which means that the orthogonal triangle PQA will be equilateral, with a 45° angle $\angle$QPA. A similar argument shows that the triangle PQB is also a 45° orthogonal triangle. It follows that $|\text{AB}| = 2|\text{QA}| = 2|\text{PQ}| = 2(1 - R)$. But the arc subtended by the angle $\Delta\omega$ is equal to the radius of the circle (i.e., 1) times the angle $\Delta\omega$. This arc is approximately equal to $|\text{AB}|$ and therefore, $\Delta\omega = |\text{AB}| = 2(1 - R)$.

Eq. (6.4.4) can be used as the *design criterion* that determines the value of R for a given bandwidth $\Delta\omega$. The filter's causal impulse response can be obtained from Eq. (6.4.3) using partial fractions. We find, for $n \geq 0$:

$$\boxed{h(n) = \frac{G}{\sin\omega_0}R^n\sin(\omega_0 n + \omega_0)}$$

The difference equation for the filter follows from Eq. (6.4.3). We have:

$$Y(z) = H(z)X(z) = \frac{G}{1 + a_1 z^{-1} + a_2 z^{-2}}X(z)$$

which gives

$$(1 + a_1 z^{-1} + a_2 z^{-2})Y(z) = GX(z)$$

and in the time domain:

$$y(n) + a_1 y(n-1) + a_2 y(n-2) = Gx(n)$$

or,

$$y(n) = -a_1 y(n-1) - a_2 y(n-2) + Gx(n) \tag{6.4.5}$$

A block diagram realization is shown in Fig. 6.4.4. The corresponding sample processing algorithm is obtained by introducing the internal states

$$\begin{aligned} w_1(n) &= y(n-1) \\ w_2(n) &= y(n-2) = w_1(n-1) \end{aligned}$$

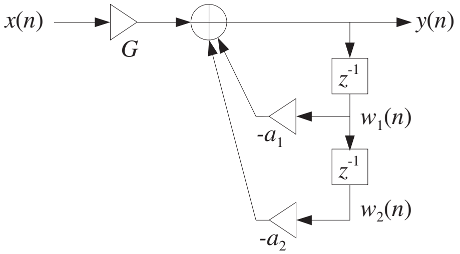


Fig. 6.4.4 Direct form realization of resonator filter.

The difference equation may be replaced by the system:

$$y(n) = -a_1 w_1(n) - a_2 w_2(n) + Gx(n)$$
$$w_2(n+1) = w_1(n)$$
$$w_1(n+1) = y(n)$$

which gives rise to the following sample processing algorithm:

```
for each input sample x do:
    $y = -a_1 w_1 - a_2 w_2 + Gx$
    $w_2 = w_1$
    $w_1 = y$
```

Example 6.4.1: Design a 2-pole resonator filter with peak at $f_0 = 500$ Hz and width $\Delta f = 32$ Hz, operating at the sampling rate of $f_s = 10$ kHz.

Solution: The normalized resonator frequency will be

$$\omega_0 = \frac{2\pi f_0}{f_s} = 0.1\pi \quad \text{[radians/sample]}$$

and the corresponding width:

$$\Delta\omega = \frac{2\pi \Delta f}{f_s} = 0.02$$

Eq. (6.4.4) gives then

$$2(1-R) = 0.02 \quad \Rightarrow \quad R = 0.99$$

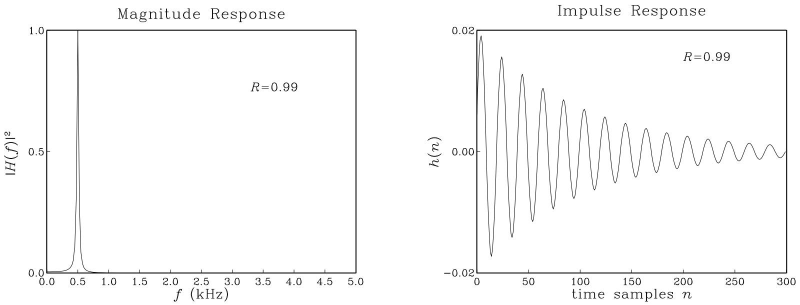
With this value of R, we find the filter parameters:

$$G = 0.0062, \quad a_1 = -1.8831, \quad a_2 = 0.9801$$

and the filter transfer function

$$H(z) = \frac{0.0062}{1 - 1.8831z^{-1} + 0.9801z^{-2}}$$

The magnitude response and impulse response $h(n)$ are shown below:



The effective time constant of this filter is about $n_{\text{eff}} = \ln\epsilon / \ln R = 458$ time samples. The graph only plots until $n = 300$. □

A slight generalization of the resonator filter is to place a pair of zeros near the poles along the same directions as the poles, that is, at locations:

$$z_1 = re^{j\omega_0}, \quad z_1^* = re^{-j\omega_0}$$

where r is restricted to the range $0 \le r \le 1$. The transfer function becomes:

$$\boxed{H(z) = \frac{(1 - re^{j\omega_0}z^{-1})(1 - re^{-j\omega_0}z^{-1})}{(1 - Re^{j\omega_0}z^{-1})(1 - Re^{-j\omega_0}z^{-1})} = \frac{1 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}} \tag{6.4.6}$$

where the filter coefficients are given in terms of the parameters r, R, and ω_0:

$$\boxed{\begin{array}{ll} b_1 = -2r\cos\omega_0 & b_2 = r^2 \\ a_1 = -2R\cos\omega_0 & a_2 = R^2 \end{array}} \tag{6.4.7}$$

The corresponding magnitude squared response is:

$$|H(\omega)|^2 = \frac{\big(1 - 2r\cos(\omega - \omega_0) + r^2\big)\big(1 - 2r\cos(\omega + \omega_0) + r^2\big)}{\big(1 - 2R\cos(\omega - \omega_0) + R^2\big)\big(1 - 2R\cos(\omega + \omega_0) + R^2\big)}$$

Figure 6.4.5 shows the pole/zero pattern. When $r < R$, the pole "wins" over the zero, in the sense that it is closer to the unit circle than the zero, giving rise to a peak in the frequency response at $\omega = \omega_0$. The resonator case may be thought of as a special case with $r = 0$. When $r > R$, the zero wins over the pole, giving rise to a dip in the frequency response. In particular, if $r = 1$, one gets an exact zero, a notch, at $\omega = \omega_0$.

When the pole and zero are very near each other, that is, $r \lesssim R$ or $r \gtrsim R$, the frequency response remains essentially flat for frequencies far from $\omega = \pm\omega_0$, because the distances of the moving point $e^{j\omega}$ to the pole/zero pairs are almost equal, giving

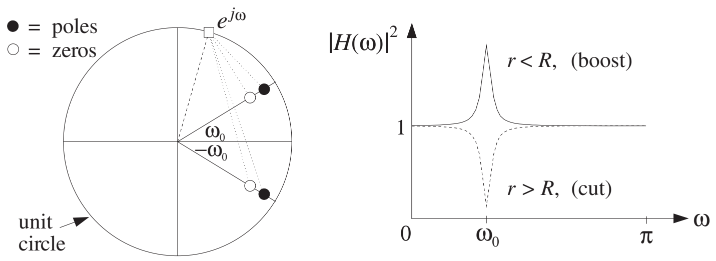


Fig. 6.4.5 Parametric equalizer filter.

$|H(\omega)| \simeq 1$. Only near the vicinity of $\pm\omega_0$ does $|H(\omega)|$ vary dramatically, developing a peak or a dip.

Such a filter can be thought of as a simple *parametric equalizer* filter, providing a "boost" if $r < R$, or a "cut" if $r > R$. The height of the boost or cut relative to 1 is controlled by the closeness of r to R. The width of the peaks or dips is controlled by the closeness of R to the unit circle.

Later on, we will reconsider such pole/zero designs of equalization filters for digital audio systems based on analog designs and the bilinear transformation method, and will derive more precise design criteria that are based on the desired values of the bandwidth and gain of the peak or dip. Such second-order filters can be cascaded together to provide boosting or cutting at multiple frequencies.

Example 6.4.2: Using the numerical values $R = 0.98$, $\omega_0 = 0.4\pi$ for the pole, determine the parametric equalizer transfer functions of Eq. (6.4.6) for a boost corresponding to $r = 0.965$, a cut with $r = 0.995$, and an exact notch with $r = 1$.

Solution: The transfer function coefficients are computed by Eq. (6.4.7). The resulting transfer functions are in the three cases of $r = 0.965$, $r = 0.995$, $r = 1$:

$$H(z) = \frac{1 - 0.5964z^{-1} + 0.9312z^{-2}}{1 - 0.6057z^{-1} + 0.9604z^{-2}}$$

$$H(z) = \frac{1 - 0.6149z^{-1} + 0.9900z^{-2}}{1 - 0.6057z^{-1} + 0.9604z^{-2}}$$

$$H(z) = \frac{1 - 0.6180z^{-1} + z^{-2}}{1 - 0.6057z^{-1} + 0.9604z^{-2}}$$

The corresponding magnitude responses $|H(\omega)|$ are shown in Fig. 6.4.6. The case $r = 1$ provides an exact notch at $\omega = \omega_0$. □

6.4.3 Notch and Comb Filters

The case $r = 1$ for a notch filter deserves some further discussion. In this case, the filter coefficients given by Eq. (6.4.7) can be written as

$$a_1 = Rb_1 = -2R\cos\omega_0, \quad a_2 = R^2 b_2 = R^2$$

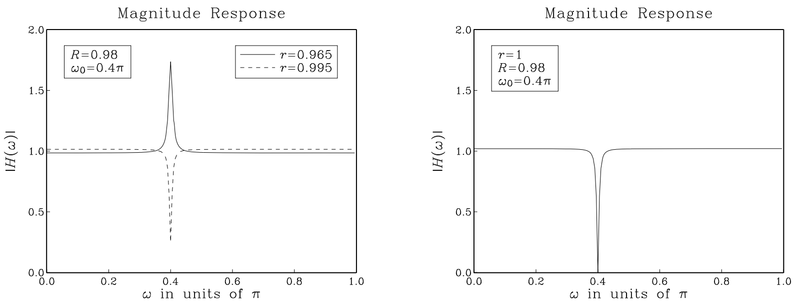


Fig. 6.4.6 Parametric equalizers of Example 6.4.2.

And, the transfer function takes the form:

$$H(z) = \frac{1 + b_1 z^{-1} + b_2 z^{-2}}{1 + Rb_1 z^{-1} + R^2 b_2 z^{-2}} = \frac{N(z)}{N(R^{-1}z)}$$

where $N(z)$ is the numerator polynomial having zeros at the two notch locations $z = e^{\pm j\omega_0}$:

$$N(z) = 1 + b_1 z^{-1} + b_2 z^{-2} = 1 - 2z^{-1}\cos\omega_0 + z^{-2} = (1 - e^{j\omega_0}z^{-1})(1 - e^{-j\omega_0}z^{-1})$$

This method can be generalized to construct a notch filter with notches at an arbitrary (finite) set of frequencies. The numerator polynomial $N(z)$ is defined as the polynomial whose zeros are *on the unit circle* at the *desired* notch locations. For example, if there are M desired notch frequencies ω_i, $i = 1, 2, \dots, M$, then $N(z)$ is defined as the Mth degree polynomial with zeros at $z_i = e^{j\omega_i}$, $i = 1, 2, \dots, M$:

$$\boxed{N(z) = \prod_{i=1}^{M}(1 - e^{j\omega_i}z^{-1})} \qquad \text{(notch polynomial)} \qquad (6.4.8)$$

The denominator polynomial is chosen as $D(z) = N(\rho^{-1}z)$, for some parameter $0 < \rho < 1$, that is,

$$D(z) = N(\rho^{-1}z) = \prod_{i=1}^{M}(1 - e^{j\omega_i}\rho z^{-1})$$

The zeros of $D(z)$ lie at the same directions as the notch zeros, but they are all pushed inside the unit circle at radius ρ. Therefore, for each desired zero $z_i = e^{j\omega_i}$, there is a corresponding pole $p_i = \rho e^{j\omega_i}$. Writing Eq. (6.4.8) in expanded form:

$$N(z) = 1 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_M z^{-M}$$

we obtain the following transfer function for the notch filter:

$$\boxed{H(z)=\frac{N(z)}{N(\rho^{-1}z)}=\frac{1+b_1z^{-1}+b_2z^{-2}+\cdots+b_Mz^{-M}}{1+\rho b_1z^{-1}+\rho^2b_2z^{-2}+\cdots+\rho^Mb_Mz^{-M}}} \tag{6.4.9}$$

that is, the denominator coefficients are chosen as the scaled versions of the numerator coefficients,

$$a_i=\rho^ib_i,\quad i=1,2,\cdots,M$$

If ρ is near one, $\rho\lesssim 1$, the distances of the movable point $e^{j\omega}$ to the pole/zero pairs $\{z_i,p_i\}=\{z_i,\rho z_i\}$ are almost equal to each other except in the near vicinity of the pair, that is, except near $\omega=\omega_i$. Thus, $H(\omega)$ remains essentially flat except in the vicinity of the desired notch frequencies.

Example 6.4.3: A DSP system operating at a sampling rate of 600 Hz, is plagued by 60 Hz power frequency interference noise and its harmonics. Design a notch filter that removes all of these harmonics, but remains flat at other frequencies.

Solution: The fundamental harmonic is

$$\omega_1=\frac{2\pi f_1}{f_s}=\frac{2\pi\cdot 60}{600}=0.2\pi\quad\text{[radians/sample]}$$

The other harmonics at $f_i=if_1$ correspond to $\omega_i=i\omega_1$. There are 10 harmonics that lie within the Nyquist interval $[0,f_s]$; namely, f_i, for $i=0,1,\cdots,9$. Because $f_s=10f_1$, all the harmonics that lie outside the Nyquist interval (if they have not been filtered out by the antialiasing prefilter) will be aliased onto harmonics inside that interval. For example, the harmonic $f_{11}=11f_s$ gets aliased with $f_{11}-f_s=11f_1-10f_1=f_1$, and so on. Therefore, our digital notch filter must be designed to have notches at the 10 frequencies within the Nyquist interval:

$$\omega_i=i\omega_1=\frac{2\pi i}{10},\quad i=0,1,\ldots,9$$

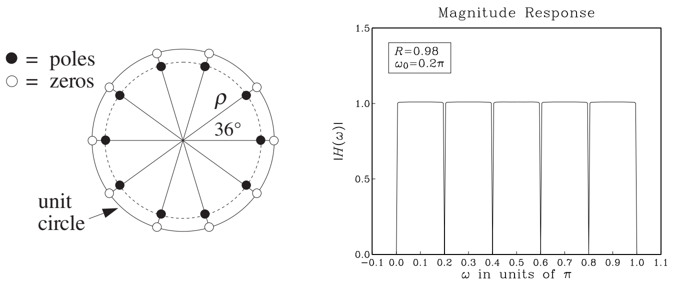
These 10 frequencies are none other than the tenth roots of unity, that is, the 10 roots of the polynomial:

$$N(z)=1-z^{-10}=\prod_{i=0}^{9}(1-e^{j\omega_i}z^{-1})$$

Our notch filter is then obtained by

$$H(z)=\frac{N(z)}{N(\rho^{-1}z)}=\frac{1-z^{-10}}{1-\rho^{10}z^{-10}}=\frac{1-z^{-10}}{1-Rz^{-10}}$$

where we set $R=\rho^{10}$. The following figure shows the resulting pole/zero pattern for this transfer function, and the corresponding magnitude response (computed only between $0\le\omega\le\pi$):



where we chose $R=0.98$, or $\rho=R^{1/10}=(0.98)^{1/10}=0.9980$. The radius ρ of the poles is very close to the unit circle resulting in very sharp notches at the desired harmonics. At other frequencies the magnitude response is essentially flat. □

Example 6.4.4: Repeat the previous example when the sampling rate is $f_s=1200$ Hz. Then, design another notch filter that excludes the DC and AC harmonics at $f=0$ and $f=f_s/2$.

Solution: Now the fundamental harmonic is $\omega_1=2\pi\cdot 60/1200=0.1\pi$, and the 20 harmonics in the Nyquist interval will be

$$\omega_i=i\omega_1=\frac{2\pi i}{20},\quad i=0,1,\ldots,19$$

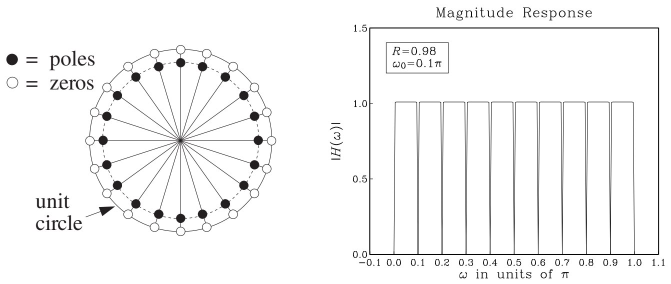
They correspond to the 20th roots of unity, that is, the roots of:

$$N(z)=1-z^{-20}=\prod_{i=0}^{19}(1-e^{j\omega_i}z^{-1})$$

The notch filter will be:

$$H(z)=\frac{N(z)}{N(\rho^{-1}z)}=\frac{1-z^{-20}}{1-\rho^{20}z^{-20}}=\frac{1-z^{-20}}{1-Rz^{-20}}$$

where we set $R=\rho^{20}$. The following figure shows the resulting pole/zero pattern and magnitude response, with the values $R=0.98$ or $\rho=R^{1/20}=(0.98)^{1/20}=0.9990$:



If we want to exclude the $\omega = \omega_0 = 0$ and $\omega = \omega_{10} = \pi$ harmonics from the notch filter, we must divide them out of $N(z)$. They contribute a factor

$$(1 - z^{-1})(1 + z^{-1}) = 1 - z^{-2}$$

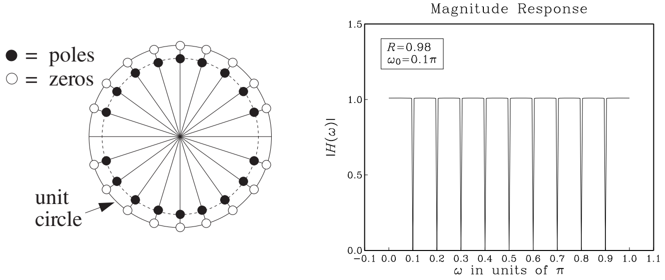
Therefore, the new $N(z)$ that has notches at all the harmonics but at $z = \pm 1$ will be:

$$N(z) = \frac{1 - z^{-20}}{1 - z^{-2}} = \frac{1 - z^{-10}}{1 - z^{-2}}(1 + z^{-10}) = (1 + z^{-2} + z^{-4} + z^{-6} + z^{-8})(1 + z^{-10})$$

Therefore, we find for the notch transfer function:

$$H(z) = \frac{N(z)}{N(\rho^{-1}z)} = \frac{(1 + z^{-2} + z^{-4} + z^{-6} + z^{-8})(1 + z^{-10})}{(1 + \rho^2 z^{-2} + \rho^4 z^{-4} + \rho^6 z^{-6} + \rho^8 z^{-8})(1 + \rho^{10} z^{-10})}$$

The resulting pole/zero pattern and magnitude response are shown below:



The value of ρ was the same as above, such that $\rho^{20} = R = 0.98$. □

A variant of the above method of constructing notch filters is the generalization of the parametric equalizer filter. It corresponds to moving the notch zeros *into* the unit circle and *behind* the poles, that is, replacing each notch zero by:

$$z_i = e^{j\omega_i} \quad \longrightarrow \quad z_i = re^{j\omega_i}$$

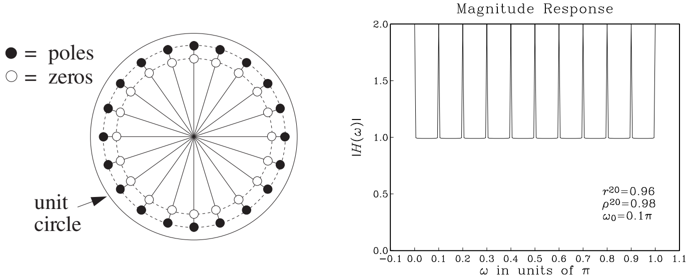
where $r \lesssim \rho$. This makes the poles win over the zeros, changing all the notch dips into sharp peaks at frequencies $\omega = \omega_i$. The corresponding transfer function is obtained from Eq. (6.4.9) by scaling z in the numerator:

$$\boxed{H(z) = \frac{N(r^{-1}z)}{N(\rho^{-1}z)} = \frac{1 + rb_1 z^{-1} + r^2 b_2 z^{-2} + \cdots + r^M b_M z^{-M}}{1 + \rho b_1 z^{-1} + \rho^2 b_2 z^{-2} + \cdots + \rho^M b_M z^{-M}}} \qquad (6.4.10)$$

Example 6.4.5: Starting with the notch polynomial $N(z) = 1 - z^{-20}$ of Example 6.4.4, we obtain the following filter, which will exhibit sharp peaks instead of dips if $r \lesssim \rho$:

$$H(z) = \frac{N(r^{-1}z)}{N(\rho^{-1}z)} = \frac{1 - r^{20}z^{-20}}{1 - \rho^{20}z^{-20}}$$

The pole/zero pattern and magnitude response are shown below:



The values of the parameters were $r^{20} = 0.96$ and $\rho^{20} = 0.98$, which correspond to $r = 0.9980$ and $\rho = 0.9990$. □

The notching and peaking filters of Eqs. (6.4.9) and (6.4.10) are referred to generically as *comb filters*. Notching comb filters are typically used to *cancel periodic interference*, such as the power frequency pickup and its harmonics. Peaking comb filters are used to *enhance periodic signals* in noise. The noise/interference reduction and signal enhancement capabilities of comb filters and their design will be discussed further in Chapters 16 and 12.

Comb filters arise also in the construction of digital reverb processors, where they represent the effects of multiple reflections of a sound wave off the walls of a listening space. They also arise in the design of digital periodic waveform generators. These topics will be discussed in Chapter 16.

6.5 Deconvolution, Inverse Filters, and Stability

In many applications, it is necessary to undo a filtering operation and recover the input signal from the available output signal. The output signal $y(n)$ is related to the input by the convolutional equation:

$$\boxed{y(n) = h(n) * x(n)} \qquad (6.5.1)$$

The objective of such "deconvolution" methods is to recover $x(n)$ from the knowledge of $y(n)$ and the filter $h(n)$. In theory, this can be accomplished by *inverse filtering*, that is, filtering $y(n)$ through the inverse filter

$$\boxed{H_{\text{inv}}(z) = \frac{1}{H(z)}} \qquad (6.5.2)$$

Indeed, working in the z-domain we have from Eq. (6.5.1):

$$Y(z) = H(z)X(z) \quad \Rightarrow \quad X(z) = \frac{1}{H(z)}Y(z) = H_{\text{inv}}(z)Y(z)$$

which becomes in the time domain:

$$\boxed{x(n)=h_{\text{inv}}(n)*y(n)} \tag{6.5.3}$$

where $h_{\text{inv}}(n)$ is the impulse response of the inverse filter $H_{\text{inv}}(z)$. This operation is illustrated in Fig. 6.5.1.

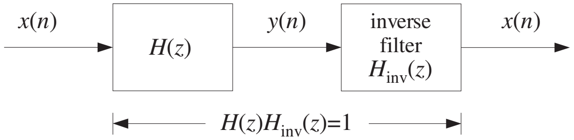


Fig. 6.5.1 Inverse filtering recovers original input.

Two typical applications of inverse filtering are *channel equalization* in digital voice or data transmission and the *equalization* of room or car acoustics in audio systems.

In channel equalization, the effect of a channel can be modeled as a linear filtering operation of the type of Eq. (6.5.1), where the transfer function $H(z)$ incorporates the effects of amplitude and phase distortions introduced by the channel. The signals $x(n)$ and $y(n)$ represent the transmitted and received signals, respectively. The inverse filter—called a *channel equalizer* in this context—is placed at the receiving end and its purpose is to undo the effects of the channel and recover the signal $x(n)$ that was transmitted. The overall processing system is shown in Fig. 6.5.2.

Often the channel itself is not known in advance, as for example in making a phone connection when the channel is established dynamically depending on how the call is routed to its destination. In such cases, the channel's transfer function must be determined (usually using adaptive signal processing techniques) before it can be inverted.

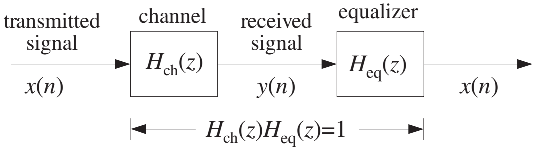


Fig. 6.5.2 Channel equalizer.

The sound generated by an audio system in a listening room is changed by the reverberation and absorption characteristics of the room's wall geometry and objects. The effect of the room can be modeled by a reverberation impulse response $h_{\text{room}}(n)$, so that the actual sound wave impinging on a listener's ears is a distorted version of the original sound wave $x(n)$ produced by the audio system:

$$y_{\text{room}}(n)=h_{\text{room}}(n)*x(n) \tag{6.5.4}$$

The impulse response $h_{\text{room}}(n)$ depends on where one sits in the room, but it can be measured and then deconvolved away by an inverse filtering operation:

$$Y_{\text{room}}(z)=H_{\text{room}}(z)X(z) \quad\Rightarrow\quad X(z)=\frac{1}{H_{\text{room}}(z)}Y_{\text{room}}(z)$$

In addition to removing the local reverberation effects of a room, one may want to *add* the reverberation ambience of a concert hall that increases the warmth and richness of the sound. If the same audio signal $x(n)$ were listened to in a concert hall with reverberation impulse response $h_{\text{hall}}(n)$, the actual sound wave would be

$$y_{\text{hall}}(n)=h_{\text{hall}}(n)*x(n) \tag{6.5.5}$$

Available DSP audio effects processors can simulate the reverberation characteristics of typical concert halls and can implement the above filtering operation. An idealized audio effects processor is shown in Fig. 6.5.3.

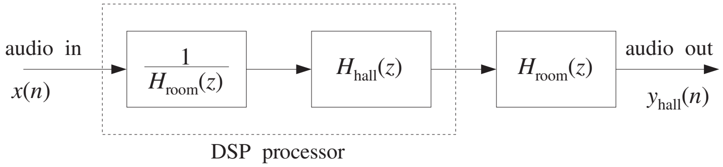


Fig. 6.5.3 An ideal audio effects processor.

First, it *deconvolves* the room acoustics by prefiltering the audio signal $x(n)$ by the inverse filter of the room's transfer function, anticipating the room's effect, and then it *convolves* it with the reverberation response of a desired concert hall. The effective transfer function of the arrangement is:

$$H_{\text{eff}}(z)=H_{\text{room}}(z)\cdot H_{\text{hall}}(z)\cdot\frac{1}{H_{\text{room}}(z)}=H_{\text{hall}}(z)$$

Thus, with the DSP effects processor, the sound wave produced in a room sounds as it does in a concert hall, Eq. (6.5.5). We will discuss audio effects processors in more detail in Chapter 16.

There are many other applications of inverse filtering in such diverse fields as identification and control systems, communication systems, image enhancement and restoration, digital magnetic recording, oil exploration, geophysics, ocean engineering, electromagnetic theory, scattering theory, radio astronomy, medical tomography, and spectroscopic analysis, as well as in many areas of applied mathematics, such as numerical analysis and statistics.

There are two major issues that arise in the practical application of inverse filtering. One is the requirement of *stability* of the inverse filter $h_{\text{inv}}(n)$. Without this requirement, the inverse filtering equation (6.5.3) would be unstable, resulting in numerical nonsense.

The other issue is the presence of *noise* in the data, that is, the available signal $y(n)$ may be (and almost invariably is) contaminated with noise, so that Eq. (6.5.1) is replaced by:

$$\boxed{y(n)=h(n)*x(n)+v(n)} \tag{6.5.6}$$

where $v(n)$ is the noise. Even if there is an exact and stable inverse filter $h_{\text{inv}}(n)$, the deconvolution of the noisy data $y(n)$ will give rise to the signal:

$$\boxed{\hat{x}(n)=h_{\text{inv}}(n)*y(n)=x(n)+\hat{v}(n)} \tag{6.5.7}$$

where $\hat{v}(n)$ is the filtered noise:

$$\hat{v}(n)=h_{\text{inv}}(n)*v(n)$$

Depending on the nature of the inverse filter $h_{\text{inv}}(n)$, even if the measurement noise $v(n)$ is very weak, it is quite possible for the filtered noise $\hat{v}(n)$ to be a much amplified version of $v(n)$, rendering $\hat{x}(n)$ a very poor and almost useless estimate of the desired signal $x(n)$. There exist signal processing techniques that try to address this noise problem to some extent. But they are beyond the scope of this book. See [45] for some discussion and references.

The impulse response $h(n)$ of the system $H(z)$ is assumed to be both stable and causal. This implies that the poles of $H(z)$ must lie strictly *inside* the unit circle. But the zeros of $H(z)$ do not have to lie inside the unit circle—they can be anywhere on the z-plane. Writing $H(z)$ in the ratio of two polynomials,

$$H(z)=\frac{N(z)}{D(z)}$$

we conclude that the zeros of $N(z)$ may be anywhere on the z-plane. Therefore, the inverse filter

$$H_{\text{inv}}(z)=\frac{1}{H(z)}=\frac{D(z)}{N(z)}$$

can have poles outside the unit circle. In this case, the *stable* inverse z-transform $h_{\text{inv}}(n)$ will necessarily be *anticausal*. As an example, consider the case of a filter $H(z)$ to be inverted:

$$H(z)=\frac{1-1.25z^{-1}}{1-0.5z^{-1}}=2.5-\frac{1.5}{1-0.5z^{-1}}$$

It has a causal and stable impulse response given by:

$$h(n)=2.5\delta(n)-1.5(0.5)^n u(n)$$

The corresponding inverse filter $H_{\text{inv}}(z)$ will be

$$H_{\text{inv}}(z)=\frac{1}{H(z)}=\frac{1-0.5z^{-1}}{1-1.25z^{-1}}=0.4+\frac{0.6}{1-1.25z^{-1}}$$

and because it has a pole outside the unit circle, its stable impulse response will be anticausal:

$$h_{\text{inv}}(n)=0.4\delta(n)-0.6(1.25)^n u(-n-1)=\begin{cases}0 & \text{if } n\ge 1\\ 0.4 & \text{if } n=0\\ -0.6(1.25)^n & \text{if } n\le -1\end{cases}$$

Such a stable and anticausal impulse response can be handled using the approximation technique discussed in Section 3.5. That is, the infinitely long anticausal tail is clipped off at some large negative time $n=-D$, replacing the exact $h_{\text{inv}}(n)$ by its clipped approximation:

$$\boxed{\widetilde{h}_{\text{inv}}(n)=\begin{cases}h_{\text{inv}}(n) & \text{if } n\ge -D\\ 0 & \text{if } n<-D\end{cases}} \tag{6.5.8}$$

The approximate impulse response $\widetilde{h}_{\text{inv}}(n)$ has only a finitely anticausal part and can be made causal by delaying it by D time units, as discussed in Section 3.5. The deconvolution error arising from using the approximate response can be determined as follows. Let $\widetilde{x}(n)$ be the deconvolved output using $\widetilde{h}_{\text{inv}}(n)$, that is,

$$\widetilde{x}(n)=\widetilde{h}_{\text{inv}}(n)*y(n)$$

Subtracting it from the exact output $x(n)$ of Eq. (6.5.3), we have

$$\begin{aligned}x(n)-\widetilde{x}(n)&=h_{\text{inv}}(n)*y(n)-\widetilde{h}_{\text{inv}}(n)*y(n)=\big(h_{\text{inv}}(n)-\widetilde{h}_{\text{inv}}(n)\big)*y(n)\\&=\sum_{m=-\infty}^{\infty}\big(h_{\text{inv}}(m)-\widetilde{h}_{\text{inv}}(m)\big)y(n-m)\\&=\sum_{m=-\infty}^{-D-1}h_{\text{inv}}(m)y(n-m)\end{aligned}$$

where all the terms for $m\ge -D$ were dropped because $\widetilde{h}_{\text{inv}}(m)$ agrees with $h_{\text{inv}}(m)$ there, and $\widetilde{h}_{\text{inv}}(m)=0$ for $m<-D$. Assuming the signal $y(n)$ is bounded by some maximum value $|y(n)|\le A$, we find

$$\boxed{|x(n)-\widetilde{x}(n)|\le A\sum_{m=-\infty}^{-D-1}|h_{\text{inv}}(m)|} \tag{6.5.9}$$

This is a general result for the deconvolution error. The upper bound gets smaller as D is increased. In particular, for the above example, we have:

$$|x(n)-\widetilde{x}(n)|\le A\sum_{m=-\infty}^{-D-1}|0.6(1.25)^m|=2.4A(1.25)^{-D}$$

which can be made as small as desired by choosing D larger.

For a more general inverse filter $H_{\text{inv}}(z)$ having more than one pole *outside* the unit circle, the pole *nearest* the circle controls the decay time constant of the negative-time tail of $h_{\text{inv}}(n)$, because it converges to zero the slowest. Therefore, it controls the choice of the delay D. If we denote the minimum magnitude of these poles by

$$a=\min|p_{\text{outside}}|>1$$

then for large D, the upper bound in Eq. (6.5.9) will behave essentially like the term a^{-D}, which gives the approximation error bound, for some constant B:

$$|x(n)-\tilde{x}(n)| \le Ba^{-D}$$

In summary, when the inverse filter transfer function $H_{\text{inv}}(z)$ has some poles *outside* the unit circle, one must choose the *anticausal but stable* impulse response $h_{\text{inv}}(n)$ and clip it at some large negative time D (and delay it by D to make it causal). The choice of D is dictated by the outside pole closest to the unit circle. The resulting deconvolution error arising from using the clipped filter can be made as small as desired by choosing the clipping delay D larger.

6.6 Problems

6.1 Using z-transforms, determine the transfer function $H(z)$ and from it the *causal* impulse response $h(n)$ of the linear systems described by the following I/O difference equations:

a. $y(n)=-0.8y(n-1)+x(n)$

b. $y(n)=0.8y(n-1)+x(n)$

c. $y(n)=0.8y(n-1)+x(n)+x(n-1)$

d. $y(n)=0.8y(n-1)+x(n)-0.5x(n-1)$

e. $y(n)=0.8y(n-1)+x(n)+0.25x(n-2)$

f. $y(n)=0.9y(n-1)-0.2y(n-2)+x(n)+x(n-1)-6x(n-2)$

In each case, determine also the *frequency response* $H(\omega)$, the *pole/zero* pattern of the transfer function on the z-plane, draw a rough sketch of the *magnitude response* $|H(\omega)|$ over the right half of the Nyquist interval $0\le\omega\le\pi$, and finally, draw the *direct* and *canonical* block diagram realizations of the difference equation and state the corresponding *sample-by-sample* filtering algorithms.

6.2 A unit-step signal $x(n)=u(n)$ is applied at the input of the linear systems:

a. $y(n)=x(n)+6x(n-1)+11x(n-2)+6x(n-3)$

b. $y(n)=x(n)-x(n-4)$

Using z-transforms, determine the corresponding output signals $y(n)$, for all $n\ge 0$.
Repeat for the alternating-step input $x(n)=(-1)^n u(n)$.

6.3 Repeat Problem 6.2 for the following systems:

a. $y(n)=0.25y(n-2)+x(n)$ b. $y(n)=-0.25y(n-2)+x(n)$

6.4 A unit-step signal $x(n)=u(n)$ is applied at the inputs of the systems of Problem 6.1.

a. Using z-transforms, derive expressions for the corresponding output signals $y(n)$ for all $n\ge 0$, and determine which part of $y(n)$ is the steady-state part and which the transient part.

b. Repeat for the input $x(n)=(-1)^n u(n)$.

c. Repeat for the input $x(n)=(0.5)^n u(n)$ applied only to Problem 6.1(d).

d. Repeat for the input $x(n)=(0.5)^n\cos(\pi n/2)u(n)$ applied to Problem 6.1(e) only.

e. Repeat for the *unstable* input $x(n)=2^n u(n)$ applied only to the system 6.1(f). Why is the output stable in this case?

6.5 Determine the transfer function $H(z)$ and the corresponding I/O difference equation relating $x(n)$ and $y(n)$ of the linear filters having the following impulse responses:

a. $h(n)=\delta(n-5)$ e. $h(n)=(-0.8)^n[u(n)-u(n-8)]$

b. $h(n)=u(n-5)$ f. $h(n)=(0.8)^n u(n)+(-0.8)^n u(n)$

c. $h(n)=(0.8)^n u(n)$ g. $h(n)=2(0.8)^n\cos(\pi n/2)u(n)$

d. $h(n)=(-0.8)^n u(n)$ h. $h(n)=(0.8j)^n u(n)+(-0.8j)^n u(n)$

In each case, determine also the *frequency response* $H(\omega)$, the *pole/zero* pattern of the transfer function on the z-plane, draw a rough sketch of the *magnitude response* $|H(\omega)|$ over the right half of the Nyquist interval $0\le\omega\le\pi$, and finally, draw the direct and canonical realizations implementing the I/O difference equation and state the corresponding *sample-by-sample* processing algorithms.

6.6 Find the transfer function $H(z)$ and express it as the *ratio* of two polynomials of the system having impulse response:

$$h(n)=\sum_{m=0}^{\infty}(0.5)^m\delta(n-8m)=\delta(n)+(0.5)\delta(n-8)+(0.5)^2\delta(n-16)+\cdots$$

Then, draw a block diagram realization and write its sample processing algorithm.

6.7 A digital reverberation processor has frequency response:

$$H(\omega)=\frac{-0.5+e^{-j\omega 8}}{1-0.5e^{-j\omega 8}}$$

where ω is the digital frequency in [radians/sample]. Determine the *causal* impulse response $h(n)$, for all $n\ge 0$, and sketch it versus n. [*Hint:* Do not use partial fractions.]

6.8 The first few Fibonacci numbers are:

$$\mathbf{h}=[0,1,1,2,3,5,8,13,21,\dots]$$

where each is obtained by summing the previous two.

a. Determine the linear system $H(z)$ whose causal impulse response is $\mathbf{h}$, and express it as a rational function in z^{-1}.

b. Using partial fractions, derive an expression for the nth Fibonacci number in terms of the poles of the above filter.

c. Show that the ratio of two successive Fibonacci numbers converges to the *Golden Section*, that is, the positive solution of the quadratic equation $\phi^2=\phi+1$, namely, $\phi=(1+\sqrt{5})/2$.

d. Show that the filter's poles are the two numbers $\{\phi,-\phi^{-1}\}$. Show that the geometric sequence:

$$\mathbf{y}=[0,1,\phi,\phi^2,\phi^3,\dots]$$

satisfies the same recursion as the Fibonacci sequence (for $n\ge 3$). Show that $\mathbf{y}$ may be considered to be the output of the filter $\mathbf{h}$ for a particular input. What is that input?

See [35,36] for some remarkable applications and properties of Fibonacci numbers.

6.9 Pell's series [35,36] is obtained by summing twice the previous number and the number before (i.e., $h_n = 2h_{n-1} + h_{n-2}$):

$$\mathbf{h} = [0, 1, 2, 5, 12, 29, \dots\ ]$$

Determine the linear system $H(z)$ whose causal impulse response is $\mathbf{h}$, and express it as a rational function in z^{-1}. Using partial fractions, derive an expression for the nth Pell number in terms of the poles of the above filter. Show that the ratio of two successive Pell numbers converges to the positive solution of the quadratic equation $\theta^2 = 2\theta + 1$, that is, $\theta = 1 + \sqrt{2}$. Show that the filter's poles are the two numbers $\{\theta, -\theta^{-1}\}$. Show that the geometric sequence:

$$\mathbf{y} = [0, 1, \theta, \theta^2, \theta^3, \dots\ ]$$

satisfies the same recursion as the Pell sequence (for $n \geq 3$). Show that $\mathbf{y}$ may be considered to be the output of the filter $\mathbf{h}$ for a particular input. What is that input?

6.10 For a particular causal filter, it is observed that the input signal $(0.5)^n u(n)$ produces the output signal $(0.5)^n u(n) + (0.4)^n u(n)$. What *input* signal produces the output signal $(0.4)^n u(n)$?

6.11 For a particular filter, it is observed that the input signal $a^n u(n)$ causes the output signal $a^n u(n) + b^n u(n)$ to be produced. What output signal is produced by the input $c^n u(n)$, where $c = (a + b)/2$?

6.12 The signal $(0.7)^n u(n)$ is applied to the input of an unknown causal LTI filter, and the signal $(0.7)^n u(n) + (0.5)^n u(n)$ is observed at the output. What is the causal input signal that will cause the output $(0.5)^n u(n)$? What is the transfer function $H(z)$ of the system? Determine its causal impulse response $h(n)$, for all $n \geq 0$.

6.13 Design a resonator filter of the form $H(z) = \dfrac{1}{1 + a_1 z^{-1} + a_2 z^{-2}}$, which has a peak at $f_0 = 250$ Hz and a 3-dB width of $\Delta f = 20$ Hz and is operating at a rate of $f_s = 5$ kHz. What are the values of a_1 and a_2? Show that the *time constant* of the resonator is given approximately by

$$n_{\text{eff}} = -\frac{2 \ln \epsilon}{\Delta\omega}$$

which is valid for small $\Delta\omega$. For the designed filter, calculate the 40-dB value of n_{eff}, that is, corresponding to $\epsilon = 10^{-2}$. Compare the approximate and exact values of n_{eff}.

6.14 For *any* stable and causal filter, let τ_{40} and τ_{60} denote its 40-dB and 60-dB time constants, expressed in seconds. Show that they are related by: $\tau_{60} = 1.5\tau_{40}$.

6.15 Show that the 60-dB time constant of a resonator filter is given approximately by:

$$\tau_{60} = \frac{2.2}{\Delta f}$$

where τ_{60} is in seconds and Δf is the 3-dB width in Hz. When is the approximation valid?

6.16 It is desired to generate the following *periodic* waveform:

$$h(n) = [1, 2, 3, 4, 0, 0, 0, 0, 1, 2, 3, 4, 0, 0, 0, 0, \cdots\ ]$$

where the dots indicate the periodic repetition of the 8 samples $[1, 2, 3, 4, 0, 0, 0, 0]$.

a. Determine the filter $H(z)$ whose impulse response is the above periodic sequence. Express $H(z)$ as a ratio of two polynomials of degree less than 8.

b. Draw the canonical and direct realization forms of $H(z)$. Write the corresponding sample processing algorithms.

6.17 A digital sawtooth generator filter has a periodic impulse response:

$$\mathbf{h} = [0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, \cdots\ ]$$

where the dots indicate the periodic repetition of the length-4 sequence $\{0, 1, 2, 3\}$.

a. Determine the transfer function $H(z)$.

b. Draw the *direct* and *canonical* realization forms. Factor $H(z)$ into second-order sections with real coefficients. Draw the corresponding *cascade* realization.

c. For each of the above three realizations, write the corresponding I/O time-domain difference equations and sample-by-sample processing algorithms.

d. Using partial fractions, do an inverse z-transform of $H(z)$ and determine a closed form expression for the above impulse response $h(n)$ in the form

$$h(n) = A + B(-1)^n + 2C\cos\left(\frac{\pi n}{2}\right) + 2D\sin\left(\frac{\pi n}{2}\right), \qquad n \geq 0$$

What are the values of A, B, C, D ?

6.18 Consider the system: $H(z) = \dfrac{1 + z^{-1} + z^{-2} + z^{-3}}{1 - z^{-7}}$.

a. Without using partial fractions, determine the causal impulse response of the system. Explain your reasoning.

b. Draw the *canonical* realization form of the system. Write the I/O *difference equations* and the *sample processing algorithm* describing this realization.

c. The length-3 input signal $\mathbf{x} = [3, 2, 1]$ is applied as input to the system. Using any method, determine the output signal $y(n)$ for all $n \geq 0$. Explain your method.

6.19 A causal filter has transfer function: $H(z) = \dfrac{1 + z^{-1} + z^{-2} + z^{-3}}{1 - z^{-2}}$.

a. Determine the numerical values of the causal impulse response $h(n)$, for all $n \geq 0$.

b. Draw the canonical realization form of this filter and write the sample processing algorithm describing it.

6.20 A filter is described by the following sample processing algorithm:

```
for each input $x$ do:
    $w_0 = x + w_1$
    $y = w_0 + w_2$
    $w_2 = w_1$
    $w_1 = w_0$
```

a. Determine the transfer function $H(z)$ of this filter.

b. Show that it is *identically* equal to that of Problem 6.19.

6.21 A biomedical signal, sampled at a rate of 240 Hz, is plagued by 60 Hz power frequency interference and its harmonics.

Design a digital notch filter $H(z)$ that removes all these harmonics, but remains essentially flat at other frequencies.

[*Hint*: You may assume, although it is not necessary, that the signal has been prefiltered by a perfect antialiasing prefilter matched to the above sampling rate. Therefore, only the harmonics that lie in the $0 \le f < 240$ Hz Nyquist interval are relevant.]

6.22 A digital filter has transfer function, where $0 < a < 1$:

$$H(z) = \frac{1 - z^{-16}}{1 - az^{-16}}$$

a. What are the poles and zeros of this filter? Show them on the z-plane.

b. Draw a rough sketch of its magnitude response $|H(\omega)|$ over the frequency interval $0 \le \omega \le 2\pi$.

c. Determine the causal/stable impulse response $h(n)$ for all $n \ge 0$. Sketch it as a function of n. [*Hint*: Do not use PF expansions.]

d. Draw the *canonical* realization form and write the corresponding sample processing algorithm. (You may make use of the `delay` routine to simplify the algorithm.)

6.23 Find the *causal* impulse response $h(n)$, for all $n \ge 0$, of $H(z) = \dfrac{0.3 + 0.15z^{-1}}{1 - 0.5z^{-1}}$.

6.24 Let $H(z) = \dfrac{1 - a}{1 - az^{-1}}$ be a first-order lowpass filter (also called a first-order smoother), where $0 < a < 1$. Draw the canonical realization. Draw another realization that uses only one multiplier, (that is, a), one delay, and one adder and one subtractor. For both realizations, write the sample-by-sample processing algorithms. What would you say is the purpose of the chosen gain factor $1 - a$?

6.25 Let $H(z) = \dfrac{3 - 3z^{-1} - z^{-2}}{1 - 1.5z^{-1} - z^{-2}}$. Determine all possible impulse responses $h(n)$, for all n, and the corresponding ROCs.

6.26 A discrete system is described by the difference equation

$$y(n) = 2.5y(n-1) - y(n-2) + 3x(n) + 3x(n-2)$$

Using z-transforms, find *all* possible impulse responses $h(n)$ and indicate their causality and stability properties.

For the causal filter, determine the output $y(n)$ if the input is $x(n) = g(n) - 2g(n-1)$, where $g(n) = \cos(\pi n/2)u(n)$.

6.27 A signal $x(n)$ has frequency bandwidth $0 \le |\omega| \le \omega_c$, where $\omega_c < \pi$. The signal is applied to a lowpass filter $H(\omega)$ resulting in the output $y(n)$. Assuming that the filter has an approximately flat passband over $0 \le |\omega| \le \omega_c$ and is zero outside that band, and assuming that the filter has linear phase with a phase delay $d(\omega) = D$, show that the resulting output will be approximately equal to the delayed input $y(n) = Gx(n - D)$, where G is the filter's passband gain.

6.28 Consider a causal/stable filter $H(z) = \dfrac{N(z)}{(1 - p_1z^{-1})(1 - p_2z^{-1})\cdots(1 - p_Mz^{-1})}$, where the M poles are inside the unit circle $|p_i| < 1$, and the numerator $N(z)$ is a polynomial in z^{-1} of degree strictly less than M. Show that the impulse response can be expressed in the form:

$$h(n) = \sum_{i=1}^{M} A_i p_i^n u(n), \qquad \text{where} \quad A_i = \frac{N(p_i)}{\prod_{j\ne i}(1 - p_j p_i^{-1})}$$

6.29 The input-on behavior of the above filter may be studied by applying to it a one-sided sinusoid that starts at $n = 0$ and continues till $n = \infty$. The input-off behavior may be studied by applying a sinusoid that has been on since $n = -\infty$ and turns off at $n = 0$. Using z-transforms, show that the corresponding outputs are in the two cases:

$$e^{j\omega_0 n}u(n) \xrightarrow{H} y(n) = H(\omega_0)e^{j\omega_0 n}u(n) + \sum_{i=1}^{M} B_i p_i^n u(n)$$

$$e^{j\omega_0 n}u(-n-1) \xrightarrow{H} y(n) = H(\omega_0)e^{j\omega_0 n}u(-n-1) - \sum_{i=1}^{M} B_i p_i^n u(n)$$

Thus, the transient behavior for $n \ge 0$ is the same in both cases except for a sign. Show that the coefficients B_i are related to A_i of the previous problem by:

$$B_i = \frac{p_i A_i}{p_i - e^{j\omega_0}}, \qquad i = 1, 2, \dots, M$$

Using these results and linear superposition, derive the steady-state result of Eq. (6.3.2), which is valid for double-sided sinusoids.

6.30 Let $H(z) = \dfrac{3 - 5z^{-1} + z^{-2}}{(1 - 0.5z^{-1})(1 - 2z^{-1})}$. Determine the stable but anticausal impulse response $h(n)$ of this system. Let $\tilde{h}(n)$ denote the causal approximation to $h(n)$ obtained by clipping off the anticausal tail of $h(n)$ at some large negative time $n = -D$. What is $\tilde{H}(z)$?

Suppose the signal $x(n) = \delta(n) - 2\delta(n-1)$ is applied at the input of the true and approximate systems resulting in the outputs $y(n)$ and $\tilde{y}(n)$, respectively. Using z-transforms, determine an expression for the output error $e(n) = y(n) - \tilde{y}(n)$.

7

Digital Filter Realizations

7.1 Direct Form

In this section, we discuss in detail the *direct form* realizations of digital filters, otherwise known as *direct form I* (DF-I) realizations. We begin by considering a simple second-order filter with transfer function

$$\boxed{H(z)=\frac{N(z)}{D(z)}=\frac{b_0+b_1z^{-1}+b_2z^{-2}}{1+a_1z^{-1}+a_2z^{-2}}} \tag{7.1.1}$$

having I/O difference equation:

$$y_n=-a_1y_{n-1}-a_2y_{n-2}+b_0x_n+b_1x_{n-1}+b_2x_{n-2} \tag{7.1.2}$$

The direct form realization is the block diagram representation of this difference equation. It is depicted in Fig. 7.1.1.

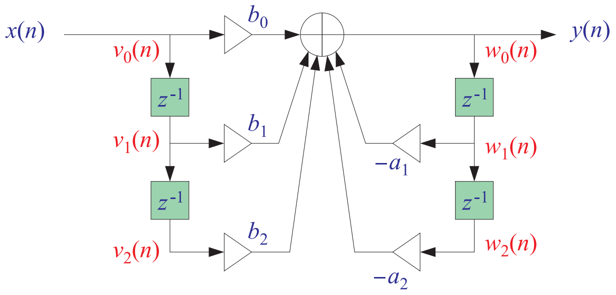


Fig. 7.1.1 Direct form realization of second-order IIR filter.

The main feature of this realization is a single adder that accumulates all the terms in the right-hand side of Eq. (7.1.2) and whose output is $y(n)$.

All the b-multiplier terms are *feeding forward*. They correspond to the *numerator* polynomial of $H(z)$ and to the x-dependent, *non-recursive* terms of the difference equation (7.1.2).

The a-multiplier terms are *feeding back*. They correspond to the *denominator* polynomial of $H(z)$ and to the y-dependent, *recursive* terms of Eq. (7.1.2). Notice the change in sign: The a-multipliers in the block diagram and in the difference equation are the *negatives* of the denominator polynomial coefficients.

The FIR direct form of Chapter 4 is obtained as a special case of this by setting the feedback coefficients to zero $a_1=a_2=0$.

The sample-by-sample processing algorithm can be derived by defining the internal states of the filter to be:

$$\begin{aligned} v_0(n)&=x(n) \\ v_1(n)&=x(n-1)=v_0(n-1) \\ v_2(n)&=x(n-2)=v_1(n-1) \end{aligned} \quad \text{and} \quad \begin{aligned} w_0(n)&=y(n) \\ w_1(n)&=y(n-1)=w_0(n-1) \\ w_2(n)&=y(n-2)=w_1(n-1) \end{aligned}$$

The quantities $v_1(n)$, $v_2(n)$, $w_1(n)$, and $w_2(n)$ are the internal states of the filter, representing the *contents* of the delay registers of the block diagram at time n. Replacing n by $n+1$ in the above definitions, we find the time updates:

$$\begin{aligned} v_1(n+1)&=v_0(n) \\ v_2(n+1)&=v_1(n) \end{aligned} \quad \text{and} \quad \begin{aligned} w_1(n+1)&=w_0(n) \\ w_2(n+1)&=w_1(n) \end{aligned}$$

Therefore, we may replace Eq. (7.1.2) by the system:

$$\boxed{\begin{aligned} &v_0(n)=x(n) \\ &w_0(n)=-a_1w_1(n)-a_2w_2(n)+b_0v_0(n)+b_1v_1(n)+b_2v_2(n) \\ &y(n)=w_0(n) \\ &v_2(n+1)=v_1(n),\quad w_2(n+1)=w_1(n) \\ &v_1(n+1)=v_0(n),\quad w_1(n+1)=w_0(n) \end{aligned}}$$

It can be replaced by the following repetitive sample processing algorithm:

```
for each input sample x do:
    $v_0 = x$
    $w_0 = -a_1w_1 - a_2w_2 + b_0v_0 + b_1v_1 + b_2v_2$
    $y = w_0$
    $v_2 = v_1,\quad w_2 = w_1$
    $v_1 = v_0,\quad w_1 = w_0$
```
(7.1.3)

Note that the state updating must be done in reverse order (from the bottom up in the block diagram).

The direct form realization can be generalized easily to the case of arbitrary numerator and denominator polynomials. One simply extends the structure downward by adding more delays and corresponding multipliers. In the general case, the transfer function is:

$$\boxed{H(z) = \frac{N(z)}{D(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_L z^{-L}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_M z^{-M}}} \tag{7.1.4}$$

having an Lth degree numerator and Mth degree denominator. The corresponding I/O difference equation is:

$$y_n = -a_1 y_{n-1} - a_2 y_{n-2} - \cdots - a_M y_{n-M} + b_0 x_n + b_1 x_{n-1} + \cdots + b_L x_{n-L} \tag{7.1.5}$$

Figure 7.1.2 shows the case $L = M$. To derive the sample processing algorithm in the general case, we define the internal state signals:

$$\begin{aligned} v_i(n) &= x(n-i), \quad i = 0, 1, \dots, L \\ w_i(n) &= y(n-i), \quad i = 0, 1, \dots, M \end{aligned} \tag{7.1.6}$$

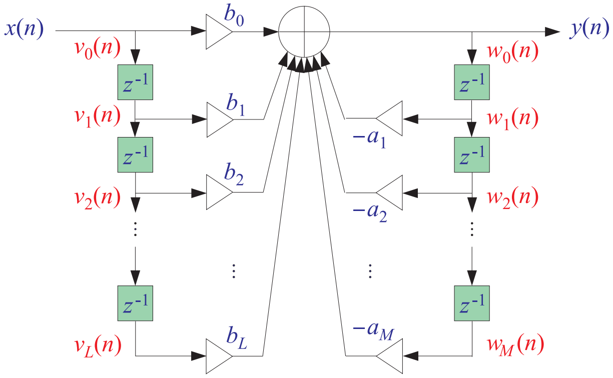


Fig. 7.1.2 Direct form realization of Mth order IIR filter.

They are updated in time by

$$\begin{aligned} v_i(n+1) &= v_{i-1}(n), \quad i = 1, 2, \dots, L \\ w_i(n+1) &= w_{i-1}(n), \quad i = 1, 2, \dots, M \end{aligned} \tag{7.1.7}$$

These can be shown easily, for example:

$$w_i(n+1) = y\big((n+1)-i\big) = y\big(n-(i-1)\big) = w_{i-1}(n)$$

Then, the difference equation (7.1.5) is be written as follows:

$$w_0(n) = -a_1 w_1(n) - \cdots - a_M w_M(n) + b_0 v_0(n) + b_1 v_1(n) + \cdots + b_L v_L(n)$$

Together with the time updates (7.1.7), it leads to the following sample processing algorithm for the direct form realization:

```
for each input sample x do:
    $v_0 = x$
    $w_0 = -a_1 w_1 - \cdots - a_M w_M + b_0 v_0 + b_1 v_1 + \cdots + b_L v_L$
    $y = w_0$
    $v_i = v_{i-1}, \quad i = L, L-1, \dots, 1$
    $w_i = w_{i-1}, \quad i = M, M-1, \dots, 1$
```

(7.1.8)

Again, the state updating must be done in *reverse* order to avoid overwriting v_i and w_i. Before filtering the first input sample, the internal states must be *initialized to zero*, that is, at time $n = 0$ set:

$$[v_1, v_2, \dots, v_L] = [0, 0, \dots, 0], \qquad [w_1, w_2, \dots, w_M] = [0, 0, \dots, 0]$$

The following C routine `dir.c` is an implementation of this algorithm:

```
/* dir.c - IIR filtering in direct form */

double dir(M, a, L, b, w, v, x)                 usage: y = dir(M, a, L, b, w, v, x);
double *a, *b, *w, *v, x;                       v, w are internal states
int M, L;                                       denominator and numerator orders
{
       int i;

       v[0] = x;                                current input sample
       w[0] = 0;                                current output to be computed

       for (i=0; i<=L; i++)                     numerator part
              w[0] += b[i] * v[i];

       for (i=1; i<=M; i++)                     denominator part
              w[0] -= a[i] * w[i];

       for (i=L; i>=1; i--)                     reverse-order updating of v
              v[i] = v[i-1];

       for (i=M; i>=1; i--)                     reverse-order updating of w
              w[i] = w[i-1];

       return w[0];                             current output sample
}
```

Note that `b`, `a` are the numerator and denominator coefficient vectors:

$$\mathbf{b} = [b_0, b_1, b_2, \dots, b_L], \qquad \mathbf{a} = [1, a_1, a_2, \dots, a_M] \tag{7.1.9}$$

They, and the internal state vectors `w`, `v`, must be declared and allocated in the main program by

```
double *a, *b, *w, *v;
a = (double *) calloc(M+1, sizeof(double));       (M+1)-dimensional
b = (double *) calloc(L+1, sizeof(double));       (L+1)-dimensional
a[0] = 1;                                         always so
w = (double *) calloc(M+1, sizeof(double));       (M+1)-dimensional
v = (double *) calloc(L+1, sizeof(double));       (L+1)-dimensional
```

Note that `calloc` initializes the internal states `w,v` to zero. The following program segment illustrates the usage of `dir`:

```
for (n = 0; n < Ntot; n++)
      y[n] = dir(M, a, L, b, w, v, x[n]);
```

Example 7.1.1: Draw the direct form realization of the following filter:

$$H(z)=\frac{2-3z^{-1}+4z^{-3}}{1+0.2z^{-1}-0.3z^{-2}+0.5z^{-4}}$$

and determine the corresponding difference equation and sample processing algorithm.

Solution: The difference equation is:

$$y_n=-0.2y_{n-1}+0.3y_{n-2}-0.5y_{n-4}+2x_n-3x_{n-1}+4x_{n-3}$$

The direct form realization is shown in Fig. 7.1.3. The sample processing algorithm is:

```
for each input sample x do:
    $v_0 = x$
    $w_0 = -0.2w_1 + 0.3w_2 - 0.5w_4 + 2v_0 - 3v_1 + 4v_3$
    $y = w_0$
    $w_4 = w_3$
    $w_3 = w_2,\quad v_3 = v_2$
    $w_2 = w_1,\quad v_2 = v_1$
    $w_1 = w_0,\quad v_1 = v_0$
```

The filter coefficient and state vectors are in this example:

$$\mathbf{a}=[a_0,a_1,a_2,a_3,a_4]=[1,0.2,-0.3,0.0,0.5]$$

$$\mathbf{b}=[b_0,b_1,b_2,b_3]=[2,-3,0,4]$$

$$\mathbf{w}=[w_0,w_1,w_2,w_3,w_4],\quad \mathbf{v}=[v_0,v_1,v_2,v_3]$$

There are four delays for the feedback coefficients and three for the feed forward ones, because the denominator and numerator polynomials have orders 4 and 3. Note also that some coefficients are zero and therefore their connections to the adder are not shown. □

The direct form algorithm (7.1.8) can be written more simply in terms of the dot product routine `dot` and delay routine `delay` in much the same way as was done in Chapter 4 for the FIR filtering routine `fir`. The output y in Eq. (7.1.8) involves essentially the *dot products* of the filter coefficient vectors with the internal state vectors. These dot products are:

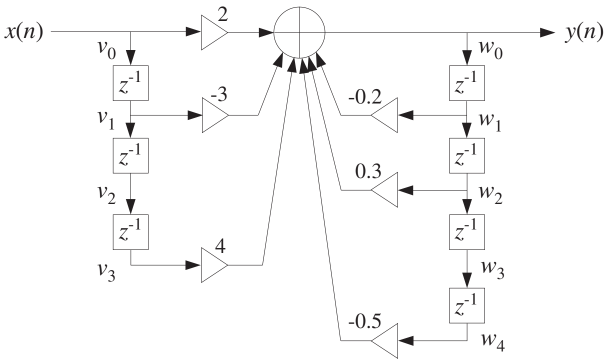


Fig. 7.1.3 Direct form realization of Example 7.1.1.

$$\text{dot}(L,\mathbf{b},\mathbf{v})=\mathbf{b}^T\mathbf{v}=b_0v_0+b_1v_1+\cdots+b_Lv_L$$

$$\text{dot}(M,\mathbf{a},\mathbf{w})=\mathbf{a}^T\mathbf{w}=a_0w_0+a_1w_1+\cdots+a_Mw_M$$

where $a_0=1$. If we set $w_0=0$, then the second dot product is precisely the contribution of the feedback (denominator) coefficients to the filter output, that is,

$$w_0=0\quad\Rightarrow\quad \text{dot}(M,\mathbf{a},\mathbf{w})=a_1w_1+\cdots+a_Mw_M$$

Therefore, we may replace Eq. (7.1.8) by the more compact version:

```
for each input sample x do:
    $v_0 = x$
    $w_0 = 0$
    $w_0 = \text{dot}(L,\mathbf{b},\mathbf{v}) - \text{dot}(M,\mathbf{a},\mathbf{w})$        (7.1.10)
    $y = w_0$
    $\text{delay}(L,\mathbf{v})$
    $\text{delay}(M,\mathbf{w})$
```

The following routine `dir2.c` is an implementation of this algorithm. It is the IIR version of the routine `fir2` of Chapter 4.

```
/* dir2.c - IIR filtering in direct form */

double dot();
void delay();

double dir2(M, a, L, b, w, v, x)          usage: y = dir2(M, a, L, b, w, v, x);
double *a, *b, *w, *v, x;
int M, L;
{
       v[0] = x;                                    current input sample
```

```
        w[0] = 0;                                   needed for dot(M,a,w)

        w[0] = dot(L, b, v) - dot(M, a, w);         current output

        delay(L, v);                                update input delay line

        delay(M, w);                                update output delay line

        return w[0];
}
```

7.2 Canonical Form

The *canonical realization form*, otherwise known as *direct form II* (DF-II), can be obtained from the direct form in the following way. Starting with the second-order filter of Eq. (7.1.1) we may group the five terms in the right-hand side of Eq. (7.1.2) into two subgroups: the recursive terms and the non-recursive ones, that is,

$$y_n = (b_0x_n + b_1x_{n-1} + b_2x_{n-2}) + (-a_1y_{n-1} - a_2y_{n-2})$$

This regrouping corresponds to splitting the big adder of the direct form realization of Fig. 7.1.1 into two parts, as shown in Fig. 7.2.1.

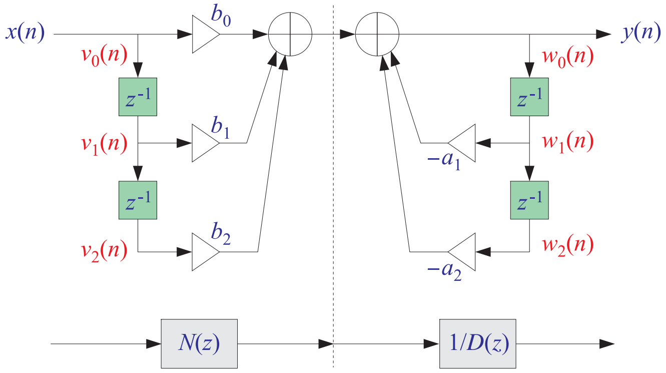


Fig. 7.2.1 Regrouping of direct form terms.

We can think of the resulting realization as the cascade of two filters: one consisting only of the feed-forward terms and the other of the feedback terms. It is easy to verify that these two filters are the numerator $N(z)$ and the inverse of the denominator $1/D(z)$, so that their cascade will be

$$H(z) = N(z)\cdot\frac{1}{D(z)}$$

which is the original transfer function given in Eq. (7.1.1). Mathematically, the order of the cascade factors can be changed so that

$$H(z) = \frac{1}{D(z)}\cdot N(z)$$

which corresponds to changing the order of the block diagrams representing the factors $N(z)$ and $1/D(z)$, as shown in Fig. 7.2.2.

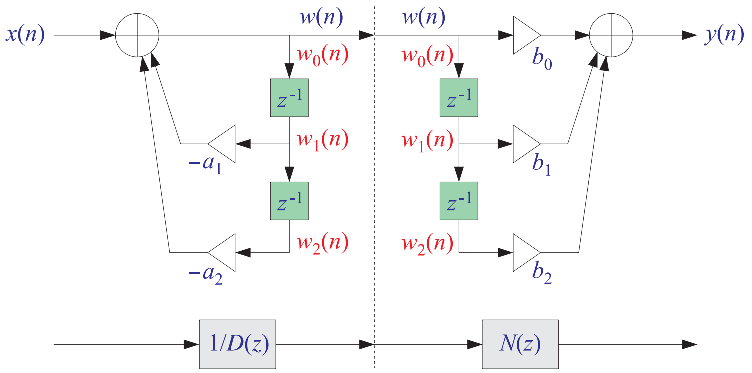


Fig. 7.2.2 Interchanging $N(z)$ and $1/D(z)$.

The output signal of the first filter $1/D(z)$ becomes the input to the second filter $N(z)$. If we denote that signal by $w(n)$, we observe that it gets delayed in the same way by the two sets of delays of the two filters, that is, the two sets of delays have the *same contents*, namely, the numbers $w(n-1)$, $w(n-2)$.

Therefore, there is no need to use two separate sets of delays. The two sets can be merged into one, shared by both the first and second filters $1/D(z)$ and $N(z)$. This leads to the *canonical realization form* depicted in Fig. 7.2.3.

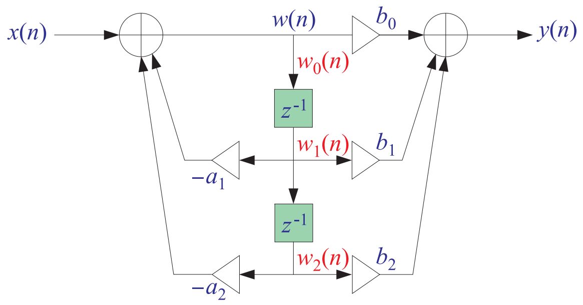


Fig. 7.2.3 Canonical realization form of second-order IIR filter.

The I/O difference equations describing the time-domain operation of this realization can be obtained by writing the conservation equations at each adder, with the input adder written first:

$$\boxed{\begin{aligned} w(n) &= x(n) - a_1 w(n-1) - a_2 w(n-2) \\ y(n) &= b_0 w(n) + b_1 w(n-1) + b_2 w(n-2) \end{aligned}} \tag{7.2.1}$$

The computed value of $w(n)$ from the first equation is passed into the second to compute the final output $y(n)$. It is instructive also to look at this system in the z-domain. Taking z-transforms of both sides, we find

$$\begin{aligned} W(z) &= X(z) - a_1 z^{-1} W(z) - a_2 z^{-2} W(z) \\ Y(z) &= b_0 W(z) + b_1 z^{-1} W(z) + b_2 z^{-2} W(z) \end{aligned}$$

which can be solved for $W(z)$ and $Y(z)$:

$$\begin{aligned} W(z) &= \frac{1}{1 + a_1 z^{-1} + a_2 z^{-2}} X(z) = \frac{1}{D(z)} X(z) \\ Y(z) &= (b_0 + b_1 z^{-1} + b_2 z^{-2}) W(z) = N(z) W(z) \end{aligned}$$

Eliminating $W(z)$, we find that the transfer function from $X(z)$ to $Y(z)$ is the original one, namely, $N(z)/D(z)$:

$$Y(z) = N(z) W(z) = N(z) \frac{1}{D(z)} X(z) = \frac{N(z)}{D(z)} X(z)$$

At each time n, the quantities $w(n-1)$ and $w(n-2)$ in Eq. (7.2.1) are the contents of the two shared delay registers. Therefore, they are the internal states of the filter. To determine the corresponding sample processing algorithm, we redefine these internal states by:

$$\begin{aligned} w_0(n) &= w(n) \\ w_1(n) &= w(n-1) = w_0(n-1) \\ w_2(n) &= w(n-2) = w_1(n-1) \end{aligned} \quad \Rightarrow \quad \begin{aligned} w_1(n+1) &= w_0(n) \\ w_2(n+1) &= w_1(n) \end{aligned}$$

Therefore, the system (7.2.1) can be rewritten as:

$$\boxed{\begin{aligned} & w_0(n) = x(n) - a_1 w_1(n) - a_2 w_2(n) \\ & y(n) = b_0 w_0(n) + b_1 w_1(n) + b_2 w_2(n) \\ & w_2(n+1) = w_1(n) \\ & w_1(n+2) = w_0(n) \end{aligned}}$$

which translates to the following sample processing algorithm:

```
for each input sample x do:                (7.2.2)
    $w_0 = x - a_1 w_1 - a_2 w_2$
    $y = b_0 w_0 + b_1 w_1 + b_2 w_2$
    $w_2 = w_1$
    $w_1 = w_0$
```

where, again, the states w_2 and w_1 must be updated in reverse order. The canonical form for the more general case of Eq. (7.1.4) is obtained following similar steps. That is, we define

$$Y(z) = N(z) W(z) \quad \text{and} \quad W(z) = \frac{1}{D(z)} X(z)$$

and rewrite them in the form:

$$\begin{aligned} (1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_M z^{-M}) W(z) &= X(z) \\ Y(z) &= (b_0 + b_1 z^{-1} + \cdots + b_L z^{-L}) W(z) \end{aligned}$$

or, equivalently

$$\begin{aligned} W(z) &= X(z) - (a_1 z^{-1} + a_2 z^{-2} + \cdots + a_M z^{-M}) W(z) \\ Y(z) &= (b_0 + b_1 z^{-1} + \cdots + b_L z^{-L}) W(z) \end{aligned}$$

which become in the time domain:

$$\boxed{\begin{aligned} w(n) &= x(n) - a_1 w(n-1) - \cdots - a_M w(n-M) \\ y(n) &= b_0 w(n) + b_1 w(n-1) + \cdots + b_L w(n-L) \end{aligned}} \tag{7.2.3}$$

The block diagram realization of this system is shown in Fig. 7.2.4 for the case $M = L$. If $M \neq L$ one must draw the *maximum* number of common delays, that is, $K = \max(M, L)$. Defining the internal states by

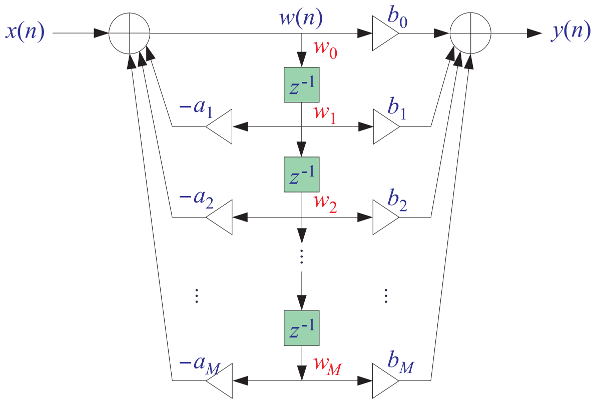


Fig. 7.2.4 Canonical realization form of Mth order IIR filter.

$$w_i(n) = w(n-i) = w_{i-1}(n-1), \quad i = 0, 1, \dots, K$$

we may rewrite the system (7.2.3) in the form:

$$\boxed{\begin{aligned} &w_0(n) = x(n) - a_1 w_1(n) - \cdots - a_M w_M(n) \\ &y(n) = b_0 w_0(n) + b_1 w_1(n) + \cdots + b_L w_L(n) \\ &w_i(n+1) = w_{i-1}(n), \quad i = K, K-1, \dots, 1 \end{aligned}} \tag{7.2.4}$$

This leads to the following sample-by-sample filtering algorithm:

```
for each input sample x do:
    $w_0 = x - a_1 w_1 - a_2 w_2 - \cdots - a_M w_M$
    $y = b_0 w_0 + b_1 w_1 + \cdots + b_L w_L$
    $w_i = w_{i-1}, \quad i = K, K-1, \dots, 1$
```
(7.2.5)

Again, the state updating must be done in reverse order. Before the first input sample, the internal states must be initialized to zero, that is, $[w_1, w_2, \dots, w_K] = [0, 0, \dots, 0]$. The following C routine `can.c` is an implementation of this algorithm:

```
/* can.c - IIR filtering in canonical form */

double can(M, a, L, b, w, x)                usage: y = can(M, a, L, b, w, x);
double *a, *b, *w, x;                       w = internal state vector
int M, L;                                   denominator and numerator orders
{
       int K, i;
       double y = 0;

       K = (L <= M) ? M : L;                K = max(M, L)

       w[0] = x;                            current input sample

       for (i=1; i<=M; i++)                 input adder
              w[0] -= a[i] * w[i];

       for (i=0; i<=L; i++)                 output adder
              y += b[i] * w[i];

       for (i=K; i>=1; i--)                 reverse updating of w
              w[i] = w[i-1];

       return y;                            current output sample
}
```

The vectors `a,b` must be allocated just as for the direct form routine `dir`. The state vector `w` must be allocated to dimension $K+1$, that is,

```
w = (double *) calloc(K+1, sizeof(double));        w = [w0, w1, ..., wK]
```

The same program segment illustrating the usage of `dir` also illustrates the usage of `can`. The only difference is that now there is only one state vector `w` instead of `w,v`:

```
for (n = 0; n < Ntot; n++)
       y[n] = can(M, a, L, b, w, x[n]);
```

Comparing Figs. 7.1.2 and 7.2.4, we note that: (a) The direct form requires *twice* as many delays; (b) both have exactly the *same* multiplier coefficients; (c) the direct form has only one adder whose output is the system output; and (d) the canonical form has *two* adders, one at the input and one at the output. Between the two, the canonical form is usually preferred in practice.

Note also that for FIR filters the denominator polynomial is trivial $D(z) = 1$ and thus, the direct and canonical forms are identical to the direct form of Chapter 4.

Example 7.2.1: Draw the canonical realization form of Example 7.1.1 and write the corresponding difference equations and sample processing algorithm.

Solution: Interchanging the feedback and feed-forward parts of Fig. 7.1.3 and merging the common delays, we obtain the canonical realization shown in Fig. 7.2.5. The difference equations describing this block diagram in the time domain are obtained from the two adder equations:

$$\begin{aligned} w(n) &= x(n) - 0.2w(n-1) + 0.3w(n-2) - 0.5w(n-4) \\ y(n) &= 2w(n) - 3w(n-1) + 4w(n-3) \end{aligned}$$

The corresponding sample processing algorithm is:

```
for each input sample x do:
    $w_0 = x - 0.2w_1 + 0.3w_2 - 0.5w_4$
    $y = 2w_0 - 3w_1 + 4w_3$
    $w_4 = w_3$
    $w_3 = w_2$
    $w_2 = w_1$
    $w_1 = w_0$
```

Here, the maximum number of delays is $K = \max(M, L) = \max(4, 3) = 4$. □

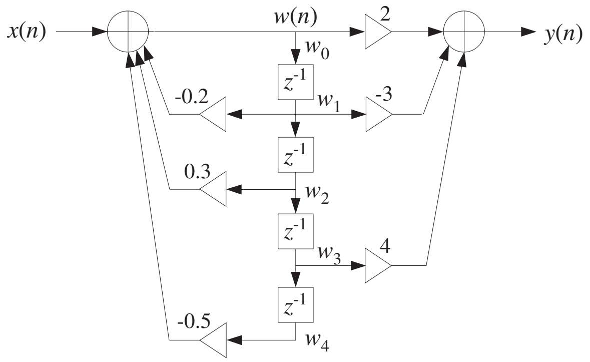


Fig. 7.2.5 Canonical realization form of Example 7.1.1.

Following the discussion of Eq. (7.1.10), we can derive an alternative version of the canonical form algorithm (7.2.5) that uses the dot product and delay routines, as follows:

```
for each input sample x do:                              (7.2.6)
    $w_0 = 0$
    $w_0 = x - \text{dot}(M, \mathbf{a}, \mathbf{w})$
    $y = \text{dot}(L, \mathbf{b}, \mathbf{w})$
    $\text{delay}(K, \mathbf{w})$
```

The C routine `can2.c` is an implementation.

```
/* can2.c - IIR filtering in canonical form */

double dot();
void delay();

double can2(M, a, L, b, w, x)              usage: y = can2(M, a, L, b, w, x);
double *a, *b, *w, x;
int M, L;
{
       int K;
       double y;

       K = (L <= M) ? M : L;                     K = max(M, L)

       w[0] = 0;                                 needed for dot(M,a,w)

       w[0] = x - dot(M, a, w);                  input adder

       y = dot(L, b, w);                         output adder

       delay(K, w);                              update delay line

       return y;                                 current output sample
}
```

7.3 Transposed Form

The *transposed realization* (also known as the observer-canonical form) is shown in Fig. 7.3.1 for a 2nd order transfer function.
Its sample processing algorithm is as follows,

```
initialize $v_1, v_2$                                    (7.3.1)
for each input sample $x$, do,
    $y = b_0 x + v_1$
    $v_1 = b_1 x - a_1 y + v_2$
    $v_2 = b_2 x - a_2 y$
```

The contents of the two delay registers, $v_1(n), v_2(n)$, are the internal state variables. Since the corresponding inputs to the delays must be the next values, $v_1(n+1), v_2(n+2)$, it follows that this realization is described by the following system of *first-order* difference equations,

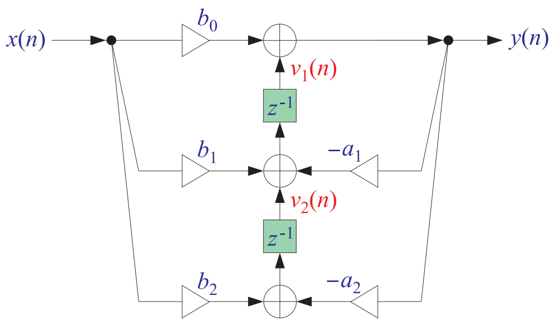


Fig. 7.3.1 Transposed realization.

$$\boxed{\begin{aligned} y(n) &= b_0 x(n) + v_1(n) \\ v_1(n+1) &= b_1 x(n) - a_1 y(n) + v_2(n) \\ v_2(n+1) &= b_2 x(n) - a_2 y(n) \end{aligned}} \qquad \text{(transposed realization)} \qquad (7.3.2)$$

Every realization has a *transposed version* obtained by the following transposition rules:

- replace adders by nodes
- replace nodes by adders
- reverse all flows
- exchange input with output

In this sense, the above transposed realization is recognized to be the transposed version of the canonical form. The canonical realization is perhaps the most widely used realization, however, it can often suffer from overflows and coefficient quantization effects. It has the advantage that it can be implemented in DSP hardware using circular delay-line buffers which reduce the number of operations per time update.

The transposed realization is fairly robust in terms of overflows and coefficient quantization, and is used by MATLAB's built-in function, **filter**.

Lattice/ladder realizations, discussed in Chap. 8, are the best realizations in terms of overflows and coefficient quantization [337–341], but at the price of effectively doubling the number of multiplier coefficients.

Each block diagram realization represents a particular way of arranging the required computations (the sample processing algorithm) of going from the input sample $x(n)$ to the output sample $y(n)$, and generally, the contents of the delay registers that appear in the block diagram are chosen as the temporary internal state variables that facilitate the output computation at successive time instants.

The reason for the existence of several realizations is that such output computations can be rearranged in different but mathematically equivalent ways. To make this point clearer, consider the case of a 2nd order transfer function

$$H(z)=\frac{b_0+b_1z^{-1}+b_2z^{-2}}{1+a_1z^{-1}+a_2z^{-2}}=\frac{N(z)}{D(z)}$$

and demonstrate the algebraic equivalence of the direct, canonical, and transposed realizations working in the z-domain. For the direct form, we have,

$$Y(z)=H(z)X(z)=\left[\frac{b_0+b_1z^{-1}+b_2z^{-2}}{1+a_1z^{-1}+a_2z^{-2}}\right]X(z)$$

$$[1+a_1z^{-1}+a_2z^{-2}]Y(z)=[b_0+b_1z^{-1}+b_2z^{-2}]X(z)$$

$$Y(z)+a_1z^{-1}Y(z)+a_2z^{-2}Y(z)=b_0X(z)+b_1z^{-1}X(z)+b_2z^{-2}X(z)$$

which leads to the time-domain difference equation,

$$y(n)+a_1y(n-1)+a_2y(n-2)=b_0x(n)+b_1x(n-1)+b_2x(n-2)\,,\quad\text{or,}$$

$$y(n)=-a_1y(n-1)-a_2y(n-2)+b_0x(n)+b_1x(n-1)+b_2x(n-2)$$

For the canonical realization, we have already seen how this is done,

$$\begin{aligned}W(z)&=\frac{1}{D(z)}X(z)\\ Y(z)&=N(z)W(z)=N(z)\cdot\frac{1}{D(z)}\cdot X(z)=H(z)X(z)\end{aligned}$$

which become,

$$D(z)W(z)=X(z)\quad\Rightarrow\quad[1+a_1z^{-1}+a_2z^{-2}]W(z)=X(z)$$

$$Y(z)=N(z)W(z)=[b_0+b_1z^{-1}+b_2z^{-2}]W(z)$$

and in the time domain,

$$\begin{aligned}w(n)&=-a_1w(n-1)-a_2w(n-2)+x(n)\\ y(n)&=b_0w(n)+b_1w(n-1)+b_2w(n-2)\end{aligned}$$

For the transposed realization, we rearrange the terms by regrouping like powers of z^{-1},

$$Y(z)=H(z)X(z)=\left[\frac{b_0+b_1z^{-1}+b_2z^{-2}}{1+a_1z^{-1}+a_2z^{-2}}\right]X(z)$$

$$[1+a_1z^{-1}+a_2z^{-2}]Y(z)=[b_0+b_1z^{-1}+b_2z^{-2}]X(z)$$

$$Y(z)=-a_1z^{-1}Y(z)-a_2z^{-2}Y(z)+b_0X(z)+b_1z^{-1}X(z)+b_2z^{-2}X(z)$$

$$Y(z)=b_0X(z)+z^{-1}\Big[b_1X(z)-a_1Y(z)+z^{-1}\big(b_2X(z)-a_2Y(z)\big)\Big]$$

$$Y(z)=b_0X(z)+\underbrace{z^{-1}\Big[b_1X(z)-a_1Y(z)+\underbrace{z^{-1}\big(b_2X(z)-a_2Y(z)\big)\Big]}_{V_2(z)}}_{V_1(z)}$$

so that the transposed block diagram realizes the last equation, or, in the time domain,

$$\begin{aligned}zV_2(z)&=b_2X(z)-a_2Y(z)\\ zV_1(z)&=b_1X(z)-a_1Y(z)+V_2(z)\end{aligned}\quad\Rightarrow\quad\begin{aligned}v_2(n+1)&=b_2x(n)-a_2y(n)\\ v_1(n+1)&=b_1x(n)-a_1y(n)+v_2(n)\end{aligned}$$

7.4 State-Space Realizations

Block diagram realizations can also be cast in *state-space form* with the contents of the delays that appear in the block diagram chosen to represent the internal states of the realization.

A so-called ABCD state-space realization is depicted in Fig. 7.4.1 and has the following standard form, written as a system of *first-order difference* equations,

$$\boxed{\begin{aligned}\mathbf{s}(n+1)&=A\mathbf{s}(n)+Bx(n)\\ y(n)&=C\mathbf{s}(n)+Dx(n)\end{aligned}}\qquad\text{(ABCD state-space realization)}\qquad(7.4.1)$$

where the state vector $\mathbf{s}(n)$ and the matrices A,B,C,D have appropriate dimensions. The corresponding sample processing algorithm for computing the output sample and updating the state vector can be stated as follows, where the operations must be done in the indicated order,

```
initialize s, then,
for each input sample x, do,
    y = Cs + Dx,    output
    s = As + Bx,    next state
```

(ABCD sample processing algorithm) (7.4.2)

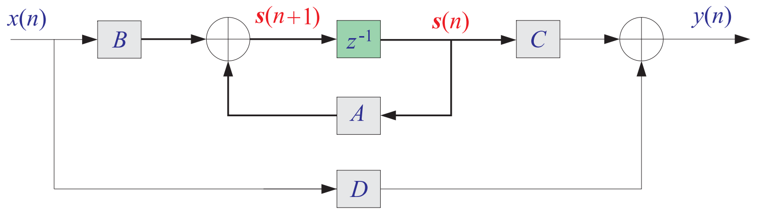


Fig. 7.4.1 ABCD state-space realization.

For example, the state vectors for the canonical and transposed realizations of our 2nd order example are the two-dimensional vectors chosen as the contents of the two delay registers that appear in their respective block diagrams, that is,

$$\mathbf{s}(n)=\begin{bmatrix}w_1(n)\\ w_2(n)\end{bmatrix}=\text{canonical}\,,\qquad\mathbf{s}(n)=\begin{bmatrix}v_1(n)\\ v_2(n)\end{bmatrix}=\text{transposed}\qquad(7.4.3)$$

The corresponding A, B, C, D matrices have dimensions, 2×2, 2×1, 1×2, and 1×1, respectively, and are given as follows in the two cases,

$$\begin{aligned}
&\text{(canonical):} \quad A = \begin{bmatrix} -a_1 & -a_2 \\ 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad C = [c_1, c_2], \quad D = b_0 \\
&\text{(transposed):} \quad A = \begin{bmatrix} -a_1 & 1 \\ -a_2 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}, \quad C = [1, 0], \quad D = b_0
\end{aligned} \tag{7.4.4}$$

where we defined the parameters,

$$\begin{aligned}
c_1 &= b_1 - b_0 a_1 \\
c_2 &= b_2 - b_0 a_2
\end{aligned}$$

Using the state-vector definition in Eq. (7.4.3), we may derive the state-space form of the transposed realization by rewriting Eq. (7.3.2) in the following way,

$$\begin{aligned}
y(n) &= b_0 x(n) + v_1(n) \\
v_1(n+1) &= b_1 x(n) - a_1 y(n) + v_2(n) = b_1 x(n) - a_1 [b_0 x(n) + v_1(n)] + v_2(n) \\
v_2(n+1) &= b_2 x(n) - a_2 y(n) = b_2 x(n) - a_2 [b_0 x(n) + v_1(n)]
\end{aligned}$$

or,

$$\begin{aligned}
v_1(n+1) &= -a_1 v_1(n) + v_2(n) + (b_1 - b_0 a_1)\,x(n) = -a_1 v_1(n) + v_2(n) + c_1 x(n) \\
v_2(n+1) &= -a_2 v_1(n) + (b_2 - b_0 a_2)\,x(n) = -a_2 v_1(n) + c_2 x(n) \\
y(n) &= v_1(n) + b_0 x(n)
\end{aligned}$$

or, reassembled in ABCD form,

$$\begin{aligned}
\begin{bmatrix} v_1(n+1) \\ v_2(n+1) \end{bmatrix} &= \begin{bmatrix} -a_1 & 1 \\ -a_2 & 0 \end{bmatrix} \begin{bmatrix} v_1(n) \\ v_2(n) \end{bmatrix} + \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} x(n) \\
y(n) &= [1, 0] \begin{bmatrix} v_1(n) \\ v_2(n) \end{bmatrix} + b_0 x(n)
\end{aligned}$$

Similarly, we have for the canonical form,

$$\begin{aligned}
w_1(n+1) &= w_0(n) = -a_1 w_1(n) - a_2 w_2(n) + x(n) \\
w_2(n+1) &= w_1(n) \\
y(n) &= b_0 w_0(n) + b_1 w_1(n) + b_2 w_2(n) \\
&= b_0 [-a_1 w_1(n) - a_2 w_2(n) + x(n)] + b_1 w_1(n) + b_2 w_2(n) \\
&= (b_1 - b_0 a_1) w_1(n) + (b_2 - b_0 a_2) w_2(n) + b_0 x(n) \\
&= c_1 w_1(n) + c_2 w_2(n) + b_0 x(n)
\end{aligned}$$

or, reassembled in ABCD form,

$$\begin{aligned}
\begin{bmatrix} w_1(n+1) \\ w_2(n+1) \end{bmatrix} &= \begin{bmatrix} -a_1 & -a_2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} w_1(n) \\ w_2(n) \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} x(n) \\
y(n) &= [c_1, c_2] \begin{bmatrix} w_1(n) \\ w_2(n) \end{bmatrix} + b_0 x(n)
\end{aligned}$$

Note that the ABCD parameters of the canonical and transposed cases are related to each other by the following mappings, which actually apply more generally to all transposed realizations and are effectively equivalent to the four transposition rules mentioned above,

$$\boxed{\begin{aligned} A &\longrightarrow A^T \\ B &\longrightarrow C^T \\ C &\longrightarrow B^T \\ D &\longrightarrow D \end{aligned}} \quad \text{(transposition mapping)} \tag{7.4.5}$$

In terms of the ABCD state-space parameters, the transfer function can be obtained by taking z-transforms of both sides of Eqs. (7.4.1) and eliminating the state variable,

$$\begin{aligned}
z\,\mathbf{S}(z) &= A\,\mathbf{S}(z) + B X(z) & \quad & \mathbf{S}(z) = (zI - A)^{-1} B X(z) \\
Y(z) &= C\,\mathbf{S}(z) + D X(z) & \Rightarrow \quad & Y(z) = C\,(zI - A)^{-1} B X(z) + D X(z)\,, \quad \text{or,}
\end{aligned}$$

$$\boxed{H(z) = \frac{Y(z)}{X(z)} = C\,(zI - A)^{-1} B + D} \tag{7.4.6}$$

where I denotes the identity matrix. We note that the transposition mapping (7.4.5) leaves (7.4.6) invariant. The corresponding impulse response is obtained by inverting Eq. (7.4.6) causally,

$$h(n) = C A^{n-1} B\, u(n-1) + D\,\delta(n) \tag{7.4.7}$$

We demonstrate Eq. (7.4.6) explicitly for the canonical realization with parameters given by Eq. (7.4.4),

$$\begin{aligned}
zI - A &= \begin{bmatrix} z & 0 \\ 0 & z \end{bmatrix} - \begin{bmatrix} -a_1 & -a_2 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} z + a_1 & -a_2 \\ -1 & z \end{bmatrix} \\
\det(zI - A) &= z^2 + a_1 z + a_2 \\
(zI - A)^{-1} &= \frac{1}{\det(zI - A)} \begin{bmatrix} z & a_2 \\ 1 & z + a_1 \end{bmatrix} = \frac{1}{z^2 + a_1 z + a_2} \begin{bmatrix} z & a_2 \\ 1 & z + a_1 \end{bmatrix} \\
C(zI - A)^{-1} B &= \frac{1}{z^2 + a_1 z + a_2} [c_1, c_2] \begin{bmatrix} z & a_2 \\ 1 & z + a_1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{c_1 z + c_2}{z^2 + a_1 z + a_2}
\end{aligned}$$

so that the complete transfer function is,

$$
\begin{aligned}
H(z) &= C(zI-A)^{-1}B + D = \frac{c_1 z + c_2}{z^2 + a_1 z + a_2} + b_0 \\
&= \frac{(b_1 - b_0 a_1)z + (b_2 - b_0 a_2)}{z^2 + a_1 z + a_2} + b_0 = \frac{b_0 z^2 + b_1 z + b_2}{z^2 + a_1 z + a_2} \\
&= \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}
\end{aligned}
$$

MATLAB's built-in function, **tf2ss**, maps a transfer function defined by numerator and denominator coefficients, num,den, to an ABCD state space form that is by default the canonical realization,

```
[A,B,C,D] = tf2ss(num,den);    % canonical state-space form
```

For example, for our 2nd order $H(z)$, it generates the parameters of the canonical form in Eq. (7.4.4),

```
num = [b0,b1,b2];
den = [1,a1,a2];
[A,B,C,D] = tf2ss(num,den);    % canonical state-space form
```

State-space realizations can also be optimized to minimize roundoff noise [342–346]. For example, in the case of a 2nd order section with complex-conjugate poles, the optimum state-space form is constructed by the following steps [345], assuming a transfer function of the form,

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{(1 - pz^{-1})(1 - p^* z^{-1})}$$

and setting,

$$
\begin{aligned}
& p = \sigma + j\omega \\
& \sigma = -\frac{a_1}{2}\,, \quad \omega = \sqrt{a_2 - \frac{a_1^2}{4}}\,, \quad \left(\text{must have of course}\,,\ a_2 > \frac{a_1^2}{4}\right) \\
& q_1 = b_1 - b_0 a_1\,, \quad q_2 = b_2 - b_0 a_2 \\
& \alpha_r = \frac{q_1}{2}\,, \quad \alpha_i = -\frac{q_1\sigma + q_2}{2\omega} \\
& \alpha = \alpha_r + j\alpha_i \\
& P = \frac{|\alpha|}{1 - |p|^2}\,, \quad Q = \mathrm{Im}\left[\frac{\alpha}{1 - p^2}\right], \quad k = \sqrt{\frac{P+Q}{P-Q}} \\
& B_1 = \sqrt{\frac{|\alpha| - \alpha_i}{P - Q}}\,, \quad B_2 = -\mathrm{sign}(\alpha_r)\sqrt{\frac{|\alpha| + \alpha_i}{P + Q}} \\
& C_1 = \frac{\alpha_r}{B_1}\,, \quad C_2 = \frac{\alpha_r}{B_2}
\end{aligned}
\tag{7.4.8}
$$

which define the $ABCD$ state-space parameters:

$$A = \begin{bmatrix} \sigma & \omega k \\ -\omega/k & \sigma \end{bmatrix}, \quad B = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}, \quad C = [C_1, C_2]\,, \quad D = b_0 \tag{7.4.9}$$

In the special case when $\alpha_r = 0$ and $\alpha_i > 0$, we have:

$$B_1 = 0\,, \quad B_2 = -\sqrt{\frac{2|\alpha|}{P+Q}}\,, \quad C_1 = \sqrt{2|\alpha|(P-Q)}\,, \quad C_2 = 0 \tag{7.4.10}$$

and in the case, $\alpha_r = 0$ and $\alpha_i < 0$:

$$B_1 = \sqrt{\frac{2|\alpha|}{P-Q}}\,, \quad B_2 = 0\,, \quad C_1 = 0\,, \quad C_2 = -\sqrt{2|\alpha|(P+Q)} \tag{7.4.11}$$

7.5 Cascade Form

A *second-order section* (SOS) is a second-order transfer function of the form (7.1.1). Its canonical realization is depicted in Fig. 7.2.3. In the time domain it operates according to the I/O system of difference equations given by Eq. (7.2.1) and the corresponding sample processing algorithm of Eq. (7.2.2).

It can be implemented by the routine can with $M = L = 2$ and three-dimensional coefficient and state arrays a,b,w. However, it proves convenient to write a special version of can as it applies to this specific case. The following C routine sos.c implements a second-order section:

```
/* sos.c - IIR filtering by single second order section */

double sos(a, b, w, x)                        a, b, w are 3-dimensional
double *a, *b, *w, x;                         a[0]= 1 always
{
       double y;

       w[0] = x - a[1] * w[1] - a[2] * w[2];
       y = b[0] * w[0] + b[1] * w[1] + b[2] * w[2];

       w[2] = w[1];
       w[1] = w[0];

       return y;
}
```

where a, b, w must be declared to be three-dimensional arrays in the main program, for example by

```
a = (double *) calloc(3, sizeof(double));          a = [1, a1, a2]
b = (double *) calloc(3, sizeof(double));          b = [b0, b1, b2]
w = (double *) calloc(3, sizeof(double));          w = [w0, w1, w2]
```

The *cascade realization form* of a general transfer function assumes that the transfer function is the product of such second-order sections:

$$\boxed{H(z)=\prod_{i=0}^{K-1}H_i(z)=\prod_{i=0}^{K-1}\frac{b_{i0}+b_{i1}z^{-1}+b_{i2}z^{-2}}{1+a_{i1}z^{-1}+a_{i2}z^{-2}}}\tag{7.5.1}$$

Any transfer function of the form (7.1.4) can be factored into second-order factors with *real-valued* coefficients, provided Eq. (7.1.4) has real coefficients.

The maximum order of the numerator and denominator polynomials in Eq. (7.5.1) is $2K$, that is, twice the number of second-order sections. By “second order” we really mean “up to second order”, and therefore, if some of the z^{-2} coefficients b_{i2} or a_{i2} are zero, the actual numerator and denominator orders will be $L \le 2K$ and $M \le 2K$.

A block diagram realization of Eq. (7.5.1) can be obtained by cascading together the block diagram realizations of the SOS filters $H_i(z)$. Each SOS may be realized in its canonical, direct, or transposed realizations. However, the convention is to realize all of them in their *canonical* form, as shown in Fig. 7.5.1.

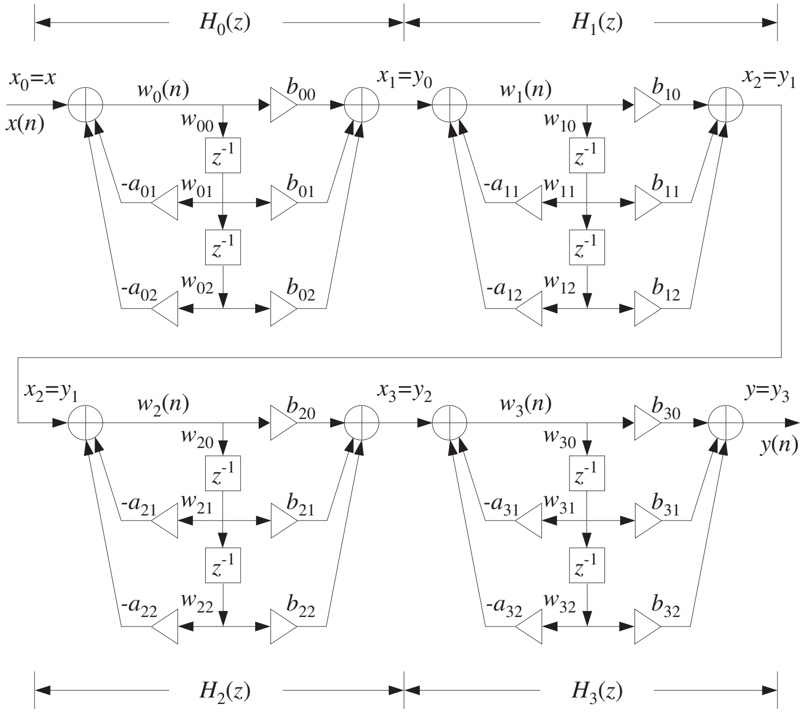


Fig. 7.5.1 Cascade of second-order sections.

Let us denote by $x_i(n)$, $y_i(n)$ the input and output signals of the ith section $H_i(z)$. Then, the overall input is the input to $H_0(z)$, namely, $x(n)= x_0(n)$, and the overall output is the output from the last SOS $H_{K-1}(z)$, namely, $y(n)= y_{K-1}(n)$. For the

intermediate stages, the output $y_i(n)$ of the ith section becomes the input to the $(i+1)$th section $H_{i+1}(z)$, that is,

$$x_{i+1}(n)= y_i(n),\quad i = 0,1,\dots,K-1$$

Each section has its own internal state vector $\mathbf{w}_i(n)= [w_{i0}(n), w_{i1}(n), w_{i2}(n)]$, $i = 0,1,\dots,K-1$, where the numbers $w_{i1}(n)$, $w_{i2}(n)$ are the contents of the section’s delay registers at the nth time instant.

The I/O difference equations describing the time-domain operation of the realization are obtained by writing the difference equations (7.2.1) for each SOS and passing the output of each to the input of the next:

$$\boxed{\begin{array}{l}
x_0(n)= x(n)\\
\textit{for } i = 0,1,\dots,K-1 \textit{ do:}\\
\qquad w_i(n)= x_i(n)-a_{i1}w_i(n-1)-a_{i2}w_i(n-2)\\
\qquad y_i(n)= b_{i0}w_i(n)+b_{i1}w_i(n-1)+b_{i2}w_i(n-2)\\
\qquad x_{i+1}(n)= y_i(n)\\
y(n)= y_{K-1}(n)
\end{array}}\tag{7.5.2}$$

It can be translated to the following sample processing algorithm:

```
for each input sample x do:
    $x_0 = x$
    for $i = 0,1,\dots,K-1$ do:
        $w_{i0} = x_i - a_{i1}w_{i1} - a_{i2}w_{i2}$
        $y_i = b_{i0}w_{i0} + b_{i1}w_{i1} + b_{i2}w_{i2}$
        $w_{i2} = w_{i1}$
        $w_{i1} = w_{i0}$
        $x_{i+1} = y_i$
    $y = y_{K-1}$
```
(7.5.3)

where, the internal state vector $\mathbf{w}_i$ of the ith section is defined at time n by:

$$\begin{aligned}
w_{i0}(n) &= w_i(n)\\
w_{i1}(n) &= w_i(n-1)\,,\quad \text{for}\quad i = 0,1,\dots,K-1\\
w_{i2}(n) &= w_i(n-2)
\end{aligned}$$

To keep track of the coefficients of the sections and the internal states, we arrange them into $K{\times}3$ matrices whose ith rows hold the corresponding parameters of the ith section. For example, if $K = 4$ as in Fig. 7.5.1, we define

$$A=\begin{bmatrix}1 & a_{01} & a_{02}\\ 1 & a_{11} & a_{12}\\ 1 & a_{21} & a_{22}\\ 1 & a_{31} & a_{32}\end{bmatrix},\quad B=\begin{bmatrix}b_{00} & b_{01} & b_{02}\\ b_{10} & b_{11} & b_{12}\\ b_{20} & b_{21} & b_{22}\\ b_{30} & b_{31} & b_{32}\end{bmatrix},\quad W=\begin{bmatrix}w_{00} & w_{01} & w_{02}\\ w_{10} & w_{11} & w_{12}\\ w_{20} & w_{21} & w_{22}\\ w_{30} & w_{31} & w_{32}\end{bmatrix}$$

The ith rows of these matrices are the three-dimensional coefficient vectors and states of the ith section, that is,

$$\begin{aligned}\mathbf{a}_i &= [1, a_{i1}, a_{i2}]\\ \mathbf{b}_i &= [b_{i0}, b_{i1}, b_{i2}], \qquad \text{for} \quad i = 0, 1, \dots, K-1\\ \mathbf{w}_i &= [w_{i0}, w_{i1}, w_{i2}]\end{aligned} \tag{7.5.4}$$

In this notation, we may rewrite the sample processing algorithm (7.5.3) as K *successive calls* to the basic SOS routine `sos`:

```
for each input sample $x$ do:
     $y = x$
     for $i = 0, 1, \dots, K-1$ do:
          $y = \text{sos}(\mathbf{a}_i, \mathbf{b}_i, \mathbf{w}_i, y)$
```

(7.5.5)

where y denotes both the input and output of each section. The last computed y is the final output. The C implementation of this algorithm is given by the following routine `cas.c`:

```
/* cas.c - IIR filtering in cascade of second-order sections */

double sos();                                   single second-order section

double cas(K, A, B, W, x)
int K;
double **A, **B, **W, x;                        A, B, W are K×3 matrices
{
       int i;
       double y;

       y = x;                                   initial input to first SOS

       for (i=0; i<K; i++)
              y = sos(A[i], B[i], W[i], y);     output of ith section

       return y;                                final output from last SOS
}
```

The coefficient and state matrices `A`, `B`, `W` must be dimensioned to size $K\times 3$ and allocated in the main program, for example, by

```
double **A, **B, **W;

A = (double **) calloc(K, sizeof(double *));          allocate K rows
B = (double **) calloc(K, sizeof(double *));
W = (double **) calloc(K, sizeof(double *));
for (i=0; i<K; i++) {
   A[i] = (double *) calloc(3, sizeof(double));       allocate each row
   B[i] = (double *) calloc(3, sizeof(double));
   W[i] = (double *) calloc(3, sizeof(double));
   }
```

Alternatively, if the value of K is known in advance, we may declare:

```
double A[K][3], B[K][3], W[K][3];
```

In that case, the declarations inside `cas` must also be modified to read:

```
double A[][3], B[][3], W[][3];
```

The quantities `A[i]`, `B[i]`, `W[i]` are the ith rows of `A`, `B`, `W`, as given by Eq. (7.5.4). The states `W` must be initialized to zero before the first call to `cas`; this is accomplished indirectly by `calloc`. The usage of `cas` is the same as `can`; that is,

```
for (n = 0; n < Ntot; n++)
       y[n] = cas(K, A, B, W, x[n]);
```

Example 7.5.1: Draw the cascade and canonical realizations of the following filter:

$$\begin{aligned}H(z) &= \left[\frac{3-4z^{-1}+2z^{-2}}{1-0.4z^{-1}+0.5z^{-2}}\right]\left[\frac{3+4z^{-1}+2z^{-2}}{1+0.4z^{-1}+0.5z^{-2}}\right] = H_0(z)H_1(z)\\ &= \frac{9-4z^{-2}+4z^{-4}}{1+0.84z^{-2}+0.25z^{-4}}\end{aligned}$$

Write the corresponding I/O difference equations and sample processing algorithms.

Solution: The cascade realization is shown in Fig. 7.5.2 and the canonical one in Fig. 7.5.3. The I/O difference equations describing the cascade realization in the time domain are:

$$\begin{aligned}w_0(n) &= x(n) + 0.4w_0(n-1) - 0.5w_0(n-2)\\ x_1(n) &= 3w_0(n) - 4w_0(n-1) + 2w_0(n-2)\\ w_1(n) &= x_1(n) - 0.4w_1(n-1) - 0.5w_1(n-2)\\ y(n) &= 3w_1(n) + 4w_1(n-1) + 2w_1(n-2)\end{aligned}$$

where $x_1(n)$ is the output of $H_0(z)$ and the input to $H_1(z)$. The corresponding sample processing algorithm is:

```
for each input sample $x$ do:
     $w_{00} = x + 0.4w_{01} - 0.5w_{02}$
     $x_1 = 3w_{00} - 4w_{01} + 2w_{02}$
     $w_{02} = w_{01}$
     $w_{01} = w_{00}$
     $w_{10} = x_1 - 0.4w_{11} - 0.5w_{12}$
     $y = 3w_{10} + 4w_{11} + 2w_{12}$
     $w_{12} = w_{11}$
     $w_{11} = w_{10}$
```

The coefficient and state matrices in the routine `cas` are in this case:

$$A = \begin{bmatrix} 1 & -0.4 & 0.5\\ 1 & 0.4 & 0.5\end{bmatrix}, \quad B = \begin{bmatrix} 3 & -4 & 2\\ 3 & 4 & 2\end{bmatrix}, \quad W = \begin{bmatrix} w_{00} & w_{01} & w_{02}\\ w_{10} & w_{11} & w_{12}\end{bmatrix}$$

For the canonical case, we have the coefficient vectors for the numerator and denominator polynomials:

$$\mathbf{b} = [9,\ 0,\ -4,\ 0,\ 4], \quad \mathbf{a} = [1.00,\ 0.00,\ 0.84,\ 0.00,\ 0.25]$$

The difference equation at the input and output adders of Fig. 7.5.3 are:

$$w(n) = x(n) - 0.84w(n-2) - 0.25w(n-4)$$
$$y(n) = 9w(n) - 4w(n-2) + 4w(n-4)$$

Defining the internal states as $w_i(n) = w(n-i)$, $i = 0, 1, 2, 3, 4$, we find the sample processing algorithm:

```
for each input sample x do:
    $w_0 = x - 0.84w_2 - 0.25w_4$
    $y = 9w_0 - 4w_2 + 4w_4$
    $w_4 = w_3$
    $w_3 = w_2$
    $w_2 = w_1$
    $w_1 = w_0$
```

The total number of internal states in the cascade and the canonical realizations is the same, namely, four. □

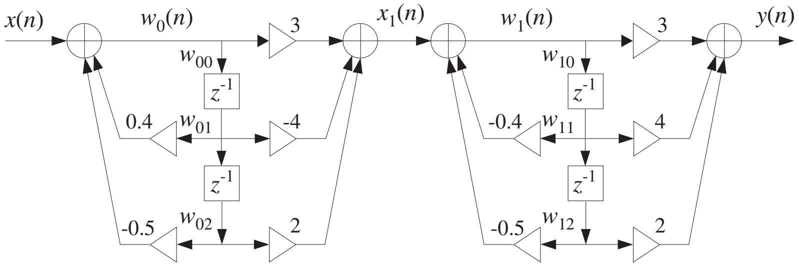


Fig. 7.5.2 Cascade realization of Example 7.5.1.

Example 7.5.2: Consider the filter

$$H(z) = \left[\frac{1+z^{-1}+z^{-2}}{1-0.7z^{-2}}\right]\left[\frac{1-z^{-2}}{1-0.6z^{-1}+0.4z^{-2}}\right]\left[\frac{1-z^{-1}+z^{-2}}{1+0.5z^{-1}+0.3z^{-2}}\right]$$
$$= \frac{1-z^{-6}}{1-0.1z^{-1}-0.3z^{-2}+0.09z^{-3}-0.16z^{-4}-0.014z^{-5}-0.084z^{-6}}$$

To illustrate the usage of the routines `cas` and `can`, we generated and filtered the following "step" input signal of length 100:

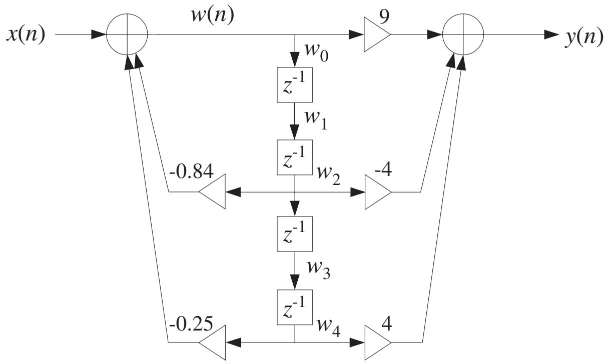


Fig. 7.5.3 Canonical realization of Example 7.5.1.

$$x(n) = \begin{cases} 2, & \text{if } 0 \le n \le 24 \\ 0, & \text{if } 25 \le n \le 49 \\ -1, & \text{if } 50 \le n \le 74 \\ 1, & \text{if } 75 \le n \le 99 \end{cases}$$

The resulting output signal $y(n)$ can be computed either by the routine `cas` or by `can`. The cascade realization coefficient matrices are:

$$A = \begin{bmatrix} 1 & 0 & -0.7 \\ 1 & -0.6 & 0.4 \\ 1 & 0.5 & 0.3 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 1 & -1 & 1 \end{bmatrix}$$

Similarly, the canonical form coefficients of the sixth degree numerator and denominator polynomials of $H(z)$ are:

$$\mathbf{b} = [1, 0, 0, 0, 0, 0, -1]$$
$$\mathbf{a} = [1, -0.1, -0.3, 0.09, -0.16, -0.014, -0.084]$$

These quantities, as well as the cascade state matrix W and canonical internal state vector $\mathbf{w}$, must be declared, allocated, and initialized in the main program as discussed above (with $K = 3$, $L = M = 6$). The output signal can be generated by the for-loop:

```
for (n=0; n<100; n++) {
       ycas[n] = cas(3, A, B, W, x[n]);
       ycan[n] = can(6, a, 6, b, w, x[n]);
       }
```

The two output signals $y_{\text{cas}}(n)$ and $y_{\text{can}}(n)$ generated by the two routines `cas` and `cas` are, of course, the same. This output signal is shown in Fig. 7.5.4.

Notice also that for this particular example, the pole closest to the unit circle is that of the first section, that is, $p = \pm\sqrt{0.7} = \pm 0.8367$. Therefore, the $\epsilon = 1\% = 0.01$ time constant

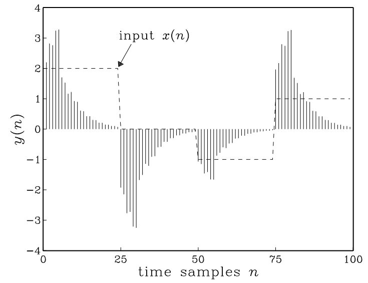


Fig. 7.5.4 Output of Example 7.5.2.

will be $n_{\text{eff}} = \ln\epsilon/\ln(0.8367) \simeq 26$. Because the filter has a zero at $z = 1$, its unit-step response will be $H(0) = H(z)\big|_{z=1} = 0$. As the step input changes level every 25 samples, the output tries to settle to its zero steady-state value with a time constant of about n_{eff}. □

7.6 *Cascade to Canonical*

To pass from the direct or canonical realization, Eq. (7.1.4), to the cascade realization, Eq. (7.5.1), requires factoring the numerator and denominator polynomials into their second-order factors.

This can be done by finding the roots of these polynomials and then pairing them in complex conjugate pairs. The procedure is outlined below. Given the M zeros p_i, $i = 1, 2, \dots, M$ of the denominator polynomial of Eq. (7.1.4), we can factor it into its root factors:

$$\begin{aligned} D(z) &= 1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_M z^{-M} \\ &= (1 - p_1 z^{-1})(1 - p_2 z^{-1})\cdots(1 - p_M z^{-1}) \end{aligned}$$

The root factors of any real-valued roots can be left as they are or combined in pairs. For example, if both p_1 and p_2 are real, we may combine them into the SOS with *real* coefficients:

$$(1 - p_1 z^{-1})(1 - p_2 z^{-1}) = \big(1 - (p_1 + p_2)z^{-1} + p_1 p_2 z^{-2}\big)$$

If any roots are complex, they must appear in complex-conjugate pairs, for example, if p_1 is a complex root, then $p_2 = p_1^*$ must also be a root. Combining the root factors of conjugate pairs results into an SOS with *real* coefficients, for example

$$\begin{aligned} (1 - p_1 z^{-1})(1 - p_1^* z^{-1}) &= 1 - (p_1 + p_1^*)z^{-1} + p_1 p_1^* z^{-2} \\ &= 1 - 2\text{Re}(p_1)z^{-1} + |p_1|^2 z^{-2} \end{aligned}$$

This identity was also used in Chapter 5. Using the polar representation of the complex number $p_1 = R_1 e^{j\theta_1}$, we have $\text{Re}(p_1) = R_1\cos\theta_1$ and $|p_1|^2 = R_1^2$, and we can write the above identity in the alternative form:

$$\begin{aligned} (1 - p_1 z^{-1})(1 - p_1^* z^{-1}) &= 1 - 2\text{Re}(p_1)z^{-1} + |p_1|^2 z^{-2} \\ &= 1 - 2R_1\cos(\theta_1)z^{-1} + R_1^2 z^{-2} \end{aligned}$$

Once the denominator and numerator polynomials have been factored into their quadratic factors, each quadratic factor from the numerator may be paired with a quadratic factor from the denominator to form a second-order section.

This pairing of numerator and denominator factors and the ordering of the resulting SOSs is *not unique*, but the overall transfer function will be the same. In practice, however, the particular pairing/ordering may make a difference.

In a hardware realization, the internal multiplications in each SOS will generate a certain amount of roundoff error which is then propagated into the next SOS. The net roundoff error at the overall output will depend on the particular pairing/ordering of the quadratic factors. The optimal ordering is the one that generates the *minimum* net roundoff error. Finding this optimal ordering is a difficult problem and is beyond the scope of this book.

Some examples will illustrate the above factoring technique. The most tedious part is finding the actual roots of the numerator and denominator polynomials. For high-order polynomials, one must use a root-finding routine from a numerical software package such as MATLAB or Mathematica. Some special cases of high-order polynomials can be handled by hand, as seen below.

Example 7.6.1: Determine the cascade realization form of the filter:

$$H(z) = \frac{1 - 1.5z^{-1} + 0.48z^{-2} - 0.33z^{-3} + 0.9376z^{-4} - 0.5328z^{-5}}{1 + 2.2z^{-1} + 1.77z^{-2} + 0.52z^{-3}}$$

Solution: Using MATLAB, we find the five roots of the numerator polynomial:

$$z = 0.9, \quad -0.5 \pm 0.7j, \quad 0.8 \pm 0.4j$$

They lead to the following root factors, already paired in conjugate pairs:

$$\begin{aligned} &(1 - 0.9z^{-1}) \\ &\big(1 - (-0.5 + 0.7j)z^{-1}\big)\big(1 - (-0.5 - 0.7j)z^{-1}\big) = (1 + z^{-1} + 0.74z^{-2}) \\ &\big(1 - (0.8 + 0.4j)z^{-1}\big)\big(1 - (0.8 - 0.4j)z^{-1}\big) = (1 - 1.6z^{-1} + 0.8z^{-2}) \end{aligned}$$

Similarly, we find the roots of the denominator:

$$p = -0.8, \quad -0.7 \pm 0.4j$$

giving the root factors:

$$(1+0.8z^{-1})$$

$$\big(1-(-0.7+0.4j)z^{-1}\big)\big(1-(-0.7-0.4j)z^{-1}\big) = (1+1.4z^{-1}+0.65z^{-2})$$

Therefore, a possible pairing/ordering of SOS factors for $H(z)$ will be:

$$H(z) = \frac{1-0.9z^{-1}}{1+0.8z^{-1}}\cdot\frac{1+z^{-1}+0.74z^{-2}}{1+1.4z^{-1}+0.65z^{-2}}\cdot(1-1.6z^{-1}+0.8z^{-2})$$

The coefficient matrices A and B needed for programming this filter by the routine `cas` will be:

$$A = \begin{bmatrix} 1 & 0.8 & 0 \\ 1 & 1.4 & 0.65 \\ 1 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & -0.9 & 0 \\ 1 & 1 & 0.74 \\ 1 & -1.6 & 0.8 \end{bmatrix}$$

The first-order section may be considered as special case of an SOS of the form (7.1.1) with zero z^{-2} coefficients, that is, $b_2 = a_2 = 0$. Similarly, the last quadratic factor is a special case of an FIR SOS, that is, with $a_1 = a_2 = 0$ (but $a_0 = 1$). □

Example 7.6.2: Determine the cascade form of the filter:

$$H(z) = \frac{1-0.48z^{-2}+0.5476z^{-4}}{1+0.96z^{-2}+0.64z^{-4}}$$

Solution: Even though the polynomials have degree 4, the z^{-1} and z^{-3} terms are missing, and we may think of the polynomials as *quadratic* in the variable z^{-2}. That is, we can find the roots of the denominator by solving the quadratic equation

$$1+0.96z^{-2}+0.64z^{-4} = 0 \quad\Rightarrow\quad (z^2)^2+0.96(z^2)+0.64 = 0$$

which has two solutions:

$$z^2 = \frac{-0.96\pm\sqrt{0.96^2-4\times 0.64}}{2} = -0.48\pm 0.64j$$

Taking square roots, we obtain the four roots of the denominator:

$$p = \pm\sqrt{-0.48\pm 0.64j} = \pm(0.4\pm 0.8j)$$

Pairing them in conjugate pairs gives the quadratic factors:

$$\big(1-(0.4+0.8j)z^{-1}\big)\big(1-(0.4-0.8j)z^{-1}\big) = 1-0.8z^{-1}+0.8z^{-2}$$
$$\big(1+(0.4+0.8j)z^{-1}\big)\big(1+(0.4-0.8j)z^{-1}\big) = 1+0.8z^{-1}+0.8z^{-2}$$

Similarly, we find for the numerator polynomial:

$$1-0.48z^{-2}+0.5476z^{-4} = 0 \quad\Rightarrow\quad z^2 = 0.24\pm 0.7j$$

and taking square roots:

$$z = \pm\sqrt{0.24\pm 0.7j} = \pm(0.7\pm 0.5j)$$

The quadratic factors are:

$$\big(1-(0.7+0.5j)z^{-1}\big)\big(1-(0.7-0.5j)z^{-1}\big) = 1-1.4z^{-1}+0.74z^{-2}$$
$$\big(1+(0.7+0.5j)z^{-1}\big)\big(1+(0.7-0.5j)z^{-1}\big) = 1+1.4z^{-1}+0.74z^{-2}$$

Thus, we find for $H(z)$:

$$H(z) = \frac{1-1.4z^{-1}+0.74z^{-2}}{1-0.8z^{-1}+0.8z^{-2}}\cdot\frac{1+1.4z^{-1}+0.74z^{-2}}{1+0.8z^{-1}+0.8z^{-2}}$$

which is one possible ordering of the quadratic factors. □

Example 7.6.3: As another special case, determine the cascade form of the filter:

$$H(z) = \frac{1+z^{-8}}{1-0.0625z^{-8}}$$

Solution: The roots of the numerator are the 8 solutions of:

$$1+z^{-8} = 0 \quad\Rightarrow\quad z^8 = -1 = e^{j\pi} = e^{j\pi}e^{2\pi jk} = e^{j(2k+1)\pi}$$

where we multiplied by $e^{2\pi jk} = 1$ for integer k. Taking eighth roots of both sides we find:

$$z_k = e^{j(2k+1)\pi/8}, \quad k = 0,1,\dots,7$$

We have the following conjugate pairs, as shown in Fig. 7.6.1: $\{z_0,z_7\}$, $\{z_1,z_6\}$, $\{z_2,z_5\}$, and $\{z_3,z_4\}$, which lead to the quadratic factors:

$$(1-z_0z^{-1})(1-z_7z^{-1}) = 1-2\cos\Big(\frac{\pi}{8}\Big)z^{-1}+z^{-2} = 1-1.8478z^{-1}+z^{-2}$$
$$(1-z_1z^{-1})(1-z_6z^{-1}) = 1-2\cos\Big(\frac{3\pi}{8}\Big)z^{-1}+z^{-2} = 1-0.7654z^{-1}+z^{-2}$$
$$(1-z_2z^{-1})(1-z_5z^{-1}) = 1-2\cos\Big(\frac{5\pi}{8}\Big)z^{-1}+z^{-2} = 1+0.7654z^{-1}+z^{-2}$$
$$(1-z_3z^{-1})(1-z_4z^{-1}) = 1-2\cos\Big(\frac{7\pi}{8}\Big)z^{-1}+z^{-2} = 1+1.8478z^{-1}+z^{-2}$$

Similarly, the filter poles are the roots of the denominator:

$$1-0.0625z^{-8} = 0 \quad\Rightarrow\quad z^8 = 0.0625 = 0.0625e^{2\pi jk} = (0.5)^4e^{2\pi jk}$$

which has the eight solutions:

$$p_k = \sqrt{0.5}e^{2\pi jk/8}, \quad k = 0,1,\dots,7$$

Of these, $p_0 = \sqrt{0.5}$ and $p_4 = \sqrt{0.5}e^{2\pi j4/8} = -\sqrt{0.5}$ are real and may be paired together into one SOS. The rest are complex and can be paired in conjugate pairs: $\{p_1, p_7\}$, $\{p_2, p_6\}$, $\{p_3, p_5\}$, resulting into the quadratic factors:

$$(1 - p_0z^{-1})(1 - p_4z^{-1}) = (1 - \sqrt{0.5}z^{-1})(1 + \sqrt{0.5}z^{-1}) = 1 - 0.5z^{-2}$$

$$(1 - p_1z^{-1})(1 - p_7z^{-1}) = 1 - \sqrt{2}\cos\big(\frac{2\pi}{8}\big)z^{-1} + 0.5z^{-2} = 1 - z^{-1} + 0.5z^{-2}$$

$$(1 - p_2z^{-1})(1 - p_6z^{-1}) = 1 - \sqrt{2}\cos\big(\frac{4\pi}{8}\big)z^{-1} + 0.5z^{-2} = 1 + 0.5z^{-2}$$

$$(1 - p_3z^{-1})(1 - p_5z^{-1}) = 1 - \sqrt{2}\cos\big(\frac{6\pi}{8}\big)z^{-1} + 0.5z^{-2} = 1 + z^{-1} + 0.5z^{-2}$$

Finally, we obtain the factorization of $H(z)$:

$$H(z) = \left[\frac{1 - 1.8478z^{-1} + z^{-2}}{1 - 0.5z^{-2}}\right]\cdot\left[\frac{1 - 0.7654z^{-1} + z^{-2}}{1 - z^{-1} + 0.5z^{-2}}\right]\cdot$$
$$\cdot\left[\frac{1 + 0.7654z^{-1} + z^{-2}}{1 + 0.5z^{-2}}\right]\cdot\left[\frac{1 + 1.8478z^{-1} + z^{-2}}{1 + z^{-1} + 0.5z^{-2}}\right]$$

The coefficient matrices A and B will be in this case:

$$A = \begin{bmatrix} 1 & 0 & -0.5 \\ 1 & -1 & 0.5 \\ 1 & 0 & 0.5 \\ 1 & 1 & 0.5 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & -1.8478 & 1 \\ 1 & -0.7654 & 1 \\ 1 & 0.7654 & 1 \\ 1 & 1.8478 & 1 \end{bmatrix}$$

This filter acts as a notch/comb filter, where the zero dips are shifted by $\pi/8$ compared to the pole peaks. The pole zero pattern and magnitude response $|H(\omega)|$ are shown in Fig. 7.6.1.

This example was only meant to illustrate the factorization procedure. Its canonical form realization is much more efficient than the cascade one, since it involves only one multiplier and an 8-fold delay. The canonical realization and the corresponding sample processing algorithm are shown in Fig. 7.6.2. Here, $\mathbf{w} = [w_0, w_1, \dots, w_8]$ is the 9-dimensional internal state vector. □

Example 7.6.4: Sharpen the poles and zeros of the previous filter and determine the cascade form of the resulting filter.

Solution: To sharpen the zeros of the filter, we must place poles "behind" the zeros, that is, replace the numerator polynomial $N(z) = 1 + z^{-8}$ by

$$H_1(z) = \frac{N(z)}{N(\rho^{-1}z)} = \frac{1 + z^{-8}}{1 + \rho^8z^{-8}}$$

For example, we may choose $\rho^8 = 0.94$, or $\rho = 0.9923$. The factorization of $N(\rho^{-1}z)$ into SOSs is obtained from that of $N(z)$ by replacing z by z/ρ or z^{-1} by ρz^{-1} in each factor. This gives:

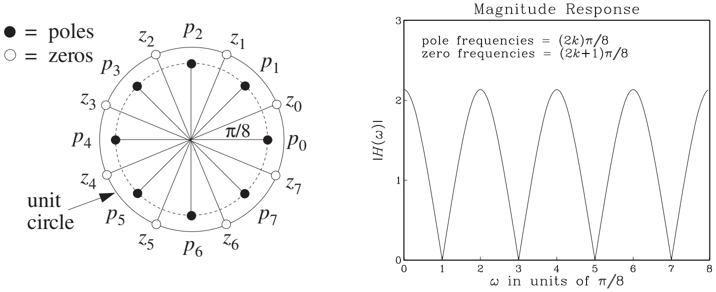
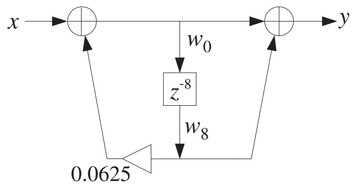


Fig. 7.6.1 Pole/zero pattern and magnitude response of Example 7.6.3.



```
for each input x do:
    $w_0 = x + 0.0625w_8$
    $y = w_0 + w_8$
    delay(8, w)
```

Fig. 7.6.2 Canonical realization of Example 7.6.3.

$$H_1(z) = \frac{N(z)}{N(\rho^{-1}z)} = \left[\frac{1 - 1.8478z^{-1} + z^{-2}}{1 - 1.8478\rho z^{-1} + \rho^2z^{-2}}\right]\cdot\left[\frac{1 - 0.7654z^{-1} + z^{-2}}{1 - 0.7654\rho z^{-1} + \rho^2z^{-2}}\right]\cdot$$
$$\cdot\left[\frac{1 + 0.7654z^{-1} + z^{-2}}{1 + 0.7654\rho z^{-1} + \rho^2z^{-2}}\right]\cdot\left[\frac{1 + 1.8478z^{-1} + z^{-2}}{1 + 1.8478\rho z^{-1} + \rho^2z^{-2}}\right]$$

To sharpen the poles, we must do two things: first push the existing poles closer to the unit circle, and second, place zeros "behind" the poles. This can be done by the substitution of the denominator polynomial by

$$\frac{1}{1 - 0.0625z^{-8}} \quad\longrightarrow\quad H_2(z) = \frac{1 - r^8z^{-8}}{1 - R^8z^{-8}}$$

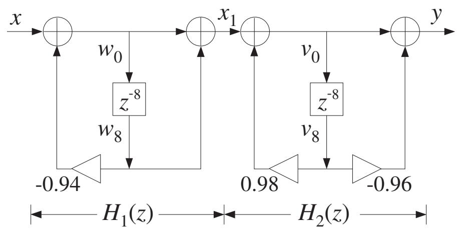
where $r \lesssim R$. For example, we may choose $r^8 = 0.96$ or $r = 0.9949$, and $R^8 = 0.98$ or $R = 0.9975$. The SOS factors of the numerator and denominator can be found in the same fashion as for the polynomial $(1 - 0.0625z^{-8})$. The factorization of $H_2(z)$ is then:

$$H_2(z) = \frac{1 - r^8z^{-8}}{1 - R^8z^{-8}} = \left[\frac{1 - r^2z^{-2}}{1 - R^2z^{-2}}\right]\cdot\left[\frac{1 - \sqrt{2}rz^{-1} + r^2z^{-2}}{1 - \sqrt{2}Rz^{-1} + R^2z^{-2}}\right]\cdot$$
$$\cdot\left[\frac{1 + r^2z^{-2}}{1 + R^2z^{-2}}\right]\cdot\left[\frac{1 + \sqrt{2}rz^{-1} + r^2z^{-2}}{1 + \sqrt{2}Rz^{-1} + R^2z^{-2}}\right]$$

Thus, the new transfer function will be

$$H(z)=H_1(z)H_2(z)=\left[\frac{1+z^{-8}}{1+0.94z^{-8}}\right]\cdot\left[\frac{1-0.96z^{-8}}{1-0.98z^{-8}}\right]$$

Again, the simplest realization is to realize $H_1(z)$ and $H_2(z)$ in cascade, with each realized in its canonical form. This realization and its sample processing algorithm are shown below; the magnitude response $|H(\omega)|$ is shown in Fig. 7.6.3. □



```
for each input x do:
    $w_0 = x - 0.94w_8$
    $x_1 = w_0 + w_8$
    $\text{delay}(8, \mathbf{w})$
    $v_0 = x_1 + 0.98v_8$
    $y = v_0 - 0.96v_8$
    $\text{delay}(8, \mathbf{v})$
```

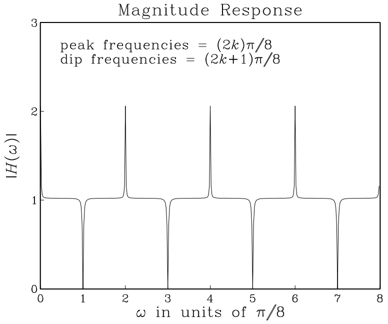


Fig. 7.6.3 Pole/zero sharpening in Example 7.6.4.

The reverse process of going from the cascade realization, Eq. (7.5.1), to the canonical one, Eq. (7.1.4), is much easier. It amounts to multiplying out the second-order numerator and denominator factors to get the full degree polynomials:

$$N(z)=\prod_{i=0}^{K-1}(b_{i0}+b_{i1}z^{-1}+b_{i2}z^{-2})=b_0+b_1z^{-1}+b_2z^{-2}+\cdots+b_Lz^{-L}$$

$$D(z)=\prod_{i=0}^{K-1}(1+a_{i1}z^{-1}+a_{i2}z^{-2})=1+a_1z^{-1}+a_2z^{-2}+\cdots+a_Mz^{-M}$$

where L and M will be at most $2K$, depending on how many sections are full second-order or first-order sections.

The polynomial multiplications may be done in the time domain using convolution. For example, using the definitions (7.1.9) and (7.5.4) for the coefficient vectors, we may write these convolutions in the form:

$$\begin{aligned}\mathbf{b}&=\mathbf{b}_0*\mathbf{b}_1*\cdots*\mathbf{b}_{K-1}\\\mathbf{a}&=\mathbf{a}_0*\mathbf{a}_1*\cdots*\mathbf{a}_{K-1}\end{aligned}\tag{7.6.1}$$

These convolutions may be implemented recursively by defining:

$$\mathbf{d}_i=\mathbf{a}_0*\mathbf{a}_1*\cdots*\mathbf{a}_{i-1}$$

and noting that $\mathbf{d}_i$ satisfies the recursion:

$$\mathbf{d}_{i+1}=\mathbf{d}_i*\mathbf{a}_i\tag{7.6.2}$$

where it must be initialized to $\mathbf{d}_0=\delta$, that is, a delta function, $d_0(n)=\delta(n)$. We may take $\mathbf{d}_0$ to be the one-dimensional vector: $\mathbf{d}_0=\delta=[1]$. A few of the recursion steps will be:

$$\begin{aligned}\mathbf{d}_1&=\mathbf{d}_0*\mathbf{a}_0=\delta*\mathbf{a}_0=\mathbf{a}_0\\\mathbf{d}_2&=\mathbf{d}_1*\mathbf{a}_1=\mathbf{a}_0*\mathbf{a}_1\\\mathbf{d}_3&=\mathbf{d}_2*\mathbf{a}_2=(\mathbf{a}_0*\mathbf{a}_1)*\mathbf{a}_2\\\mathbf{d}_4&=\mathbf{d}_3*\mathbf{a}_3=(\mathbf{a}_0*\mathbf{a}_1*\mathbf{a}_2)*\mathbf{a}_3,\quad\text{etc.}\end{aligned}$$

The recursion ends at $i=K-1$ with the desired answer $\mathbf{a}=\mathbf{d}_K$. Note that the intermediate vector $\mathbf{d}_i$ has order $2i$ and length $2i+1$; similarly, the resulting vector $\mathbf{d}_{i+1}$ has length $2(i+1)+1=2i+3$.

During the recursion (7.6.2), there is no need to save the intermediate vectors $\mathbf{d}_i$. Therefore, the recursion can be stated in the following algorithmic form:

```
$\mathbf{d}=\delta$
for $i=0,1,\dots,K-1$ do:
    $\mathbf{d}=\mathbf{a}_i*\mathbf{d}$
$\mathbf{a}=\mathbf{d}$
```

A variation, which helps the implementation in terms of the routine conv of Chapter 4, is to keep updating **a** during each step:

```
$\mathbf{a}=\delta$
for $i=0,1,\dots,K-1$ do:
    $\mathbf{d}=\mathbf{a}_i*\mathbf{a}$
    $\mathbf{a}=\mathbf{d}$
```
(7.6.3)

and similarly for the numerator vector **b**. The following C routine cas2can.c is an implementation of this algorithm:

```
/* cas2can.c - cascade to canonical */

#include <stdlib.h>                                          declares calloc

void conv();
void cas2can(K, A, a)                                   a is (2K + 1)-dimensional
double **A, *a;                                         A is Kx3 matrix
int K;                                                  K = no. of sections
{
       int i,j;
       double *d;

       d = (double *) calloc(2*K+1, sizeof(double));

       a[0] = 1;                                        initialize

       for(i=0; i<K; i++) {
              conv(2, A[i], 2*i+1, a, d);               d = a[i] * a
              for(j=0; j<2*i+3; j++)                    a = d
                     a[j] = d[j];
              }
}
```

Its inputs are the number of sections K and the coefficient matrix A, whose rows hold the coefficients of the successive sections, as in the routine `cas`. Its output is the $(2K)$-dimensional vector $\mathbf{a}$. It must be called separately on the numerator and denominator coefficient matrices.

Example 7.6.5: To illustrate the usage of `cas2can`, we apply it to the cascade realization of Example 7.6.1:

$$H(z)=\left[\frac{1-0.9z^{-1}}{1+0.8z^{-1}}\right]\cdot\left[\frac{1+z^{-1}+0.74z^{-2}}{1+1.4z^{-1}+0.65z^{-2}}\right]\cdot\left[1-1.6z^{-1}+0.8z^{-2}\right]$$

The routine `cas2can` must be called twice with inputs the coefficient matrices A and B:

$$A=\begin{bmatrix}1 & 0.8 & 0\\ 1 & 1.4 & 0.65\\ 1 & 0 & 0\end{bmatrix},\quad B=\begin{bmatrix}1 & -0.9 & 0\\ 1 & 1 & 0.74\\ 1 & -1.6 & 0.8\end{bmatrix}$$

The quantities `A`, `B`, `a`, `b` must be dimensioned in the main program as in Section 7.5 or Example 7.5.2. Then, the two calls:

```
cas2can(K, A, a);                         denominator coefficients
cas2can(K, B, b);                         numerator coefficients
```

will return the vectors:

$$\mathbf{a}=[1,\ 2.2,\ 1.77,\ 0.52,\ 0,\ 0]$$

$$\mathbf{b}=[1,\ -1.5,\ 0.48,\ -0.33,\ 0.9376,\ -0.5328]$$

which define the canonical realization of Example 7.6.1. □

7.7 Hardware Realizations and Circular Buffers

Hardware realizations of FIR filters with DSP chips were discussed in Section 4.3.4. IIR filters can be realized in a similar fashion.

Consider, for example, a second-order section (7.1.1) realized in its canonical form shown in Fig. 7.2.3. A hardware realization by a typical DSP chip is shown in Fig. 7.7.1. The filter coefficients $\{b_0, b_1, b_2, a_1, a_2\}$ are stored in RAM or ROM on board the chip; the internal states $\{w_0, w_1, w_2\}$ are stored in RAM.

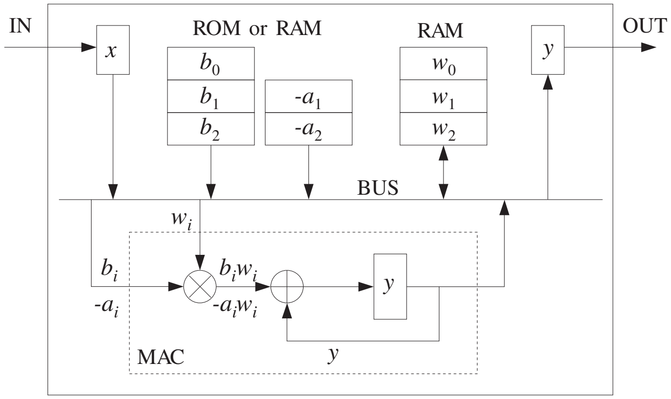


Fig. 7.7.1 Typical DSP chip realization of a second-order section.

As in Section 4.3.4, the sample processing algorithm (7.2.2) can be rewritten in a form that imitates individual instructions of the DSP chip, such as MAC and data shifting instructions:

```
for each input sample x do:
    $w_0 := x - a_1 w_1$
    $w_0 := w_0 - a_2 w_2$
    $y := b_2 w_2$
    $w_2 := w_1,\quad y := y + b_1 w_1$
    $w_1 := w_0,\quad y := y + b_0 w_0$
```
(7.7.1)

In a modern DSP chip, each line in the above algorithm can be implemented with a single MAC-type instruction; therefore, a single SOS can be implemented with *five* instructions per input sample.

Note that the states w_1 and w_2 cannot be updated until after w_0 has been computed. The MAC instructions for computing w_0 proceed forward, that is, from the lowest a_i coefficient to the highest. This is convenient because once w_0 is computed, the combined data shift/MAC instructions for computing y can be started, but proceeding backwards from the highest b_i coefficient to the lowest. In the general case, we can rewrite Eq. (7.2.5), where for simplicity we assumed $L = M$:

```
for each input sample x do:
    $w_0 := x$
    for $i = 1, 2, \dots, M$ do:
        $w_0 := w_0 - a_i w_i$
    $y := b_M w_M$
    for $i = M-1, \dots, 1, 0$ do:
        $w_{i+1} := w_i$
        $y := y + b_i w_i$
```

(7.7.2)

The following C routine `can3.c` is an implementation.

```
/* can3.c - IIR filtering in canonical form, emulating a DSP chip */

double can3(M, a, b, w, x)                    usage: y = can3(M, a, b, w, x);
double *a, *b, *w, x;                         w = internal state vector
int M;                                        a, b have order M
{
       int i;
       double y;

       w[0] = x;                                     read input sample

       for (i=1; i<=M; i++)                          forward order
              w[0] -= a[i] * w[i];                   MAC instruction

       y = b[M] * w[M];

       for (i=M-1; i>=0; i--) {                      backward order
              w[i+1] = w[i];                         data shift instruction
              y += b[i] * w[i];                      MAC instruction
              }

       return y;                                     output sample
}
```

Assuming that each MAC operation in Eq. (7.7.2) can be done with one instruction, and in particular that the combined data move/MAC instructions in the second for-loop can be also done with a single instruction, we count the *total number of instructions* for the filtering of each input sample by an Mth order IIR filter to be:

$$N_{\text{instr}} = 2(M+1) + C \qquad \text{(order-}M\text{ IIR filter)} \tag{7.7.3}$$

where we have added a constant C to account for any additional overhead (such as loop overhead) in instructions. Its value is typically of the order of 10 or less, depending on the particular DSP chip. In Section 4.3.4, we had arrived at a similar result for an FIR filter, which we rewrite now in the form:

$$N_{\text{instr}} = (M+1) + C \qquad \text{(order-}M\text{ FIR filter)} \tag{7.7.4}$$

The total time for processing each input sample will be then

$$\boxed{T_{\text{proc}} = N_{\text{instr}} T_{\text{instr}}} \tag{7.7.5}$$

where T_{instr} is the time for a basic instruction, such as MAC or MACD. Recall from Section 4.3.4 that T_{instr} is of the order of 30–80 nanoseconds, which corresponds to an instruction rate of $f_{\text{instr}} = 1/T_{\text{instr}} = 12.5$–$33.3$ MIPS (million instructions per second). The processing time per sample imposes an *upper limit* on the sampling rate f_s at which the filter may be operated, that is,

$$\boxed{f_s = \frac{1}{T_{\text{proc}}} = \frac{1}{N_{\text{instr}} T_{\text{instr}}} = \frac{f_{\text{instr}}}{N_{\text{instr}}}} \tag{7.7.6}$$

where the quantity $1/T_{\text{proc}}$ is the chip's *computational rate*, that is, the number of samples that can be *processed* per second.

It is impossible to give a processor-independent count of the number of instructions for a particular filter. The precise count, as well as the total processing time T_{proc} per sample, depend on the DSP chip's architecture, instruction set, how memory accessing is used, processor wait states introduced for slow memory, and the way a filter realization is programmed in the chip's assembly language, for example, using in-line code or not.

The above results must be used only as rough guidelines in evaluating the performance of a DSP chip. Our discussion was based on counting the number of MACs in the sample processing algorithm for the particular filter.

The transposed realizations for both IIR and FIR filters can be implemented also by the same number of instructions given by Eqs. (7.7.3) and (7.7.4). The transposed sample processing algorithm uses only plain MAC instructions—not requiring combined data shift/MAC instructions. Therefore, in the early generations of DSP chips, it had a computational advantage in the number of instructions over the canonical realizations.

For a cascade of second-order sections, to find the total processing time we must calculate the time it takes to process a single SOS and then multiply it by the number of sections. We saw in Eq. (7.7.1) that it takes about five instructions per SOS; therefore, the processing time for a single SOS will be approximately (ignoring any other overhead):

$$T_{\text{SOS}} \simeq 5T_{\text{instr}} \tag{7.7.7}$$

For K second-order sections that are either cascaded or arranged in parallel, but which are implemented by the *same* DSP, the total number of instructions will be:

$$N_{\text{instr}} = 5K + C \qquad (K\text{-section IIR filter}) \tag{7.7.8}$$

where C is any additional overhead for the K-section filter. Therefore, the total processing time will be:

$$T_{\text{proc}} = (5K + C)T_{\text{instr}} = KT_{\text{SOS}} + CT_{\text{instr}} \tag{7.7.9}$$

Ignoring the possible small overhead term, we find the maximum sampling rate f_s for implementing K second-order sections:

$$f_s = \frac{1}{T_{\text{proc}}} = \frac{1}{KT_{\text{SOS}}} = \frac{f_{\text{instr}}}{5K} \tag{7.7.10}$$

For *parallel* implementations (see Problem 5.18), we may speed up the throughput rate by using K different DSP chips operating in parallel, each being dedicated to performing a single SOS filtering operation in T_{SOS} seconds. In this case, the total processing time is $T_{\rm SOS}$ because all of the DSPs finish simultaneously, and therefore, the throughput rate is K times *faster* than in the case of a single DSP:

$$T_{\rm proc} = T_{\rm SOS} \quad\Rightarrow\quad f_s = \frac{1}{T_{\rm proc}} = \frac{1}{T_{\rm SOS}} \tag{7.7.11}$$

For a *cascade* implementation, one may also use K DSP chips—one for each SOS—to speed up processing. However, because the output of each section becomes the input of the next section, it is not possible to run all K DSP chips simultaneously. Each DSP must wait $T_{\rm SOS}$ seconds for the DSP before it to finish.

One solution is to *pipeline* the filtering operations of the successive sections, so that all DSPs are working together, but each is processing the input from the previous sampling instant. This can be accomplished by inserting unit delays between the DSPs, as shown in Fig. 7.7.2.

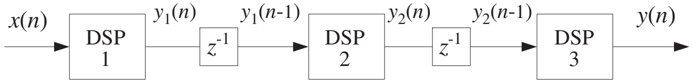


Fig. 7.7.2 Pipelining the operation of multiple DSP processors.

At the nth time instant, while DSP-1 is working on the current input sample $x(n)$, DSP-2 is working on the sample $y_1(n-1)$ which was produced by DSP-1 at the previous time instant and was saved in the delay register until now, and DSP-3 is working on the sample $y_2(n-1)$ which was produced by DSP-2 earlier, and so on. The effect of introducing these delays is only an overall delay in the output. For example, in the case shown in Fig. 7.7.2, the overall transfer function changes from $H(z)= H_1(z)H_2(z)H_3(z)$ to:

$$H(z)= H_1(z)\,z^{-1}\,H_2(z)\,z^{-1}\,H_3(z)= z^{-2}H_1(z)H_2(z)H_3(z)$$

which corresponds to delaying the overall output by two sampling units. For K sections, the overall delay will be $z^{-(K-1)}$.

Example 7.7.1: The AT&T DSP32C floating point DSP chip [103,104] can execute a basic MAC-type instruction in four clock cycles, that is, $T_{\rm instr} = 4T_{\rm clock}$. Therefore, its instruction rate is $f_{\rm instr} = f_{\rm clock}/4$. A typical MAC instruction represents two *floating point operations*: one addition and one multiplication. Therefore, the chip achieves a computational rate of $f_{\rm FLOPS} = 2f_{\rm instr} = f_{\rm clock}/2$ FLOPS.

At a clock rate of $f_{\rm clock} = 50$ MHz, it achieves an instruction rate of $f_{\rm instr} = 50/4 = 12.5$ MIPS, and a computational rate of $f_{\rm FLOPS} = 50/2 = 25$ MFLOPS (megaflops). The time per instruction is $T_{\rm instr} = 1/f_{\rm instr} = 1/12.5 = 80$ nanoseconds.

An order-M FIR filter can be implemented (with in-line code) with

$$N_{\rm instr} = (M+1)+11 = M+12 \qquad \text{(instructions per sample)}$$

Therefore, the processing time per sample will be

$$T_{\rm proc} = (M+12)\,T_{\rm instr}$$

For a 100-tap FIR filter ($M = 99$) with the DSP32C running at 50 MHz, we have $T_{\rm proc} = (99+12)80$ nsec $= 8.9$ μsec, achieving a maximum throughput rate of $f_s = 1/T_{\rm proc} = 112.4$ kHz.

A K-section IIR filter can be implemented (with in-line code) with

$$N_{\rm instr} = 5K+10 \qquad \text{(instructions per sample)}$$

It also requires a number of machine-cycle wait states:

$$N_{\rm wait} = 2K+1 \qquad \text{(wait states per sample)}$$

Therefore, the total processing time for K sections will be:

$$T_{\rm proc} = N_{\rm instr}T_{\rm instr} + N_{\rm wait}T_{\rm clock}$$

Writing $T_{\rm instr} = 4T_{\rm clock} = 4/f_{\rm clock}$, we have

$$T_{\rm proc} = \frac{4N_{\rm instr}+N_{\rm wait}}{f_{\rm clock}} = \frac{4(5K+10)+2K+1}{f_{\rm clock}}$$

For one SOS, $K = 1$, and a 50 MHz clock, we find $T_{\rm proc} = 1.26$ μsec, which translates to maximum sampling rate of $f_s = 1/T_{\rm proc} = 793.6$ kHz. For a 5-section filter $K = 5$, we find $T_{\rm proc} = 3.02$ μsec and $f_s = 331.1$ kHz. And, for a 10-section filter, $K = 10$, we have $T_{\rm proc} = 5.22$ μsec and $f_s = 191.6$ kHz. □

We saw in Section 4.3.4 that *circular addressing* was an efficient way to implement FIR filters and delay lines, and was supported by most of the current DSP chip families. All of the IIR filtering routines—direct, canonical, and cascade—can also be implemented using circular addressing.

The following routine `ccan.c` implements the canonical realization of Fig. 7.2.4 using circular buffers, and replaces `can`. For simplicity, we assume that the numerator and denominator polynomials have the same order M.

```
/* ccan.c - circular buffer implementation of canonical realization */

void wrap();                                    defined in Section 4.3.4

double ccan(M, a, b, w, p, x)                   usage: y = ccan(M, a, b, w, &p, x);
double *a, *b, *w, **p, x;                      p = circular pointer to buffer w
int M;                                          a, b have common order M
{
       int i;
       double y = 0, s0;

       **p = x;                                         read input sample x

       s0  = *(*p)++;                                   s0 = x
       wrap(M, w, p);                                   p now points to s1

       for (a++, i=1; i<=M; i++) {                      start with a incremented to a1
              s0 -= (*a++) * (*(*p)++);
```

```
            wrap(M, w, p);
            }

        **p = s0;                                   p has wrapped around once

        for (i=0; i<=M; i++) {                      numerator part
            y += (*b++) * (*(*p)++);
            wrap(M, w, p);                          upon exit, p has wrapped
            }                                       around once again

        (*p)--;                                     update circular delay line
        wrap(M, w, p);

        return y;                                   output sample
}
```

Like the FIR routine `cfir`, it uses a circular pointer p that always points at the effective starting address of the circular buffer. Here the internal state vector is defined at time n by

$$\mathbf{s}(n)=\begin{bmatrix} s_0(n) \\ s_1(n) \\ \vdots \\ s_M(n) \end{bmatrix}=\begin{bmatrix} w(n) \\ w(n-1) \\ \vdots \\ w(n-M) \end{bmatrix} \tag{7.7.12}$$

Upon entry, the circular pointer p points at the w-register holding $s_0(n)=w(n)$. The value of $s_0(n)$ is not known—only its address. The first for-loop computes the numerical value of $s_0(n)$ and puts it in the correct w-register. This is so because after the loop, p has been post-incremented a total of $M+1$ times and has wrapped completely around.

The second for-loop then computes the contribution of the numerator part. The pointer p is incremented $M+1$ times and cycles around once more. Finally, in preparation for the next time step, p is circularly decremented and points at the w-register that will hold the next value $w(n+1)$.

As we mentioned in Section 4.3.4, in DSP chips that support circular addressing, the incrementing or decrementing pointer wraps around automatically and there is no need to use the routine `wrap`.

The following program segment illustrates the proper initialization and usage of the routine. Note that p must be passed by address because it is changed after each call:

```
double *a, *b, *w, *p;
a = (double *) calloc(M+1, sizeof(double));
b = (double *) calloc(M+1, sizeof(double));
w = (double *) calloc(M+1, sizeof(double));     initializes w to zero
a[0] = 1;                                       not used in the routine

p = w;                                          initialize p

for (n = 0; n < Ntot; n++)
        y[n] = ccan(M, a, b, w, &p, x[n]);      p is passed by address
```

The operations carried out by the routine `ccan` can be restated in a slightly different form by the following sample processing algorithm:

```
for each input sample x do:
    for i = 1, 2, ..., M determine states:
        $s_i = \text{tap}(M, \mathbf{w}, p, i)$
    $s_0 = x - a_1 s_1 - \cdots - a_M s_M$
    $y = b_0 s_0 + b_1 s_1 + \cdots + b_M s_M$
    $*p = s_0$
    $\text{cdelay}(M, \mathbf{w}, \&p)$
```

where for convenience, we used the routine `tap` to compute the current states.

Example 7.7.2: Write the circular-buffer version of the sample processing algorithm of Example 7.2.1 or 7.1.1, whose canonical realization is depicted in the block diagram of Fig. 7.2.5.

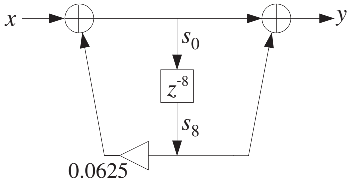
Solution: Here, the buffer w is a five-dimensional array, initialized to zero. The circular pointer p is initialized by $p=w$. We have:

```
for each input sample x do:
    $s_1 = \text{tap}(4, \mathbf{w}, p, 1)$
    $s_2 = \text{tap}(4, \mathbf{w}, p, 2)$
    $s_3 = \text{tap}(4, \mathbf{w}, p, 3)$
    $s_4 = \text{tap}(4, \mathbf{w}, p, 4)$
    $s_0 = x - 0.2s_1 + 0.3s_2 - 0.5s_4$
    $y = 2s_0 - 3s_1 + 4s_3$
    $*p = s_0$
    $\text{cdelay}(4, \mathbf{w}, \&p)$
```

The statement $*p = s_0$ puts the computed value of the 0th component s_0 into the w-register pointed to by p. Then, `cdelay` decrements p circularly. □

Example 7.7.3: Determine the circular-buffer version of the sample processing algorithm of Example 7.6.3, whose realization is depicted in Fig. 7.6.2.

Solution: Here, the buffer $\mathbf{w}$ is nine-dimensional. The algorithm is stated below, where only the output s_8 of the 8-fold delay line is needed:



```
for each input sample x do:
    $s_8 = \text{tap}(8, \mathbf{w}, p, 8)$
    $s_0 = x + 0.0625 s_8$
    $y = s_0 + s_8$
    $*p = s_0$
    $\text{cdelay}(8, \mathbf{w}, \&p)$
```

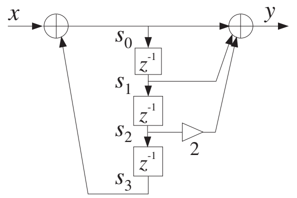
Note that the output s_8 was available before the input s_0 could be computed. □

Example 7.7.4: The input signal $\mathbf{x}=[1,3,2,5,4,6]$ is applied to the filter:

$$H(z)=\frac{1+z^{-1}+2z^{-2}}{1-z^{-3}}$$

Draw the canonical realization and write its circular buffer sample processing algorithm. Iterate the algorithm nine times for $n = 0, 1, \dots, 8$ and compute the corresponding output $y(n)$. Make a table of the circular buffer entries $\mathbf{w}$ and the filter's internal states $\mathbf{s}$.

Solution: The block diagram realization and its circular sample processing algorithm are:



```
for each input sample x do:
    $s_1 = \text{tap}(3, \mathbf{w}, p, 1)$
    $s_2 = \text{tap}(3, \mathbf{w}, p, 2)$
    $s_3 = \text{tap}(3, \mathbf{w}, p, 3)$
    $s_0 = x + s_3$
    $y = s_0 + s_1 + 2s_2$
    $*p = s_0$
    $\text{cdelay}(3, \mathbf{w}, \&p)$
```

The entries of the linear buffer $\mathbf{w} = [w_0, w_1, w_2, w_3]$ over which the circular pointer circulates are shown in the following table.

n	x	w_0	w_1	w_2	w_3	s_0	s_1	s_2	s_3	$y = s_0 + s_1 + 2s_2$
0	1	1↑	0	0	0	1	0	0	0	1
1	3	1	0	0	3↑	3	1	0	0	4
2	2	1	0	2↑	3	2	3	1	0	7
3	5	1	6↑	2	3	6	2	3	1	14
4	4	7↑	6	2	3	7	6	2	3	17
5	6	7	6	2	8↑	8	7	6	2	27
6	0	7	6	6↑	8	6	8	7	6	28
7	0	7	7↑	6	8	7	6	8	7	29
8	0	8↑	7	6	8	8	7	6	8	27

In each row, only one w_i changes, that is, the one pointed to by p. These entries, indicated by an up-arrow, can be filled only after the value of s_0 has been calculated from $s_0 = x + s_3$. The internal states s_i are pointed to by $p + i$, $i = 0, 1, 2, 3$ and wrap around if necessary. For example, at time $n = 3$, p is pointing to w_1; therefore, $p + 1$, $p + 2$, point to w_2, w_3, but $p + 3$ wraps around and points to w_0, so that $s_3 = w_0 = 1$.

According to Eq. (7.7.12), the states s_i are the delayed replicas of the signal $w(n)$ running through the intermediate delays. In the z-domain, this signal is

$$W(z) = \frac{1}{D(z)} X(z) = \frac{1 + 3z^{-1} + 2z^{-2} + 5z^{-3} + 4z^{-4} + 6z^{-5}}{1 - z^{-3}}$$

Its inverse z-transform is the period-3 replication of the numerator:

$$\begin{array}{cccccccccccl}
1 & 3 & 2 & 5 & 4 & 6 & & & & & & = x(n) \\
 & & & 1 & 3 & 2 & 5 & 4 & 6 & & & = x(n-3) \\
 & & & & & & 1 & 3 & 2 & 5 & \cdots & = x(n-6) \\
 & & & & & & & & & 1 & \cdots & \\
\hline
1 & 3 & 2 & 6 & 7 & 8 & 6 & 7 & 8 & 6 & \cdots & = w(n)
\end{array}$$

Thus, the s_0 column holds $w(n)$, s_1 holds $w(n-1)$, and so on. Similarly, the output signal $y(n)$ can be constructed from $w(n)$ by

$$Y(z) = N(z)W(z) = (1 + z^{-1} + 2z^{-2})W(z) \quad \Rightarrow \quad y(n) = w(n) + w(n-1) + 2w(n-2)$$

Adding up the delayed/scaled replicas of $w(n)$, we have:

$$\begin{array}{cccccccccccl}
1 & 3 & 2 & 6 & 7 & 8 & 6 & 7 & 8 & \cdots & = w(n) \\
 & 1 & 3 & 2 & 6 & 7 & 8 & 6 & 7 & \cdots & = w(n-1) \\
 & & 2 & 6 & 4 & 12 & 14 & 16 & 12 & \cdots & = 2w(n-2) \\
\hline
1 & 4 & 7 & 14 & 17 & 27 & 28 & 29 & 27 & \cdots & = y(n)
\end{array}$$

which agrees with the values computed in the above table. □

A second-order section can be implemented by `ccan` by setting $M = 2$. Alternatively, we can use the following specialized version `csos.c`, which replaces `sos`:

```
/* csos.c - circular buffer implementation of a single SOS */

void wrap();

double csos(a, b, w, p, x)                         a, b, w are 3-dimensional
double *a, *b, *w, **p, x;                         p is circular pointer to w
{
       double y, s0;

       *(*p) = x;                                  read input sample x

       s0  = *(*p)++;           wrap(2, w, p);
       s0 -= a[1] * (*(*p)++);  wrap(2, w, p);
       s0 -= a[2] * (*(*p)++);  wrap(2, w, p);

       *(*p) = s0;                                 p has wrapped around once

       y  = b[0] * (*(*p)++);  wrap(2, w, p);
       y += b[1] * (*(*p)++);  wrap(2, w, p);
       y += b[2] * (*(*p));                        p now points to s2

       return y;
}
```

As required, the pointer p points at the w-register containing $w(n)$ and cycles around modulo-3, because the state vector is three-dimensional:

$$\mathbf{s}(n) = \begin{bmatrix} w(n) \\ w(n-1) \\ w(n-2) \end{bmatrix}$$

After the first three post-increments, p cycles around completely. The last two post-increments leave p pointing at the register containing $s_2(n) = w(n-2)$, which is where it should be pointing at the beginning of the next call.

The sample processing algorithm implemented by `csos` can be restated in the following form:

```
for each input sample x do:
    $s_1 = \text{tap}(2, \mathbf{w}, p, 1)$
    $s_2 = \text{tap}(2, \mathbf{w}, p, 2)$
    $s_0 = x - a_1 s_1 - a_2 s_2$
    $y = b_0 s_0 + b_1 s_1 + b_2 s_2$
    $*p = s_0$
    $\text{cdelay}(2, \mathbf{w}, \&p)$
```

As in Eq. (7.5.5), the cascade of K second-order sections can be realized by K *successive calls* to the routine `csos`:

```
for each input sample x do:
    $y = x$
    for $i = 0, 1, \dots, K-1$ do:
        $y = \text{csos}(\mathbf{a}_i, \mathbf{b}_i, \mathbf{w}_i, \&p_i, y)$
```

where each of the K sections has its own three-dimensional buffer $\mathbf{w}_i$ and corresponding circular pointer p_i. The following routine `ccas.c` is an implementation, replacing `cas`:

```
/* ccas.c - circular buffer implementation of cascade realization */

double csos();                             circular-buffer version of single SOS

double ccas(K, A, B, W, P, x)
int K;
double **A, **B, **W, **P, x;              P = array of circular pointers
{
       int i;
       double y;

       y = x;

       for (i=0; i<K; i++)
              y = csos(A[i], B[i], W[i], P+i, y);      note, P + i = &P[i]

       return y;
}
```

As in the case of `cas`, we save the individual buffers $\mathbf{w}_i$ as the rows of the matrix W. Similarly, we save the individual pointers p_i in an *array* of pointers P. The declaration and allocation of A, B, and W are the same as in Section 7.5. The declaration of P and initialization and usage of `ccas` is illustrated below:

```
double **P;

P = (double **) calloc(K, sizeof(double *));      array of K pointers
for (i=0; i<K; i++)
       P[i] = W[i];                                P[i] = ith row of W

for (n = 0; n < Ntot; n++)
       y[n] = ccas(K, A, B, W, P, x[n]);
```

Example 7.7.5: Write the circular version of the cascade realization of Example 7.5.1, depicted in Fig. 7.5.2.

Solution: Let p_0 and p_1 denote the circular pointers of the two sections. Initially they point at the first elements of the three-dimensional buffers $\mathbf{w}_0$ and $\mathbf{w}_1$ of the two sections, that is, $p_0 = w_0$ and $p_1 = w_1$. The sample processing algorithm is:

```
for each input sample x do:
    $s_1 = \text{tap}(2, \mathbf{w}_0, p_0, 1)$
    $s_2 = \text{tap}(2, \mathbf{w}_0, p_0, 2)$
    $s_0 = x + 0.4s_1 - 0.5s_2$
    $x_1 = 3s_0 - 4s_1 + 2s_2$
    $*p_0 = s_0$
    $\text{cdelay}(2, \mathbf{w}_0, \&p_0)$
    $s_1 = \text{tap}(2, \mathbf{w}_1, p_1, 1)$
    $s_2 = \text{tap}(2, \mathbf{w}_1, p_1, 2)$
    $s_0 = x_1 - 0.4s_1 - 0.5s_2$
    $y = 3s_0 + 4s_1 + 2s_2$
    $*p_1 = s_0$
    $\text{cdelay}(2, \mathbf{w}_1, \&p_1)$
```

where the output x_1 of the first section becomes the input to the second. □

We can also write versions of the routines that manipulate the offset index q instead of the circular pointer p, in the same fashion as the FIR routine `cfir2` of Section 4.3.4. The following routine `ccan2.c` replaces `ccan`:

```
/* ccan2.c - circular buffer implementation of canonical realization */

void wrap2();                              defined in Section 4.3.4

double ccan2(M, a, b, w, q, x)
double *a, *b, *w, x;                      q = circular pointer offset index
int M, *q;                                 a, b have common order M
{
       int i;
       double y = 0;

       w[*q] = x;                                 read input sample x

       for (i=1; i<=M; i++)
              w[*q] -= a[i] * w[(*q+i)%(M+1)];

       for (i=0; i<=M; i++)
              y += b[i] * w[(*q+i)%(M+1)];

       (*q)--;                                    update circular delay line
       wrap2(M, q);

       return y;                                  output sample
}
```

Its usage is illustrated by the following program segment. Note that q must be passed by address:

```
int q;
double *a, *b, *w;
a = (double *) calloc(M+1, sizeof(double));
b = (double *) calloc(M+1, sizeof(double));
w = (double *) calloc(M+1, sizeof(double));     initializes w to zero
a[0] = 1;                                        not used in the routine

q = 0;                                           initialize q

for (n = 0; n < Ntot; n++)
       y[n] = ccan2(M, a, b, w, &q, x[n]);      p is passed by address
```

Similarly, the following routines `csos2.c` and `ccas2.c` replace `csos` and `ccas`:

```
/* csos2.c - circular buffer implementation of a single SOS */

void wrap2();

double csos2(a, b, w, q, x)
double *a, *b, *w, x;                   a, b, w are 3-dimensional arrays
int *q;                                 q is circular offset relative to w
{
       double y;

       w[*q] = x - a[1] * w[(*q+1)%3] - a[2] * w[(*q+2)%3];

       y = b[0] * w[*q] + b[1] * w[(*q+1)%3] + b[2] * w[(*q+2)%3];

       (*q)--;
       wrap2(2, q);

       return y;
}
```

and

```
/* ccas2.c - circular buffer implementation of cascade realization */

double csos2();                         circular-buffer version of single SOS

double ccas2(K, A, B, W, Q, x)
int K, *Q;                              Q = array of circular pointer offsets
double **A, **B, **W, x;
{
       int i;
       double y;

       y = x;

       for (i=0; i<K; i++)
              y = csos2(A[i], B[i], W[i], Q+i, y);     note, Q + i = &Q[i]

       return y;
}
```

The ith SOS has its own offset index q_i. Therefore, the quantity Q is defined as an array of K integers. The usage and initialization of `ccas2` is illustrated below. The quantities A, B, W are declared and allocated as usual, Q must be declared as:

```
int *Q;

Q = (double *)  calloc(K, sizeof(double));          array of K integers
for (i=0; i<K; i++)
       Q[i] = 0;                                    initialize Q[i]

for (n = 0; n < Ntot; n++)
       y[n] = ccas2(K, A, B, W, Q, x[n]);
```

7.8 Problems

7.1 A system has transfer function:

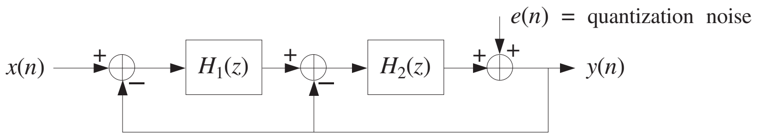
$$H(z)=\frac{z^{-1}+2z^{-2}+3z^{-3}+4z^{-4}}{1-z^{-5}}$$

a. *Without* using partial fractions, determine the *causal* impulse response $h(n)$ of this system, for *all* $n\geq 0$, and sketch it versus n.

b. Draw the *direct* and *canonical* realization forms. Write the *difference equations* describing these realizations. Then, write the corresponding *sample processing* algorithms.

c. Factor this transfer function in the form $H(z)=H_1(z)H_2(z)$, where $H_1(z)$ is the ratio of two first-order polynomials, and $H_2(z)$ has numerator of degree 3 and denominator of degree 4. Draw the corresponding cascade realization, with each factor realized in its *canonical* form. Write the *difference equations* describing this realization, and the corresponding *sample processing* algorithm.

7.2 A discrete-time model for a second-order delta-sigma A/D converter is shown below:



a. Show that the output z-transform $Y(z)$ is related to $X(z)$ and $E(z)$ by a transfer function relationship of the form:

$$Y(z)=H_x(z)X(z)+H_e(z)E(z)$$

Express the transfer functions $H_x(z)$ and $H_e(z)$ *in terms of* the loop filters $H_1(z)$ and $H_2(z)$.

b. Determine $H_1(z)$ and $H_2(z)$ in order for $H_x(z)$ to act as a single delay and $H_e(z)$ as a second-order noise shaper, that is,

$$H_x(z)=z^{-1}\quad\text{and}\quad H_e(z)=(1-z^{-1})^2$$

7.3 A digital filter has transfer function:

$$H(z)=\frac{z^{-1}(1+2z^{-2})(1+3z^{-2})}{1-z^{-6}}$$

a. Draw the *direct* form realization (direct form I). Write the I/O difference equation and corresponding sample processing algorithm for this realization.

b. Draw the *canonical* form realization (direct form II). Write the I/O difference equations and corresponding sample processing algorithm for this realization.

c. Factor $H(z)$ into *second-order sections* with real-valued coefficients, and draw the corresponding *cascade* realization. Write the I/O difference equations and corresponding sample processing algorithm for this realization.

d. *Without* using partial fractions, determine the causal impulse response $h(n)$ of this filter for all n. Explain your reasoning.

7.4 A linear system is described by the system of difference equations:

$$\begin{aligned} v(n) &= x(n)+v(n-1) \\ y(n) &= v(n)+v(n-2)+v(n-4) \end{aligned}$$

Determine the transfer function from $x(n)$ to $y(n)$. Draw the direct, the canonical, and the cascade of SOS realizations (with real coefficients). In each case, state the sample-by-sample processing algorithm.

7.5 Draw the three realizations: (1) direct, (2) canonical, and (3) cascade of second-order sections for the following filter:

$$H(z)=\frac{(2-3z^{-1})(1+z^{-2})}{1-0.25z^{-4}}$$

For each realization write the corresponding: (a) I/O difference equations and (b) sample processing algorithm.

7.6 A filter has transfer function:

$$H(z)=\frac{5}{1+0.25z^{-2}}-\frac{4}{1-0.25z^{-2}}=\frac{1-2.25z^{-2}}{(1+0.25z^{-2})(1-0.25z^{-2})}$$

a. Determine *all possible* impulse responses $h(n)$ and their ROCs.

b. Draw the *direct realization form* of $H(z)$.

c. Draw the *canonical realization form.*

d. Draw the *cascade form.*

In all cases, write all the *I/O difference equations* describing the realization in the time domain, and the *sample processing algorithm* implementing it.

7.7 Draw the *direct* and the *canonical* realizations of the system:

$$H(z)=\frac{1-2z^{-2}+z^{-4}}{1-0.4096z^{-4}}$$

a. Write the I/O difference equations and state the sample processing algorithms of these two realizations. [*Hint:* $0.4096=(0.8)^4$.]

b. Factor the above transfer function into second-order sections (with real coefficients). Draw the cascade realization (with each SOS realized in its canonical form). Write *all* the I/O difference equations and state the sample processing algorithm describing this realization.

7.8 An allpass digital reverberator with delay of 10 time units and having input $x(n)$ and overall output $y(n)$, is described by the system of difference equations:

$$\begin{aligned} w(n) &= 0.75w(n-10)+x(n) \\ y(n) &= -0.75w(n)+w(n-10) \end{aligned}$$

a. Draw a block diagram realization of this filter. The realization must use only one 10-fold delay.

b. Write the sample processing algorithm for this filter. Then, convert this algorithm into a C routine that implements it.

c. Show that the magnitude response of the filter is identically equal to one, that is, $|H(\omega)|=1$ for all ω.

7.9 A filter is described by the following sample processing algorithm relating the input and output samples x and y:

```
for each input sample x do:
    $w_0 = x + 0.64w_4$
    $y = w_0 + w_3$
    $w_4 = w_3$
    $w_3 = w_2$
    $w_2 = w_1$
    $w_1 = w_0$
```

Determine the transfer function $H(z)$ of this filter. Factor $H(z)$ into factors of order up to two (with real-valued coefficients) and draw the corresponding cascade realization. State the sample processing algorithm for that realization.

7.10 For the following three filters,

$$H(z)=(1+z^{-2})^3\,,\qquad H(z)=\frac{1}{1+0.81z^{-2}}\,,\qquad H(z)=\frac{1-z^{-4}}{1-0.9z^{-1}}$$

a. Determine *all possible* impulse responses $h(n)$, corresponding ROCs, stability, and causality properties.

b. Draw the *direct, canonical*, and *cascade of SOS* realization forms. Write the I/O difference equations for each realization. State the sample-by-sample processing algorithm for each realization.

c. Determine the corresponding pole/zero plots and then make a rough sketch of the magnitude responses $|H(\omega)|$ versus ω.

7.11 Consider a system with transfer function:

$$H(z)=\frac{(1-\sqrt{2}z^{-1}+z^{-2})(1+\sqrt{2}z^{-1}+z^{-2})}{(1+0.81z^{-2})(1-0.81z^{-2})}$$

a. Determine the poles and zeros of this filter and place them on the z-plane. Then, draw a rough sketch of the *magnitude response* of the filter versus frequency.

b. Draw the *cascade* of second-order sections realization. Write the I/O difference equations for this realization. State the corresponding sample-by-sample processing algorithm.

Repeat for the *canonical* and *direct* form realizations.

7.12 Consider a stable system with transfer function $H(z)=\dfrac{\dfrac{1}{16}+z^{-4}}{1+\dfrac{1}{16}z^{-4}}$.

a. Determine the poles and zeros of $H(z)$ and place them on the complex z-plane. Pair them in conjugate pairs to write $H(z)$ as a cascade of second-order sections with real coefficients.

b. Draw the direct, canonical, and cascade realization forms. In each case, write the corresponding sample processing algorithm.

c. Determine the impulse response $h(n)$ for all n. And, finally show that $|H(\omega)| = 1$ for all ω, that is, it is an allpass filter.

7.13 *Computer Experiment: IIR Filtering.* A digital filter has transfer function:

$$H(z)= H_0(z)H_1(z)H_2(z)H_3(z)$$

where

$$H_0(z) = \frac{0.313(1+z^{-1})}{1-0.373z^{-1}}, \qquad H_1(z) = \frac{0.147(1+2z^{-1}+z^{-2})}{1-1.122z^{-1}+0.712z^{-2}}$$

$$H_2(z) = \frac{0.117(1+2z^{-1}+z^{-2})}{1-0.891z^{-1}+0.360z^{-2}}, \quad H_3(z) = \frac{0.103(1+2z^{-1}+z^{-2})}{1-0.780z^{-1}+0.190z^{-2}}$$

a. Draw the cascade realization of $H(z)$ and write all the difference equations required for the time operation of this filter. Write the sample-by-sample processing algorithm implementing the cascade realization.

b. Using the routine `cas2can`, determine the canonical and direct realizations of $H(z)$ and draw them. Write the corresponding sample processing algorithms and difference equations for the two realizations.

c. Generate a length-100 input signal defined as

$$x(n)=\begin{cases} 1 & \text{if } 0 \le n < 50 \\ 0 & \text{if } 50 \le n < 100 \end{cases}$$

Using the cascade routine `cas` compute the filter output y_n for $0 \le n \le 99$. Repeat using the routines `dir` and `can`. In three parallel columns, print the signal samples y_n computed by the three routines `cas`, `dir`, `can`.

d. On the same graph, plot the two signals x_n and y_n versus n. You will be observing the input-on transients, steady-state response to a constant input, and input-off transients. What is the theoretical value of the steady-state response to a constant input?

e. Send a unit impulse as input. For the cascade realization, using the routine `cas`, compute the corresponding output impulse response for $0 \le n < 50$, and plot it versus n. Repeat for the canonical realization using the routine `can`. Do you get identical results?

7.14 *Computer Experiment: IIR Filtering in Canonical Form.* Write a stand-alone C program, say `canfilt.c`, that implements the canonical form of the IIR sample processing algorithm, Eq. (7.2.5). The program must have usage:

```
canfilt a.dat b.dat  < x.dat  > y.dat
```

It must read and dynamically allocate the denominator and numerator coefficient vectors **a**, **b** from two input files, say `a.dat` and `b.dat`, and must allocate the internal state vector **w**. Using the routine `can.c`, it must keep processing input samples, reading them one at a time from `stdin` or from a file `x.dat`, and writing the computed output samples to `stdout` or a file `y.dat`. Filtering must stop when the end-of-file of the input file is encountered.

Using this program, calculate the filter outputs required in Problem 7.13.

7.15 *Computer Experiment: IIR Filtering in Cascade Form.* Write a stand-alone C program, say `casfilt.c`, that implements the cascade form of the IIR sample processing algorithm, Eq. (7.5.3). The program must have usage:

```
casfilt A.dat B.dat  < x.dat  > y.dat
```

It must read and dynamically allocate the $K\times3$ denominator and numerator coefficient matrices A and B from two input files, say `A.dat` and `B.dat` (stored in row-wise fashion). and must allocate the internal $K\times3$ state matrix W. Using the routine `cas.c`, it must keep processing input samples, reading them one at a time from `stdin` or from a file `x.dat`, and writing the computed output samples to `stdout` or a file `y.dat`. Filtering must stop when the end-of-file of the input file is encountered.

Alternatively or additionally, write a MATLAB version, say `casfilt.m`, that reads the input vector **x** and the matrices A and B and computes the output vector **y**. It may use the MATLAB functions `sos.m` and `cas.m` of Appendix C. Its usage must be:

```
y = casfilt(B, A, x);
```

Using these programs, calculate the filter outputs required in Problem 7.13.

7.16 *Computer Experiment: Comb Filtering.* Consider the two comb filters discussed in Examples 7.6.3 and 7.6.4. To understand the difference in their time-domain operation, consider as input to both filters the “noisy” signal:

$$x(n)= s(n)+v(n), \qquad \text{where} \quad \begin{aligned} s(n) &= A_0\cos(w_0 n)+A_2\cos(w_1 n) \\ v(n) &= A_2\cos(w_2 n)+A_3\cos(w_3 n) \end{aligned}$$

where $n = 0, 1, \dots, 499$, and $A_0 = 1$, $A_1 = A_2 = A_3 = 0.5$. The frequency components of the “desired” signal $s(n)$ are chosen to lie in the flat part, between the zeros, of the frequency response of the sharpened filter shown in Fig. 7.6.3: $w_0 = 0.50\pi/8$, $w_1 = 0.75\pi/8$. The frequency components of the “noise” part $v(n)$ are chosen to be two of the comb’s zeros: $w_2 = \pi/8$, $w_3 = 3\pi/8$.

Plot the desired and noisy signals $s(n)$ and $x(n)$. Compute the corresponding output signals $y_1(n)$, $y_2(n)$ for $n = 0, 1, \dots, 499$ of the two filters, plot them, and compare them with the desired signal $s(n)$.

To implement the filtering operations for the first filter, use the sample processing algorithm of the canonical form given in Example 7.6.3, and for the second filter use the cascade of the two combs given in Example 7.6.4. Moreover, to make a fair comparison *normalize* the two magnitude responses to unity gain at one of the desired frequencies, say at w_0.

7.17 *Computer Experiment: Comb Impulse Response.* First, derive closed-form analytical expressions for the impulse responses of the two comb filters of Examples 7.6.3 and 7.6.4. [*Hint:* You may do partial fractions in the variable z^{-8}.]

Then, using their sample processing algorithms, compute the impulse responses by sending in a unit impulse input. Compare the computed values and the analytical expressions. For each filter, compute the impulse responses $h(n)$ for $0 \le n < 200$.

7.18 *Computer Experiment: IIR Filtering Using Circular Buffers.* For the canonical and cascade realizations, repeat all the questions of Problem 7.13, using the circular buffer routines `ccan.c`, `ccas.c`, and `csos.c`, instead of the standard linear buffer routines `can`, `cas`, and `sos`.

Repeat this problem using the alternative circular buffer routines `ccan2.c`, `ccas2.c`, and `csos2.c`.

7.19 Consider a filter with transfer function: $H(z)=\dfrac{6-2z^{-3}}{1-0.5z^{-3}}$.

a. Draw the *canonical realization* form of $H(z)$ and write the corresponding sample processing algorithm both in its *linear* and *circular* buffer versions.

b. Determine the causal impulse response $h(n)$ in two ways: (i) by doing and inverse z-transform on $H(z)$, and (ii) by sending in a unit impulse and iterating the circular buffer sample processing algorithm. Perform seven iterations for $n = 0, 1, \dots, 6$ filling the entries of the following table:

n	x	w_0	w_1	w_2	w_3	s_0	s_1	s_2	s_3	y
0	1	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$
1	0	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
6	0	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$

where $[w_0, w_1, w_2, w_3]$ is the linear buffer over which the circular pointer circulates and s_i are the internal states (i.e., the tap outputs) of the triple delay z^{-3}.

c. Suppose the length-3 input signal $\mathbf{x} = [1, 2, 3]$ is applied. Compute the corresponding output for $n = 0, 1, \dots, 6$ by iterating the circular version of the sample processing algorithm and filling the w_i and s_i entries of the above table.

7.20 Consider a second-order denominator polynomial for an IIR filter with conjugate poles:

$$1 + a_1 z^{-1} + a_2 z^{-2} = (1 - pz^{-1})(1 - p^* z^{-1})$$

Taking differentials of both sides of this identity, show that small changes $\{da_1, da_2\}$ in the polynomial coefficients induce the small change in the pole location:

$$dp = -\frac{p\,da_1 + da_2}{p - p^*}$$

7.21 Consider a fourth-order denominator polynomial for an IIR filter with conjugate poles:

$$1 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} + a_4 z^{-4} = (1 - p_0 z^{-1})(1 - p_0^* z^{-1})(1 - p_1 z^{-1})(1 - p_1^* z^{-1})$$

Show that small changes $\{da_1, da_2, da_3, da_4\}$ in the coefficients induce the following changes in the pole locations, which may not be too small if p_0 is close to p_1:

$$dp_0 = -\frac{p_0^3 da_1 + p_0^2 da_2 + p_0 da_3 + da_4}{(p_0 - p_0^*)(p_0 - p_1)(p_0 - p_1^*)}$$

$$dp_1 = -\frac{p_1^3 da_1 + p_1^2 da_2 + p_1 da_3 + da_4}{(p_1 - p_0^*)(p_1 - p_0)(p_1 - p_1^*)}$$

Such pole sensitivity formulas can be generalized to polynomials of arbitrary degree; see [2,3,266].

7.22 Consider the transposed realization of a third-order IIR filter shown in Fig. 7.8.1. First, determine the transfer function $H(z)$. Then, using the indicated variables $v_i(n)$, $i = 0, 1, 2, 3$, write the difference equations describing the time-domain operation of this realization. Then, rewrite them as a sample-by-sample processing algorithm that transforms each input x into an output sample y.

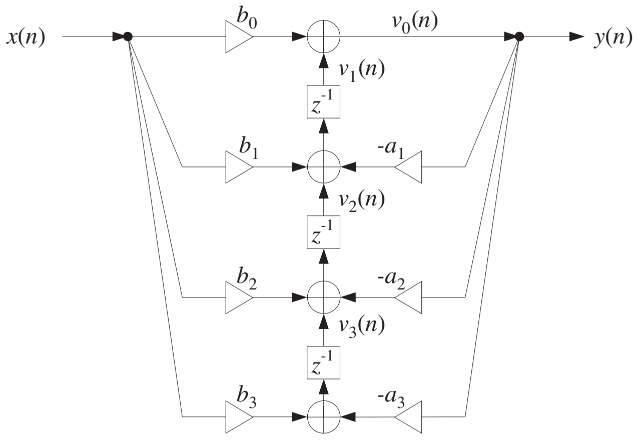


Fig. 7.8.1 Transpose of canonical realization.

7.23 *Computer Experiment: Transposed Realization Form.* Generalize the transposed structure of Fig. 7.8.1 to an arbitrary transfer function with numerator and denominator polynomials of degree M. State the corresponding sample processing algorithm in this case. Then, convert it into a C routine, say `transp.c`, that has usage:

```
y = transp(M, a, b, v, x);
```

where **a**, **b**, and **v** are $(M+1)$-dimensional vectors.

7.24 *Computer Experiment: Filtering in Transposed Form.* Repeat the filtering questions of Problem 7.13. Use the transpose of the canonical form implemented by the routine `transp.c` of Problem 7.23. For the cascade form, realize each second-order section in its transposed form. The output signals must agree exactly with those of Problem 7.13.

7.25 For the filter of Example 7.5.1, draw the transpose realizations of the canonical form. Also draw a cascade realization in which every second-order section is realized in its transposed form. In all cases, write the corresponding I/O difference equations and sample processing algorithms.

7.26 A general feedback system is shown in Fig. 7.8.2, where the output of filter $H_1(z)$ is fed back into filter $H_2(z)$ and then back to the input, and where the delay z^{-1} can be positioned at the four points A, B, C, or D.

For each case, determine the transfer function of the overall closed-loop system, that is, from $x(n)$ to $y(n)$. Assuming that the I/O equations of the filters $H_1(z)$ and $H_2(z)$ are known, state the corresponding sample processing algorithms in the four cases. How does moving the position of the delay change the order of computations? Finally, if the same input $x(n)$ is fed into the four filters, determine the relationships among the four output signals $y_A(n)$, $y_B(n)$, $y_C(n)$, and $y_D(n)$.

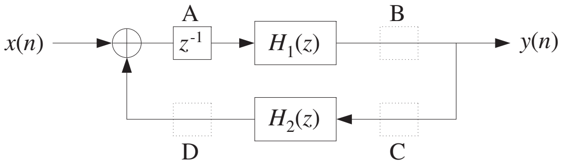


Fig. 7.8.2 General feedback system.

7.27 Consider the block diagram of Fig. 7.8.3, where the feedback delay z^{-1} can be positioned at points A or B. For the two cases, introduce appropriate internal state variables and write the difference equations describing the time-domain operations of the overall feedback system. Then, translate these difference equations into sample processing algorithms. What is the effect of moving the delay from A to B on the order of computations? What happens if that delay is removed altogether?

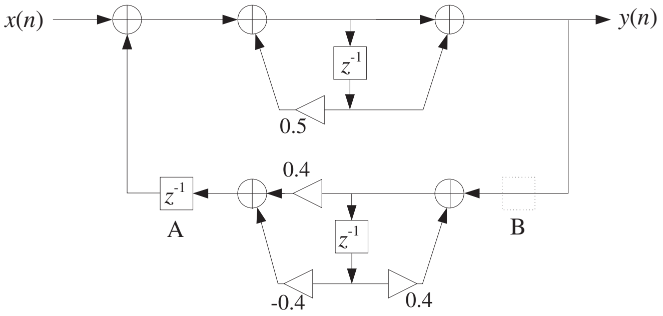


Fig. 7.8.3 Feedback system of Problem 7.27.

7.28 The three block diagrams (a,b,c) shown below are equivalent descriptions of the same lowpass reverberator filter of the form:

$$H(z) = \frac{1}{1 - G(z)z^{-4}} \qquad \text{where} \quad G(z) = \frac{0.2}{1 - 0.8z^{-1}}$$

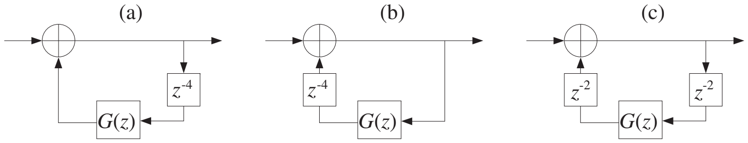
a. Draw a realization of the lowpass feedback filter $G(z)$. Replace the block $G(z)$ by its realization.

b. Write the sample processing algorithms describing the time-domain operation of *each* of the three block diagrams (a), (b), and (c).

c. In block diagram (a), replace the 4-fold delay z^{-4} by its reverberating version:

$$z^{-4} \to \frac{z^{-4}}{1 - 0.8z^{-4}}$$

Draw the block diagram realization of the new overall filter. Your diagram must be obtained from diagram (a) by replacing the 4-fold delay z^{-4} by the block diagram realization of the given reverberating version. Write the sample processing algorithm for the new filter.



8

Lattice Realizations

8.1 Overview

Lattice realizations, originally proposed by Gray and Markel [337–339], can realize an arbitrary rational transfer function $H(z)= B(z)/A(z)$ as a cascade of simpler first-order sections—albeit these sections have two inputs and two outputs each, as shown below in Fig. 8.1.1 for a third-order example,

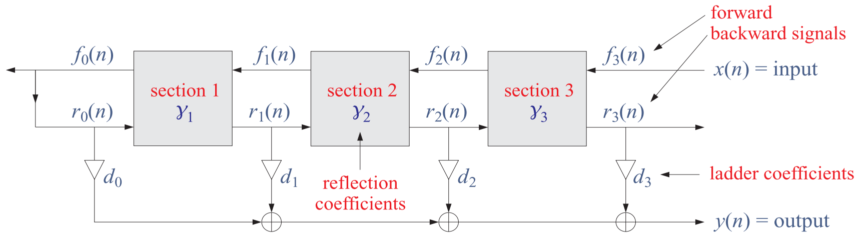


Fig. 8.1.1 Typical lattice/ladder filter structure.

In its simpler, standard, form, each section contains one delay element z^{-1} and one multiplier γ referred to as a “reflection coefficient”, whereas the normalized form contains two multipliers, the reflection coefficient γ and its associated “transmission” coefficient, $\tau = \sqrt{1-\gamma^2}$.[†]

Starting with input signal $x(n)$, the upper inputs $f(n)$ to each section are referred to as “forward” signals and the lower outputs $r(n)$ as “reverse” or “backward” signals. The lower outputs are linearly combined with the “ladder” coefficients d to generate the overall output signal $y(n)$. This structure is often referred as a *lattice/ladder realization* to indicate the fact that it contains both lattice sections and ladder coefficients

The *normalized lattice realization* is one of the best, if not the best, realization with regard to robustness and stability under coefficient quantization, roundoff noise accumulation, and protection against internal overflows [337–339]. These nice properties can be traced to the fact that all internal branches are scaled by multipliers (the γs and τs) that have magnitude less than one.

8.2 Applications of Lattice Structures

Lattice filter structures have a long history and arise in diverse fields of science, engineering, and mathematics.

They arise naturally in the description of *wave propagation* in multilayer structures, such as dielectric structures used in the design of thin-film anti-reflection coatings, dielectric mirrors, and optical interference filters, such as narrow-band transmission filters for wavelength-division multiplexing (WDM), as well as in the design of broadband terminations of multi-segment transmission lines.

The terminology of referring to the γ-coefficients as reflection coefficients arises from this wave propagation context, where the γs do indeed represent the elementary reflection coefficients at the multilayer interfaces. The multilayers themselves are the lattice sections, and the wave propagation equations, referred to as the *layer recursions*, relating the incident and reflected fields (i.e., the forward and backward signals) at each interface are the exactly the same as those used in the digital filter transfer function context.

Other wave propagation applications include the making acoustic tube models for the analysis and synthesis of speech, with the layer recursions having exactly the same structure as in the electromagnetic context. The layer recursions are also used in speech recognition, disguised as the so-called Schur algorithm.

Wave propagation in multilayer structures also find application in geophysical deconvolution and inverse scattering problems for oil exploration, in structures for acoustic noise control, and in vibration control for dampening vibrations of long structures.

Lattice filter structures arise in time-series analysis in the theory of linear prediction and the realization of optimum Wiener filters, including adaptive filtering versions which are perhaps the fastest and most stable of all known adaptive filtering algorithms. The layer recursions are related to the Levinson-Durbin algorithm for linear prediction, and the γ-coefficients are also known in this context as “partial-correlation” coefficients (PARCOR).

The layer recursions—known as the Schur recursions in linear prediction—are intimately connected to the mathematical theory of lossless bounded real functions in the z-plane, and positive real functions in the s-plane, and find application in analysis, synthesis, and stability of linear networks. The so-called *Schur-Cohn stability* test states that a transfer function $H(z)$ is stable and causal (i.e. all its poles are inside the unit circle on the z-plane) if and only if all the γ-coefficients extracted from $H(z)$ by applying the layer recursions have magnitude less than one, $|\gamma| < 1$.

Most of the above applications are discussed extensively in the author’s books on *Electromagnetic Waves and Antennas* [46] and *Applied Optimum Signal Processing* [45], both of which are available online.

[†] stability and causality of $H(z)$ requires that all γs have magnitude less than one, $|\gamma| < 1$.

8.3 Standard, Rearranged, and Normalized Lattice

Standard Lattice Realization

An example of a standard lattice realization is depicted in Figs. 8.3.1 for an order-3 transfer function,

$$H(z)=\frac{B(z)}{A(z)}=\frac{b_0+b_1z^{-1}+b_2z^{-2}+b_3z^{-3}}{1+a_1z^{-1}+a_2z^{-2}+a_3z^{-3}} \tag{8.3.1}$$

We discuss below how to map the direct-form numerator and denominator coefficients, b_0, b_1, b_2, b_3, and, a_1, a_2, a_3, to the reflection coefficients, $\gamma_1, \gamma_2, \gamma_2$, and the ladder coefficients, d_0, d_1, d_2, d_3.

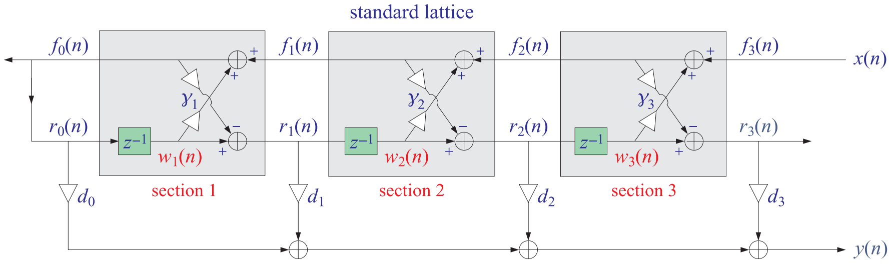
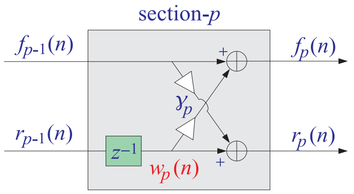


Fig. 8.3.1 Standard lattice realization.

To derive the lattice realization for the filter, $H(z)=B(z)/A(z)$, it proves convenient to consider first the "analysis" lattice part that only realizes the denominator polynomial, $A(z)$. It is obtained by reversing the directions of the $f(n)$ signals, as depicted in the upper part of Fig. 8.3.2, so that the overall input to this structure is $f_0(n)$ and both the $f(n)$ and $r(n)$ signals run to the right, feeding forward with no feedback connections.

For the pth section, we have the following input/output equations relating the two inputs, $f_{p-1}(n), r_{p-1}(n)$, to the two outputs, $f_p(n), r_p(n)$, where for a general order M filter there will be M sections, starting with a common overall input, $f_0(n)=r_0(n)$,



$$\boxed{\begin{aligned} f_p(n) &= f_{p-1}(n)-\gamma_p r_{p-1}(n-1)\\ r_p(n) &= r_{p-1}(n-1)-\gamma_p f_{p-1}(n)\end{aligned}} \tag{8.3.2}$$

for, $p=1,2,\dots,M$. These are known as "forward" layer recursions. For the purpose of later developing a sample processing algorithm, we may also express these equations in terms of the content of the delay z^{-1}, denoted by $w_p(n)$ for the pth section,

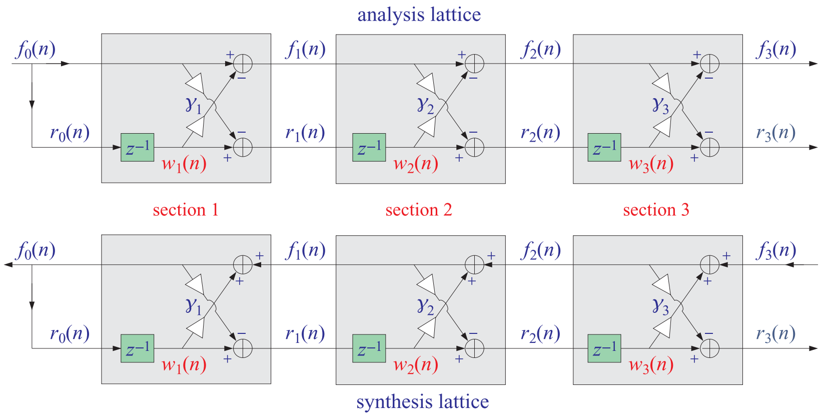


Fig. 8.3.2 Analysis and synthesis lattice structures.

$$\boxed{\begin{aligned} w_p(n) &= r_{p-1}(n-1)\\ f_p(n) &= f_{p-1}(n)-\gamma_p w_p(n)\\ r_p(n) &= w_p(n)-\gamma_p f_{p-1}(n)\\ w_p(n+1) &= r_{p-1}(n)\end{aligned}}\qquad p=1,2,\dots,M \tag{8.3.3}$$

In the z-domain, Eqs. (8.3.2) are expressed in terms of the z-transforms, $F_p(z), R_p(z)$, of the signals, $f_p(n), r_p(n)$,

$$\boxed{\begin{aligned} F_p(z) &= F_{p-1}(z)-\gamma_p z^{-1}R_{p-1}(z)\\ R_p(z) &= z^{-1}R_{p-1}(z)-\gamma_p F_{p-1}(z)\end{aligned}}\qquad p=1,2,\dots,M \tag{8.3.4}$$

which can be written in the 2×2 matrix form,

$$\begin{bmatrix}F_p(z)\\R_p(z)\end{bmatrix}=\begin{bmatrix}1 & -\gamma_p z^{-1}\\ -\gamma_p & z^{-1}\end{bmatrix}\begin{bmatrix}F_{p-1}(z)\\R_{p-1}(z)\end{bmatrix},\quad p=1,2,\dots,M \tag{8.3.5}$$

The iteration of the forward recursions allows one to express the overall output signals in terms of the overall input signals, by multiplying out the 2×2 transition matrices, for example, we have for $M=3$,

$$\begin{bmatrix}F_3(z)\\R_3(z)\end{bmatrix}=\begin{bmatrix}1 & -\gamma_3 z^{-1}\\ -\gamma_3 & z^{-1}\end{bmatrix}\begin{bmatrix}1 & -\gamma_2 z^{-1}\\ -\gamma_2 & z^{-1}\end{bmatrix}\begin{bmatrix}1 & -\gamma_1 z^{-1}\\ -\gamma_1 & z^{-1}\end{bmatrix}\begin{bmatrix}F_0(z)\\R_0(z)\end{bmatrix}$$

or, since we assumed a common input, $F_0(z)=R_0(z)$,

$$\begin{bmatrix}F_3(z)\\R_3(z)\end{bmatrix}=\begin{bmatrix}1 & -\gamma_3 z^{-1}\\ -\gamma_3 & z^{-1}\end{bmatrix}\begin{bmatrix}1 & -\gamma_2 z^{-1}\\ -\gamma_2 & z^{-1}\end{bmatrix}\begin{bmatrix}1 & -\gamma_1 z^{-1}\\ -\gamma_1 & z^{-1}\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}F_0(z)$$

We may define the overall transfer functions from the common input $F_0(z)=R_0(z)$ to the two outputs, $F_3(z), R_3(z)$, by the relationship,

$$A_3(z)=\frac{F_3(z)}{F_0(z)}\,,\quad A_3^R(z)=\frac{R_3(z)}{R_0(z)}=\frac{R_3(z)}{F_0(z)}$$

where, $A_3(z), A_3^R(z)$, will be the two order-3 polynomials resulting from the matrix operation,

$$\begin{bmatrix} A_3(z) \\ A_3^R(z) \end{bmatrix} = \begin{bmatrix} 1 & -\gamma_3 z^{-1} \\ -\gamma_3 & z^{-1} \end{bmatrix} \begin{bmatrix} 1 & -\gamma_2 z^{-1} \\ -\gamma_2 & z^{-1} \end{bmatrix} \begin{bmatrix} 1 & -\gamma_1 z^{-1} \\ -\gamma_1 & z^{-1} \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{8.3.6}$$

It can be verified that $A_3^R(z)$ will be the reverse of $A_3(z)$. We note that the reverse of a polynomial is obtained by reversing its coefficients, for example,

$$\begin{aligned} A(z) &= a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}\,, &\quad \mathbf{a} &= [a_0,\, a_1,\, a_2,\, a_3] \\ A^R(z) &= a_3 + a_2 z^{-1} + a_1 z^{-2} + a_0 z^{-3}, &\quad \mathbf{a}^R &= [a_3,\, a_2,\, a_1,\, a_0] \end{aligned}$$

Denoting resulting polynomial by, $A_3(z)= 1 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}$, we would like to associate it with the denominator polynomial $A(z)$ of our transfer function Eq. (8.3.1). Although Eq. (8.3.6) establishes a relationship between the polynomial coefficients, $\{a_1, a_2, a_3\}$, and the reflection coefficients $\{\gamma_1, \gamma_2, \gamma_3\}$, it is much more convenient to express this relationship recursively with the help of the lower-order polynomials of order 2 and order 1, defined by,

$$\begin{aligned} \begin{bmatrix} A_2(z) \\ A_2^R(z) \end{bmatrix} &= \begin{bmatrix} 1 & -\gamma_2 z^{-1} \\ -\gamma_2 & z^{-1} \end{bmatrix} \begin{bmatrix} 1 & -\gamma_1 z^{-1} \\ -\gamma_1 & z^{-1} \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\ \begin{bmatrix} A_1(z) \\ A_1^R(z) \end{bmatrix} &= \begin{bmatrix} 1 & -\gamma_1 z^{-1} \\ -\gamma_1 & z^{-1} \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \end{aligned} \tag{8.3.7}$$

With these definitions, we obtain the following recursions that start with the order-0 polynomials, $A_0(z)=A_0^R(z)=1$, and proceed to order-3,

$$\begin{aligned} \begin{bmatrix} A_0(z) \\ A_1^R(z) \end{bmatrix} &= \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\ \begin{bmatrix} A_1(z) \\ A_1^R(z) \end{bmatrix} &= \begin{bmatrix} 1 & -\gamma_1 z^{-1} \\ -\gamma_1 & z^{-1} \end{bmatrix} \begin{bmatrix} A_0(z) \\ A_0^R(z) \end{bmatrix} \\ \begin{bmatrix} A_2(z) \\ A_2^R(z) \end{bmatrix} &= \begin{bmatrix} 1 & -\gamma_2 z^{-1} \\ -\gamma_2 & z^{-1} \end{bmatrix} \begin{bmatrix} A_1(z) \\ A_1^R(z) \end{bmatrix} \\ \begin{bmatrix} A_3(z) \\ A_3^R(z) \end{bmatrix} &= \begin{bmatrix} 1 & -\gamma_3 z^{-1} \\ -\gamma_3 & z^{-1} \end{bmatrix} \begin{bmatrix} A_2(z) \\ A_2^R(z) \end{bmatrix} \end{aligned} \tag{8.3.8}$$

More generally, for an order-M denominator, we have the forward recursions, also known as the *forward Levinson recursions* because of their connection to linear prediction, for $p = 1, 2, \dots, M$, starting with, $A_0(z)=A_0^R(z)=1$,

$$\begin{bmatrix} A_p(z) \\ A_p^R(z) \end{bmatrix} = \begin{bmatrix} 1 & -\gamma_p z^{-1} \\ -\gamma_p & z^{-1} \end{bmatrix} \begin{bmatrix} A_{p-1}(z) \\ A_{p-1}^R(z) \end{bmatrix} \qquad \text{(forward Levinson)} \tag{8.3.9}$$

The polynomial $A_p(z)$, to be referred to as a linear prediction polynomial, is the transfer function from the overall input $F_0(z)$ to the $F_p(z)$ output of the pth lattice section, while $A_p^R(z)$ is the transfer function to the $R_p(z)$ output, for $p = 1, 2, \dots, M$,

$$\begin{aligned} F_p(z) &= A_p(z) F_0(z) \\ R_p(z) &= A_p^R(z) F_0(z) \end{aligned} \tag{8.3.10}$$

In the time-domain, the forward Levinson recursion (8.3.9) reads as follows, building an order-p filter from and order-$(p-1)$ one,

$$\mathbf{a}_p = \begin{bmatrix} \mathbf{a}_{p-1} \\ 0 \end{bmatrix} - \gamma_p \begin{bmatrix} 0 \\ \mathbf{a}_{p-1}^R \end{bmatrix}, \quad p = 1, 2, \dots, M \qquad \text{(forward Levinson)} \tag{8.3.11}$$

where $\mathbf{a}_p = [1, a_{p1}, a_{p2}, \dots, a_{pp}]^T$ is the column vector of the coefficients of $A_p(z)$, starting with $\mathbf{a}_0 = \mathbf{a}_0^R = [1]$. Explicitly, we have,

$$\begin{bmatrix} 1 \\ a_{p,1} \\ a_{p,2} \\ \vdots \\ a_{p,p-1} \\ a_{p,p} \end{bmatrix} = \begin{bmatrix} 1 \\ a_{p-1,1} \\ a_{p-1,2} \\ \vdots \\ a_{p-1,p-1} \\ 0 \end{bmatrix} - \gamma_p \begin{bmatrix} 0 \\ a_{p-1,p-1} \\ a_{p-1,p-2} \\ \vdots \\ a_{p-1,1} \\ 1 \end{bmatrix} \quad\Rightarrow\quad \begin{aligned} a_{p,1} &= a_{p-1,1} - \gamma_p a_{p-1,p-1} \\ a_{p,2} &= a_{p-1,2} - \gamma_p a_{p-1,p-2} \\ &\vdots \\ a_{p,p-1} &= a_{p-1,p-1} - \gamma_p a_{p-1,1} \\ a_{p,p} &= -\gamma_p \end{aligned}$$

The inverse of this recursion is the *backward Levinson recursion*, which starts with the final order-M filter, $\mathbf{a}_M = [1, a_{M1}, a_{M2}, \dots, a_{MM}]^T$, and proceeds downwards to order-0, extracting the reflection coefficients γ_p in the process, for, $p = M, M-1, \dots, 1$,

$$\begin{aligned} \gamma_p &= -a_{p,p} \\ \mathbf{a}_{p-1} &= \frac{\mathbf{a}_p + \gamma_p \mathbf{a}_p^R}{1-\gamma_p^2}\,, \quad p = M, M-1, \dots, 1 \end{aligned} \qquad \text{(backward Levinson)} \tag{8.3.12}$$

or, explicitly,

$$\begin{bmatrix} 1 \\ a_{p-1,1} \\ a_{p-1,2} \\ \vdots \\ a_{p-1,p-1} \\ 0 \end{bmatrix} = \frac{1}{1-\gamma_p^2} \left(\begin{bmatrix} 1 \\ a_{p,1} \\ a_{p,2} \\ \vdots \\ a_{p,p-1} \\ a_{p,p} \end{bmatrix} + \gamma_p \begin{bmatrix} a_{p,p} \\ a_{p,p-1} \\ \vdots \\ a_{p,2} \\ a_{p,1} \\ 1 \end{bmatrix} \right)$$

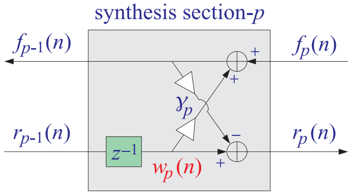
where by construction, the bottom entry of $\mathbf{a}_{p-1}$ is zero, thus, the polynomial order is successively reduced to order 0. To summarize, we use the forward recursion (8.3.11) to generate the denominator polynomial coefficients from the reflection coefficients, and use the backward recursion, to generate the reflection coefficients from the polynomial coefficients,

$$\begin{aligned} [\gamma_1,\, \gamma_2,\, \dots,\, \gamma_M] &\longrightarrow \text{Eq. (8.3.11)} \longrightarrow [a_1,\, a_2,\, \dots,\, a_M] \\ [a_1,\, a_2,\, \dots,\, a_M] &\longrightarrow \text{Eq. (8.3.12)} \longrightarrow [\gamma_1,\, \gamma_2,\, \dots,\, \gamma_M] \end{aligned} \tag{8.3.13}$$

Next, we consider the synthesis structure shown at the bottom of Fig. 8.3.2 in which the directions of the $f_p(n)$ signals are reversed and are order-decreasing. The input/output equations are obtained by simply rearranging the terms of Eq. (8.3.2),

$$f_p(n)=f_{p-1}(n)-\gamma_p r_{p-1}(n-1) \quad\Rightarrow\quad f_{p-1}(n)=f_p(n)+\gamma_p r_{p-1}(n-1)$$

so that now the incoming signals are, $f_p(n)$ and $r_{p-1}(n)$, and the outgoing ones, $f_{p-1}(n)$ and $r_p(n)$, for $p=0,1,\dots,M$,



$$\boxed{\begin{aligned} f_{p-1}(n) &= f_p(n)+\gamma_p r_{p-1}(n-1)\\ r_p(n) &= r_{p-1}(n-1)-\gamma_p f_{p-1}(n)\end{aligned}} \tag{8.3.14}$$

Starting with the rightmost input, $X(z)=F_M(z)=A_M(z)F_0(z)$, the transfer function from $X(z)$ to the backward output $R_p(z)$ of the pth lattice section is, for $p=0,1,\dots,M$,

$$R_p(z)=A_p^R(z)F_0(z)=\frac{A_p^R(z)}{A_M(z)}A_M(z)F_0(z)=\frac{A_p^R(z)}{A_M(z)}F_M(z)=\frac{A_p^R(z)}{A_M(z)}X(z) \tag{8.3.15}$$

The backward signals $r_p(n)$ can be linearly combined with the ladder coefficients d_p to generate the overall output $y(n)$ as shown in Fig. 8.1.1, from which the overall transfer function can be obtained,

$$Y(z)=\sum_{p=0}^{M}d_p R_p(z)=\frac{\displaystyle\sum_{p=0}^{M}d_p A_p^R(z)}{A_M(z)}X(z) \tag{8.3.16}$$

thus, resulting in the overall transfer function from $X(z)$ to $Y(z)$,

$$H(z)=\frac{Y(z)}{X(z)}=\frac{\displaystyle\sum_{p=0}^{M}d_p A_p^R(z)}{A_M(z)}=\frac{B_M(z)}{A_M(z)} \tag{8.3.17}$$

where the numerator polynomial $B_M(z)$ is expressible as a linear combination of the reversed polynomials, $A_p^R(z)$,

$$\boxed{B_M(z)=\sum_{p=0}^{M}d_p A_p^R(z)} \tag{8.3.18}$$

This can be written in the time domain in the matrix form,

$$\mathbf{b}_M=\sum_{p=0}^{M}d_p\mathbf{a}_p^R=[\mathbf{a}_0^R,\mathbf{a}_1^R,\cdots,\mathbf{a}_M^R]\begin{bmatrix}d_0\\d_1\\\vdots\\d_M\end{bmatrix}=A\,\mathbf{d} \tag{8.3.19}$$

where A is a unit upper-tridiagonal matrix consisting of the reversed prediction polynomials. For example, for $M=3$, we have,

$$A=[\mathbf{a}_0^R,\mathbf{a}_1^R,\mathbf{a}_2^R,\mathbf{a}_3^R]=\begin{bmatrix}1&a_{11}&a_{22}&a_{33}\\0&1&a_{21}&a_{32}\\0&0&1&a_{31}\\0&0&0&1\end{bmatrix}$$

and Eq. (8.3.19) becomes,

$$\mathbf{b}=\begin{bmatrix}b_0\\b_1\\b_2\\b_3\end{bmatrix}=\begin{bmatrix}1&a_{11}&a_{22}&a_{33}\\0&1&a_{21}&a_{32}\\0&0&1&a_{31}\\0&0&0&1\end{bmatrix}\begin{bmatrix}d_0\\d_1\\d_2\\d_3\end{bmatrix}=A\,\mathbf{d} \tag{8.3.20}$$

Solving for the ladder weights in terms of the numerator coefficients $\mathbf{b}$ can be done efficiently with MATLAB's backslash operator,

$$\mathbf{d}=A^{-1}\mathbf{b}=A\backslash\mathbf{b} \tag{8.3.21}$$

To summarize, given the numerator and denominator coefficients, $\mathbf{b},\mathbf{a}$, of a transfer function, one subjects $\mathbf{a}$ to the backward Levinson recursions generating the reflection coefficients γ_p and all the reversed prediction polynomials, gathered in the matrix A, and then, solves for the ladder coefficients d_p using Eq. (8.3.21).

Conversely, if the γ_p and d_p coefficients are given, one subjects the γ_ps to the forward Levinson recursion generating all the prediction polynomials, gathered in the matrix A, and then constructs the numerator coefficients $\mathbf{b}$ from Eq. (8.3.19).

The overall time-domain filtering equations for an order-M filter, generalizing the block diagram in Fig. 8.3.1, are obtained by doing the backward recursion Eq. (8.3.14) starting the with last section and proceeding to the first one.

The resulting difference equations can be converted into a sample-by-sample processing algorithm using the delay contents, $w_p(n)=r_{p-1}(n-1)$, as auxiliary internal state variables. However, since, $w_p(n+1)=r_{p-1}(n)$ and we are going downwards in order, the signal $r_{p-1}(n)$ is not yet available until the next lower section, in other words, what can be updated at the pth pass is the quantity, $w_{p+1}(n+1)=r_p(n)$. Thus, in structuring the order-decreasing recursions, we must carry out the operations in the order,

```
for $p=M-1,M-2,\dots,1$, do,
    $f_{p-1}(n)=f_p(n)+\gamma_p w_p(n)$
    $r_p(n)=w_p(n)-\gamma_p f_{p-1}(n)$
    $w_{p+1}(n+1)=r_p(n)$
end
```

Putting all the steps together that include the proper initialization at stage M for a given input sample $x(n)$, and the successive computation of the output sample $y(n)$ using the ladder weights, we arrive at the following implementation for the difference equations and sample processing algorithm,

```
for each time instant $n$, do,
    $f_M(n) = x(n)$
    $f_{M-1}(n) = f_M(n) + \gamma_M w_M(n)$
    $r_M(n) = w_M(n) - \gamma_M f_{M-1}(n)$
    $y(n) = d_M r_M(n)$
    for $p = M-1, M-2, \dots, 1$, do,
        $f_{p-1}(n) = f_p(n) + \gamma_p w_p(n)$
        $r_p(n) = w_p(n) - \gamma_p f_{p-1}(n)$
        $w_{p+1}(n+1) = r_p(n)$
        $y(n) = y(n) + d_p r_p(n)$
    end
    $r_0(n) = f_0(n)$
    $w_1(n+1) = r_0(n)$
    $y(n) = y(n) + d_0 r_0(n)$
end
```

```
for each input sample $x$, do,
    $f = x$
    $f = f + \gamma_M w_M$
    $r = w_M - \gamma_M f$
    $y = d_M r$
    for $p = M-1, M-2, \dots, 1$, do,
        $f = f + \gamma_p w_p$
        $r = w_p - \gamma_p f$
        $w_{p+1} = r$
        $y = y + d_p r$
    end
    $r = f$
    $w_1 = r$
    $y = y + d_0 r$
end
```

(8.3.22)

so that upon exit from the nth time pass, we have accumulated all of the output terms,

$$y(n) = d_0 r_0(n) + d_1 r_1(n) + \cdots + d_M r_M(n)$$

Rearranged Lattice Realization

A rearranged version of the standard lattice realization is shown in Figs. 8.3.3 in which the intermediate signals $f_p(n), r_p(n)$ running between the lattice sections are the same as those of the standard lattice, as are the coefficients, except that each section has one more multiplier denoted by, $\tau_p^2 = 1 - \gamma_p^2$.

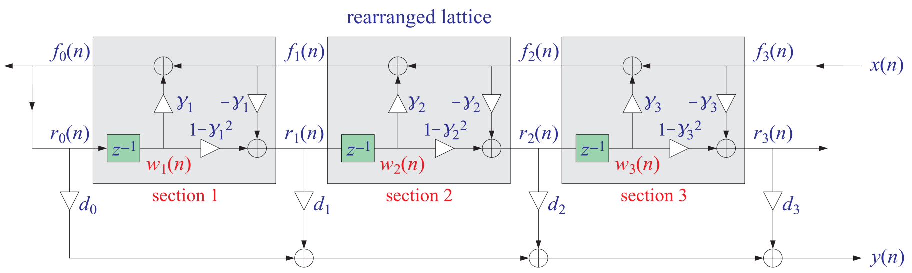


Fig. 8.3.3 Rearranged lattice realization.

The difference equations and sample processing algorithm are in this case,

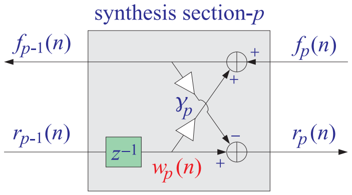
```
for each time instant $n$, do,
    $f_M(n) = x(n)$
    $r_M(n) = \tau_M^2 w_M(n) - \gamma_M f_M(n)$
    $f_{M-1}(n) = f_M(n) + \gamma_M w_M(n)$
    $y(n) = d_M r_M(n)$
    for $p = M-1, M-2, \dots, 1$, do,
        $r_p(n) = \tau_p^2 w_p(n) - \gamma_p f_p(n)$
        $f_{p-1}(n) = f_p(n) + \gamma_p w_p(n)$
        $w_{p+1}(n+1) = r_p(n)$
        $y(n) = y(n) + d_p r_p(n)$
    end
    $r_0(n) = f_0(n)$
    $w_1(n+1) = r_0(n)$
    $y(n) = y(n) + d_0 r_0(n)$
end
```

```
for each input sample $x$, do,
    $f = x$
    $r = \tau_M^2 w_M - \gamma_M f$
    $f = f + \gamma_M w_M$
    $y = d_M r$
    for $p = M-1, M-2, \dots, 1$, do,
        $r = \tau_p^2 w_p - \gamma_p f$
        $f = f + \gamma_p w_p$
        $w_{p+1} = r$
        $y = y + d_p r$
    end
    $r = f$
    $w_1 = r$
    $y = y + d_0 r$
end
```

(8.3.23)

In the sample processing algorithms, the variables f, r are recycled from one lattice section to the next, and in the rearranged case, r must be updated before f to avoid overwriting the correct value of f. The sample processing algorithms can easily be converted into MATLAB functions.

The rearranged lattice is derived simply by substituting the $f_{p-1}(n)$ equation of the standard lattice into the $r_p(n)$ equation,

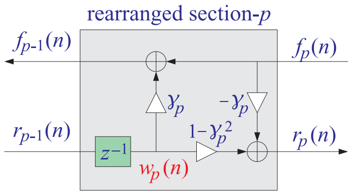


$$\begin{aligned} f_{p-1}(n) &= f_p(n) + \gamma_p w_p(n) \\ r_p(n) &= w_p(n) - \gamma_p f_{p-1}(n) \end{aligned}$$

that is,

$$\begin{aligned} f_{p-1}(n) &= f_p(n) + \gamma_p w_p(n) \\ r_p(n) &= w_p(n) - \gamma_p f_{p-1}(n) \\ &= w_p(n) - \gamma_p [f_p(n) + \gamma_p w_p(n)] \\ &= (1 - \gamma_p^2) w_p(n) - \gamma_p f_p(n) \\ &= \tau_p^2 w_p(n) - \gamma_p f_p(n), \qquad \text{where } \tau_p = \sqrt{1 - \gamma_p^2}, \quad \tau_p^2 = 1 - \gamma_p^2 \end{aligned}$$

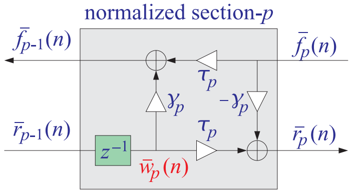
Thus, the pth section is now described as follows, for, $p = M, M-1, \dots, 1$,



$$\boxed{\begin{aligned} f_{p-1}(n) &= f_p(n) + \gamma_p w_p(n) \\ r_p(n) &= \tau_p^2 w_p(n) - \gamma_p f_p(n) \end{aligned}} \qquad (8.3.24)$$

Normalized Lattice Realization

The signals, $f_p(n), r_p(n)$, are the same in the standard and rearranged cases. By appropriate rescaling of these signals, we may obtain the so-called *normalized lattice realization* in which the coefficient, $\tau_p^2 = 1 - \gamma_p^2$, multiplying the $w_p(n)$ signal is shared equally between the right-going signal $w_p(n)$ and the upper left-going signal $f_p(n)$. The resulting pth section with the rescaled signals, $\bar{f}_p(n), \bar{r}_p(n), \bar{w}_p(n)$, becomes then,



$$\boxed{\begin{aligned} \bar{w}_p(n) &= \bar{r}_{p-1}(n-1) \\ \bar{f}_{p-1}(n) &= \tau_p \bar{f}_p(n) + \gamma_p \bar{w}_p(n) \\ \bar{r}_p(n) &= \tau_p \bar{w}_p(n) - \gamma_p \bar{f}_p(n) \end{aligned}} \qquad (8.3.25)$$

Figs. 8.3.3 shows a third-order example, with three such normalized sections. Because the rescaling of the signals also requires the rescaling of the ladder coefficients, which are denoted now by c_p.

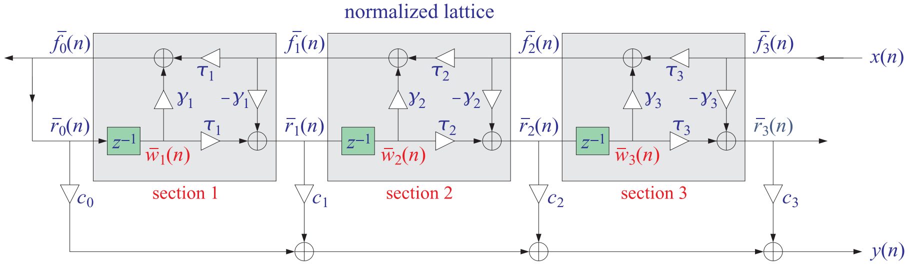


Fig. 8.3.4 Normalized lattice realization.

In order to maintain the equivalence of the normalized and the standard/rearranged forms, the rescaled signals, $\bar{f}_p(n), \bar{r}_p(n)$, and ladder coefficients c_p must be defined as follows in terms of the unscaled signals, $f_p(n), r_p(n)$, and ladder coefficients d_p. For the general order-M case, we have the relationships,

$$\boxed{\begin{aligned} f_p(n) &= \sigma_p \bar{f}_p(n) \\ r_p(n) &= \sigma_p \bar{r}_p(n) \\ d_p &= \frac{\sigma_M}{\sigma_p}\, c_p \end{aligned}} \qquad p = 0, 1, \dots, M \qquad (8.3.26)$$

where,

$$\begin{aligned} \tau_p &= (1-\gamma_p^2)^{1/2}, \quad p = 1, 2, \dots, M \\ \sigma_0 &= 1 \\ \sigma_p &= \tau_1 \tau_2 \cdots \tau_p = \tau_p \sigma_{p-1}, \quad p = 1, 2, \dots, M \end{aligned} \qquad (8.3.27)$$

In the normalized form, the overall input and output signals, $x(n), y(n)$, will have z-transforms, noting that, $R_p(z) = A_p^R(z)\, F_0(z)$,

$$\begin{aligned} X(z) &= \bar{F}_M(z) = \sigma_M^{-1} F_M(z) = \sigma_M^{-1} A_M(z)\, F_0(z) \\ Y(z) &= \sum_{p=0}^{M} c_p \bar{R}_p(z) = \sum_{p=0}^{M} c_p \sigma_p^{-1} R_p(z) = \sum_{p=0}^{M} c_p \sigma_p^{-1} A_p^R(z)\, F_0(z) \end{aligned}$$

so that the resulting transfer function will be,

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\displaystyle\sum_{p=0}^{M} c_p \sigma_p^{-1} A_p^R(z)\, F_0(z)}{\sigma_M^{-1} A_M(z)\, F_0(z)}$$

or,

$$H(z) = \frac{\displaystyle\sum_{p=0}^{M} c_p \sigma_p^{-1} \sigma_M A_p^R(z)}{A_M(z)} = \frac{B_M(z)}{A_M(z)} \qquad (8.3.28)$$

which becomes identical to the transfer function of the standard lattice provided that the d_p and c_p coefficients are related as in Eq. (8.3.26),

$$d_p = c_p \sigma_p^{-1} \sigma_M, \quad p = 0, 1, \dots, M \qquad (8.3.29)$$

which imply that the numerator polynomial is the same in the two realization forms,

$$B_M(z) = \sum_{p=0}^{M} c_p \sigma_p^{-1} \sigma_M A_p^R(z) = \sum_{p=0}^{M} d_p A_p^R(z)$$

The difference equations and sample processing algorithm of the normalized lattice are derived from minor changes to the rearranged case,

```
for each time instant $n$, do,
    $\bar{f}_M(n) = \bar{x}(n)$
    $\bar{r}_M(n) = \tau_M \bar{w}_M(n) - \gamma_M \bar{f}_M(n)$
    $\bar{f}_{M-1}(n) = \tau_M \bar{f}_M(n) + \gamma_M \bar{w}_M(n)$
    $y(n) = c_M \bar{r}_M(n)$
    for $p = M-1, M-2, \dots, 1$, do,
        $\bar{r}_p(n) = \tau_p \bar{w}_p(n) - \gamma_p \bar{f}_p(n)$
        $\bar{f}_{p-1}(n) = \tau_p \bar{f}_p(n) + \gamma_p \bar{w}_p(n)$
        $\bar{w}_{p+1}(n+1) = \bar{r}_p(n)$
        $y(n) = y(n) + c_p \bar{r}_p(n)$
    end
    $\bar{r}_0(n) = \bar{f}_0(n)$
    $\bar{w}_1(n+1) = \bar{r}_0(n)$
    $y(n) = y(n) + c_0 \bar{r}_0(n)$
end
```

```
for each input sample $x$, do,
    $f = x$
    $r = \tau_M w_M - \gamma_M f$
    $f = \tau_M f + \gamma_M w_M$
    $y = c_M r$
    for $p = M-1, M-2, \dots, 1$, do,
        $r = \tau_p w_p - \gamma_p f$
        $f = \tau_p f + \gamma_p w_p$
        $w_{p+1} = r$
        $y = y + c_p r$
    end
    $r = f$
    $w_1 = r$
    $y = y + c_0 r$
end
```

(8.3.30)

8.4 Direct to/from Lattice Transformations

Going back and forth from the direct-form coefficients to the lattice coefficients of the normalized or standard forms can be done with the following MATLAB functions, **dir2nl** and **nl2dir**, which use the recursions (8.3.11) and (8.3.12), also successively building the matrix A that is needed to generate the ladder coefficients,

```
[g,d] = dir2nl(b,a,type);   % type = 'n', 's'
                             % for normalized or standard forms
                             % default type = 'n'
                             % when type='n', d represents the c ladder coefficients

[b,a] = nl2dir(g,d,type);   % default type = 'n'
                             % when type='n', d represents the c ladder coefficients
```

The essential code in these functions is given below. We note also that the built-in MATLAB functions, **tf2latc** and **latc2tf**, perform similar operations, but only for the standard lattice. By their convention, their reflection coefficients k_p are the negatives of ours, that is, $k_p = -\gamma_p$. The ladder coefficients d_p are the same.

```
nl2dir.m     gamma,d --> b,a
---------------------------------------
M = length(g);
d = d(:); g = g(:);

a = 1;  A = eye(M+1);

for p = 1:M,
   a = [a; 0] - g(p)*[0; flip(a)];
   A(1:p+1,p+1) = flip(a);
end

if type == 'n'
   t = sqrt(1-g.^2);
   s = [1; cumprod(t)] / prod(t);
   d = d./s;
end

b = A*d;
```

```
dir2nl.m     b,a --> gamma,d
---------------------------------------
g = zeros(M,1);

A(1:M+1,M+1) = flip(a);

for p = M:-1:1
   g(p) = -a(end);
   a = (a + g(p)*flip(a))/(1-g(p)^2);
   a(end) = [];
   A(1:p,p) = flip(a);
end

d = A\b;

if type == 'n'
   t = sqrt(1-g.^2);
   s = [1; cumprod(t)] / prod(t);
   d = s.*d;
end
```

8.5 Filtering in Lattice Realizations

The filtering operations in the lattice realization are implemented by the following MATLAB function, **nlfilt**, either in the normalized or the standard forms. Its inputs and outputs are described below.

```
% nlfilt.m - filtering in the normalized lattice form
%
% Usage: y = nlfilt(g,d,x,type)
%        y = nlfilt(g,d,x)       (default type = ’n’)
%
% g = gamma reflection coefficients = [g_1, ..., g_M]
% d = ladder weights = [d_0, d_1, ..., d_M]
% x = input signal vector (row or column)
% type = ’n’, ’s’, for normalized or standard/rearranged lattice, default ’n’
%
% y = output signal (same size as x)
%
% notes: implements the sample processing algorithm of the normalized lattice
%
%        normalization factors / transmission coefficients, tau = sqrt(1-g.^2)
%
%        type ’s’ standard lattice is implemented in its rearranged form

function y = nlfilt(g,d,x,type)

if nargin==0, help nlfilt; return; end
if nargin==3, type = ’n’; end

M = length(g);

if length(d) ~= M+1, disp(’d must have length M+1’); return; end

t = 1-g.^2;                      % transmission coefficients squared

if type == ’n’
   tf = sqrt(t);                 % normalization gain factors
   tr = tf;
else
   tf = ones(M,1);
   tr = t;
end

y = zeros(size(x));              % initialize y to same size as x
w = zeros(M,1);                  % internal states

for n = 1:length(x),             % lattice/ladder sample processing algorithm
  f = x(n);                      % f is f_M

  for p = M:-1:1,                % run over lattice sections
    r = tr(p)*w(p) - g(p)*f;
    f = tf(p)*f + g(p)*w(p);
    if p<M, w(p+1) = r; end      % next w_{p+1}(n+1) = r_p(n), skips p=M case
    y(n) = y(n) + d(p+1)*r;      % accumulate ladder output
  end

  y(n) = y(n) + d(1)*f;          % d(1) stands for d_0, last f is f_0 = r_0
  w(1) = f;                      % next w_1(n+1) = r_0(n) = f_0(n)
end
```

8.6 Frequency Response of Lattice Forms

The frequency response $H(\omega)$ of a lattice filter can be computed by mapping the coefficients, $\{\gamma_p, c_p\}$, to the direct form ones, $\{a_i, b_i\}$ and using, for example, the built-in MATLAB function **freqz**,

```
[b,a] = lat2dir(g,d,type);

omega = ...  % define desired vector of digital frequencies [rads/sample]

H = freqz(b,a,omega);
```

Alternatively, the **lat2dir** function can be modified in order to compute the frequency response of the lattice. The essential code implemented into the function, **lat2freq**, is given below.

```
lat2freq.m     gamma,d --> frequency response
---------------------------------------------
M = length(g);       % g,d assumed columns

if type == ’n’
   t = sqrt(1-g.^2);
   s = [1; cumprod(t)] / prod(t);
   d = d./s;
end

a = 1;
B = d(1);            % numerator DTFT

for p = 1:M,
   a = [a; 0] - g(p)*[0; flip(a)];
   B = B + d(p+1) * freqz(flip(a),1,w);
end

A = freqz(a,1,w);    % denominator DTFT

H = B./A;            % frequency response H(w)
```

8.7 Schur-Cohn Stability Test

A necessary and sufficient condition for the denominator polynomial $A_M(z)$ to have all its zeros inside the unit circle on the z-plane is that all the reflection coefficients extracted from $A_M(z)$ by the backward Levinson recursion have magnitude less that one, that is, $|\gamma_p| < 1$, for $p = 1, 2, \dots, M$. For a proof, see Ref. [45].

Thus, the causality/stability of the IIR filter, $H(z) = B_M(z)/A_M(z)$, can be determined by subjecting $A_M(z)$ to the backward Levinson recursion and extracting the reflection coefficients. This is known as the Schur-Cohn stability test.

8.8 Lattice Filter Examples

Next, we discuss a number of examples that illustrate explicitly the operations of going from the direct to/from the lattice forms, as well as the filtering operations in the standard or normalized forms.

Example 1

Consider the following order-3 transfer function,

$$H(z)=\frac{B_3(z)}{A_3(z)}=\frac{7+1.14z^{-1}-3.9z^{-2}+z^{-3}}{1+0.1z^{-1}-0.26z^{-2}+0.6z^{-3}}$$

with filter coefficient vectors,

$$\mathbf{a}_3=\begin{bmatrix}1\\0.1\\-0.26\\0.6\end{bmatrix},\quad \mathbf{b}_3=\begin{bmatrix}7\\1.14\\-3.9\\1\end{bmatrix}$$

We will construct the lower order polynomials $A_2(z), A_1(z), A_0(z)$ from the backward Levinson recursion, and extract the reflection coefficients $\gamma_1, \gamma_2, \gamma_3$ in the process, and also express $B_3(z)$ as a linear combination of the reversed polynomials in the form,

$$B_3(z)=d_0A_0^R(z)+d_1A_1^R(z)+d_2A_2^R(z)+d_3A_3^R(z) \tag{8.8.1}$$

The steps are as follows. First we extract γ_3 and calculate $\mathbf{a}_2$,

$$\gamma_3=-a_{33}=-0.6 \quad\Rightarrow\quad \mathbf{a}_2=\frac{\mathbf{a}_3+\gamma_3\mathbf{a}_3^R}{1-\gamma_3^2}=\frac{\begin{bmatrix}1\\0.1\\-0.26\\0.6\end{bmatrix}+(-0.6)\begin{bmatrix}0.6\\-0.26\\0.1\\1\end{bmatrix}}{1-(-0.6)^2}=\begin{bmatrix}1\\0.4\\-0.5\\0\end{bmatrix}$$

and after deleting the bottom zero in $\mathbf{a}_2$, we repeat the recursion to get γ_2 and $\mathbf{a}_1$,

$$\gamma_2=-a_{22}=0.5 \quad\Rightarrow\quad \mathbf{a}_1=\frac{\mathbf{a}_2+\gamma_2\mathbf{a}_2^R}{1-\gamma_2^2}=\frac{\begin{bmatrix}1\\0.4\\-0.5\end{bmatrix}+0.5\begin{bmatrix}-0.5\\0.4\\1\end{bmatrix}}{1-0.5^2}=\begin{bmatrix}1\\0.8\\0\end{bmatrix}$$

and after deleting the bottom zero of $\mathbf{a}_1$, we have,

$$\mathbf{a}_1=\begin{bmatrix}1\\0.8\end{bmatrix} \quad\Rightarrow\quad \gamma_1=-a_{11}=-0.8$$

thus, $[\gamma_1,\gamma_2,\gamma_3]=[-0.8,\,0.5,\,-0.6]$. The feed-forward numerator coefficients d_p are calculated by solving the equation (8.3.20), which reads here,

$$\begin{bmatrix}1&0.8&-0.5&0.6\\0&1&0.4&-0.26\\0&0&1&0.1\\0&0&0&1\end{bmatrix}\begin{bmatrix}d_0\\d_1\\d_2\\d_3\end{bmatrix}=\begin{bmatrix}7\\1.14\\-3.9\\1\end{bmatrix} \quad\Rightarrow\quad \begin{bmatrix}d_0\\d_1\\d_2\\d_3\end{bmatrix}=\begin{bmatrix}2\\3\\-4\\1\end{bmatrix}$$

with the solution obtained easily with the backward substitution. Finally, we verify Eq. (8.8.1)

$$\begin{aligned}B_3(z)&=d_0A_0^R(z)+d_1A_1^R(z)+d_2A_2^R(z)+d_3A_3^R(z)\\&=2+3\big(0.8+z^{-1}\big)+(-4)\big(-0.5+0.4z^{-1}+z^{-2}\big)+\big(0.6-0.26z^{-1}+0.1z^{-2}+z^{-3}\big)\\&=7+1.14z^{-1}-3.9z^{-2}+z^{-3}\end{aligned}$$

Finally, we note that the normalized lattice feed-forward coefficients c_p are related to the d_p coefficients as follows in the order-3 case,

$$\begin{array}{l}\sigma_0=1\\ \sigma_1=\tau_1\\ \sigma_2=\tau_1\tau_2\\ \sigma_3=\tau_1\tau_2\tau_3\end{array} \quad\Rightarrow\quad d_p=\frac{\sigma_3}{\sigma_p}c_p \quad\Rightarrow\quad \begin{array}{l}d_0=\tau_1\tau_2\tau_3\,c_0\\ d_1=\tau_2\tau_3\,c_1\\ d_2=\tau_3\,c_2\\ d_3=c_3\end{array}$$

where

$$\begin{aligned}\tau_1&=\sqrt{1-\gamma_1^2}=\sqrt{1-(-0.8)^2}=0.6\\ \tau_2&=\sqrt{1-\gamma_2^2}=\sqrt{1-(0.5)^2}=\frac{\sqrt{3}}{2}=0.8660\\ \tau_3&=\sqrt{1-\gamma_3^2}=\sqrt{1-(-0.6)^2}=0.8\end{aligned}$$

Example 2

A third-order normalized lattice filter has reflection coefficients and ladder coefficients:

$$[\gamma_1,\gamma_2,\gamma_3]=[0.5,\,-0.6,\,-0.5]$$

$$[c_0,c_1,c_2,c_3]=\frac{1}{6}[40,\ 15\sqrt{3},\ 8\sqrt{3},\ 6]\approx[6.6667,\ 4.3301,\ 2.3094,\ 1]$$

Determine the transfer function, $H(z)=B_3(z)/A_3(z)$, of this filter, as well as the lower order linear prediction polynomials, $A_1(z), A_2(z)$, and the ladder coefficients d_0, d_1, d_2, d_3 in the standard lattice form. Moreover, state the sample processing algorithm in its standard lattice form and implement it in MATLAB applied to the particular input,

$$\mathbf{x}=[1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8]$$

We start by building the polynomials $A_1(z), A_2(z), A_3(z)$ using the forward Levinson recursion expressed in vectorial form. Starting with the 0th order polynomial, $\mathbf{a}_0 = \mathbf{a}_0^R = [1]$, we construct the order-1 polynomial $A_1(z)$,

$$\mathbf{a}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} - \gamma_1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} - 0.5 \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -0.5 \end{bmatrix}$$

$$A_1(z) = 1 - 0.5z^{-1}$$

Then, we construct the order-2 polynomial,

$$\mathbf{a}_2 = \begin{bmatrix} \mathbf{a}_1 \\ 0 \end{bmatrix} - \gamma_2 \begin{bmatrix} 0 \\ \mathbf{a}_1^R \end{bmatrix} = \begin{bmatrix} 1 \\ -0.5 \\ 0 \end{bmatrix} - (-0.6) \begin{bmatrix} 0 \\ -0.5 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -0.8 \\ 0.6 \end{bmatrix}$$

$$A_2(z) = 1 - 0.8z^{-1} + 0.6z^{-2}$$

and finally the order-3 polynomial, which is the denominator of $H(z)$,

$$\mathbf{a}_3 = \begin{bmatrix} \mathbf{a}_2 \\ 0 \end{bmatrix} - \gamma_3 \begin{bmatrix} 0 \\ \mathbf{a}_2^R \end{bmatrix} = \begin{bmatrix} 1 \\ -0.8 \\ 0.6 \\ 0 \end{bmatrix} - (-0.5) \begin{bmatrix} 0 \\ 0.6 \\ -0.8 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -0.5 \\ 0.2 \\ 0.5 \end{bmatrix}$$

$$A_3(z) = 1 - 0.5z^{-1} + 0.2z^{-2} + 0.5z^{-3}$$

Next, we transform the ladder coefficients c_p into standard un-normalized ones d_p,

$$\tau_1 = \sqrt{1-\gamma_1^2} = \frac{\sqrt{3}}{2}\,, \quad \tau_2 = \sqrt{1-\gamma_2^2} = 0.8\,, \quad \tau_3 = \sqrt{1-\gamma_3^2} = \frac{\sqrt{3}}{2}$$

$$d_0 = \tau_1\tau_2\tau_3 c_0 = \frac{\sqrt{3}}{2} \cdot 0.8 \cdot \frac{\sqrt{3}}{2} \cdot \frac{40}{6} = 4$$

$$d_1 = \tau_2\tau_3 c_1 = 0.8 \cdot \frac{\sqrt{3}}{2} \cdot \frac{15\sqrt{3}}{6} = 3$$

$$d_2 = \tau_3 c_2 = \frac{\sqrt{3}}{2} \cdot \frac{8\sqrt{3}}{6} = 2$$

$$d_3 = c_3 = 1$$

and calculate the numerator polynomial coefficients from Eq. (8.3.20),

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 1 & a_{11} & a_{22} & a_{33} \\ 0 & 1 & a_{21} & a_{32} \\ 0 & 0 & 1 & a_{31} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} 1 & -0.5 & 0.6 & -0.5 \\ 0 & 1 & -0.8 & 0.2 \\ 0 & 0 & 1 & -0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 4 \\ 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 4.2 \\ 1.6 \\ 1.5 \\ 1 \end{bmatrix}$$

Thus, the overall transfer function will be,

$$H(z) = \frac{B_3(z)}{A_3(z)} = \frac{4.2 + 1.6z^{-1} + 1.5z^{-2} + z^{-3}}{1 - 0.5z^{-1} + 0.2z^{-2} + 0.5z^{-3}}$$

The sample processing algorithm using the intermediate signals, f_3, f_2, f_1, f_0, and r_3, r_2, r_1, r_0, and the delayed ones w_3, w_2, w_1 will be as follows,

```
for each input sample $x$, do,
    $f_3 = x$
    $f_2 = f_3 + \gamma_3 w_3$
    $r_3 = w_3 - \gamma_3 f_2$
    $y = d_3 r_3$

    $f_1 = f_2 + \gamma_2 w_2$
    $r_2 = w_2 - \gamma_2 f_1$
    $w_3 = r_2$
    $y = y + d_2 r_2 = d_2 r_2 + d_3 r_3$

    $f_0 = f_1 + \gamma_1 w_1$
    $r_1 = w_1 - \gamma_1 f_0$
    $w_2 = r_1$
    $y = y + d_1 r_1 = d_1 r_1 + d_2 r_2 + d_3 r_3$

    $r_0 = f_0$
    $w_1 = r_0$
    $y = y + d_0 r_0 = d_0 r_0 + d_1 r_1 + d_2 r_2 + d_3 r_3$
```

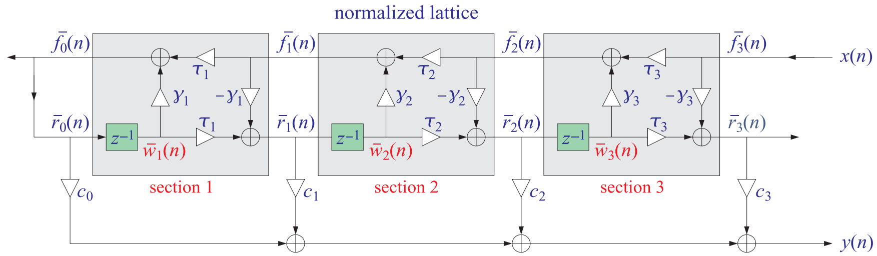
The MATLAB implementation for this example is as follows,

```
g1 = 0.5; g2 = -0.6; g3 = -0.5;    % reflection coefficients
d0 = 4; d1 = 3; d2 = 2; d3 = 1;    % ladder coefficients
b = [4.2, 1.6, 1.5, 1];            % numerator coefficients
a = [1, -0.5, 0.2, 0.5];           % denominator coefficients

x = [1 2 3 4 5 6 7 8]’;      % input signal

w1 = 0; w2 = 0; w3 = 0;      % initial states
y = zeros(size(x));

for n = 1:length(x)
    f = x(n);                % section 3
    f = f + g3*w3;
    r = w3 - g3*f;
    y(n) = y(n) + d3*r;

    f = f + g2*w2;           % section 2
    r = w2 - g2*f;
    w3 = r;
```

```
        y(n) = y(n) + d2*r;

        f = f + g1*w1;           % section 1
        r = w1 - g1*f;
        w2 = r;
        y(n) = y(n) + d1*r;

        r = f;
        w1 = r;
        y(n) = y(n) + d0*r;      % final output sample
    end

    yf = filter(b,a,x);          % compare with output from FILTER

    [x,y,yf]

    %  1    4.2000    4.2000
    %  2   12.1000   12.1000
    %  3   22.5100   22.5100
    %  4   32.3350   32.3350
    %  5   39.5155   39.5155
    %  6   44.2357   44.2358
    %  7   48.5473   48.5473
    %  8   54.4687   54.4687
```

Example 3

Here, we consider a concrete MATLAB implementation of another third-order example realized in its normalized lattice form. The sample processing algorithm implementing the bottom block diagram on page-7 using the intermediate signals $\bar{f}_3, \bar{f}_2, \bar{f}_1, \bar{f}_0$, and $\bar{r}_3, \bar{r}_2, \bar{r}_1, \bar{r}_0$, and the delayed ones $\bar{w}_3, \bar{w}_2, \bar{w}_1$ will be as follows,



```
for each input sample $x$, do,
    $\bar{f}_3 = x$
    $\bar{r}_3 = \tau_3 \bar{w}_3 - \gamma_3 \bar{f}_3$
    $\bar{f}_2 = \tau_3 \bar{f}_3 + \gamma_3 \bar{w}_3$
    $y = c_3 \bar{r}_3$

    $\bar{r}_2 = \tau_2 \bar{w}_2 - \gamma_2 \bar{f}_2$
    $\bar{f}_1 = \tau_2 \bar{f}_2 + \gamma_2 \bar{w}_2$
    $\bar{w}_3 = \bar{r}_2$
    $y = y + c_2 \bar{r}_2$,   or,   $y = c_3 \bar{r}_3 + c_2 \bar{r}_2$

    $\bar{r}_1 = \tau_1 \bar{w}_1 - \gamma_1 \bar{f}_1$
    $\bar{f}_0 = \tau_1 \bar{f}_1 + \gamma_1 \bar{w}_1$
    $\bar{w}_2 = \bar{r}_1$
    $y = y + c_1 \bar{r}_1$,   or,   $y = c_3 \bar{r}_3 + c_2 \bar{r}_2 + c_1 \bar{r}_1$

    $\bar{r}_0 = \bar{f}_0$
    $y = y + c_0 \bar{r}_0$,   or,   $y = c_3 \bar{r}_3 + c_2 \bar{r}_2 + c_1 \bar{r}_1 + c_0 \bar{r}_0$
    $\bar{w}_1 = \bar{r}_0$
```

The following MATLAB program is a concrete implementation.

```
    c = [1 2 -4 5];                 % ladder coefficients
    ga = [0.6, -0.8, -0.6];         % reflection coefficients
    tau = sqrt(1-ga.^2);            % tau = [0.8, 0.6, 0.8]

    g1 = ga(1); t1 = tau(1);        % simplified notation
    g2 = ga(2); t2 = tau(2);
    g3 = ga(3); t3 = tau(3);
    c0 = c(1);
    c1 = c(2);
    c2 = c(3);
    c3 = c(4);

    [b,a] = lat2dir(ga,c,'n');      % direct-form coefficients


    % [b,a]
    %     0.2480    1.0000
    %     5.1760   -0.6000
    %    -6.2000    0.1520
    %     5.0000    0.6000

    x = [1 2 3 4 5]';               % input signal

    y = zeros(size(x));             % output has same size as x
```

```
w1=0; w2=0; w3=0;                % initialize states

for n=1:length(x)
   f3 = x(n);                    % section 3
   r3 = t3*w3 - g3*f3;
   f2 = t3*f3 + g3*w3;
   y(n) = c3*r3;

   r2 = t2*w2 - g2*f2;           % section 2
   f1 = t2*f2 + g2*w2;
   w3 = r2;
   y(n) = y(n) + c2*r2;

   r1 = t1*w1 - g1*f1;           % section 1
   f0 = t1*f1 + g1*w1;
   w2 = r1;
   y(n) = y(n) + c1*r1;

   r0 = f0;                      % wrap up
   y(n) = y(n) + c0*r0;
   w1 = r0;
end

[x,y]                            % print x,y
%  x      y
% ------------
%  1    0.2480
%  2    5.8208
%  3    8.3508
%  4   13.0969
%  5   16.4403

Err = norm(y - nlfilt(ga,c,x))    % compare with nlfilt()
% Err =
%      0

Err = norm(y - filter(b,a,x))     % compare with filter()
% Err =
%      4.4440e-15
```

8.9 Quantization Effects in Digital Filters

There are two types of quantization effects in digital filters besides the quantization of the input and output signals: *roundoff errors* in the internal computations of the filter and *coefficient quantization*.

Coefficient quantization takes place whenever the filter coefficients are *rounded* from their exact values to a finite number of digits (or, bits for hardware implementations). The direct and canonical realizations tend to be *extremely sensitive* to such roundings, whereas the cascade realization remains very *robust*.

For *higher-order* filters whose poles are *closely clustered* in the z-plane, small changes in the denominator coefficients can cause large shifts in the location of the poles. If any of the poles moves outside the unit circle, the filter will become *unstable*, rendering

it completely unusable. But even if the poles do not move outside, their large shifts may distort the *frequency response* of the filter so that it no longer satisfies the design specifications.

The sensitivity of the poles to the quantization of the filter coefficients can be quantified as follows. Consider an order-M monic denominator polynomial $A(z)$ with M poles p_j, $j = 1, 2, \dots, M$, that are assumed to be distinct for simplicity, that is, with $a_0 = 1$,

$$A(z) = \sum_{m=0}^{M} a_m z^{-m} = \prod_{j=1}^{M}(1 - p_j z^{-1})$$

Separating out the ith pole, we may write,

$$A(z) = \sum_{m=0}^{M} a_m z^{-m} = (1 - p_i z^{-1})F(z)\,, \quad F(z) = \prod_{\substack{j=1\\ j\neq i}}^{M}(1 - p_j z^{-1})$$

and taking differentials of both sides, we obtain the identity in z,

$$dA(z) = \sum_{m=0}^{M}(da_m)\,z^{-m} = -(dp_i)\,z^{-1}F(z) + (1 - p_i z^{-1})\,dF(z)$$

then, setting $z = p_i$, will isolate the dp_i term,

$$\sum_{m=0}^{M}(da_m)\,p_i^{-m} = -(dp_i)\,p_i^{-1}F(p_i) = -(dp_i)p_i^{-1}\prod_{\substack{j=1\\ j\neq i}}^{M}(1 - p_j p_i^{-1}) = -(dp_i)p_i^{-M}\prod_{\substack{j=1\\ j\neq i}}^{M}(p_i - p_j)$$

from which we obtain the partial derivatives,

$$\frac{\partial p_i}{\partial a_m} = -\frac{p_i^{M-m}}{\displaystyle\prod_{\substack{j=1\\ j\neq i}}^{M}(p_i - p_j)}\,, \quad \text{for } i, m = 1, 2, \dots, M \tag{8.9.1}$$

Thus, if the poles p_j are closely clustered with each other, as they tend to be when one designs filters with very sharp specifications, then, the partial derivatives will be very large, implying that tiny shifts in the coefficients a_m can cause large shifts in the poles and could shift them outside the unit circle, rendering the filter unstable.

In practice, one must *always* check that the stability and specifications of the filter are preserved by the rounded coefficients. In using a software package to design a filter, one must always copy the designed coefficients with enough digits to guarantee these requirements. The computer experiments in Sec. 8.10 explore such quantization effects and some common pitfalls.

We do not mean to imply that the direct and canonical forms are always to be avoided; in fact, we saw in Examples 7.6.3 and 7.6.4 that the canonical forms were much simpler to implement than the cascade ones, and were also very robust under coefficient quantization.

In summary, the cascade form is recommended for the implementation of high-order narrowband lowpass, bandpass, or highpass IIR filters that have closely clustered poles. In this regard, it is convenient that many IIR filter design techniques, such as the bilinear transformation method, give the results of the design already in cascaded form.

There are other realization forms, such as cascaded second-order sections in transposed form, parallel forms, and *normalized lattice* forms that are also very robust under coefficient quantization [2].

Roundoff errors occur in the internal multiplication and accumulation operations, for example, $y := y + aw$. The product aw requires twice as many bits as the factors to be represented correctly. A roundoff error will occur if this product is rounded to the original wordlength of the two factors. Such roundoff errors can be trapped into the feedback loops of recursive filters and can get amplified, causing too much distortion in the desired output.

Special state-space realizations and the technique of quantization noise shaping (also called in this context *error spectrum shaping*) can be used to minimize the accumulation of roundoff error [14,86–92]. To prevent overflow of intermediate results in filter computations, appropriate scaling factors must be introduced at various points within the filter stages [19,93,94].

Modern DSP chips address the issues of quantization effects in two ways: by using *long* wordlengths, such as 32 bits, for coefficient storage, and by using *double-precision* accumulators that can perform several MAC operations without roundoff error before the final result is rounded out of the accumulator.

8.10 Computer Experiments – Coefficient Quantization Effects

Experiment 1

Consider the double resonator filter given in cascade and direct forms:

$$
\begin{aligned}
H(z) &= \frac{1}{1-1.8955z^{-1}+0.9930z^{-2}}\cdot\frac{1}{1-1.6065z^{-1}+0.9859z^{-2}} \\
&= \frac{1}{1-3.5020z^{-1}+5.0240z^{-2}-3.4640z^{-3}+0.9790z^{-4}}
\end{aligned}
\qquad (8.10.1)
$$

using four-digit precision to represent the coefficients.

a. Calculate the zeros of the denominators in polar form and place them on the z-plane with respect to the unit circle. Are they inside the unit circle? Plot the magnitude response of the filter in dB for $0 \le \omega \le \pi/2$. Then, using the function **filter** calculate the filter's impulse response $h(n)$ for $n = 0, 1, \dots, 599$, and plot it versus n.

b. Consider the cascade form and round the coefficients of the individual second-order sections to two-digit accuracy. Then, repeat all questions of part (a). Discuss the stability of the resulting filter and compare its magnitude response with that of part (a).

c. Consider the fourth-order direct-form denominator polynomial and round its coefficients to two-digit accuracy. Then, again, repeat all questions of part (a). Discuss the stability of the resulting filter and compare its magnitude response with that of part (a), if such comparison makes sense (why or why not?).

Experiment 2

It is desired to design and implement a digital lowpass Chebyshev type-1 filter having the following specifications: sampling rate of 20 kHz, passband frequency of 1 kHz, stopband frequency of 2 kHz, passband attenuation of 1 dB, and stopband attenuation of 50 dB. The design method described in Sec. 12.12, generates a sixth-order transfer function in cascade and direct forms:

$$
\begin{aligned}
H(z) &= G\cdot\frac{(1+z^{-1})^2}{1+a_{11}z^{-1}+a_{12}z^{-2}}\cdot\frac{(1+z^{-1})^2}{1+a_{21}z^{-1}+a_{22}z^{-2}}\cdot\frac{(1+z^{-1})^2}{1+a_{31}z^{-1}+a_{32}z^{-2}} \\
&= G\cdot\frac{(1+z^{-1})^6}{1+a_1z^{-1}+a_2z^{-2}+a_3z^{-3}+a_4z^{-4}+a_5z^{-5}+a_6z^{-6}}
\end{aligned}
$$

where the normalization gain factor is $G = 8.07322364\times 10^{-7}$ and is to remain fixed in the remainder of this problem. The full precision coefficients are given below:

$$
\begin{aligned}
\mathbf{a}_1 &= [1, a_{11}, a_{12}] = [1\,,\ -1.86711351\,,\ 0.96228613] \\
\mathbf{a}_2 &= [1, a_{21}, a_{22}] = [1\,,\ -1.84679822\,,\ 0.89920764] \\
\mathbf{a}_3 &= [1, a_{31}, a_{32}] = [1\,,\ -1.85182222\,,\ 0.86344488]
\end{aligned}
\qquad (8.10.2)
$$

where "full precision" means eight-digit precision.

a. Carry out the filter design procedure, and verify that the above 8-digit coefficients, and the gain G, are correct. Then, calculate the direct-form denominator coefficient vector:

$$\mathbf{a} = \mathbf{a}_1 * \mathbf{a}_2 * \mathbf{a}_3 = [1, a_1, a_2, a_3, a_4, a_5, a_6]$$

as well as the normalized lattice coefficients, γ_p, c_p, using your function **dir2lat**. Replace all these coefficients by their 8-digit approximations, and verify that the resulting values are given correctly by the following table,

p	a_p	γ_p	Gc_p
0	1.00000000		0.25668361
1	−5.56573395	0.98144049	0.15971532
2	13.05062482	−0.98561779	0.03518539
3	−16.49540451	0.98529921	0.00411257
4	11.84993647	−0.98329869	0.00028542
5	−4.58649943	0.96911938	0.00001405
6	0.74713457	−0.74713457	0.00000081

b. Calculate the six zeros of **a** and their magnitudes, and display them on the z-plane.

Calculate and plot in dB the magnitude response of this filter over the interval $0 \le f \le 1.2$ kHz, and verify that it meets the prescribed 1 dB passband specification.

Using the normalized lattice form implemented by your function, **nlfilt**, calculate and plot the impulse response of this filter, $h(n)$, for $n = 0, 1, \dots, 200$. Moreover, calculate it also using the function **filter**, and compare the two methods.

c. Determine the quantized version of **a**, say $\hat{\mathbf{a}}$, rounded to five-digit fractional accuracy and calculate its 6 zeros and their magnitudes. Are all the zeros inside the unit circle? Compare the zeros with the full precision zeros. Using $\hat{\mathbf{a}}$ as the filter denominator vector, calculate in dB the magnitude response.

Then, round the normalized lattice coefficients γ_p, c_p to 5-digit fractional precision and calculate the corresponding magnitude response of the normalized lattice using your function **lat2freq**, and plot it together with the above 5-digit direct form, and with the exact response of part (b).

d. Determine the quantized version of **a**, say $\hat{\mathbf{a}}$, rounded to four-digit fractional accuracy and calculate its 6 zeros and their magnitudes. Are all the zeros inside the unit circle? You should find that the direct form is now unstable and therefore the calculation of its magnitude response is meaningless (why is that?).

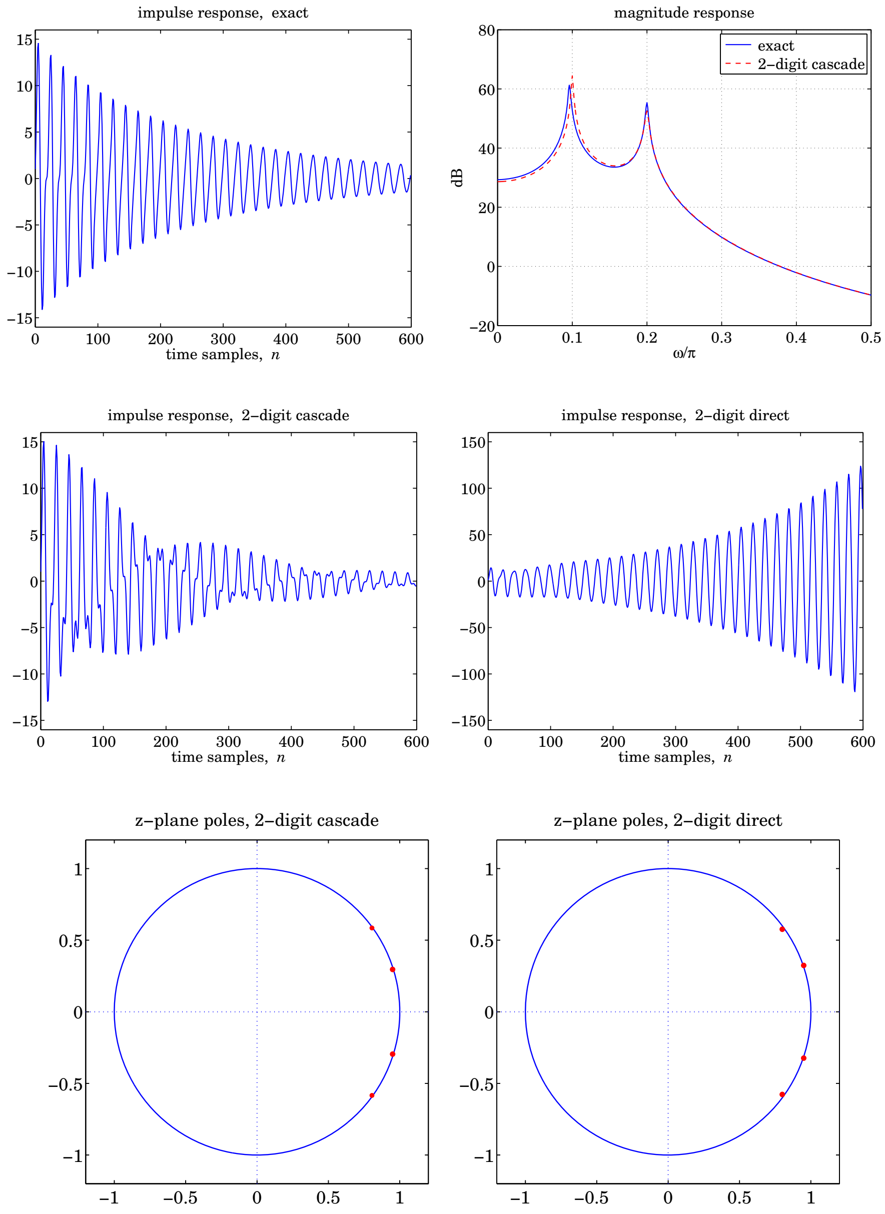
However, the normalized lattice and cascaded forms remain stable. To see this, round the normalized lattice coefficients γ_p, c_p to 4-digit precision, and verify that the resulting γ_p's have magnitude less than one, which guarantees stability by the Schur-Cohn stability criterion. Then, calculate the magnitude response of the 4-digit normalized lattice using your function **lat2freq**.

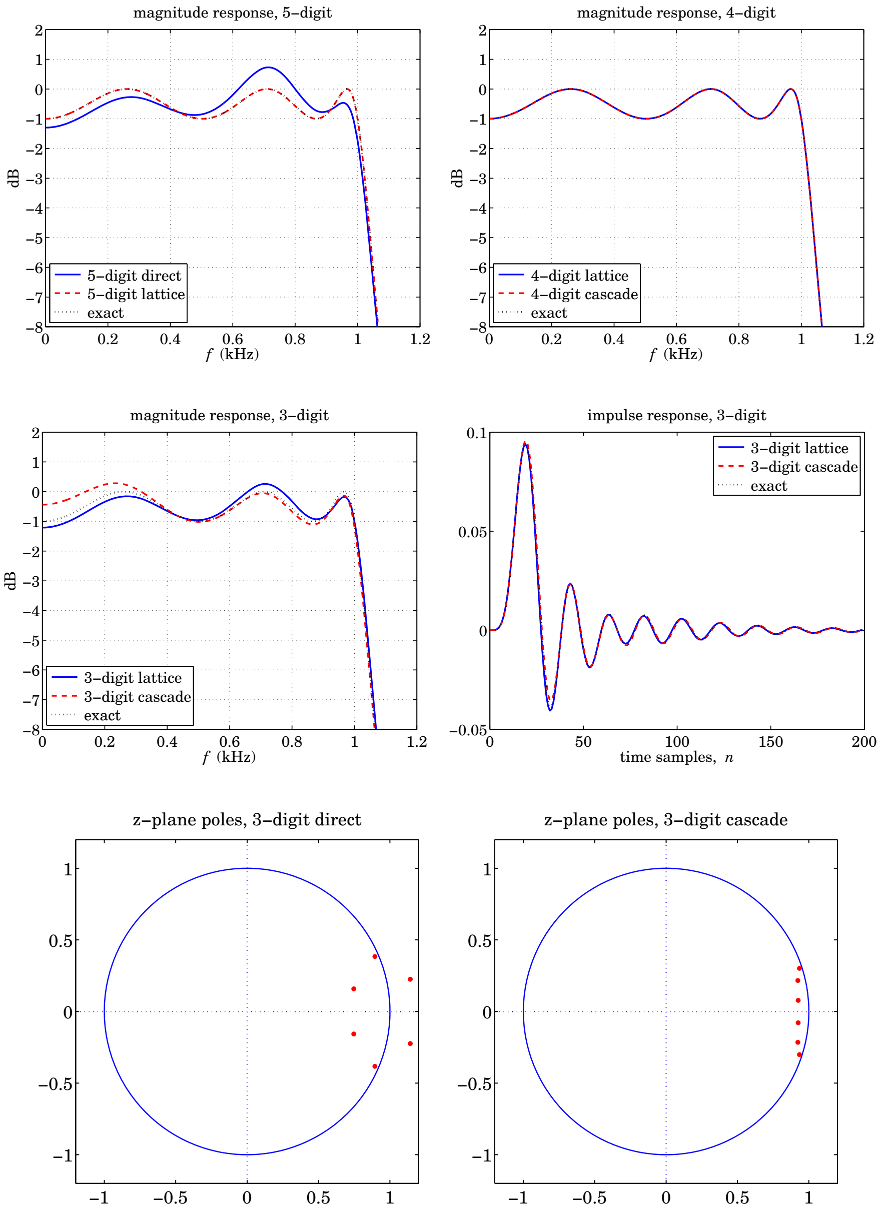
In addition, round to four-digit fractional precision the *individual* second-order coefficient vectors $\mathbf{a}_i$, $i = 1, 2, 3$, of the cascade form of Eq. (8.10.2), and compute their zeros displaying them on the z-plane, and also compute the corresponding magnitude response.

On the same graph, plot the 4-digit normalized lattice and cascade magnitude responses, and the exact response.

e. Repeat all the questions of part (d) using 3-digit fractional precision. In addition, compute and plot the corresponding impulse responses, $h(n)$, for $n = 0, 1, \dots, 200$, of the 3-digit normalized lattice and cascade forms, and the exact response.

Example Graphs for Experiments 1 & 2





9

DTFT and Spectral Analysis

The discrete Fourier transform (DFT) and its fast implementation, the fast Fourier transform (FFT), have three major uses in DSP: (a) the numerical *computation* of the frequency spectrum of a signal; (b) the efficient implementation of *convolution* by the FFT; and (c) the *coding* of waveforms, such as speech or pictures, for efficient transmission and storage [222–227,242–263,642]. The *discrete cosine transform*, which is a variant of the DFT, is especially useful for coding applications [257–259]. In this chapter, we discuss spectrum estimation methods for both deterministic and stationary random signals.

9.1 Frequency Resolution and Windowing

To compute the spectrum of an analog signal digitally, a finite-duration record of the signal is sampled and the resulting samples are transformed to the frequency domain by a DFT or FFT algorithm. The sampling rate f_s must be fast enough to minimize aliasing effects. If necessary, an analog antialiasing prefilter may precede the sampling operation.

The spectrum of the sampled signal $\hat{X}(f)$ is the replication of the desired analog spectrum $X(f)$ at multiples of the sampling rate f_s, as given by the Poisson summation formula, Eq. (1.5.19) of Chapter 1. We saw there that with the proper choice of sampling rate and prefilter, it can be guaranteed that $\hat{X}(f)$ agree with the desired $X(f)$ over the Nyquist interval, that is, by Eq. (1.5.20):

$$T\hat{X}(f)=X(f)\,,\qquad -\frac{f_s}{2}\le f\le\frac{f_s}{2} \tag{9.1.1}$$

This property is a direct consequence of the sampling theorem, following from the non-overlapping of the spectral replicas in $\hat{X}(f)$. However, if the replicas overlap, they will contribute to the right-hand side of Eq. (9.1.1), making the sampled spectrum different from the desired one:

$$T\hat{X}(f)=X(f)+\underbrace{X(f-f_s)+X(f+f_s)+\cdots}_{\text{replicas}}\,,\qquad -\frac{f_s}{2}\le f\le\frac{f_s}{2} \tag{9.1.2}$$

Because digitally we can only compute $\hat{X}(f)$, it is essential that Eq. (9.1.1) be satisfied, or that the extra terms in Eq. (9.1.2) remain small over the Nyquist interval, which happens when $X(f)$ falls off sufficiently fast with f. Example 1.5.2 illustrates the nature of the approximation of Eq. (9.1.2) for a non-bandlimited signal.

Even though $\hat{X}(f)$ is the closest approximation to $X(f)$ that we can achieve by DSP, it is still not computable because generally it requires an infinite number of samples $x(nT)$, $-\infty < n < \infty$. To make it computable, we must make a second approximation to $X(f)$, keeping only a finite number of samples, say, $x(nT)$, $0 \le n \le L-1$. This *time-windowing* process is illustrated in Fig. 9.1.1.

In terms of the time samples $x(nT)$, the original sampled spectrum $\hat{X}(f)$ and its time-windowed version $\hat{X}_L(f)$ are given by:

$$\begin{aligned}\hat{X}(f) &= \sum_{n=-\infty}^{\infty} x(nT)e^{-2\pi jfnT} \\ \hat{X}_L(f) &= \sum_{n=0}^{L-1} x(nT)e^{-2\pi jfnT}\end{aligned} \tag{9.1.3}$$

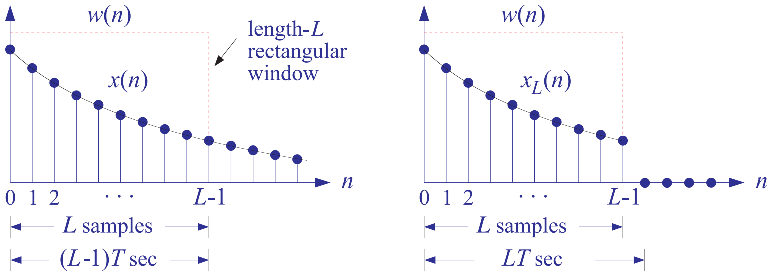


Fig. 9.1.1 Time windowing.

As seen in Fig. 9.1.1, the duration of the windowed data record from the time sample at $n = 0$ to the sample at $n = L-1$ is $(L-1)T$ seconds, where T is the sampling time interval $T = 1/f_s$. Because each sample lasts for T seconds, the last sample will last until time LT. Therefore, we may take the duration of the data record to be:

$$\boxed{T_L = LT} \tag{9.1.4}$$

The windowed signal may be thought of as an infinite signal which is zero outside the range of the window and agrees with the original one within the window. To express this mathematically, we define the *rectangular window* of length L:

$$w(n) = \begin{cases} 1, & \text{if } 0 \le n \le L-1 \\ 0, & \text{otherwise}\end{cases} \tag{9.1.5}$$

Then, define the windowed signal as follows:

$$x_L(n) = x(n)w(n) = \begin{cases} x(n), & \text{if } 0 \le n \le L-1 \\ 0, & \text{otherwise}\end{cases} \tag{9.1.6}$$

The multiplication by $w(n)$ ensures that $x_L(n)$ vanish outside the window. Equations (9.1.3) can now be expressed more simply in the form:

$$\begin{aligned}X(\omega) &= \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n} \\ X_L(\omega) &= \sum_{n=0}^{L-1} x(n)e^{-j\omega n} = \sum_{n=-\infty}^{\infty} x_L(n)e^{-j\omega n}\end{aligned} \tag{9.1.7}$$

where $\omega = 2\pi f/f_s$. Thus, $X_L(\omega)$ is the DTFT of the windowed signal $x_L(n)$ and is computable for any desired value of ω.

As the length L of the data window increases, the windowed signal $x_L(n)$ becomes a better approximation of $x(n)$, and thus, $X_L(\omega)$ a better approximation of $X(\omega)$. Example 1.5.2 illustrates this approximation as L increases.

In general, the windowing process has two major effects: First, it *reduces the frequency resolution* of the computed spectrum, in the sense that the smallest resolvable frequency difference is limited by the length of the data record, that is, $\Delta f = 1/T_L$. This is the well-known "uncertainty principle." Second, it introduces *spurious* high-frequency components into the spectrum, which are caused by the sharp clipping of the signal $x(n)$ at the left and right ends of the rectangular window. This effect is referred to as "frequency leakage."

Both effects can be understood by deriving the precise connection of the windowed spectrum $X_L(\omega)$ to the unwindowed one $X(\omega)$ of Eq. (9.1.7). Using the property that the Fourier transform of the *product* of two time functions is the *convolution* of their Fourier transforms, we obtain the frequency-domain version of $x_L(n) = x(n)w(n)$:

$$\boxed{X_L(\omega) = \int_{-\pi}^{\pi} X(\omega')W(\omega-\omega')\,\frac{d\omega'}{2\pi}} \tag{9.1.8}$$

where $W(\omega)$ is the DTFT of the rectangular window $w(n)$, that is,

$$W(\omega) = \sum_{n=0}^{L-1} w(n)e^{-j\omega n}$$

It can be thought of as the evaluation of the z-transform on the unit circle at $z = e^{j\omega}$. Setting $w(n) = 1$ in the sum, we find:

$$W(z) = \sum_{n=0}^{L-1} w(n)z^{-n} = \sum_{n=0}^{L-1} z^{-n} = \frac{1-z^{-L}}{1-z^{-1}}$$

Setting $z = e^{j\omega}$, we find for $W(\omega)$:

$$\boxed{W(\omega) = \frac{1-e^{-jL\omega}}{1-e^{-j\omega}} = \frac{\sin(\omega L/2)}{\sin(\omega/2)}\,e^{-j\omega(L-1)/2}} \tag{9.1.9}$$

The magnitude spectrum $|W(\omega)| = |\sin(\omega L/2)/\sin(\omega/2)|$ is depicted in Fig. 9.1.2. It consists of a *mainlobe* of height L and base width $4\pi/L$ centered at $\omega = 0$, and several smaller *sidelobes*.

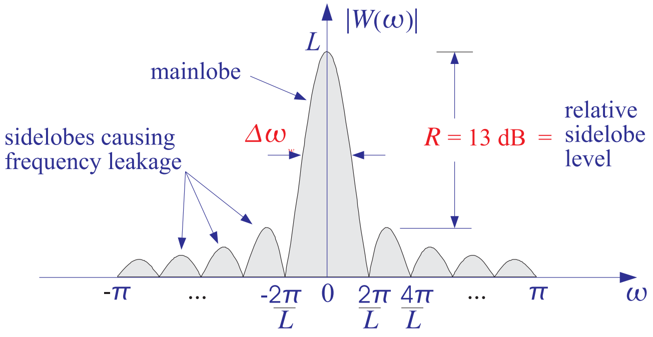


Fig. 9.1.2 Magnitude spectrum of rectangular window.

The sidelobes are between the zeros of $W(\omega)$, which are the zeros of the numerator $\sin(\omega L/2)= 0$, that is, $\omega = 2\pi k/L$, for $k = \pm 1, \pm 2, \dots$ (with $k = 0$ excluded).

The mainlobe peak at DC dominates the spectrum, because $w(n)$ is essentially a DC signal, except when it cuts off at its endpoints. The higher frequency components that have "leaked" away from DC and lie under the sidelobes represent the sharp transitions of $w(n)$ at the endpoints.

The *width* of the mainlobe can be defined in different ways. For example, we may take it to be the width of the base, $4\pi/L$, or, take it to be the 3-dB width, that is, where $|W(\omega)|^2$ drops by $1/2$. For simplicity, we will define it to be *half* the base width, that is, in units of radians per sample:

$$\boxed{\Delta\omega_w = \frac{2\pi}{L}} \qquad \text{(rectangular window width)} \tag{9.1.10}$$

In units of Hz, it is defined through $\Delta\omega_w = 2\pi\Delta f_w/f_s$. Using Eq. (9.1.4), we have:

$$\boxed{\Delta f_w = \frac{f_s}{L} = \frac{1}{LT} = \frac{1}{T_L}} \tag{9.1.11}$$

We will see shortly that the mainlobe width Δf_w determines the *frequency resolution limits* of the windowed spectrum. As L increases, the height of the mainlobe increases and its width becomes narrower, getting more concentrated around DC. However, the height of the sidelobes also increases, but *relative* to the mainlobe height, it remains approximately the same and about 13 dB down.

For example, the peak of the first sidelobe occurs approximately halfway between the two zeros $2\pi/L$ and $4\pi/L$, that is, at $\omega = 3\pi/L$. Using $W(0)= L$, we find that the *relative* heights are essentially independent of L:

$$\left|\frac{W(\omega)}{W(0)}\right|_{\omega=3\pi/L} = \left|\frac{\sin(\omega L/2)}{L\sin(\omega/2)}\right| = \left|\frac{\sin(3\pi/2)}{L\sin(3\pi/2L)}\right| \simeq \frac{1}{L\cdot(3\pi/2L)} = \frac{2}{3\pi}$$

We assumed that L was fairly large (typically, $L \geq 10$), and used the small-x approximation $\sin x \simeq x$ with $x = 3\pi/2L$. In decibels, the *relative sidelobe* level is

$$R = 20\log_{10}\left|\frac{W(\omega)}{W(0)}\right|_{\omega=3\pi/L} \simeq 20\log_{10}\left(\frac{2}{3\pi}\right) = -13.46\text{ dB}$$

More precisely, the local maximum of the first sidelobe occurs at $\omega = 2.8606\pi/L$, corresponding to an attenuation of 13.26 dB.

To illustrate the effect of the convolutional equation (9.1.8), we consider the case of a single analog complex sinusoid of frequency f_1 and its sampled version:

$$x(t)= e^{2\pi j f_1 t}, \quad -\infty < t < \infty \quad \Rightarrow \quad x(n)= e^{2\pi j f_1 nT} = e^{j\omega_1 n}, \quad -\infty < n < \infty$$

where $\omega_1 = 2\pi T f_1 = 2\pi f_1/f_s$. The spectrum of the analog signal $x(t)$ is the Fourier transform:

$$X(f)= \int_{-\infty}^{\infty} x(t)e^{-2\pi j f t}\,dt = \int_{-\infty}^{\infty} e^{-2\pi j(f-f_1)t}\,dt = \delta(f-f_1)$$

Therefore, $X(f)$ consists of a single sharp *spectral line* at $f = f_1$. For a real sinusoid $x(t)= \cos(2\pi f_1 t)$, we would get *two* half-height lines at $f = \pm f_1$. Indeed, the Fourier transform of the cosine is:

$$\cos(2\pi f_1 t)= \frac{1}{2}e^{2\pi j f_1 t} + \frac{1}{2}e^{-2\pi j f_1 t} \longrightarrow \frac{1}{2}\delta(f-f_1)+\frac{1}{2}\delta(f+f_1)$$

Assuming that f_1 lies within the Nyquist interval, that is, $|f_1| \leq f_s/2$, we may use Eq. (9.1.1) to determine the spectrum of the signal $x(n)$ for $-f_s/2 \leq f \leq f_s/2$:

$$X(\omega)= \hat{X}(f)= \frac{1}{T}X(f)= \frac{1}{T}\delta(f-f_1)$$

Using the delta function property, $|a|\delta(ax)= \delta(x)$, we can express the spectrum in terms of the digital frequency $\omega = 2\pi f/f_s = 2\pi T f$, as follows:

$$2\pi\delta(\omega-\omega_1)= \frac{1}{T}2\pi T\,\delta(2\pi T f - 2\pi T f_1)= \frac{1}{T}\delta(f-f_1)$$

Therefore, the spectrum of the sampled signal will be, over the Nyquist interval:

$$X(\omega)= 2\pi\delta(\omega-\omega_1), \qquad -\pi \leq \omega \leq \pi \tag{9.1.12}$$

Outside the Nyquist interval, the spectral line is replicated at multiples of 2π, that is, $2\pi\delta(\omega-\omega_1-2\pi m)$. This was also discussed in Section 5.4. It can be verified that Eq. (9.1.12) generates the sampled sinusoid from the inverse DTFT formula, Eq. (1.5.5):

$$x(n)= \int_{-\pi}^{\pi} X(\omega)e^{j\omega n}\frac{d\omega}{2\pi} = \int_{-\pi}^{\pi} 2\pi\delta(\omega-\omega_1)e^{j\omega n}\frac{d\omega}{2\pi} = e^{j\omega_1 n}$$

The windowed sinusoid consists of the L samples:

$$x_L(n)= e^{j\omega_1 n}, \qquad n = 0, 1, \dots, L-1$$

Its spectrum is obtained by inserting Eq. (9.1.12) into (9.1.8):

$$X_L(\omega)=\int_{-\pi}^{\pi}X(\omega')W(\omega-\omega')\,\frac{d\omega'}{2\pi}=\int_{-\pi}^{\pi}2\pi\,\delta(\omega'-\omega_1)W(\omega-\omega')\,\frac{d\omega'}{2\pi}$$

Because of the delta function $\delta(\omega'-\omega_1)$ in the integrand, we obtain:

$$X_L(\omega)=W(\omega-\omega_1) \tag{9.1.13}$$

This is the translation of $W(\omega)$ centered about ω_1, as shown in Fig. 9.1.3. Thus, the windowing process has the effect of *smearing* the sharp spectral line $\delta(\omega-\omega_1)$ at ω_1 and replacing it by $W(\omega-\omega_1)$.

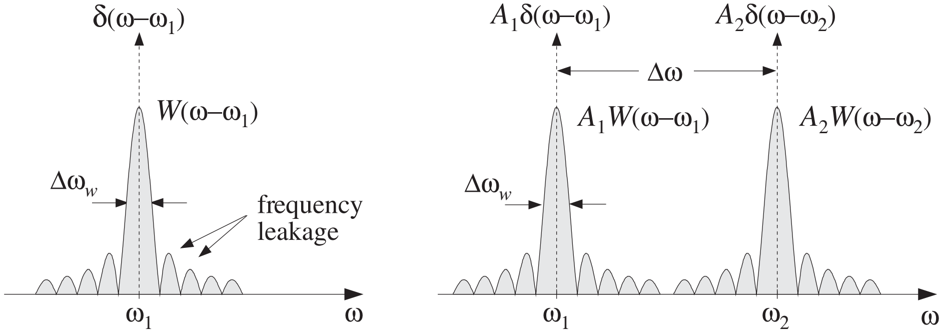


Fig. 9.1.3 Spectra of windowed single and double sinusoids.

A similar analysis can be made in the case when $x(t)$ is a linear combination of *two* complex sinusoids with frequencies f_1 and f_2 and (complex) amplitudes A_1 and A_2. We have for the analog, sampled, and windowed signals and their spectra:

$$\begin{aligned}
x(t) &= A_1e^{2\pi jf_1t}+A_2e^{2\pi jf_2t}, \quad -\infty<t<\infty\\
X(f) &= A_1\delta(f-f_1)+A_2\delta(f-f_2)\\
x(n) &= A_1e^{j\omega_1n}+A_2e^{j\omega_2n}, \quad -\infty<n<\infty\\
X(\omega) &= 2\pi A_1\delta(\omega-\omega_1)+2\pi A_2\delta(\omega-\omega_2), \quad -\pi\le\omega\le\pi\\
x_L(n) &= A_1e^{j\omega_1n}+A_2e^{j\omega_2n}, \quad 0\le n\le L-1\\
X_L(\omega) &= A_1W(\omega-\omega_1)+A_2W(\omega-\omega_2)
\end{aligned}$$

Again, the two sharp spectral lines are replaced by their smeared versions, as shown in Fig. 9.1.3. In this figure, we have taken the frequency separation, $\Delta f=|f_2-f_1|$, or $\Delta\omega=|\omega_2-\omega_1|$, of the two sinusoids to be large enough so that the mainlobes are distinct and do not overlap. However, if Δf is decreased, the mainlobes will begin merging with each other and will not appear as distinct. This will start to happen when Δf is approximately equal to the mainlobe width Δf_w.

The *resolvability* condition that the two sinusoids appear as two distinct ones is that their frequency separation Δf be *greater* than the mainlobe width:

$$\boxed{\Delta f\ge\Delta f_w=\frac{f_s}{L}} \qquad \text{(frequency resolution)} \tag{9.1.14}$$

or, in radians per sample:

$$\boxed{\Delta\omega\ge\Delta\omega_w=\frac{2\pi}{L}} \tag{9.1.15}$$

These equations can be rewritten to give the *minimum number* of samples required to achieve a desired frequency resolution Δf. The smaller the desired separation, the longer the data record:

$$\boxed{L\ge\frac{f_s}{\Delta f}=\frac{2\pi}{\Delta\omega}} \tag{9.1.16}$$

The mainlobe width of $W(\omega)$ determines the amount of achievable frequency resolution. The sidelobes, on the other hand, determine the amount of frequency leakage and are undesirable artifacts of the windowing process. They must be suppressed as much as possible because they may be confused with the mainlobes of *weaker* sinusoids that might be present.

The standard technique for suppressing the sidelobes is to use a *non-rectangular window*—a window that cuts off to zero less sharply and more gradually than the rectangular one. There are literally dozens of possible shapes for such windows, such as trapezoidal, triangular, Gaussian, raised cosine, and many others [222–227].

One of the simplest and most widely used window is the *Hamming window*. It provides a suppression of the sidelobes by at least 40 dB (more accurately, 42.67 dB). Another one that allows the user to control the desired amount of sidelobe suppression is the *Kaiser window* [224], which we will discuss later in Sec. 9.3. The Hamming window, depicted in Fig. 9.1.4, is a raised-cosine type of window defined as follows:

$$w(n)=\begin{cases}0.54-0.46\cos\left(\dfrac{2\pi n}{L-1}\right), & \text{if } 0\le n\le L-1\\ 0, & \text{otherwise}\end{cases} \tag{9.1.17}$$

At its center, $n=(L-1)/2$, the value of $w(n)$ is $0.54+0.46=1$, and at its endpoints, $n=0$ and $n=L-1$, its value is $0.54-0.46=0.08$. Because of the gradual transition to zero, the high frequencies that are introduced by the windowing process are de-emphasized. Fig. 9.1.4 shows the magnitude spectrum $|W(\omega)|$. The sidelobes are still present, but are barely visible because they are suppressed relative to the mainlobe by $R=40$ dB.

The main tradeoff in using any type of non-rectangular window is that its mainlobe becomes *wider* and shorter, thus, reducing the frequency resolution capability of the windowed spectrum. For any type of window, the effective width of the mainlobe is still *inversely proportional* to the window length:

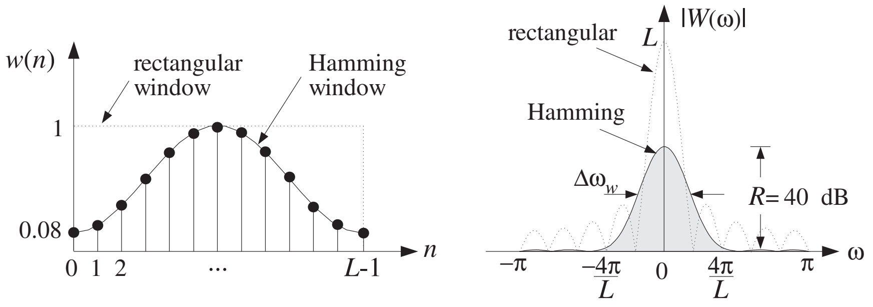


Fig. 9.1.4 Hamming window in the time and frequency domains.

$$\boxed{\Delta f_w = c\,\frac{f_s}{L} = c\,\frac{1}{T_L}} \tag{9.1.18}$$

or, in radians per sample:

$$\boxed{\Delta\omega_w = c\,\frac{2\pi}{L}} \tag{9.1.19}$$

where the constant c depends on the window used and is always $c \geq 1$.

The rectangular window has the *narrowest* width, corresponding to $c = 1$. As seen in Fig. 9.1.4, the Hamming window has approximately $c = 2$, that is, its mainlobe is twice as wide as the rectangular one. The Kaiser window has variable c that depends on the prescribed amount of relative sidelobe level R; see Eq. (11.3.25).

Given a finite data record of L samples, $x(n)$, $n = 0, 1, \dots, L-1$, the windowed signal is defined by Eq. (9.1.6); for example, for the Hamming window:

$$x_L(n) = w(n)x(n) = \left[0.54 - 0.46\cos\left(\frac{2\pi n}{L-1}\right)\right]x(n) \tag{9.1.20}$$

for $n = 0, 1, \dots, L-1$.

The corresponding spectrum $X_L(\omega)$ will still be given by Eq. (9.1.8). If $x(n)$ consists of a linear combination of sinusoids, then each sharp spectral line $\delta(\omega - \omega_i)$ of $x(n)$ will be replaced by the Hamming window spectrum $W(\omega - \omega_i)$. The frequency resolution depends now on the width of the Hamming window Δf_w. It follows that the minimum resolvable frequency difference will be:

$$\boxed{\Delta f \geq \Delta f_w = c\,\frac{f_s}{L} = c\,\frac{1}{T_L}} \tag{9.1.21}$$

This implies that the minimum data record required to achieve a given value of Δf is c-times longer than that of a rectangular window:

$$\boxed{L \geq c\,\frac{f_s}{\Delta f} = c\,\frac{2\pi}{\Delta\omega}} \tag{9.1.22}$$

In summary, the windowing process introduces artificial high-frequency components, which can be suppressed by using a non-rectangular window, but at the expense of reducing the frequency resolution. The lost frequency resolution can be recovered only by increasing the length L of the data record.

For random signals, such as sinusoids in noise, one must also deal with the *statistical reliability* of the computed spectra. In Sec. 9.6, we discuss the *periodogram averaging* and *periodogram smoothing* methods which may be used to reduce the statistical *variability* of the spectrum estimate.

The periodogram averaging method consists of dividing the total length-L data record into K contiguous segments of length N, such that $L = KN$. In order to reduce frequency leakage, a length-N non-rectangular window, such as a Hamming window, may be applied to each signal segment before its DTFT is computed.

The resulting reduction in resolution must be compensated for by increasing the length N. For a fixed total length L, this will reduce the number of segments K, thus worsening the spectrum estimate. Therefore, N must be chosen to be large enough to achieve a desired frequency resolution, but not larger. The *Welch method* is a variation of the averaging method in which the data record is divided into segments that are 50-percent overlapping—we discuss this further in Sec. 9.6.

The relationships $L = KN$ and $N = cf_s/\Delta f$ capture these tradeoffs: We want K to be large enough to get a reliable spectrum estimate, and we want N to be large enough to give us the desired resolution Δf for the particular window that we chose. Thus, together the two conditions require the total length L to be large. In some applications, this may be impossible to achieve, either because we cannot collect more data, or because beyond a certain length L, the signal will no longer remain stationary.

Parametric spectrum estimation methods, such as those based on linear prediction, maximum likelihood, and eigenvector techniques, offer the possibility of obtaining high-resolution spectrum estimates based on short data records [25,26,45].

Example 9.1.1: A signal consisting of four sinusoids of frequencies of 1, 1.5, 2.5, and 2.75 kHz is sampled at a rate of 10 kHz. What is the minimum number of samples that should be collected for the frequency spectrum to exhibit four distinct peaks at these frequencies? How many samples should be collected if they are going to be preprocessed by a Hamming window and then Fourier transformed?

Solution: The *smallest* frequency separation that must be resolved by the DFT is $\Delta f = 2.75 - 2.5 = 0.25$ kHz. Using Eq. (9.1.16) for a rectangular window, we get

$$L \geq \frac{f_s}{\Delta f} = \frac{10}{0.25} = 40 \text{ samples}$$

Because the mainlobe width of the Hamming window is twice as wide as that of the rectangular window, it follows that twice as many samples must be collected, that is, $L = 80$. This value can also be calculated from Eq. (9.1.22) with $c = 2$. □

Example 9.1.2: A 10-millisecond portion of a signal is sampled at a rate of 10 kHz. It is known that the signal consists of two sinusoids of frequencies $f_1 = 1$ kHz and $f_2 = 2$ kHz. It is also known that the signal contains a third component of frequency f_3 that lies somewhere between f_1 and f_2. (a) How close to f_1 could f_3 be in order for the spectrum of the collected samples to exhibit three distinct peaks? How close to f_2 could f_3 be? (b) What are the answers if the collected samples are windowed by a Hamming window?

Solution: The total number of samples collected is $L = f_s T_L = 10{\times}10 = 100$. The frequency resolution of the rectangular window is $\Delta f = f_s/L = 10/100 = 0.1$ kHz. Thus, the closest f_3 to f_1 and f_2 will be:

$$f_3 = f_1 + \Delta f = 1.1\ \text{kHz}, \qquad \text{and} \qquad f_3 = f_2 - \Delta f = 1.9\ \text{kHz}$$

In the Hamming case, the minimum resolvable frequency separation doubles, that is, $\Delta f = cf_s/L = 2\cdot 10/100 = 0.2$ kHz, which gives $f_3 = 1.2$ kHz or $f_3 = 1.8$ kHz. □

Example 9.1.3: The sinusoid $x(t) = \cos(2\pi f_0 t)$, where $f_0 = 50$ Hz is sampled at a rate of $f_s = 1$ kHz. The sampled signal is $x(n) = \cos(\omega_0 n)$, where $\omega_0 = 2\pi f_0/f_s = 2\pi\cdot 50/1000 = 0.1\pi$ rads/sample. A length-L portion of $x(n)$ is windowed by a rectangular and a Hamming window, that is, for $n = 0, 1, \dots, L-1$:

$$\begin{aligned} x_L(n) &= w_{\text{rec}}(n)x(n) = \cos(\omega_0 n) \\ x_L(n) &= w_{\text{ham}}(n)x(n) = \left[0.54 - 0.46\cos\left(\frac{2\pi n}{L-1}\right)\right]\cos(\omega_0 n) \end{aligned}$$

Figure 9.1.5 shows the rectangularly windowed signals, for $L = 100$ and $L = 200$. Figure 9.1.6 shows the Hamming windowed signals. Figure 9.1.7 shows the corresponding spectra, $|X_L(\omega)|$, plotted over the Nyquist subinterval, $0 \le \omega \le 0.2\pi$. The spectra were computed by successive calls to the routine `dtft` of the next section, for 200 equally spaced values of ω in the interval $0 \le \omega \le 0.2\pi$.

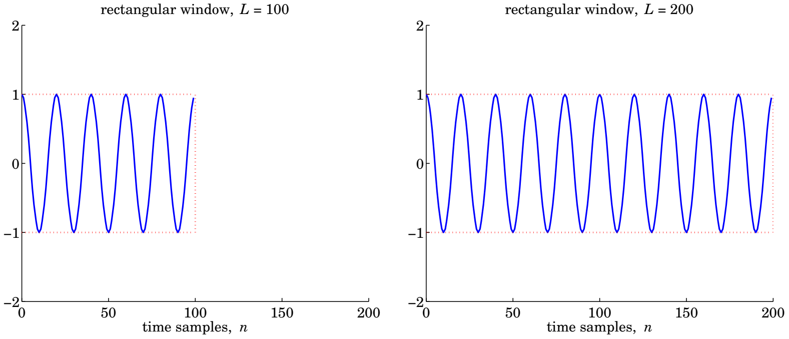


Fig. 9.1.5 Rectangularly windowed sinusoids of lengths $L = 100$ and $L = 200$.

As L doubles, both the rectangular and the Hamming mainlobe widths become narrower, with the Hamming one always lagging behind the rectangular one. Note also that as L doubles, the sidelobes of the rectangular window get more compressed, but also higher so that their *relative* depth compared to the mainlobe remains the same.

The reason why the peak height of the rectangular mainlobe is $L/2$ instead of L is that we are working with a real-valued sinusoid and looking only at its positive-frequency half-height peak. □

Example 9.1.4: The following analog signal consisting of three equal-strength sinusoids of frequencies $f_1 = 2$ kHz, $f_2 = 2.5$ kHz, and $f_3 = 3$ kHz:

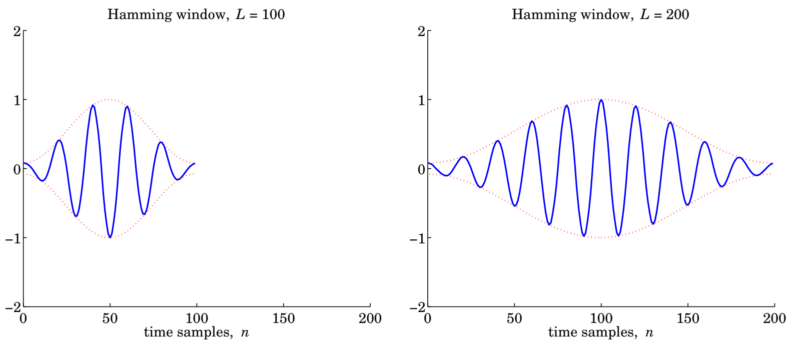


Fig. 9.1.6 Hamming windowed sinusoids of lengths $L = 100$ and $L = 200$.

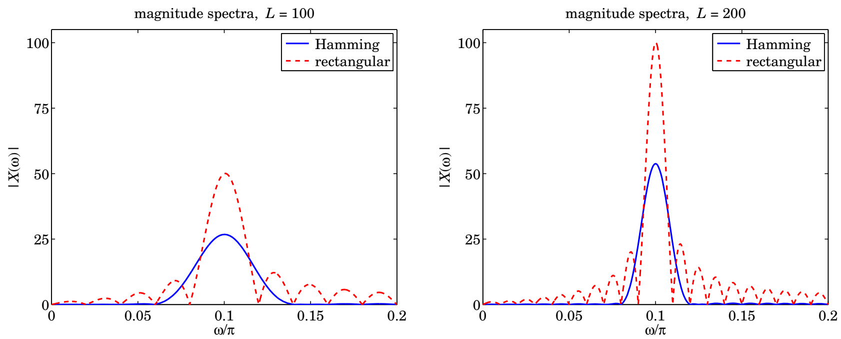


Fig. 9.1.7 Rectangular and Hamming spectra for $L = 100$ and $L = 200$.

$$x(t) = \cos(2\pi f_1 t) + \cos(2\pi f_2 t) + \cos(2\pi f_3 t)$$

where t is in milliseconds, is sampled at a rate of 10 kHz. We consider four data records of lengths $L = 10$, 20, 40, and 100 samples. They correspond to the time durations of 1, 2, 4, and 10 msec. To facilitate comparison, the same vertical scale has been used in all figures.

Figures 9.1.8 and 9.1.9 show the magnitude spectra of the rectangularly and Hamming windowed signals for the above four values of L. The spectra were computed by calling a 256-point FFT routine and plotted over an entire Nyquist interval, $0 \le f \le f_s$. For each L, the 256-point input to the FFT routine was obtained by padding $(256-L)$ zeros at the end of the L-point signal $\mathbf{x}$ to make it of length 256. (The padding operation does not affect the DTFT.)

As we will see in the next section, the three peaks in the right half of the Nyquist interval correspond to the negative-frequency peaks of the sinusoids, but they have been shifted to the right by one f_s, using the periodicity property of the spectra with respect to f_s.

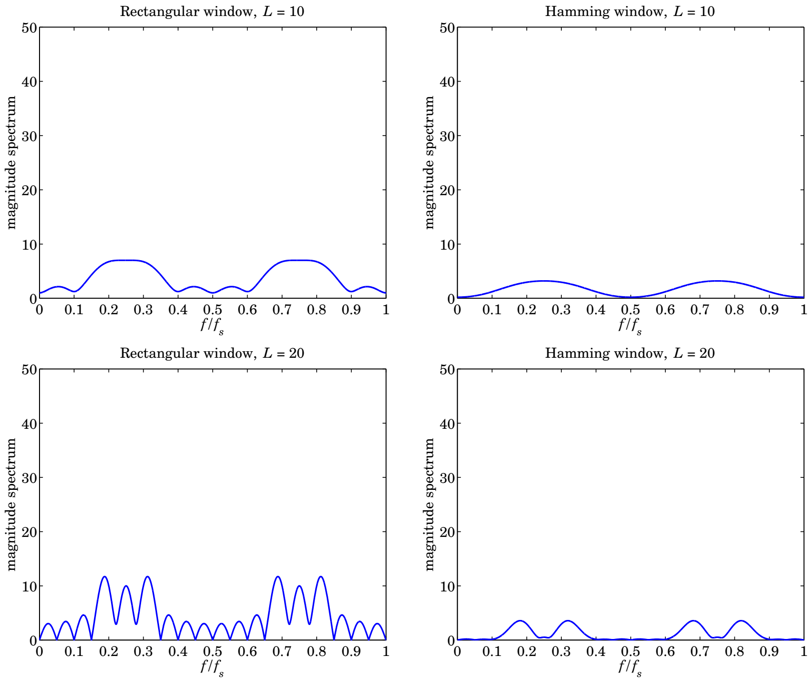


Fig. 9.1.8 Rectangular and Hamming spectra for $L = 10$ and 20.

The minimum frequency separation is $\Delta f = 2.5 - 2 = 0.5$ kHz. According to (9.1.16), the minimum length L to resolve all three sinusoids should be $L = f_s/\Delta f = 10/0.5 = 20$ samples for the rectangular window, and $L = 40$ samples for the Hamming case.

In the case $L = 10$, the signal does not have enough length to separate the sinusoids, which appear merged into one wide peak.

For $L = 20$, corresponding to the minimum acceptable length, the sinusoids begin to be separated for the rectangular window, but not yet for the Hamming window. Note also in the Hamming case, the destructive interference taking place exactly at the position of the middle sinusoid, $f_2/f_s = 0.25$.

For $L = 40$, the Hamming windowed spectra are beginning to show the separate peaks. Finally, when $L = 100$, both windows have clearly separated peaks.

The Hamming window spectra lag behind the rectangular ones in resolution, but improve with increasing L, while they provide higher sidelobe suppression. □

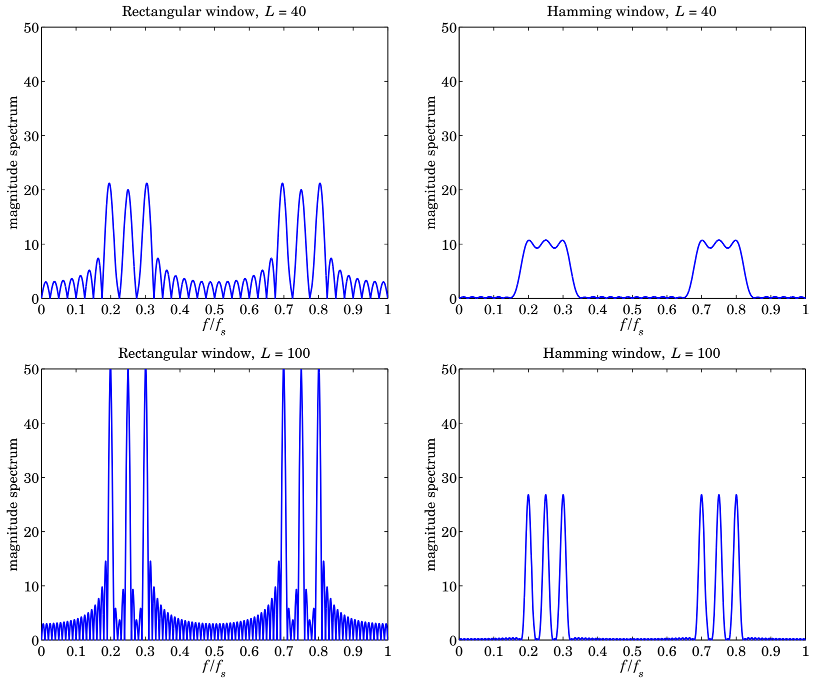


Fig. 9.1.9 Rectangular and Hamming spectra for $L = 40$ and 100.

9.2 DTFT Computation

In this section, we turn our attention to the computational aspects of the DTFT. We consider a length-L signal $x(n)$, $n = 0, 1, \dots, L-1$, which may have been prewindowed by a length-L non-rectangular window. Its DTFT, defined by Eq. (9.1.7), can be written in the simplified notation:

$$\boxed{X(\omega) = \sum_{n=0}^{L-1} x(n)e^{-j\omega n}} \qquad \text{(DTFT of length-}L\text{ signal)} \tag{9.2.1}$$

This expression may be computed at any desired value of ω in the Nyquist interval $-\pi \le \omega \le \pi$. It is customary in the context of developing computational algorithms to take advantage of the periodicity of $X(\omega)$ and map the conventional symmetric Nyquist interval $-\pi \le \omega \le \pi$ onto the right-sided one $0 \le \omega \le 2\pi$. We will refer to the latter as the *DFT Nyquist interval.* This mapping is shown in Fig. 9.2.1.

The positive-frequency subinterval $0 \le \omega \le \pi$ remains unchanged, but the negative-frequency one, $-\pi \le \omega \le 0$, gets mapped onto the second half of the DFT Nyquist interval, $\pi \le \omega \le 2\pi$.

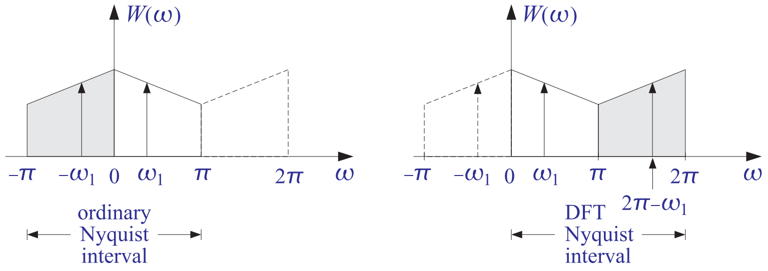


Fig. 9.2.1 Equivalent Nyquist intervals.

For example, a cosinusoidal signal $\cos(\omega_1 n)$ with two spectral peaks at $\pm\omega_1$ will be represented by the two shifted peaks:

$$\{\omega_1,\, -\omega_1\} \Leftrightarrow \{\omega_1,\, 2\pi - \omega_1\}$$

or, in Hz

$$\{f_1,\, -f_1\} \Leftrightarrow \{f_1,\, f_s - f_1\}$$

As we saw in Section 5.4, the DTFT (9.2.1) can be thought of as the evaluation of the z-transform of the sequence $x(n)$ on the unit circle:

$$X(\omega) = \sum_{n=0}^{L-1} x(n)e^{-j\omega n} = \sum_{n=0}^{L-1} x(n)z^{-n}\bigg|_{z=e^{j\omega}} = X(z)\,\big|_{z=e^{j\omega}} \tag{9.2.2}$$

Thus, $X(\omega)$ can be computed by evaluating the polynomial $X(z)$ at $z = e^{j\omega}$. Hörner's rule of synthetic division that was discussed in the problems of Chapter 2 is an efficient polynomial evaluator. It can be adapted in the following form for the evaluation of the z-transform $X(z)$:

```
for each complex z do:
    $X = 0$
    for $n = L-1$ down to $n = 0$ do:
        $X = x_n + z^{-1}X$
```

(Hörner's rule) (9.2.3)

Upon exit, X is the desired value of $X(z)$. To see how the iterations build up the z-transform, we iterate them for the case $L = 4$. Starting with $X = 0$ at $n = L - 1 = 3$, we have:

$$X = x_3 + z^{-1}X = x_3$$

$$X = x_2 + z^{-1}X = x_2 + z^{-1}x_3$$

$$X = x_1 + z^{-1}X = x_1 + z^{-1}x_2 + z^{-2}x_3$$

$$X = x_0 + z^{-1}X = x_0 + z^{-1}x_1 + z^{-2}x_2 + z^{-3}x_3 = X(z)$$

This algorithm can then be applied to any point on the unit circle $z = e^{j\omega}$ to evaluate $X(\omega)$. The C-functions, **dtft.c** and **dtftr.c**,[†] implement the computation of the DTFT at a single frequency and and at a range of frequencies., respectively.

Alternatively, one may use the built-in MATLAB function **freqz** to evaluate $X(\omega)$ at any vector of frequencies ω, with usage:

```
% ---------------------------------------------------------------------

% x = ...           % define length-L time samples
% om = ...          % define vector of digital frequencies (rads/sample)
                    % om must have dimension of at least 2

X = freqz(x,1,om);  % has the same size as the vector om

% ---------------------------------------------------------------------
```

Example 9.2.1: In Example 16.1.1, we discussed the generation of dual sinusoidal tones for DTMF touch-tone phones. Each keypress generates two frequencies ω_H and ω_L, one from the high and one from the low group of frequencies. A total of 4×4 = 16 pairs of frequencies can be generated.

Such a signal can be detected by computing its DTFT at the 4 high and 4 low group frequencies and then deciding with the help of a threshold which pair $\{X(\omega_H), X(\omega_L)\}$ of DTFT values has the largest magnitudes. The corresponding pair of frequencies $\{\omega_H, \omega_L\}$ can then be decoded into the appropriate key.

Because the DTFT is needed only at 8 positive frequencies, the use of the routine `dtft` or Goertzel's algorithm is more efficient than using an FFT. Such DTMF detectors can be implemented easily on present-day DSP chips [107–109].

The minimum duration L of the received dual tone may be estimated by requiring that the high and low groups of frequencies remain distinct, so that the DTFT will consist of one peak lying in the high group and one in the low group.

The resolvability condition depends on the minimum frequency difference *between* the groups, that is, from Fig. 16.1.3 we have

$$\Delta f = f_{H,\min} - f_{L,\max} = 1209 - 941 = 268 \text{ Hz}$$

which at sampling rate of $f_s = 8$ kHz and rectangular windowing gives the minimum length $L = f_s/\Delta f = 8000/268 \simeq 30$ samples. □

9.3 Window Parameters

We saw above that the effect of windowing a signal consisting of a sum of sinusoidal components is to broaden the (ideally infinitely narrow) spectral peaks at the sinusoid frequencies, as well as to introduce sidelobes, referred to as *leakage* because the energy of each peak spreads or leaks into its neighboring sidelobes.

If the broadened peaks are too close to each other, they will appear merged, resulting in loss of resolution. Similarly, if the amplitudes of the sinusoids are too weak, they will

[†] included in the C-function collection of this book

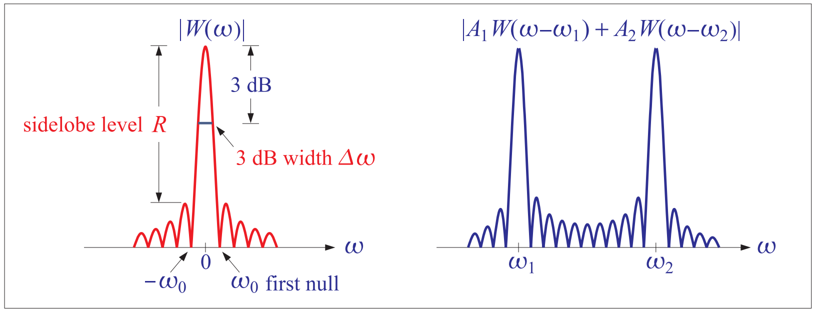
be lost under the sea of sidelobes of the stronger nearby sinusoids and their presence may not be detectable.

The choice of window determines the width $\Delta\omega$ of the mainlobe and the depth R of the sidelobes. We mentioned that the mainlobe width of a length-L window is broadened relative to the rectangular window by a factor c, such that

$$\Delta\omega_w = c\,\frac{2\pi}{L}$$

But, we were a bit vague as to the precise definition of the parameter c. In fact, different books define the mainlobe width differently: some choose it to be the width of the base of the main lobe (i.e., the null-to-null width), some choose it to be half of that, and some choose it to be the 3-dB width. In general, the mainlobe width depends on the sidelobe level—the deeper the sidelobes, the wider the mainlobe.

Here we look at the properties of several useful windows and, in order to put their discussion on a common footing, we will choose the mainlobe width to be the 3-dB width, developing approximate formulas for it in terms of the sidelobe level R. The figure below illustrates these parameters for a typical window.



For example, consider a signal consisting of two sinusoids of frequencies ω_1,ω_2 and amplitudes A_1,A_2. If the signal is windowed by a length L window $w(n)$ with DTFT $W(\omega)$, then, the two spectral lines at ω_1,ω_2 get smeared and replaced by the shifted versions of $W(\omega)$,

$$\begin{aligned} x(n) &= A_1 e^{j\omega_1 n}w(n) + A_2 e^{j\omega_2 n}w(n)\,, \quad n = 0,1,\dots,L-1 \\ X(\omega) &= A_1 W(\omega-\omega_1) + A_2 W(\omega-\omega_2) \end{aligned} \tag{9.3.1}$$

The mainlobe width could be defined to be the null-to-null width, $\Delta\omega = 2\omega_0$, as is done in some texts, or, alternatively, $\Delta\omega$ can be chosen to be the 3-dB width.

The so-called *Rayleigh resolvability criterion* states that the two peaks will be resolvable if they are separated by more than the frequency ω_0 of the first null, that is, if, $\omega_2-\omega_1 \ge \omega_0$. Since the value of ω_0 is roughly equal to the 3-dB width, an alternative resolvability criterion is to require that,

$$\boxed{\omega_2-\omega_1 \ge \Delta\omega_{3\text{dB}}} \quad \text{(frequency resolution condition)} \tag{9.3.2}$$

But resolvability is only half of the story. Even if condition (9.3.2) is satisfied, it may not be possible to detect the presence of both sinusoids if one of them is too weak to stand above the sidelobe levels of the stronger one.

Moreover, because of the interaction between mainlobes, the frequencies estimated from the maxima of the computed spectrum will be slightly different from the true frequencies of the sinusoids—an effect referred to as *biasing*. For example, it is evident from Eq. (9.3.1) that the local maximum of $|X(\omega)| = |A_1W(\omega-\omega_1)+A_2W(\omega-\omega_2)|$ near ω_1 will not necessarily be at precisely ω_1, i.e., coinciding with the maximum of $|W(\omega-\omega_1)|$, but it will be close to it provided the mainlobes are very narrow and their separation large enough.

There are literally dozens of windows—see Doerry's recent review [227] of 50+ windows— but among them there are some, such as the Kaiser, DPSS (prolate of Slepian), and Chebyshev windows, that have a user-defined sidelobe level R (specified as an attenuation in dB relative to the mainlobe peak) that can be adjusted to bring out the presence of weak sinusoids.

The price one pays for making $\Delta\omega_{3\text{dB}}$ narrower and/or making R deeper is to require an increased length L of the data record. The tradeoff between window length and 3-dB width (defined as the full width at half-power) is captured by the approximate relationship,

$$\boxed{\Delta\omega_{3\text{dB}} = 0.886\,\frac{2\pi b}{L}} \quad \Rightarrow \quad \Delta f_{3\text{dB}} = 0.886\,\frac{f_s b}{L} \tag{9.3.3}$$

where b is referred to as a "broadening factor" which depends on the window and, in the Kaiser, DPSS, and Chebyshev cases, on the sidelobe level R. For example, the rectangular window has the *narrowest* mainlobe with $b = 1$ and fixed $R = 13.26$ dB, the Hamming window has approximately $b = 1.47 + 0.97/L$ and fixed $R = 42.67$ dB, while the Kaiser window has a variable R and b,

$$b = 0.0124R + 1.0221 \tag{9.3.4}$$

which works well over the range, $20 \le R \le 120$ dB, with a 2-percent error. An even better approximation with a 0.4-percent error is,

$$b = -0.000051R^2 + 0.019549R + 0.813943\,, \quad 20 \le R \le 120 \text{ dB} \tag{9.3.5}$$

The simpler expression (9.3.4) is usually adequate, and it can also be used for the DPSS prolate window. For an odd window length, $L = 2M+1$, the Hamming and Kaiser windows are given by,

$$\begin{aligned} w(n) &= 0.54 - 0.46\cos\left(\frac{\pi n}{M}\right), \quad n = 0,1,\dots,2M\,, \quad M = \frac{L-1}{2} \\ w(n) &= \frac{I_0\left(\alpha\sqrt{1-(n-M)^2/M^2}\right)}{I_0(\alpha)}\,, \quad n = 0,1,\dots,2M \end{aligned} \tag{9.3.6}$$

The Kaiser window shape parameter α (denoted by β in MATLAB), can be calculated in terms of R by the Kaiser-Schafer approximation [224], where R is in dB,[†]

[†] A different approximation is used in the FIR filter design problem, discussed in Chap. 11.

$$\alpha = \begin{cases} 0, & R \le 13.26 \\ 0.76609\,(R-13.26)^{0.4} + 0.09834\,(R-13.26), & 13.26 < R \le 60 \\ 0.12438\,(R+6.3), & 60 < R \le 120 \end{cases} \tag{9.3.7}$$

Thus, in the Kaiser case, given a desired 3-dB width $\Delta\omega_{\text{3dB}}$ and sidelobe level R, equations (9.3.3)–(9.3.7) can be used to determine the length L of the window and the parameter α required for the calculation of the window samples $w(n)$. The window samples $w(n)$ can be computed with the built-in MATLAB function **kaiser**,

```
w = kaiser(L,alpha);            % Lx1 column vector
```

We note also that Kaiser & Schafer [224] have also derived the following approximation for the width of the mainlobe base, i.e., the null-to-null width, with R in dB,

$$\Delta\omega_{\text{null}} = \frac{2\pi c}{L-1} \quad \Rightarrow \quad \Delta f_{\text{null}} = \frac{f_s c}{L-1}, \qquad c = \frac{12(R+12)}{155} \tag{9.3.8}$$

The DPSS prolate or Slepian window is very similar to the Kaiser one, but is slightly better in that it provides a slightly narrower mainlobe for the same sidelobe level, which can be calculated by finding L, α from the same Eqs. (9.3.3), (9.3.4), and (9.3.7), and then, the window samples are obtained by invoking MATLAB's built-in "discrete-prolate-spheroidal-sequences" function, **dpss**, in the form,

```
B = alpha/pi;                   % time-bandwidth product
w = dpss(L, B, 1);              % window samples w(n), Lx1 vector
w = w/max(w);                   % normalize to unity at its middle
```

and a slightly better approximation is,

```
B = 0.95*alpha/pi + 0.14;           % time-bandwidth product
w = dpss(L, B, 1);                  % window samples w(n), Lx1 vector
w = w/max(w);                       % normalize to unity at its middle
```

This window maximizes the power contained in the mainlobe. More specifically, for a length-L window, $w(n)$, $n = 0, 1, \dots, L-1$, the time-bandwidth parameter B defines the mainlobe band, $[-\omega_c, \omega_c]$, via the relationship,

$$B = L \cdot \frac{\omega_c}{2\pi} \quad \Rightarrow \quad \omega_c = \frac{2\pi B}{L} \tag{9.3.9}$$

The window $w(n)$ is chosen to maximize the power within the mainlobe band, that is,

$$J = \frac{\dfrac{1}{2\pi}\displaystyle\int_{-\omega_c}^{\omega_c} |W(\omega)|^2 d\omega}{\dfrac{1}{2\pi}\displaystyle\int_{-\pi}^{\pi} |W(\omega)|^2 d\omega} = \max \tag{9.3.10}$$

where $W(\omega)$ is the DTFT of the window,

$$W(\omega) = \sum_{n=0}^{L-1} w(n)\, e^{-j\omega n}$$

See also EWA/Ch.23 [46] for an alternative calculation of $w(n)$ that implements this maximization procedure using an inverse-power iteration.

The Chebyshev window also has an adjustable sidelobe level R. It was originally proposed by Dolph for the design of narrow-beam arrays, and sometimes is called the Dolph-Chebyshev window. It is optimal in the sense that it provides the narrowest mainlobe for the given sidelobe level R, and as a consequence, its sidelobes are all equal to R. Its design is fully explained in EWA/Ch.22 [46]. Alternatively, one may use MATLAB's built in function **chebwin** to calculate its time samples, given its length L, and sidelobe level R in dB:

```
wc = chebwin(L,R);          % built-in function, Lx1 vector
wc = wc/max(wc);            % normalize to unity at its middle
```

Its broadening factor can be computed by the following approximation, which is accurate over the range, $13 \le R \le 180$ dB:

$$b = 0.65 + 0.0195R - 0.00005R^2 \tag{9.3.11}$$

In summary, we list the 3-dB broadening factors for the above windows, as well as their sidelobe attenuations R in dB:

window	3-dB broadening factor	sidelobe attenuation
Rectangular	$b = 1$	13.26
Hamming	$b = 1.47 + 0.97/L$	42.67
Kaiser	$b = 0.0124R + 1.0221$	variable R
DPSS	$b = 0.0124R + 1.0221$	variable R
Chebyshev	$b = 0.65 + 0.0195R - 0.00005R^2$	variable R

Example 9.3.1: The following graphs compare these windows applied to a single complex sinusoid of frequency $f_1 = 1.5$ kHz and sampled at $f_s = 10$ kHz,

$$x(n) = e^{j\omega_1 n}, \quad \omega_1 = \frac{2\pi f_1}{f_s}, \quad n = 0, 1, \dots, L-1$$

The length L was chosen to meet the design requirements of the Kaiser window that achieves a 3-dB width of $\Delta f = 0.1$ kHz and sidelobe attenuation of $R = 60$ dB. The following MATLAB code segment illustrates the calculation of the length L from the equations,

$$\Delta f = 0.886\,\frac{f_s b}{L}, \quad b = 0.0124R + 1.0221$$

```
fs = 10;  f1 = 1.5;  w1=2*pi*f1/fs;

R = 60; Df = 0.1;           % sidelobe attenuation and 3-dB width

b = 0.0124 * R + 1.0221;    % broadening factor
L = 0.886*b*fs/Df;          % solve Df = 0.886 * 2*pi*b/L for L
M = ceil((L-1)/2);          % round up to the next odd integer
L = 2*M+1;                  % resulting value, L = 157

n=0:L-1;                    % construct sinusoid
x = exp(j*w1*n);
```

For each window, the graphs plot the spectrum of the windowed sinusoid in dB with all spectra normalized to unity gain at $f = f_1$, that is,

$$X(f) = \sum_{n=0}^{L-1} w(n)\,x(n)\,e^{-2\pi j f n/f_s} = \text{DTFT of windowed signal}$$

$$X_{\text{norm}}(f) = \frac{X(f)}{|X(f_1)|} = \text{normalized spectrum}$$

$$20\log_{10}|X_{\text{norm}}(f)| = \text{spectrum in dB}$$

In addition, the window time functions, $w(n)$, $n = 0, 1, \dots, L-1$, are also compared. We note the similarity of the Kaiser and DPSS windows, with the DPSS being slightly wider in the time domain (having slightly narrower mainlobe in the frequency domain and slightly higher outer sidelobes). Visible also is the typical “wiggly” behavior of the Chebyshev window at its end points where the ending pedestals turn slightly upwards.

In the Chebyshev case for real-valued sinusoids, or for multiple sinusoids at frequencies ω_i, the interaction between the shifted windowed spectra $W(\omega \pm \omega_i)$ will cause some distortion of the equiripple nature of the sidelobes and they may no longer be equal at the designed value of R. In such cases, increasing R slightly would move the sidelobes to the desired level.

As an example illustrating such distortion, the following two graphs show the spectra of the real-valued version of the above windowed complex sinusoid, that is, $x(n) = \cos(\omega_1 n)$, $n = 0, 1, \dots, L-1$. Here, the interaction is between the positive and negative frequency terms, plotted over the positive frequency side, as well as over the positive and negative sides. Increasing the sidelobe design level to $R = 65$ brings all the sidelobes to the desired 60-dB level.

$$x(n) = w(n)\cos(\omega_1 n) \quad\Rightarrow\quad X(\omega) = \frac{1}{2}\big[W(\omega-\omega_1) + W(\omega+\omega_1)\big]$$

Example 9.3.2: Consider the following analog signal consisting of three sinusoids:

$$x(t) = \cos(2\pi f_1 t) + 10^{-3}\cos(2\pi f_2 t) + \cos(2\pi f_3 t)$$

where $f_1 = 1.5$, $f_2 = 2.5$, and $f_3 = 3.5$ kHz, and t is in msec. The middle term represents a weak sinusoid whose presence we wish to detect by sampling $x(t)$ and computing its DTFT spectrum. The sampling rate is $f_s = 10$ kHz.

We will compare the use of the rectangular, Hamming, and Kaiser, DPSS, and Chebyshev windows for this problem. Because the middle sinusoid is 60 dB below the other two, in order to detect its presence, we must use a window that has sidelobes that are suppressed by at least 60 dB.

To get some extra margin, we take the relative sidelobe level to be 10 dB deeper than required, that is, $R = 60 + 10 = 70$ dB and choose the Kaiser 3-dB width to be $\Delta f = 0.1$ kHz (i.e., one-tenth the minimum frequency separation of $f_2 - f_1 = 2.5 - 1.5 = 1$ kHz.) The length L of a Kaiser window with the above values of R and Δf is determined to be $L = 169$, from the MATLAB code segment,

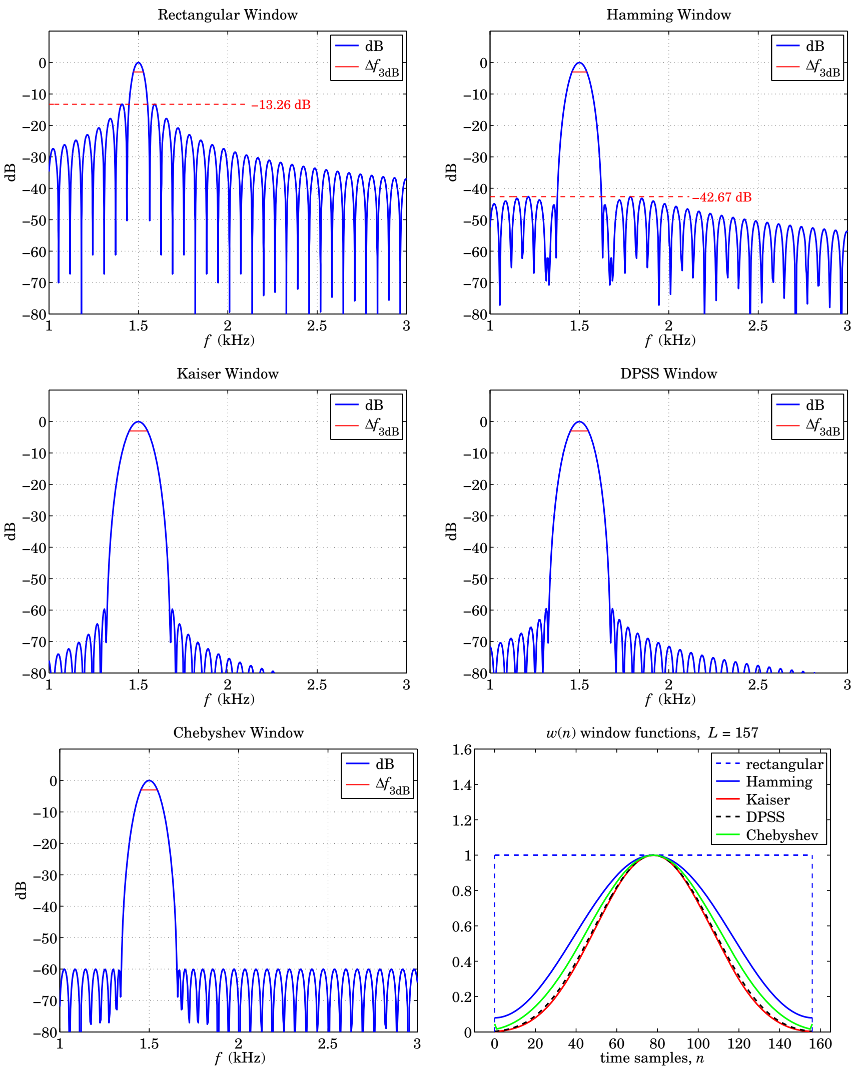


Fig. 9.3.1 Rectangular, Hamming, Kaiser, DPSS, and Chebyshev windows for $L = 157$.

```
fs = 10;
f1 = 1.5;  w1=2*pi*f1/fs;
f2 = 2.5;  w2=2*pi*f2/fs;
f3 = 3.5;  w3=2*pi*f3/fs;

A2 = 10^(-3);             % relative amplitude of middle sinusoid
```

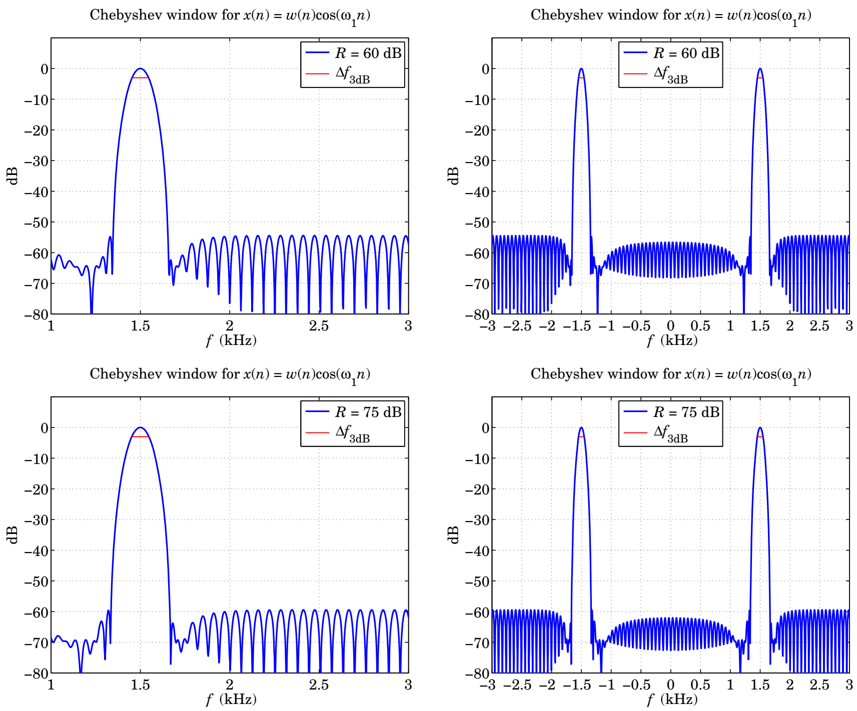


Fig. 9.3.2 Real-valued sinusoid case, designed with $L = 157$ and $R = 60$ and $R = 75$ dB.

```
R = 70; Df = 0.1;           % sidelobe attenuation and 3-dB width

b = 0.0124 * R + 1.0221;    % broadening factor
L = 0.886*b*fs/Df;          % solve Df = 0.886 * 2*pi*b/L for L
M = ceil((L-1)/2);          % round up to the next odd integer
L = 2*M+1:                  % resulting value, L = 169

n=0:L-1;                    % construct signal x(n)

x = cos(w1*n) + A2*cos(w2*n) + cos(w3*n);
```

The graphs in Fig. 9.3.3, compare the spectra of the windowed signal for the five window cases. The graphs also show the calculated 3-dB widths of one of the mainlobes.

We observe how the Kaiser, DPSS, and Chebyshev windows allow the weak sinusoid to rise above the sidelobes, whereas in the rectangular and Hamming window cases the weak sinusoid is buried under the sidelobes of the two larger mainlobes, even though all mainlobe widths are much narrower.

The DPSS mainlobes are slightly narrower than those of the Kaiser case, but they also have slightly higher far-sidelobes, i.e., the sidelobes of the DPSS window decay more slowly.

The required DTFT calculations can be done with the help of the **freqz** function, with MATLAB code similar to the following,

```
% w = window                      % define window, same size as x

f = linspace(0,fs/2,2001);        % S will have same size as f
om = 2*pi*f/fs;                   % frequencies in rads/sample

S = abs(freqz(w.*x,1,om)).^2;     % magnitude square of DTFT
S = S/max(S);                     % normalize to unity maximum
SdB = 20*log10(S);                % spectrum in dB

plot(f,Sdb);
```

As in the previous example, because of the interaction of the various sinusoidal terms, the resulting sidelobe level of the Chebyshev case is not quite at the desired design level of $R = 70$. The bottom right graph in Fig. 9.3.3 show a redesign with $R = 80$ and $L = 169$, which brings all the sidelobes dow to the desired 70-dB level.

Example 9.3.3: Consider next the following analog signal consisting of three sinusoids:

$$x(t) = \cos(2\pi f_1 t) + 10^{-3}\cos(2\pi f_2 t) + \cos(2\pi f_3 t)$$

where $f_1 = 1.5$, $f_2 = 1.6$, and $f_3 = 3.5$ kHz, and t is in msec. The middle term represents a weak sinusoid whose presence we wish to detect by sampling $x(t)$ and computing its DTFT spectrum. The sampling rate is $f_s = 10$ kHz.

If we were to design our windows exactly as in the previous example, that is, with Kaiser 3-dB width $\Delta f = 0.1$ kHz and $R = 70$ dB, with length $L = 169$, then the computed spectra would not be able to resolve the f_2 sinusoid because it is very close the first one. This is illustrated in Fig. 9.3.4 where for all windows, the f_2 mainlobe is buried under the influence of the first sinusoid.

In order to bring out the weak f_2 sinusoid, we must choose a longer length L. For example, if we redesign our Kaiser window based on the narrower width of $\Delta f = 0.02$ kHz, the resulting length would be $L = 839$. Fig. 9.3.5 shows the spectra in this case and the weak mainlobe of the f_2 sinusoid is now visible for the Kaiser, DPSS, and Chebyshev windows.

For this length L, the null-to-null width of the Kaiser window turns out to be just less than the frequency separation that needs to be resolved, $f_2 - f_1 = 0.1$ kHz. Indeed, we find from Eq. (9.3.8),

$$\Delta f_{\text{null}} = \frac{f_s}{L-1}\cdot\frac{12(R+12)}{155} = 0.076\text{ kHz}$$

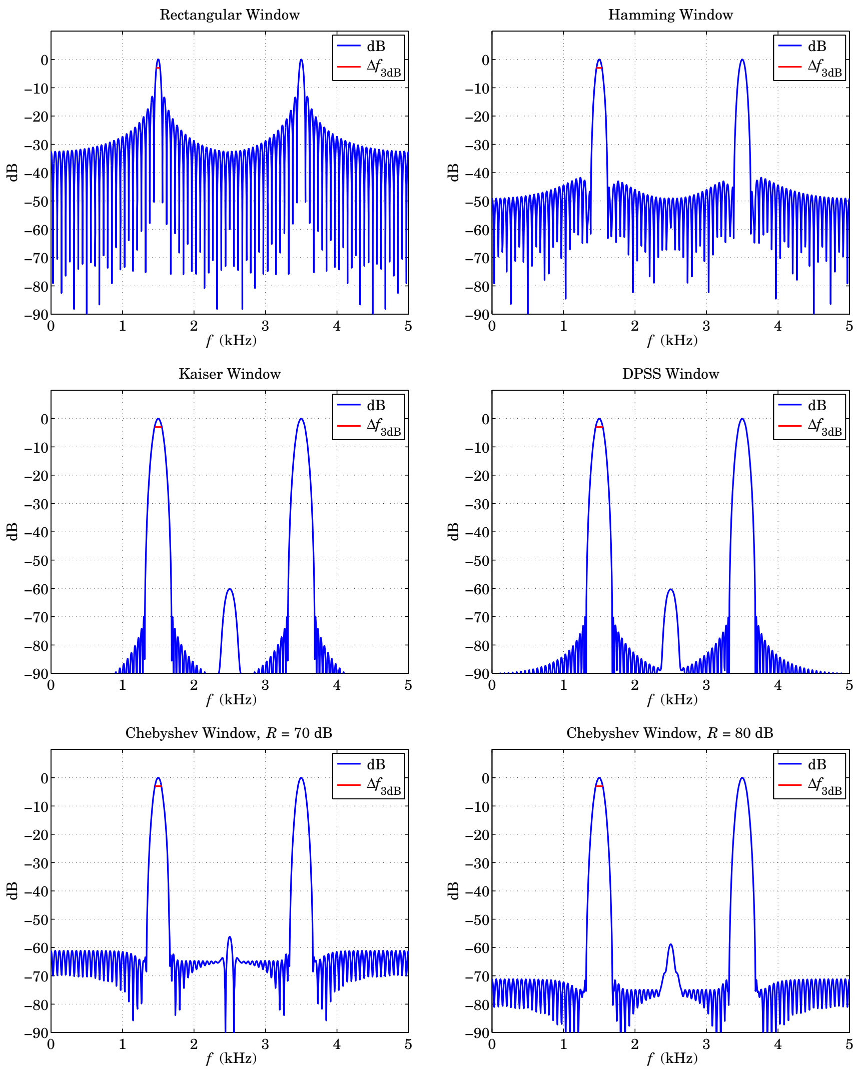
Because of the very weak amplitude of the f_2 sinusoid, the null-to-null width might be a more suitable design criterion in this case than the 3-dB width.

9.4 Additional Details on Windows

9.4.1 Rectangular Window

We saw earlier that the 3-dB width of a length-L rectangular window is given by,

$$\Delta\omega_{3\text{dB}} = 0.886\,\frac{2\pi}{L}$$


Rectangular Window

Hamming Window

Kaiser Window

DPSS Window

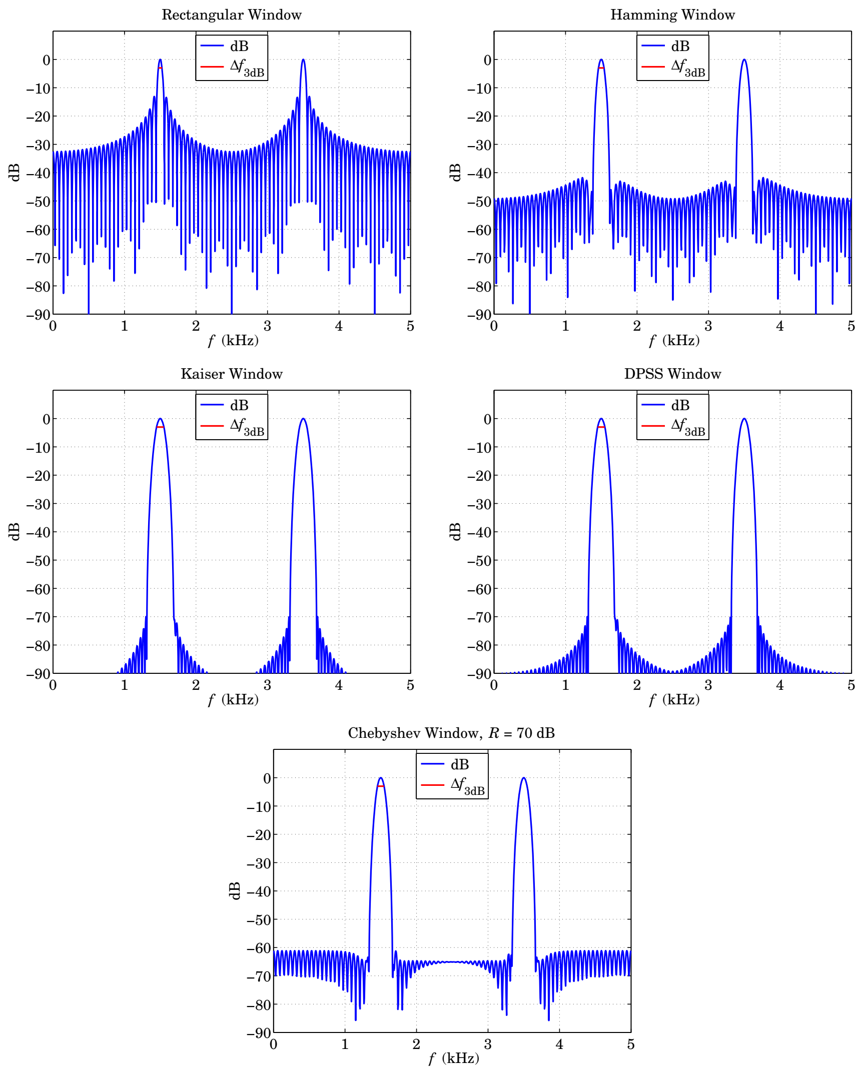
Chebyshev Window, $R = 70$ dB

Chebyshev Window, $R = 80$ dB

Fig. 9.3.3 Rectangular, Hamming, Kaiser, DPSS, and Chebyshev windows for $L = 169$.

This can be derived by considering the DTFT of the length-L rectangular window,

$$\begin{aligned} W(\omega) &= \frac{1-e^{-jL\omega}}{1-e^{-j\omega}} = e^{-j\omega(L-1)/2}\,\frac{\sin(\omega L/2)}{\sin(\omega/2)} \\ &= e^{-j\pi x(L-1)/L}\,\frac{\sin(\pi x)}{\sin(\pi x/L)}\,, \qquad \text{where } \omega = \frac{2\pi x}{L} \end{aligned} \tag{9.4.1}$$


Rectangular Window

Hamming Window

Kaiser Window

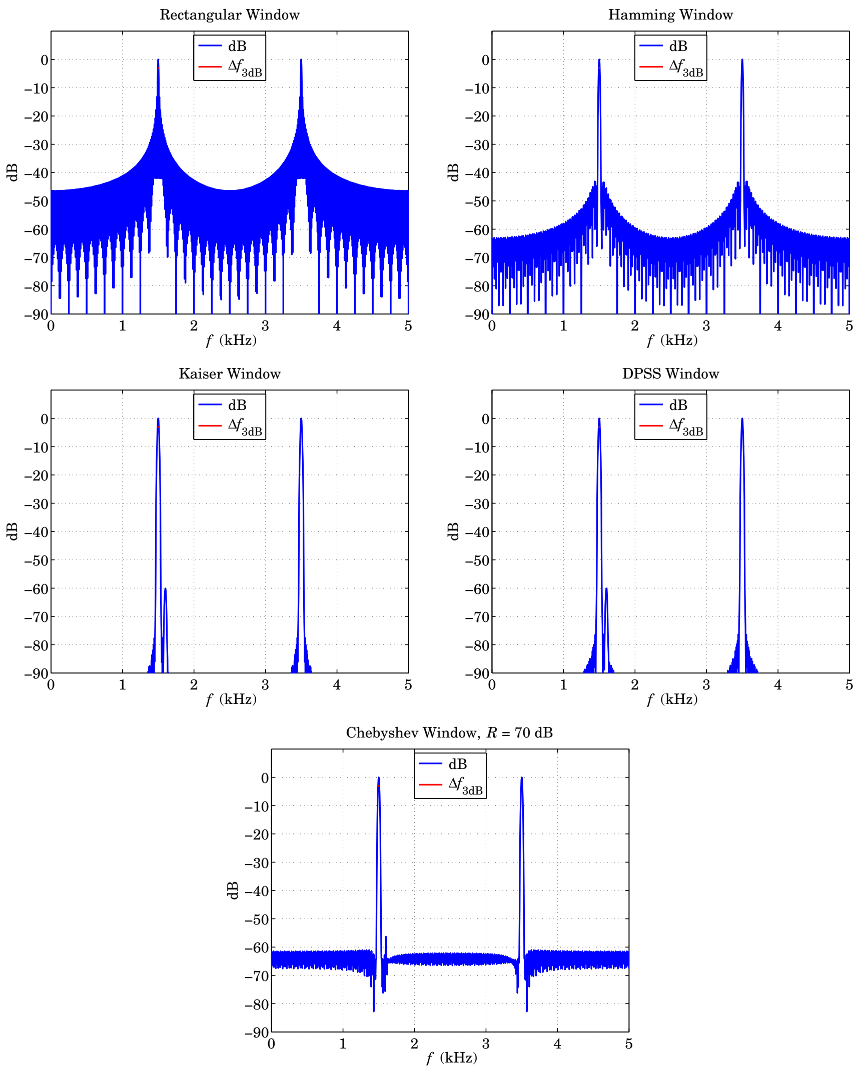
DPSS Window

Chebyshev Window, $R = 70$ dB

Fig. 9.3.4 Windows designed with $R = 70$ dB and $L = 169$.

and its magnitude, normalized to unity gain at DC,

$$\left|\frac{W(\omega)}{W(0)}\right| = \left|\frac{\sin(\omega L/2)}{L\sin(\omega/2)}\right| = \left|\frac{\sin(\pi x)}{L\sin(\pi x/L)}\right|, \quad \omega = \frac{2\pi x}{L}$$


Rectangular Window

Hamming Window

Kaiser Window

DPSS Window

Chebyshev Window, R = 70 dB

Fig. 9.3.5 Windows designed with $R = 70$ dB and $L = 839$.

For $L > 10$, this can be approximated well by a sinc-function,

$$F(x) = \frac{\sin(\pi x)}{L\sin(\pi x/L)} \approx \frac{\sin(\pi x)}{L\cdot\pi x/L} = \frac{\sin(\pi x)}{\pi x} = \operatorname{sinc}(x)$$

The 3-dB frequency is obtained by solving the half-power condition,

$$\left|\frac{W(\omega)}{W(0)}\right|^2 = \frac{1}{2} \quad\Rightarrow\quad F^2(x) = \frac{1}{2} \quad\Rightarrow\quad |F(x)| = \frac{1}{\sqrt{2}}$$

which can be implemented easily using MATLAB's **fzero** function,

```
% search near the initial value, x0 = 0.4

x3 = fzero(@(x) abs(sinc(x))-1/sqrt(2), 0.4);

% with resulting solution, x3 = 0.4429, rounded to, x3 = 0.443
```

so that the 3-dB frequency and 3-dB width are,

$$\omega_3 = \frac{2\pi x_3}{L} = 0.443\,\frac{2\pi}{L} \quad\Rightarrow\quad \Delta\omega_{3\text{dB}} = 2\omega_3 = 0.886\,\frac{2\pi}{L}$$

The highest sidelobe and the frequency at which it occurs can be found by the following MATLAB code using the minimization function **fminbnd**, searching within the interval, $1 \le x \le 2$, because as we noted earlier, the sidelobe frequency is approximately near $\omega = 2\pi x/L$, with $x \approx 1.5$,

```
xside = fminbnd(@(x) -abs(sinc(x)), 1,2);    % max of |F(x)| is the min of -|F(x)|

% resulting in, xside = 1.4303
% then, evaluate F(xside) in dB

Rside  = -20*log10(abs(sinc(xside)));

% Rside = 13.2615 dB
```

so that,

$$\omega_{\text{side}} = \frac{2\pi\,1.4303}{L}\,, \qquad -20\log_{10}\left|\frac{W(\omega_{\text{side}})}{W(0)}\right| = 13.2615\text{ dB}$$

9.4.2 Hamming Window

For the Hamming window, we may express its DTFT in terms of the DTFT of the rectangular window (9.4.1) using the modulation property of DTFTs,

$$\begin{aligned}
w(n) &= u(n) - u(n-L) = \text{length-}L\text{ rectangular window}\\
w_H(n) &= 0.54\,w(n) - 0.46\cos\left(\frac{2\pi n}{L-1}\right)w(n)\\
&= 0.54\,w(n) - 0.23\,e^{2\pi jn/(L-1)}w(n) - 0.23\,e^{-2\pi jn/(L-1)}w(n)\\
W_H(\omega) &= 0.54\,W(\omega) - 0.23\,W\left(\omega - \frac{2\pi}{L-1}\right) - 0.23\,W\left(\omega + \frac{2\pi}{L-1}\right)
\end{aligned}$$

or, using the x-variable and the sinc approximation,

$$\begin{aligned}
W(x) &= e^{-j\pi x(L-1)/L}\,\frac{\sin(\pi x)}{\sin(\pi x/L)} \approx e^{-j\pi x(L-1)/L}\,L\,\frac{\sin(\pi x)}{\pi x}\,, \qquad \omega = \frac{2\pi x}{L}\\
W_H(x) &= 0.54\,W(x) - 0.23\,W\left(x - \frac{L}{L-1}\right) - 0.23\,W\left(x + \frac{L}{L-1}\right)
\end{aligned}$$

The exact DC value, $W_H(0)$, can be expressed in the simple closed form,

$$W_H(0) = 0.54L - 0.46 \tag{9.4.2}$$

By noting that the maximum sidelobe always occurs in the range, $4 \le x \le 5$, and finding the local maximum of the ratio, $|W_H(x)/W_H(0)|$, within this x-range, and carrying out a data fit for several values of L,[†] we obtain the following approximations to the maximum sidelobe level in dB and the frequency at which it occurs,

$$\begin{aligned} \omega_{\text{side}} &= \frac{2\pi x_{\text{side}}}{L}\,, \quad x_{\text{side}} = 4.5 - \frac{2.1}{L} + \frac{2.2}{L^2} \\ R_{\text{side}} &= -20\log_{10}\left|\frac{W_H(\omega_{\text{side}})}{W_H(0)}\right| = -42.67 + \frac{1.5}{L} + \frac{513}{L^2} \end{aligned} \tag{9.4.3}$$

where both are valid for $L > 10$. For $L > 100$, we may simply use, $x_{\text{side}} = 4.5$ and $R_{\text{side}} = -42.67$. In a similar fashion, we may determine an approximate expression for the half-power 3-dB width, by solving the following condition for several values of L,

$$\left|\frac{W_H(\omega_{3\text{dB}})}{W_H(0)}\right|^2 = \frac{1}{2} \tag{9.4.4}$$

from which we obtain the approximate broadening factor for the full width $\Delta\omega_{3\text{dB}}$, valid again for $L > 10$,

$$\Delta\omega_{3\text{dB}} = 2\omega_{3\text{dB}} = 0.886\,\frac{2\pi b}{L}\,, \quad b = 1.47 + \frac{0.97}{L} \tag{9.4.5}$$

9.4.3 Kaiser Window

The symmetric Kaiser window is based on the following Fourier transform pair in continuous time,

$$2\tau_0\,\frac{\sinh\left[\sqrt{\tau_0^2\Omega_0^2 - \tau_0^2\Omega^2}\right]}{\sqrt{\tau_0^2\Omega_0^2 - \tau_0^2\Omega^2}} = \int_{-\tau_0}^{\tau_0} I_0\left[\tau_0\Omega_0\sqrt{1 - t^2/\tau_0^2}\right] e^{j\Omega t}\,dt$$

defining the time-bandwidth product, $\alpha = \tau_0\Omega_0$, and sampling at some rate $f_s = 1/T_s$, assuming an integer number of samples within τ_0,

$$\begin{aligned} \tau_0 &= MT_s \\ t &= kT_s\,, \quad -M \le k \le M \\ \omega &= \Omega T_s = \text{digital frequency} \\ \tau_0\Omega &= MT_s\Omega = M\omega \end{aligned}$$

then, the sampled and normalized (to unity at its middle) symmetric window becomes,

$$w(k) = \frac{I_0\left[\alpha\sqrt{1 - k^2/M^2}\right]}{I_0(\alpha)}\,, \quad -M \le k \le M$$

[†] specifically, we used the range, $10 \le L \le 2000$

From the sampling theorem, the DTFT of $w(k)$ will be the above continuous-time transform scaled by T_s, with all its replicas at multiples of f_s added. For small enough T_s, the replicas can be ignored approximately, resulting in the scaled and normalized spectrum,

$$W(\omega) = 2M\,\frac{\sinh\left[\sqrt{\alpha^2 - M^2\omega^2}\right]}{I_0(\alpha)\sqrt{\alpha^2 - M^2\omega^2}}\,, \quad \omega = \frac{2\pi f}{f_s} = \Omega T_s$$

$$\frac{W(\omega)}{W(0)} = \frac{\alpha}{\sinh(\alpha)}\cdot\frac{\sinh\left[\sqrt{\alpha^2 - M^2\omega^2}\right]}{\sqrt{\alpha^2 - M^2\omega^2}}$$

The causal version of the window is obtained by delaying $w(k)$ by M samples,

$$w(n) = \frac{I_0\left[\alpha\sqrt{1 - (n-M)^2/M^2}\right]}{I_0(\alpha)}\,, \quad n = 0, 1, \dots, 2M$$

The 3-dB width and sidelobe levels can be obtained from,

$$\frac{W(\omega)}{W(0)} = \frac{\alpha}{\sinh(\alpha)}\cdot\frac{\sinh\left[\sqrt{\alpha^2 - M^2\omega^2}\right]}{\sqrt{\alpha^2 - M^2\omega^2}}$$

The resulting broadening factor and the exact relationship between α and R are (see EWA/Ch.20 [46] for computational details),

$$\boxed{\begin{aligned} b &= 0.0124R + 1.0221 \\ R &= 13.26 + 20\log_{10}\left[\frac{\sinh(\alpha)}{\alpha}\right] \end{aligned}}$$

with R in dB, and the exact relationship between R and α can be replaced by the Kaiser-Schafer approximation:

$$\alpha = \begin{cases} 0, & R \le 13.26 \\ 0.76609\,(R - 13.26)^{0.4} + 0.09834\,(R - 13.26)\,, & 13.26 < R \le 60 \\ 0.12438\,(R + 6.3)\,, & 60 < R \le 120 \end{cases}$$

9.4.4 DPSS Window

The prolate window maximizes the power that resides within the mainlobe of the window. Assuming the mainlobe is within the range of digital frequencies $[-\omega_c, \omega_c]$, with ω_c to be determined from α and L, the minimization criterion is as mentioned in Eq. (9.3.10),

$$J = \frac{\dfrac{1}{2\pi}\displaystyle\int_{-\omega_c}^{\omega_c} |W(\omega)|^2\,d\omega}{\dfrac{1}{2\pi}\displaystyle\int_{-\pi}^{\pi} |W(\omega)|^2\,d\omega} = \max$$

where $W(\omega)$ is the DTFT of the window w_n,

$$W(\omega)=\sum_{n=0}^{L-1} w_n e^{-j\omega n}\,, \qquad \mathbf{w}=\begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{L-1}\end{bmatrix}$$

The performance index J can be expressed directly in terms of the window time samples $\mathbf{w}$ as a Rayleigh quotient,

$$J=\frac{\mathbf{w}^T A\mathbf{w}}{\mathbf{w}^T\mathbf{w}}=\max$$

where A is the $L\times L$ "prolate matrix" defined in terms of its matrix elements,

$$A_{nm}=\frac{\sin\big(2\pi W_c(n-m)\big)}{\pi(n-m)}\,, \quad n,m=0,1,\dots,L-1\,, \quad W_c=\frac{\omega_c}{2\pi}$$

The maximization of the Rayleigh quotient is realized by the maximum eigenvector of the prolate matrix A, that is, the eigenvector belonging to the maximum eigenvalue, say, λ_0:

$$A\mathbf{w}=\lambda_0\mathbf{w}$$

The prolate matrix is notoriously ill-conditioned having approximately $2LW_c$ eigenvalues that are very near one, and the remaining eigenvalues decreasing rapidly to zero. The following table lists the eigenvalues in decreasing order for the case $L=21$ and $W_c=0.2$, so that $2LW_c=8.4$, its condition number being, $\text{cond}(A)=5.1063\times 10^{16}$,

i	λ_i	i	λ_i
0	0.99999999998517786000	11	0.00131552671490021500
1	0.99999999795514627000	12	0.00007986915605618046
2	0.99999987170540139000	13	0.00000365494381482577
3	0.99999517388508363000	14	0.00000012731149204486
4	0.99987947149714795000	15	0.00000000336154097643
5	0.99792457099956200000	16	0.00000000006621668668
6	0.97588122145542644000	17	0.00000000000094327944
7	0.83446090480119717000	18	0.00000000000000920186
8	0.45591142240913063000	19	0.00000000000000004034
9	0.11887181858959120000	20	0.00000000000000001958
10	0.01567636516215985600		

These were generated by the following MATLAB code:

```
L = 21; Wc = 0.2;
n = 0:L-1;
f = 2*Wc*sinc(2*Wc*n);
A = toeplitz(f,f);
lambda = svd(A);
```

The eigenvectors of the prolate matrix are referred to as the *discrete prolate spheroidal sequences* (DPSS). The following approximate relationship between W_c and the Kaiser α parameter works well over the range $14\le R\le 120$ dB:

$$W_c=\frac{0.95\,\alpha/\pi+0.14}{L} \tag{9.4.6}$$

A simple way to compute the maximum eigenvector is by the power iteration, that is, start with a unit-norm initial vector $\mathbf{w}_0$, then iterate, $\mathbf{w}_k=A\mathbf{w}_{k-1}$, and normalize to unit-norm at each iteration, $\mathbf{w}_k=\mathbf{w}_k/\|\mathbf{w}_k\|$. However, because the corresponding eigenvalue λ_0 and the next highest one are so close to unity, the iteration will be very slow converging.

A more efficient approach is to apply the inverse power iteration on the matrix $Q=I-A$, that is, $\mathbf{w}_{k+1}=Q^{-1}\mathbf{w}_k=Q^{-k}\mathbf{w}_0$. This iteration converges to the minimum eigenvector of Q, which is the same as the maximum eigenvector of A. The minimum eigenvalue of Q is $1-\lambda_0$, which is very small and its inverse $(1-\lambda_0)^{-1}$ very large, causing the iteration to converge very fast. More details on this approach can be found in EWA–Ch.23 [46].

We note also that since the discrete-time DPSS functions are approximations to the sampled versions of the continuous-time prolate spheroidal wave functions (PSWF), one may construct the DPSS window from the 0th order PSWF. The spatial equivalent of this for finite apertures is discussed in EWA–Ch.20 [46], where the PSWF functions are adapted to represent space-limited signals as opposed to their original definition of frequency bandlimited signals. The construction of the time-limited PSWF window can be implemented in MATLAB via the function, **pswf**, see Appendix B.

```
% calculate alpha,L from R,Df using the Kaiser-Schafer approximation

% alpha = ...
% L = ...

n = 0:L-1;                          % time range of window
M = (L-1)/2;                        % middle of window, L must be odd

t0 = M; om0 = alpha/M;              % PSWF parameters, alpha = t0*om0

w = pswf(om0,t0,0,(n-M)*om0/t0);    % 0th order PSWF

% w = pswf(t0,om0,0,n-M);           % alternative version

w = w/max(w);                       % normalize to unity maximum
```

But for our purposes in this chapter, the built-in MATLAB function, **dpss** is perfectly adequate.

9.4.5 Chebyshev Window

Most windows have largest sidelobes near the main lobe. If a window is designed to achieve a minimum sidelobe attenuation of R dB, then typically R will be the attenuation of the sidelobes nearest to the mainlobe; the sidelobes further away will have attenuations higher than R.

Because of the tradeoff between mainlobe width and sidelobe attenuation, the extra attenuation of the furthest sidelobes will come at the expense of increased mainlobe width. If the attenuation of these sidelobes could be decreased (up to the level of the minimum R), then the mainlobe width would narrow.

It follows that for a given *minimum* desired sidelobe level R, the *narrowest* mainlobe width will be achieved by a window whose sidelobes are all *equal* to R. Conversely, for a given *maximum* desired mainlobe width, the *largest* sidelobe attenuation will be achieved by a window with equal sidelobe levels.

This "optimum" window is the Dolph-Chebyshev window, which is constructed with the help of Chebyshev polynomials. The mth Chebyshev polynomial $T_m(x)$ is:

$$T_m(x) = \cos\big(m\,\mathrm{acos}(x)\big)$$

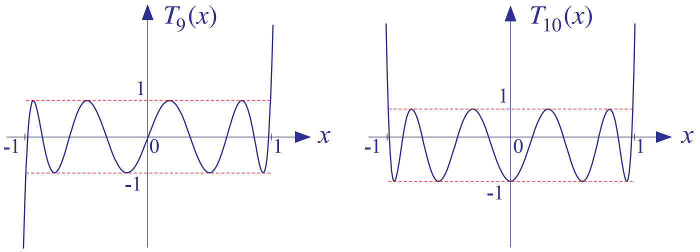
If $|x| > 1$, the inverse cosine $\mathrm{acos}(x)$ becomes imaginary, and the expression can be rewritten in terms of hyperbolic cosines:

$$T_m(x) = \cosh\big(m\,\mathrm{acosh}(x)\big)$$

Setting $x = \cos\theta$, or $\theta = \mathrm{acos}(x)$, we see that $T_m(x) = \cos(m\theta)$. Using trigonometric identities, the quantity $\cos(m\theta)$ can always be expanded as a polynomial in powers of $\cos\theta$. The expansion coefficients are precisely the coefficients of the powers of x of the Chebyshev polynomial, e.g.,

$$\begin{array}{lcl}
\cos(0\theta) = 1 & & T_0(x) = 1 \\
\cos(1\theta) = \cos\theta & & T_1(x) = x \\
\cos(2\theta) = 2\cos^2\theta - 1 & \Rightarrow & T_2(x) = 2x^2 - 1 \\
\cos(3\theta) = 4\cos^3\theta - 3\cos\theta & & T_3(x) = 4x^3 - 3x \\
\cos(4\theta) = 8\cos^4\theta - 8\cos^2\theta + 1 & & T_4(x) = 8x^4 - 8x^2 + 1
\end{array}$$

For $|x| < 1$, the Chebyshev polynomial has equal ripples, whereas for $|x| > 1$, it increases like x^m. Moreover, $T_m(x)$ is even in x if m is even, and odd in x if m is odd. The figure below depicts the Chebyshev polynomials $T_9(x)$ and $T_{10}(x)$.



The Dolph-Chebyshev window is defined such that its sidelobes will correspond to a portion of the equiripple range $|x| \le 1$ of the Chebyshev polynomial, whereas its mainlobe will correspond to a portion of the range $x > 1$.

For either even or odd L, the window spectrum $W(\omega)$ can be written in general as a polynomial of degree $L-1$ in the variable $u = \cos(\omega/2)$. Indeed, we have for the mth terms:

$$\cos(m\omega) = \cos\left(2m\frac{\omega}{2}\right) = T_{2m}(u)$$

$$\cos\big((m-1/2)\omega\big) = \cos\left((2m-1)\frac{\omega}{2}\right) = T_{2m-1}(u)$$

Thus in the odd case, the summation of such terms will result in a polynomial of maximal degree $2M = L-1$ in the variable u, and in the even case, it will result into a polynomial of degree, $2M - 1 = L - 1$.

The DTFT of the Dolph-Chebyshev window is defined by the Chebyshev polynomial of degree $L-1$ in the scaled variable $x = x_0\cos(\omega/2)$, that is,

$$\boxed{W(\omega) = T_{L-1}(x)\,, \quad x = x_0\cos\left(\frac{\omega}{2}\right)}$$

The relative sidelobe attenuation level in absolute units and in dB is defined in terms of the ratio of the mainlobe to the sidelobe heights:

$$R_a = \frac{W_{\text{main}}}{W_{\text{side}}}\,, \quad R = 20\log_{10}(R_a)\,, \quad R_a = 10^{R/20}$$

Because the mainlobe peak occurs at $\omega = 0$ or $x = x_0$, we will have $W_{\text{main}} = T_{L-1}(x_0)$, and because the sidelobe level is equal to the Chebyshev level within $|x| \le 1$, we will have $W_{\text{side}} = 1$. Thus, we find:

$$\boxed{R_a = T_{L-1}(x_0) = \cosh\big((L-1)\,\mathrm{acosh}(x_0)\big)}$$

which can be solved for x_0 in terms of R_a:

$$\boxed{x_0 = \cosh\left(\frac{\mathrm{acosh}(R_a)}{L-1}\right)}$$

Once the scale factor x_0 is determined, the window samples $w(n)$ can be computed by constructing the z-transform of the DTFT from its zeros and then doing an inverse z-transform. The $L-1$ zeros of $T_{L-1}(x)$ are easily found to be:

$$T_{L-1}(x) = \cos\big((L-1)\,\mathrm{acos}(x)\big) = 0 \quad \Rightarrow \quad x_i = \cos\left(\frac{(i-1/2)\pi}{L-1}\right)$$

for $i = 1, 2, \dots, L-1$. Solving for the corresponding frequencies through $x_i = x_0\cos(\omega_i/2)$, we find the DTFT zeros:

$$\omega_i = 2\,\mathrm{acos}\left(\frac{x_i}{x_0}\right)\,, \qquad z_i = e^{j\omega_i}\,, \qquad i = 1, 2, \dots, L-1$$

The symmetric z-transform of the window is then constructed in terms of its zeros:

$$W(z)=\prod_{i=1}^{L-1}(1-z_iz^{-1})$$

The inverse z-transform of $W(z)$ are the window coefficients $w(n)$. The typical MATLAB code is,

```
L1 = L-1;                   % number of zeros
Ra = 10^(R/20);             % sidelobe level in absolute units
x0 = cosh(acosh(Ra)/L1);    % scaling factor

i = 1:L1;
xi = cos(pi*(i-0.5)/L1);    % L1 zeros of Chebyshev polynomial
omi = 2 * acos(xi/x0);      % L1 zeros in omega-space
zi = exp(j*omi);            % L1 zeros of W(z) polynomial

w = real(poly(zi));         % zeros-to-polynomial-coefficients
                            % see also the more accurate function poly2
                            % in the EWA toolbox
```

The window coefficients resulting from this construction can be normalized to unity maximum. Alternatively, the built-in MATLAB function, **chebwin**, can be used to calculate the window samples

```
w = chebwin(L,R);     % R in dB
```

The 3-dB frequency ω_3 is defined by the half-power condition:

$$W(\omega_3)=T_{L-1}(x_3)=\frac{T_{L-1}(x_0)}{\sqrt{2}}=\frac{R_a}{\sqrt{2}}\quad\Rightarrow$$

$$\cosh\big((L-1)\,\mathrm{acosh}(x_3)\big)=\frac{R_a}{\sqrt{2}}$$

Solving for x_3 and the corresponding 3-dB angle ω_3, $x_3=x_0\cos(\omega_3/2)$,

$$x_3=\cosh\left(\frac{\mathrm{acosh}(R_a/\sqrt{2})}{L-1}\right),\qquad \omega_3=2\,\mathrm{acos}\left(\frac{x_3}{x_0}\right)$$

From ω_3, one calculates the 3-dB width, $\Delta\omega_{3\mathrm{dB}}=2\omega_3$. However, the following approximate relationship works well for the broadening factor relative to the rectangular window, over the range $13\le R\le 180$ dB,

$$\Delta\omega_{3\mathrm{dB}}=0.886\,\frac{2\pi b}{L},\qquad b=0.65+0.0195R-0.00005R^2$$

9.5 Fourier Optics, Apertures, Spatial Arrays

So far we have dealt with time-domain signals and their Fourier transforms. There is a parallel set of topics involving spatial signals and the corresponding spatial Fourier transforms in which the time and frequency variables, t,Ω, are replaced one-dimensionally by the space and wavenumber variables, x,k_x, or, three-dimensionally, x,y,z,k_x,k_y,k_z.

A finite-duration time-windowed signal is replaced by a finite-aperture spatially-windowed signal. The process of designing a spatial window or aperture function with tapered ends is referred to in this context as *apodization.*[†]

Lens systems act as Fourier transformers in the spatial domain allowing all sorts of signal processing operations, such as lowpass or highpass spatial filtering, to be implemented optically.

The problem of designing narrow-beam low-sidelobe antenna or sensor arrays is equivalent to the problem of spectrum analysis of sinusoidal signals. Typical array geometries are depicted in Fig. 9.5.1. For example, N identical linear antennas arranged at equal distances d along the x-axis, $x_n=nd$, $n=0,1,2,\dots,N-1$ with z-directed linear elements perpendicular to the xy-plane, will have an array factor, which can be thought of as a discrete-space Fourier transform (DSFT) of the antenna weights, w_n,

$$W(\psi)=\sum_{m=0}^{N-1}w_m e^{j\psi m},\qquad \psi=k_xd=kd\cos\phi=\frac{2\pi d}{\lambda}\cos\phi$$

where $k_x=k\cos\phi$ is the wavenumber along the x-direction and ϕ, the azimuthal angle, and k is the vacuum wavenumber related to the wavelength λ, by $k=2\pi/\lambda$, and ψ plays the role of digital spatial frequency in units of radians per space sample. An array that is scanned towards a particular direction ϕ_0, with $\psi_0=kd\cos\phi_0$, will have an array factor, $A(\psi)=W(\psi-\psi_0)$.

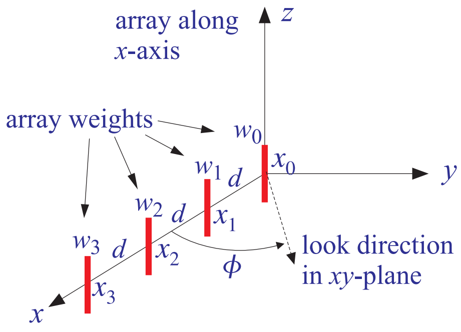


Fig. 9.5.1 One-dimensional antenna array.

The problem of determining the angles of arrival of several plane waves incident on an array is the spatial equivalent of the time-domain problem of spectral analysis of multiple sinusoids. These topics are beyond the scope of this book, but they are discussed very extensively in the EWA book, Chapters 20, 22, 23 [46]. The Table 9.5.1 illustrates the corresponding concepts between time and spatial concepts.

[†] from the Greek word for "no feet"

discrete-time signal processing	discrete-space array processing
time-domain sampling, $t_n = nT$	space-domain sampling, $x_n = nd$
sampling time interval, T	sampling space interval, d
sampling rate, $1/T$ [samples/sec]	sampling rate, $1/d$ [samples/meter]
frequency, Ω [radians/sec]	wavenumber, k_x [radians/meter]
digital frequency, $\omega = \Omega T$	digital wavenumber, $\psi = k_x d$
Nyquist interval, $-\pi \le \omega \le \pi$	Nyquist interval, $-\pi \le \psi \le \pi$
sampling theorem, $\Omega \le \pi/T$	sampling theorem, $k_x \le \pi/d$
spectral images	grating lobes or fringes
frequency response, $A(\omega)$	array factor, $A(\psi)$
z-domain, $z = e^{j\omega}$	z-domain, $z = e^{j\psi}$
transfer function, $A(z)$	transfer function, $A(z)$
DTFT and inverse DTFT	DSFT and inverse DSFT
pure sinusoid, $e^{j\omega_0 n}$	narrow beam, $e^{-j\psi_0 n}$
windowed sinusoid, $w(n)e^{j\omega_0 n}$	windowed narrow beam, $w(n)e^{-j\psi_0 n}$
resolution of multiple sinusoids	resolution of multiple beams
frequency shifting by AM modulation	phased array scanning
filter design by window method	array design by window method
bandpass FIR filter design	angular sector array design
frequency-sampling design	Woodward-Lawson design
DFT	Blass matrix
FFT	Butler matrix

Table 9.5.1 Duality between time-domain and space-domain signal processing.

9.6 *Periodogram and Its Improvements*

We begin by reviewing some basic random signal concepts, such as autocorrelation functions, power spectra, sample autocorrelations, and the periodogram.

The *autocorrelation function* of a zero-mean random signal is defined as the correlation between two samples $x(n)$ and $x(n+k)$ separated by a time lag k. It is a measure of the dependence of successive samples on the previous ones:

$$\boxed{R_{xx}(k) = E\big[x(n+k)x(n)\big]} \qquad \text{(autocorrelation function)}$$

For stationary signals, $R_{xx}(k)$ depends only on the relative time-lag k, and not on the absolute time n. Note that $R_{xx}(k)$ is a double-sided sequence and, as a consequence of stationarity, it is symmetric in k, that is, $R_{xx}(-k) = R_{xx}(k)$.

The *power spectrum* of the random signal $x(n)$ is defined as the DTFT of its autocorrelation function $R_{xx}(k)$. It represents the frequency content of the random signal $x(n)$ in an average sense:

$$\boxed{S_{xx}(\omega) = \sum_{k=-\infty}^{\infty} R_{xx}(k)e^{-j\omega k}} \qquad \text{(power spectrum)}$$

where $\omega = 2\pi f/f_s$ is the digital frequency in radians per sample. The inverse DTFT relationship expresses $R_{xx}(k)$ in terms of $S_{xx}(\omega)$,

$$\boxed{R_{xx}(k) = E\big[x(n+k)x(n)\big] = \int_{-\pi}^{\pi} S_{xx}(\omega)e^{j\omega k}\,\frac{d\omega}{2\pi}}$$

In particular, setting $k = 0$, we obtain the *average power*, or variance, of the signal $x(n)$:

$$\sigma_x^2 = R_{xx}(0) = E\big[x(n)^2\big] = \int_{-\pi}^{\pi} S_{xx}(\omega)\,\frac{d\omega}{2\pi} = \int_{-f_s/2}^{f_s/2} S_{xx}(f)\,\frac{df}{f_s}$$

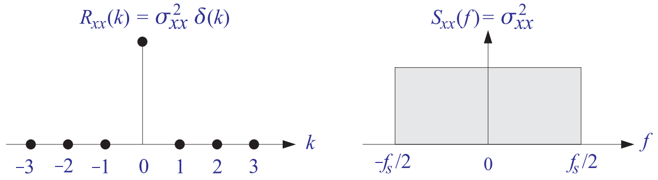
$$\boxed{S_{xx}(f) = \sum_{k=-\infty}^{\infty} R_{xx}(k)e^{-2\pi jfk/f_s}}$$

The quantity $S_{xx}(f)/f_s$ represents the *power per unit frequency* interval. Hence, the name “power spectrum” or “power spectral density” (psd). It describes how the signal’s power is *distributed* among different frequencies. Its integral over the Nyquist interval gives the *total* power in the signal.

Often it is more convenient to work with the z-transform of the autocorrelation and replace $z = e^{j\omega} = e^{2\pi jf/f_s}$ to obtain the power spectrum $S_{xx}(\omega)$ or $S_{xx}(f)$:

$$\boxed{S_{xx}(z) = \sum_{k=-\infty}^{\infty} R_{xx}(k)z^{-k}}$$

White noise has a delta-function autocorrelation and a *flat* spectrum, as shown below.



Because (by definition) successive signal samples are independent of each other, the autocorrelation function will factor for $k \neq 0$ into the product of the means which are assumed to be individually zero:

$$R_{xx}(k) = E\big[x(n+k)x(n\big] = E\big[x(n+k)\big]\cdot E\big[x(n)\big] = 0$$

whereas for $k = 0$, we get the variance

$$R_{xx}(0) = E\big[x(n)^2\big] = \sigma_x^2$$

Combining them into a single equation, we have:

$$R_{xx}(k) = \sigma_x^2\delta(k) \qquad \text{(white noise autocorrelation)}$$

Only the $k = 0$ term survives the sum giving the flat spectral density (over the Nyquist interval):

$$S_{xx}(f) = \sigma_x^2, \qquad \text{for } -\frac{f_s}{2} \le f \le \frac{f_s}{2} \qquad \text{(white noise spectrum)}$$

Given a length-N block of signal samples $x(n)$, $n = 0, 1, \dots, N-1$, one can compute an *estimate* of the statistical quantity $R_{xx}(k)$ by the so-called *sample autocorrelation* obtained by replacing the statistical average by the time average:

$$\boxed{\hat{R}_{xx}(k) = \frac{1}{N}\sum_{n=0}^{N-1-k} x(n+k)x(n)} \qquad \text{(sample autocorrelation)}$$

for $k = 0, 1, \dots, N-1$. The negative tail can be defined using the symmetry property $\hat{R}_{xx}(-k) = \hat{R}_{xx}(k)$, so that we can write, for $|k| \le N-1$,

$$\boxed{\hat{R}_{xx}(k) = \frac{1}{N}\sum_{n=0}^{N-1-|k|} x(n+|k|)x(n)} \qquad \text{(sample autocorrelation)}$$

The rule of thumb is that only about the first 5-10% of the lags are statistically reliable, that is, $0 \le k \le N/10$. The built-in MATLAB function, **xcorr**, computes $\hat{R}_{xx}(k)$, for $-M \le k \le M$, with any $M \le N-1$, with usage,

```
Rxx = xcorr(x,M);   % with M <= length(x)-1

% examples

x = 0:5;
Rxx = xcorr(x,3);
% Rxx = [14  26  40  55  40  26  14]

Rxx = xcorr(x,5);
% Rxx = [0  5  14  26  40  55  40  26  14   5  0]
```

with the outputs listed in the order:

$$[\,\hat{R}_{xx}(-M)\,,\ \cdots\,,\ \hat{R}_{xx}(-1)\,,\ \hat{R}_{xx}(0)\,,\ \hat{R}_{xx}(1)\,,\ \cdots\,,\ \hat{R}_{xx}(M)\,]$$

Alternatively, one can use the function, **acf.m**, in the ISP2e toolbox, which computes non-negative lags only up to desired maximum lag, and can also compute cross correlations. It has usage,

```
Rxy = acf(x,x,M) = autocorrelation of length-N vector x
Rxy = acf(x,y,M) = cross-correlation between two length-N vectors x,y
Rxy = acf(x,y)   = equivalent to M = N-1

x = length-N vector
y = length-N vector
M = maximum lag (typically, M <= N-1, pads zeros if M > N-1)

Rxy = [Rxy(0), Rxy(1), ..., Rxy(M)], row vector of non-negative lags
```

It can be shown that for wide-sense stationary signals, $\hat{R}_{xx}(k)$ is a good (i.e., consistent) estimate of $R_{xx}(k)$, converging to the latter for large N (in the mean-square sense):

$$\hat{R}_{xx}(k) \longrightarrow R_{xx}(k) \quad \text{as} \quad N \to \infty$$

The DTFT of $\hat{R}_{xx}(k)$ is called the *periodogram spectrum* and can be thought of as an *estimate* of the true power spectrum $S_{xx}(\omega)$:

$$\boxed{\hat{S}_{xx}(\omega) = \sum_{k=-(N-1)}^{N-1} \hat{R}_{xx}(k)e^{-j\omega k}} \qquad \text{(periodogram spectrum)} \tag{9.6.1}$$

Using the definition of $\hat{R}_{xx}(k)$, and rearranging summations, we can express the periodogram in the alternative way:

$$\boxed{\hat{S}_{xx}(\omega) = \frac{1}{N}\,|X_N(\omega)|^2} \qquad \text{(periodogram spectrum)} \tag{9.6.2}$$

where $X_N(\omega)$ is the DTFT of the length-N data block $x(n)$, which can be computed efficiently using FFTs, or, the MATLAB function **freqz**,

$$X_N(\omega) = \sum_{n=0}^{N-1} x(n)e^{-j\omega n}$$

A *modified periodogram* can also be defined by windowing the length-N data block using a length-N window, such as Hamming, and then computing the DTFT of the windowed signal, that is,

$$\begin{aligned} X_w(\omega) &= \sum_{n=0}^{N-1} w(n)\,x(n)e^{-j\omega n} \\ \hat{S}_{\text{mod}}(\omega) &= \frac{1}{N}\,\big|X_w(\omega)\big|^2 = \text{modified periodogram} \end{aligned} \tag{9.6.3}$$

It can be shown that for wide-sense stationary random signals the periodogram is *asymptotically unbiased*, that is, its mean converges to the true power spectrum $S_{xx}(\omega)$ in the limit of large N,

$$\boxed{S_{xx}(\omega) = \lim_{N\to\infty} E\Big[\hat{S}_{xx}(\omega)\Big] = \lim_{N\to\infty} E\Big[\frac{1}{N}\,|X_N(\omega)|^2\Big]}$$

Since the sample autocorrelation is an asymptotically unbiased and consistent estimate of the true autocorrelation function, $R_{xx}(k) = E[x(n+k)x(n)]$, one may hope that the periodogram itself can be taken to be an estimate of the true power spectrum of the process $x(n)$. However, although the periodogram is asymptotically unbiased, it is not a consistent estimator of the true spectrum, so that while its mean tends to the true spectrum as $N \to \infty$, but its variance does not tend to zero, resulting therefore in an unreliable estimate.

The subject of “classical spectral analysis” is concerned with the methods of improving the plain periodogram, maintaining its asymptotic unbiasedness while reducing its variance, thus, improving its statistical reliability. There are two basic methods of improvement [240]: (a) periodogram averaging and (b) periodogram smoothing, which are summarized in Fig. 9.6.1.

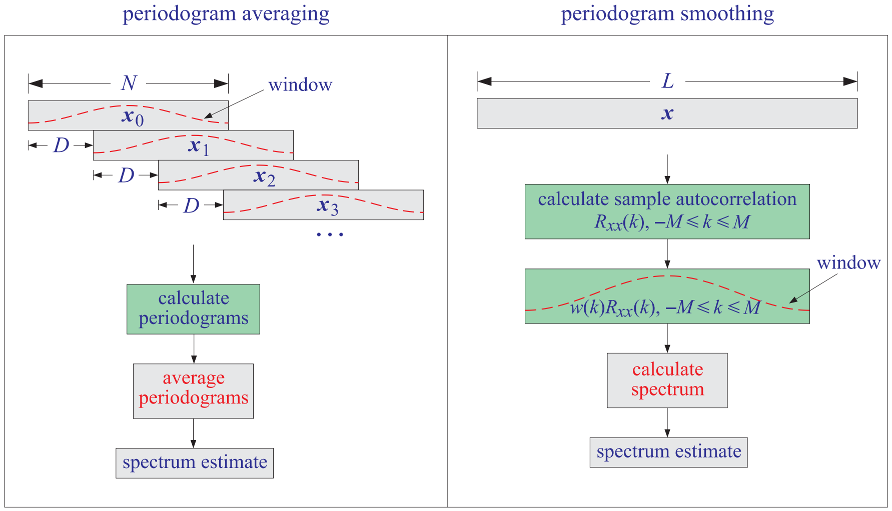


Fig. 9.6.1 Periodogram averaging and smoothing methods.

In the *periodogram averaging* method, the signal is divided into K segments that may be contiguous or overlapping (usually by 50% in the Welch method). The signal segments may be optionally pre-windowed and their periodograms are computed and averaged for each ω. Denoting by $\hat{S}_i(\omega)$, $i=1,2,\dots,K$, the periodogram of the i-th segment, the averaged periodogram is calculated by,

$$\hat{S}_{\text{av}}(\omega)=\frac{1}{K}\sum_{i=1}^{K}\hat{S}_i(\omega)=\text{averaged periodogram} \tag{9.6.4}$$

It can be shown that the variance of the resulting averaged periodogram is reduced typically by a factor of K. The larger the number of segments K, the smaller the variance and the more reliable the spectrum estimate.

The relationship between the overall signal length L, and the number of segments K and the length N of each segment, is as follows in the non-overlapping and the 50-percent overlapping cases,

$$\begin{aligned} L &= KN && \text{contiguous}\\ L &= \frac{1}{2}(K+1)N && \text{50\% overlapping}\end{aligned} \tag{9.6.5}$$

These capture the tradeoff between *statistical reliability*, which requires a large K, and *frequency resolution* for each segment, which requires a large N, since, $\Delta\omega\approx 2\pi b/N$.

Thus, if a long enough signal is available, it is possible to compute a reliable spectrum estimate.

This can fail in two ways: (i) a long data record may not be available, and (ii) even if it is available, it may be too long to represent a stationary random signal, and therefore its spectrum computation would be meaningless. These shortcomings have led to the development of *modern spectral analysis* methods, which are based on *parametric models* of the signal [25,26].

In the *periodogram smoothing* method, one computes $2M+1$ lags of the sample autocorrelation of the length-L data, with $M\ll L-1$, then windows the autocorrelation with a symmetric window of length $2M+1$, and computes the DTFT of the result, that is, given a signal block, $x(n)$, $n=0,1,\dots,L-1$, first remove its sample mean, and then,

$$\begin{aligned} \hat{R}_{xx}(k) &= \frac{1}{L}\sum_{n=0}^{L-1-|k|}x(n+|k|)x(n)\,, \quad -M\leq k\leq M\\ \hat{S}_{\text{sm}}(\omega) &= \sum_{k=-M}^{M}w(k)\hat{R}_{xx}(k)e^{-j\omega k}=\text{smoothed periodogram}\end{aligned} \tag{9.6.6}$$

Depending on the window used, the variance of the spectrum estimate is typically reduced by a factor of M/L, while the frequency resolution becomes of the order of $\Delta\omega=2\pi b/M$. We emphasize that in this method, the window is applied to the autocorrelation itself, not to the time signal. For this reason, such windows are referred to as *lag windows*.

Because multiplication in the time domain is equivalent to convolution in the frequency domain, the smoothed periodogram will be the frequency-domain convolution of the DTFT $W(\omega)$ of the lag window $w(k)$ and the ordinary periodogram $\hat{S}_{\text{per}}(\omega)$,

$$\hat{S}_{\text{sm}}(\omega)=W(\omega)*\hat{S}_{\text{per}}(\omega)=\int_{-\pi}^{\pi}W(\omega')\hat{S}_{\text{per}}(\omega-\omega')\frac{d\omega'}{2\pi} \tag{9.6.7}$$

Thus, the periodogram gets smeared or smoothed in the frequency domain by $W(\omega)$. Because $w(k)$ is real-valued and symmetric in the the interval, $|k|\leq M$, its DTFT $W(\omega)$ will also be real valued. However, it need not be positive, and in order to avoid any negative terms in the convolution integral, it should also be required that $W\omega)\geq 0$ for all ω. An example of such window is the symmetric *Bartlett* window of length $2M+1$, or, length $2M-1$, since it vanishes at $k=\pm M$,

$$w(k)=1-\frac{|k|}{M}\,,\quad -M\leq k\leq M \qquad \text{(Bartlett window)} \tag{9.6.8}$$

having a non-negative DTFT, known as Fejér’s kernel,

$$W(\omega)=\sum_{k=-M}^{M}\left(1-\frac{|k|}{M}\right)e^{-j\omega k}=\sum_{k=-M}^{M}w(k)e^{-j\omega k}=\frac{1}{M}\frac{\sin^2\left(\dfrac{M\omega}{2}\right)}{\sin^2\left(\dfrac{\omega}{2}\right)} \tag{9.6.9}$$

Another lag window is the Parzen window, defined over $-M \le k \le M$,

$$w(k)=\begin{cases} 1-6\left(\dfrac{|k|}{M}\right)^2+6\left(\dfrac{|k|}{M}\right)^3, & |k|\le \dfrac{M}{2} \\ 2\left(1-\dfrac{|k|}{M}\right)^3, & \dfrac{M}{2}<|k|\le M \end{cases} \tag{9.6.10}$$

with non-negative DTFT,

$$W(\omega)=\frac{4\left[3-2\sin^2\left(\dfrac{\omega}{2}\right)\right]\sin^4\left(\dfrac{M\omega}{2}\right)}{M^3\sin^4\left(\dfrac{\omega}{2}\right)} \tag{9.6.11}$$

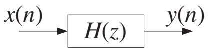
A more complete discussion of these and other lag windows can be found in Percival and Walden [240].

9.7 Filtering of Random Signals

In designing filters to remove noise, it is necessary to know the effect of filtering on the autocorrelation function and on the power spectrum of a random signal.

Suppose the input to a *strictly stable* filter $H(z)$ with impulse response $h(n)$ is a wide-sense stationary signal $x(n)$. Then, the corresponding output $y(n)$ will also be a wide-sense stationary random signal:

$$y(n)=\sum_m h(m)x(n-m)$$



It can be shown [2,45] that the power spectrum of the output is related to that of the input by:

$$\boxed{S_{yy}(\omega)=|H(\omega)|^2 S_{xx}(\omega)} \tag{9.7.1}$$

Thus, the input spectrum is reshaped by the filter spectrum. A simple way to justify this result is in terms of periodograms. The filtering equation in the z-domain is, $Y(z)=H(z)X(z)$, and in the frequency domain, $Y(\omega)=H(\omega)X(\omega)$. It follows that the output periodogram will be related to the input periodogram by a similar equation as (9.7.1):

$$\frac{1}{N}|Y(\omega)|^2=|H(\omega)|^2\cdot\frac{1}{N}|X(\omega)|^2$$

Applying this result to the special case of a *white noise input* with a flat spectral density $S_{xx}(\omega)=\sigma_x^2$ gives

$$\boxed{S_{yy}(\omega)=|H(\omega)|^2\sigma_x^2} \tag{9.7.2}$$

Similarly, in the z-transform notation:

$$\boxed{S_{yy}(z)=H(z)H(z^{-1})\sigma_x^2} \tag{9.7.3}$$

where we replaced $H(\omega)=H(z)$ and $H(\omega)^*=H(z^{-1})$, the latter following from the fact that $h(n)$ is real-valued. Indeed, with $z=e^{j\omega}$ and $z^{-1}=z^*=e^{-j\omega}$, we have:

$$H(\omega)^*=\Big(\sum_n h(n)e^{-j\omega n}\Big)^*=\sum_n h(n)e^{j\omega n}=H(z^{-1})$$

Equation (9.7.2) implies that the filtered noise $y(n)$ is no longer white. Its power spectrum acquires the shape of the filter's spectrum. Its autocorrelation function is no longer a delta function. It can be computed by taking the (stable) inverse z-transform of Eq. (9.7.3).

A measure of whether the filter attenuates or magnifies the input noise is given by the variance of the output σ_y^2. Using Eq. (9.7.1) applied to $y(n)$, we have:

$$\sigma_y^2=\int_{-\pi}^{\pi}S_{yy}(\omega)\,\frac{d\omega}{2\pi}=\sigma_x^2\int_{-\pi}^{\pi}|H(\omega)|^2\,\frac{d\omega}{2\pi}$$

which can be written in the form:

$$\boxed{NRR=\frac{\sigma_y^2}{\sigma_x^2}=\int_{-\pi}^{\pi}|H(\omega)|^2\,\frac{d\omega}{2\pi}=\sum_n h(n)^2} \quad \text{(NRR)} \tag{9.7.4}$$

where we used Parseval's equation discussed in Chapter 5.

This ratio will be referred to as the *noise reduction ratio* (NRR). If it is less than one, the input noise will be attenuated by the filter. It can be used as a useful criterion for designing noise-reducing filters—the objective being to design $H(z)$ such that (9.7.4) is minimized as much as possible.

A necessary assumption for the derivation of the results (9.7.1) or (9.7.4) is that the filter $h(n)$ be strictly stable. The stability of $h(n)$ is required to ensure that the stationary input signal $x(n)$ will generate, after the filter transients die out, a stationary output signal $y(n)$.

Thus, even marginally stable filters with poles on the unit circle are not allowed. To illustrate the problems that may arise, consider the simplest marginally stable filter, namely, an accumulator/integrator:

$$H(z)=\frac{1}{1-z^{-1}},\qquad h(n)=u(n)$$

It has I/O difference equation:

$$y(n)=y(n-1)+x(n)$$

Assuming zero initial conditions, we can write it in the convolutional form:

$$y(n)=x(n)+x(n-1)+\cdots+x(1)+x(0) \tag{9.7.5}$$

If $x(n)$ is a zero-mean, white noise signal with variance σ_x^2, the resulting accumulated output signal $y(n)$ is a version of the *random walk* process [1277].

The signal $y(n)$ is not stationary and becomes unstable as n increases, in the sense that its mean-square value (i.e., its variance) $\sigma_y^2(n)=E[y(n)^2]$ diverges. Indeed, using the property that the variance of a sum of independent random variables is equal to the sum of the individual variances, we obtain from Eq. (9.7.5):

$$\sigma_y^2(n)= E[y(n)^2]= E[x(n)^2]+E[x(n-1)^2]+\cdots+E[x(0)^2]= \sigma_x^2+\sigma_x^2+\cdots+\sigma_x^2$$

where all the terms have a common variance σ_x^2, by assumption. It follows that:

$$\sigma_y^2(n)= E[y(n)^2]= (n+1)\sigma_x^2 \tag{9.7.6}$$

Thus, the mean-square value of $y(n)$ grows linearly in n. In a digital implementation, the growing amplitude of $y(n)$ will quickly saturate the hardware registers.

Analog integrators also behave in a similar fashion, growing unstable when their input is random noise. Therefore, one should never accumulate or integrate white noise. A standard remedy is to use a so-called *leaky integrator*, which effectively stabilizes the filter by pushing its pole slightly into the unit circle. This is accomplished by the replacement, with $0<\rho\lesssim 1$

$$H(z)=\frac{1}{1-z^{-1}} \quad\longrightarrow\quad H(z)=\frac{1}{1-\rho z^{-1}}$$

More generally, a marginally stable filter can be stabilized by the substitution $z\to\rho^{-1}z$, which pushes all the marginal poles into the inside of the unit circle. The substitution amounts to replacing the transfer function and impulse response by

$$\boxed{H(z)\longrightarrow H(\rho^{-1}z),\qquad h(n)\longrightarrow \rho^n h(n)} \tag{9.7.7}$$

9.8 Computer Experiment – Sunspot Time Series

The sunspot time series has served as a historical benchmark for testing several spectrum estimation methods. Yule was the first to model it in terms of a 2nd order autoregressive model. However, in this experiment we will consider only the periodogram averaging and periodogram smoothing methods.

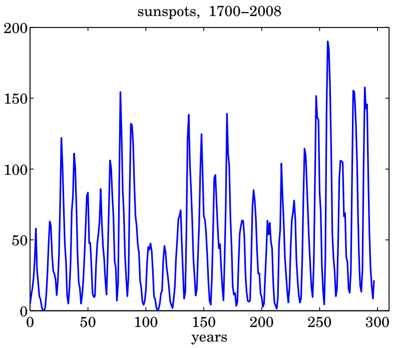
The attached file, **sunspots.dat**,[†] contains the yearly mean number of sunspots for the 309 years of 1700–2008. These can be read into MATLAB and remove their sample mean as follows,

```
Y = load('sunspots.dat');     % Y is 309x2 matrix of [years,sunspots]
x = Y(:,2);                   % number of sunspots
L = length(x);                % here, L = 309
mx = mean(x);                 % sample mean
x = x - mx;                   % zero-mean data
```

where the last line determines the sample mean of the data block and subtracts it from the data. The mean m_x is saved and can be restored at the end if necessary. The time series with its mean restored is plotted below.

In this problem the time units are "years", and the frequency units are "cycles/year", and the corresponding inverse frequencies or periods are in units of "years/cycle". The

sampling rate is $f_s=1$ samples/year. Our objective is to determine the dominant cycle present in the time series, which is known to have a period of about 11 years.



(a) Using a *rectangular window* calculate the ordinary periodogram of Eq. (9.6.2) over 1001 equally-spaced frequencies in the Nyquist interval $0\le f\le f_s/2$.

Normalize it to unity maximum, convert it into dB scales, and plot it versus f using vertical scales $[-50, 10]$ dB. Use the same vertical scales in all subsequent questions. This can be done with the help of the **freqz** function, with typical MATLAB code as follows,

```
f = linspace(0,0.5,1001);            % frequency range
om = 2*pi*f/fs;                      % frequencies in rads/sample
S = 1/L * abs(freqz(x,1,om)).^2;     % periodogram
S = S/max(S);                        % unit max
S = 10*log10(S);                     % dB
```

(b) Determine the frequency $f_{\max}$ of the dominant cycle, i.e., the frequency of the maximum of the periodogram spectrum, and calculate the estimated period $1/f_{\max}$ of the dominant cycle in years, placing that point on the graph. See some example graphs at the end. Typical code to determine the maximum and its frequency is as follows:

```
% f = ...                       % frequency range, as above
% S = ...                       % in dB, as above
[Smax,imax] = max(S);           % max S and its index
fmax = f(imax);                 % frequency at Smax
```

(c) Repeat parts (a,b) using a length-L Hamming window to calculate a modified periodogram, as in Eq. (9.6.3).

(d) Next, consider the periodogram averaging method with contiguous segments. Choose the dimension of each segment to be $N=60$ so that there are roughly $K=309/60\approx 5$ segments whose periodograms are to be averaged. The division into segments can be done conveniently using the built-in **buffer** function, with usage,

[†] included in the ISP2e zip file of MATLAB programs and data

```
    N = 60;                            % segment length
    [X,~] = buffer(x,N,0,'nodelay');   % X = complete segments only
    K = size(X,2);                     % actual number of segments
```

The “nodelay” option starts the first segment without inserting zeros at the beginning, the “0” option tells it to use contiguous segments, and the particular output usage ensures that the X output of dimension $N\times K$ will consist only of *complete* length-N column segments, ignoring any possible incomplete last segment. The row dimension of X is the actual number K of complete segments.

Using both a rectangular and Hamming window of length N, compute the periodograms of each segment over the same set of frequencies as in part (a), then, average the periodograms of the segments, normalize the result to unity maximum, convert it to dB, and plot it versus f, placing on the graph the estimated maximum, and its estimated period in years.

The computation and averaging of the segment periodograms can be done using a column-wise for-loop in MATLAB that runs over the columns of X,

```
% w = ...                                % length-N window, column
% f = ...                                % define frequency range
om = 2*pi*f/fs;                          % frequencies in rads/sample
Sav = zeros(size(om));                   % initialize average
for s = X                                % run over the columns of X
   Sav = Sav + abs(freqz(w.*s,1,om)).^2; % accumulate periodograms
end                                      % follow by normalizations, etc.
```

(e) Repeat part (d) using the Welch periodogram method, implemented with 50% overlapping segments and using both a rectangular and a Hamming window. The **buffer** function can be used again to divide the signal into overlapping and complete segments. In this part, choose $N = 104$.

The following MATLAB code segment shows the usage of **buffer**, where now the third argument $N/2$ specifies a 50% overlap,

```
    N = 104;                             % segment length
    [X,~] = buffer(x,N,N/2,'nodelay');   % X = complete segments only
    K = size(X,2);                       % actual number of segments
```

For both window cases, carry out the periodogram averaging and plotting, as in part (d), including finding the dominant cycle. Do not use the built-in function **pwelch** in this part.

(f) Next, consider the periodogram smoothing method. Choose a maximum lag of $M = 200$, and using a Bartlett window, calculate the smoothed periodogram of Eq. (9.6.6) over the same set of frequencies as in part (a).

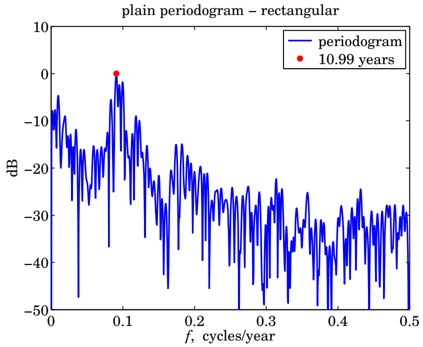
As before, normalize the smoothed periodogram to unity maximum, convert to dB, and plot it versus f, inserting on the graph the estimated maximum point and the estimated dominant cycle in years.

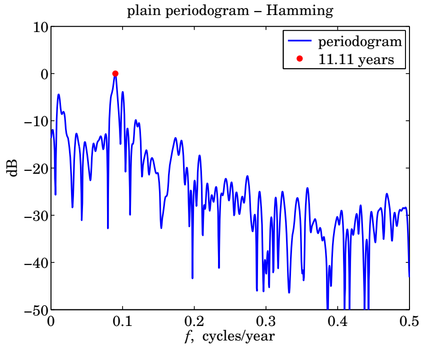
The sample autocorrelation can be calculated with the built-in **xcorr** function, and its windowed DTFT, with **freqz**,

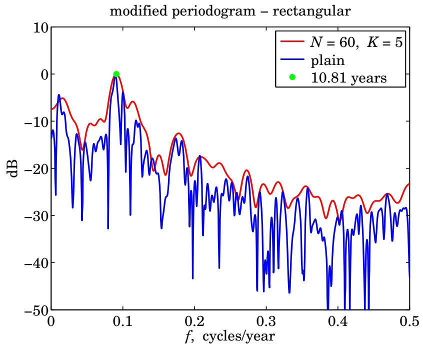
```
% M = 200;                       % Rxx length is 2*M+1
% w = ...                        % define window, same size as Rxx
% f = ...                        % define frequency range
om = 2*pi*f/fs;                  % frequencies in rads/sample
Rxx = xcorr(x,M);                % sample autocorrelation over [-M,M]
S =  abs(freqz(w.*Rxx,1,om));    % smoothed periodogram
```

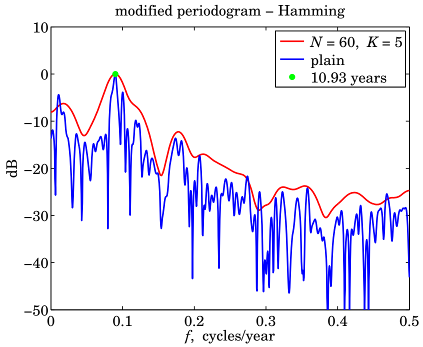
(g) Repeat part (f) with $M = 100$ and with $M = 50$, noting the changes in the amount of smoothing, as well the reduction in resolution.

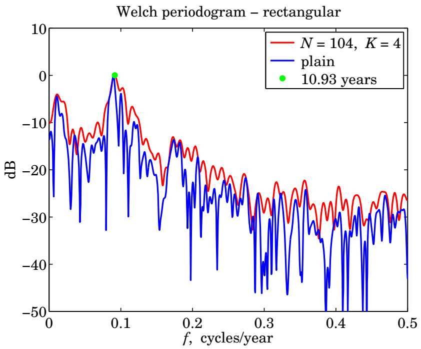
Some typical graphs are shown below.

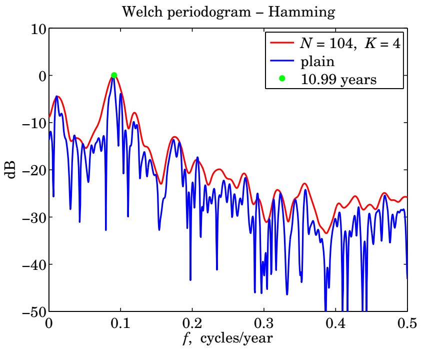


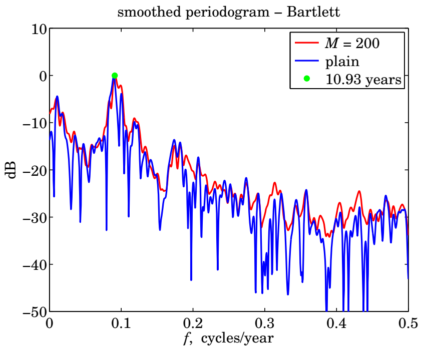












9.9 Problems

9.1 A 128-millisecond portion of an analog signal is sampled at a rate of 8 kHz and the resulting L samples are saved for further processing. What is L? The 256-point DFT of these samples is computed. What is the frequency spacing in Hz of the computed DFT values? What is the total number of required multiplications: (a) if the computations are done directly using the definition of the DFT, (b) if the L samples are first wrapped modulo 256 and then the 256-point DFT is computed, and (c) if a 256-point FFT is computed of the wrapped signal?

9.2 A 10 kHz sinusoidal signal is sampled at 80 kHz and 64 samples are collected and used to compute the 64-point DFT of this signal. At what DFT indices $k = 0, 1, \dots, 63$ would you expect to see any peaks in the DFT?

9.3 A 5 kHz sinusoidal signal is sampled at 40 kHz and 16 periods of the signal are collected. What is the length N of the collected samples? Suppose an N-point DFT is performed. Then, at what DFT indices, $k = 0, 1, \dots, N-1$, do you expect to see any peaks in the DFT spectrum?

In general, how is the number of periods contained in the N samples related to the DFT index at which you get a peak?

9.4 An 18 kHz sinusoid is sampled at a rate of 8 kHz and a 16-point DFT of a finite portion of the signal is computed. At what DFT indices in the range $0 \le k \le 15$ do you expect to see any peaks in the DFT spectrum? Would it matter if first we folded the 18 kHz frequency to lie within the Nyquist interval and then computed the DFT? Explain.

9.5 It is known that the frequency spectrum of a narrowband signal has a peak of width of 20 Hz but it is not known where this peak is located. To find out, an FFT must be computed and plotted versus frequency. If the signal is sampled at a rate of 8 kHz, what would be the minimum number of samples L that must be collected in order for the peak to be resolvable by the length-L data window? What is the duration in seconds of this data segment? What would be the minimum size N of the FFT in order for the N FFT spectral values to represent the L time samples accurately?

9.6 *Computer Experiment: Rectangular and Hamming Windows.* Using the routine `dtftr.c`, reproduce the results and graphs of Example 9.1.3.

9.7 *Computer Experiment: Frequency Resolution and Windowing.* Reproduce the results and graphs of Example 9.1.4. The spectra of the windowed signals must be computed by first windowing them using a length-L window and then padding $256 - L$ zeros at their ends to make them of length-256, and finally calling a 256-point FFT routine.

9.8 *Computer Experiment: Physical versus Computational Resolution.* Reproduce the results of Figs. 10.3.1 and 10.3.2. The theoretical DTFTs may be computed by 256-point FFTs. The 32-point and 64-point DFTs may be extracted from the 256-point FFTs by keeping every 8th point ($256/32 = 8$) and every 4th point ($256/64 = 4$.)

9.9 A dual-tone multi-frequency (DTMF) transmitter (touch-tone phone) encodes each keypress as a sum of two sinusoidal tones, with one frequency taken from group A and one from group B, where:

$$\text{group A} = 697,\ 770,\ 852,\ 941 \text{ Hz}$$

$$\text{group B} = 1209,\ 1336,\ 1477 \text{ Hz}$$

A digital DTMF receiver computes the spectrum of the received dual-tone signal and determines the two frequencies that are present, and thus, the key that was pressed.

What is the *smallest* number of time samples L that we should collect at a sampling rate of 8 kHz, in order for the group-A frequencies to be resolvable from the group-B frequencies? What is L if a Hamming window is used prior to computing the spectrum?

9.10 Suppose we collect 256 samples of the above DTMF signal and compute a 256-point FFT. Explain why each keypress generates substantial signal energy in 2 out of 7 possible DFT frequency bins (and their negatives). What are the indices k for these 7 bins? [*Hint:* Round k to its nearest integer. Do not ignore negative frequencies.]

Note that in practice, it may be more economical to just compute the value of $X(k)$ at those 14 k's instead of computing a full 256-point FFT.

9.11 *Computer Experiment: DTMF Sinusoids.* Consider the following artificial signal consisting of the sum of all seven DTMF sinusoids:

$$x(t)=\sum_{a=1}^{4}\sin(2\pi f_a t)+\sum_{b=1}^{3}\sin(2\pi f_b t)$$

where f_a and f_b are the group A and B frequencies given in Problem 9.9 and t is in seconds. (In practice, of course, only one f_a term and one f_b term will be present.)

The signal $x(t)$ is sampled at a rate of 8 kHz and 256 samples are collected, say, $x(n)$, $n = 0, 1, \dots, 255$. The spectrum of this signal should consist of seven peaks clustered in two clearly separable groups (and, seven more negative-frequency peaks).

a. Plot the signal $x(n)$ versus n.

b. Compute the 256-point DFT or FFT of the signal $x(n)$ and plot the corresponding magnitude spectrum $|X(f)|$ only over the frequency range $0 \le f \le 4$ kHz.

c. Window the signal $x(n)$ by a length-256 Hamming window $w(n)$, that is, $x_{\text{ham}}(n) = w(n)x(n)$, and plot it versus n. Then, compute its 256-point DFT and plot the magnitude spectrum $|X_{\text{ham}}(f)|$ over $0 \le f \le 4$ kHz.

10

DFT/FFT Algorithms

10.1 Discrete Fourier Transform

In the previous chapter, we defined the DTFT of a length-L signal, $x(n)$, to be,

$$X(\omega) = \sum_{n=0}^{L-1} x(n)\, e^{-j\omega n} \tag{10.1.1}$$

where $\omega = 2\pi f/f_s$ is the digital frequency in [rads/sample], and noted that $X(\omega)$ is periodic in ω with period 2π, so that we only need to evaluate $X(\omega)$ within the symmetric Nyquist interval, $-\pi \le \omega \le \pi$, or, within the right-sided Nyquist interval, $0 \le \omega \le 2\pi$, and mentioned that $X(\omega)$ can be evaluated at any range of ωs using, for example, MATLAB's built-in function **freqz**,

```
% x = ...                % define length-L signal x(n)
% omega = ...            % choose a range of omega's
X = freqz(x,1,omega);    % DTFT X(omega) of same length as omega
```

The N-point *discrete Fourier transform* (DFT) of a length-L signal is defined to be the DTFT evaluated at N equally-spaced frequencies spanning the full right-sided Nyquist interval, $0 \le \omega \le 2\pi$. These so-called "DFT frequencies" are defined in radians per sample as follows:

$$\boxed{\omega_k = \frac{2\pi k}{N}}\,, \qquad k = 0, 1, \dots, N-1 \tag{10.1.2}$$

or, in Hz

$$\boxed{f_k = \frac{k f_s}{N}}\,, \qquad k = 0, 1, \dots, N-1 \tag{10.1.3}$$

Thus, the N-point DFT will be, for $k = 0, 1, \dots, N-1$:

$$\boxed{X(\omega_k) = \sum_{n=0}^{L-1} x(n)\, e^{-j\omega_k n}} \qquad \text{(N-point DFT of length-L signal)} \tag{10.1.4}$$

Usually, $X(\omega_k)$ is denoted by X_k in terms of the DFT index $k = 0, 1, \dots, N-1$. The C-function[†] **dft** can be used to evaluate the array X_k. Alternatively, the MATLAB computation in terms of **freqz** is,

```
% x = ...                  % define length-L signal x(n)
k = 0:N-1;                 % DFT index
wk = 2*pi*k/N;             % N-dimensional vector of DFT frequencies
Xk = freqz(x,1,wk);        % N-dimensional DFT vector
```

Note that the value at $k = N$, corresponding to $\omega_N = 2\pi$, is not computed because by periodicity it equals the value at $\omega_0 = 0$, that is, $X(\omega_N) = X(\omega_0)$, or,

$$X_N = X_0$$

More generally, the periodicity of $X(\omega)$ with period 2π is reflected in the periodicity of the DFT, $X_k = X(\omega_k)$, in the index k with period N, which follows from,

$$\omega_{k+N} = \frac{2\pi(k+N)}{N} = \frac{2\pi k}{N} + 2\pi = \omega_k + 2\pi$$

so that,

$$X_{k+N} = X(\omega_{k+N}) = X(\omega_k + 2\pi) = X(\omega_k) = X_k \tag{10.1.5}$$

A consequence of this property is also the mapping of the negative-half. $-\pi <\leq \omega \leq 0$, of the symmetric Nyquist interval, $-\pi \leq \pi$, into the second-half, $\pi \leq \omega \leq 2\pi$, of the right-sided Nyquist interval, $0 \leq \omega \leq 2\pi$, that is,

$$X_{-k} = X_{N-k}\,, \quad k = 1, 2\dots, N-1 \tag{10.1.6}$$

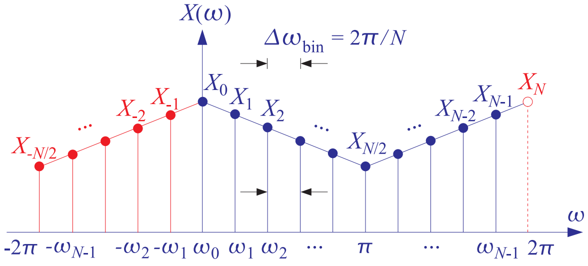

$X(\omega)$
$\Delta\omega_{\text{bin}} = 2\pi/N$
X_0
X_1
X_2
$\dots$
$X_{N/2}$
$\dots$
$X_{N\text{-}2}$
$X_{N\text{-}1}$
X_N
$X_{\text{-}N/2}$
$\dots$
$X_{\text{-}2}$
$X_{\text{-}1}$
ω
$\text{-}2\pi$ $\text{-}\omega_{N\text{-}1}$ $\text{-}\omega_2$ $\text{-}\omega_1$ ω_0 ω_1 ω_2 $\cdots$ π $\cdots$ $\omega_{N\text{-}1}$ 2π

Fig. 10.1.1 Mapping of negative frequencies into positive frequencies in N-point DFT.

so that, for example, $X_{-1} = X_{N-1}$, $X_{-2} = X_{N-2}$, and so on. The symmetric and right-sided Nyquist interval DFT values are displayed in Fig. 10.1.1 (see also Fig. 9.2.1). Also shown is the computational bin width, i.e., the separation between the equally-spaced computed frequency points, $\omega_k = k\Delta\omega_{\text{bin}}$, $k = 0, 1, \dots, N-1$,

$$\Delta\omega_{\text{bin}} = \frac{2\pi}{N} \quad \text{or,} \quad \Delta f_{\text{bin}} = \frac{f_s}{N} \tag{10.1.7}$$

[†] found in the C-function collection of this book.

The built-in MATLAB function, **fftshift**, implements the remapping of the right-half of an ordinary DFT to the negative-half of a symmetric DFT. For example, the following 16-point DFT computed for indices, $k = 0, 1, \dots, 15$,

$$X_k = [\underbrace{16,\ 15,\ 14,\ 13,\ 12,\ 11,\ 10,\ 9,}_{\text{positive-half}}\ \underbrace{8,\ 9,\ 10,\ 11,\ 12,\ 13,\ 14,\ 15}_{\text{negative-half}}]$$

will get remapped to the symmetric one, with indices, $-8 \leq k \leq 7$,

$$X_k^{\text{shifted}} = [\underbrace{8,\ 9,\ 10,\ 11,\ 12,\ 13,\ 14,\ 15,}_{\text{negative-half}}\ \underbrace{16,\ 15,\ 14,\ 13,\ 12,\ 11,\ 10,\ 9}_{\text{positive-half}}]$$

and shown in Fig. 10.1.2, implemented with MATLAB code,

```
X = [16, 15, 14, 13, 12, 11, 10, 9, 8, 9, 10, 11, 12, 13, 14, 15];

Xshifted = fftshift(X);

% resulting into,
% Xshifted = [8, 9, 10, 11, 12, 13, 14, 15, 16, 15, 14, 13, 12, 11, 10, 9]
```

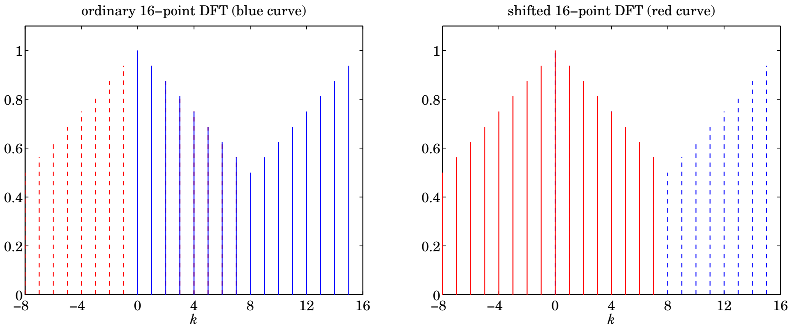


Fig. 10.1.2 FFTSHIFT example.

For the case of real-valued signals $x(n)$, the Hermitian conjugation symmetry property of the DTFT, $X^*(\omega) = X(-\omega)$, can be combined with the periodicity property (10.1.6) resulting in,

$$X_k^* = X_{-k} = X_{N-k}\,, \quad k = 0, 1, 2\dots, N-1 \tag{10.1.8}$$

so that for example, $X_0^* = X_0 = X_N$ (i.e., real-valued), $X_1^* = X_{N-1}$, $X_2^* = X_{N-2}$, etc.

The N computed values $X(\omega_k)$ can also be thought of as the evaluation of the z-transform $X(z)$ at the following z-points on the unit circle:

$$X(\omega_k) = X(z_k) = \sum_{n=0}^{L-1} x(n) z_k^{-n} \tag{10.1.9}$$

where

$$\boxed{z_k = e^{j\omega_k} = e^{2\pi jk/N}}\,, \qquad k = 0, 1, \dots, N-1 \tag{10.1.10}$$

These are recognized as the Nth *roots of unity*, that is, the N solutions of the equation $z^N = 1$. They are evenly spaced around the unit circle at relative angle increments of $2\pi/N$, as shown in Fig. 10.1.3.

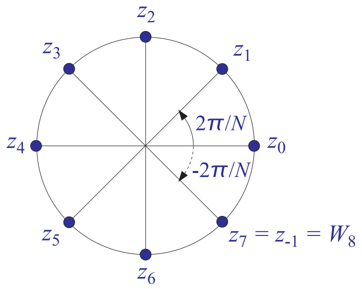


Fig. 10.1.3 Nth roots of unity, for $N = 8$.

10.2 Zero Padding

In principle, the two lengths L and N can be specified *independently* of each other: L is the number of *time samples* in the data record and can even be infinite; N is the number of *frequencies* at which we choose to evaluate the DTFT.

Most discussions of the DFT assume that $L = N$. The reason for this will be discussed later. If $L < N$, we can pad $N - L$ zeros at the end of the data record to make it of length N. If $L > N$, we may reduce the data record to length N by *wrapping* it modulo-N—a process to be discussed in Section 10.5.

Padding any number of zeros at the *end* of a signal has no effect on its DTFT. For example, padding D zeros will result into a length-$(L+D)$ signal:

$$\begin{aligned}
\mathbf{x} &= [x_0, x_1, \dots, x_{L-1}] \\
\mathbf{x}_D &= [x_0, x_1, \dots, x_{L-1}, \underbrace{0, 0, \dots, 0}_{D \text{ zeros}}]
\end{aligned}$$

Because $x_D(n) = x(n)$ for $0 \le n \le L-1$ and $x_D(n) = 0$ for $L \le n \le L+D-1$, the corresponding DTFTs will remain the same:

$$\begin{aligned}
X_D(\omega) &= \sum_{n=0}^{L+D-1} x_D(n)e^{-j\omega n} = \sum_{n=0}^{L-1} x_D(n)e^{-j\omega n} + \sum_{n=L}^{L+D-1} x_D(n)e^{-j\omega n} \\
&= \sum_{n=0}^{L-1} x(n)e^{-j\omega n} = X(\omega)
\end{aligned}$$

Therefore, their evaluation at the N DFT frequencies will be the same: $X_D(\omega_k) = X(\omega_k)$. We note also that padding the D zeros to the *front* of the signal will be equivalent to a delay by D samples, which in the z-domain corresponds to multiplication by z^{-D} and in the frequency domain by $e^{-j\omega D}$. Therefore, the signals:

$$\begin{aligned}
\mathbf{x} &= [x_0, x_1, \dots, x_{L-1}] \\
\mathbf{x}_D &= [\underbrace{0, 0, \dots, 0}_{D \text{ zeros}}, x_0, x_1, \dots, x_{L-1}]
\end{aligned} \tag{10.2.1}$$

will have DTFTs and DFTs:

$$\begin{aligned}
X_D(\omega) &= e^{-j\omega D}X(\omega) \\
X_D(\omega_k) &= e^{-j\omega_k D}X(\omega_k), \quad k = 0, 1, \dots, N-1
\end{aligned} \tag{10.2.2}$$

10.3 Physical versus Computational Resolution

The bin width Δf_{bin} represents the spacing between the DFT frequencies at which the DTFT is computed and must not be confused with the frequency resolution width $\Delta f = f_s/L$ of Eq. (9.1.14), which refers to the minimum resolvable frequency separation between two sinusoidal components. To avoid confusion, we will refer to Eq. (9.1.14) as the *physical* frequency resolution and to Eq. (10.1.7) as the *computational* frequency resolution.

The interplay between physical and computational resolution is illustrated in Fig. 10.3.1 for the triple sinusoidal signal of Example 9.1.4. The $N = 32$ and $N = 64$ point DFTs of the rectangularly windowed signals of lengths $L = 10$ and $L = 20$ are shown together with their full DTFTs (computed here as 256-point DFTs).

It is evident from these graphs that if the length L of the signal is not large enough to provide sufficient physical resolution, then there is no point increasing the length N of the DFT—that would only put more points on the wrong curve.

Another issue related to physical and computational resolution is the question of how accurately the DFT represents the peaks in the spectrum. For each sinusoid that is present in the signal, say, at frequency f_0, the DTFT will exhibit a mainlobe peak arising from the shifted window $W(f - f_0)$. When we evaluate the N-point DFT, we would like the peak at f_0 to *coincide* with one of the N DFT frequencies (10.1.3). This will happen if there is an integer $0 \le k_0 \le N-1$, such that

$$f_0 = f_{k_0} = \frac{k_0 f_s}{N} \qquad \Rightarrow \qquad \boxed{k_0 = N\frac{f_0}{f_s}} \tag{10.3.1}$$

Similarly, the peak at the negative frequency, $-f_0$, or at the equivalent shifted one, $f_s - f_0$, will correspond to the integer, $-k_0$, or to the shifted one $N - k_0$:

$$-f_0 = -k_0\frac{f_s}{N} \qquad \Rightarrow \qquad f_s - f_0 = f_s - k_0\frac{f_s}{N} = (N - k_0)\frac{f_s}{N}$$

In summary, for each sinusoid with peaks at $\pm f_0$, we would like our DFT to show these peaks at the integers:

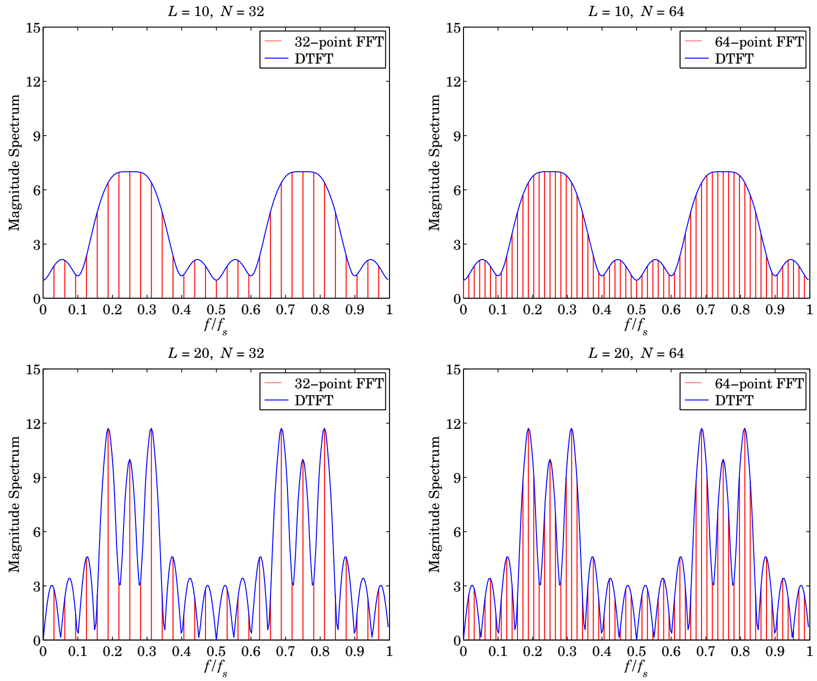


Fig. 10.3.1 Physical versus computational resolution in DTFT computation.

$$\{f_0,\,-f_0\} \quad\Rightarrow\quad \{f_0,\,f_s-f_0\} \quad\Rightarrow\quad \{k_0,\,N-k_0\} \tag{10.3.2}$$

In general, this is not possible because k_0 computed from Eq. (10.3.1) is not an integer, and the DFT will miss the exact peaks. However, for large N, we may *round* k_0 to the nearest integer and use the corresponding DFT frequency as an *estimate* of the actual peak frequency.

A pitfall of using the DFT can be seen in the lower two graphs of Fig. 10.3.1, where it appears that the DFT correctly identifies the three peaks in the spectrum, for both $N = 32$ and $N = 64$.

However, this is misleading for two reasons: First, it is a numerical accident in this example that the mainlobe maxima coincide with the DFT frequencies. Second, it can be seen in the figure that these maxima correspond to the *wrong* frequencies and not to the correct ones, which are:

$$\frac{f_1}{f_s} = 0.20, \quad \frac{f_2}{f_s} = 0.25, \quad \frac{f_3}{f_s} = 0.30 \tag{10.3.3}$$

This phenomenon, whereby the maxima of the peaks in the spectrum do not quite correspond to the correct frequencies, is called *biasing* and is caused by the lack of adequate *physical resolution*, especially when the sinusoidal frequencies are too close to each other and the sum of terms $W(f-f_0)$ interact strongly.

Using Eq. (10.3.1), we can calculate the DFT indices k and $N-k$ to which the true frequencies (10.3.3) correspond. For $N = 32$, we have:

$$k_1 = N\frac{f_1}{f_s} = 32\cdot 0.20 = 6.4, \quad N-k_1 = 25.6$$

$$k_2 = N\frac{f_2}{f_s} = 32\cdot 0.25 = 8, \quad N-k_2 = 24$$

$$k_3 = N\frac{f_3}{f_s} = 32\cdot 0.30 = 9.6, \quad N-k_3 = 22.4$$

Similarly, for $N = 64$, we find:

$$k_1 = N\frac{f_1}{f_s} = 64\cdot 0.20 = 12.8, \quad N-k_1 = 51.2$$

$$k_2 = N\frac{f_2}{f_s} = 64\cdot 0.25 = 16, \quad N-k_2 = 48$$

$$k_3 = N\frac{f_3}{f_s} = 64\cdot 0.30 = 19.2, \quad N-k_3 = 44.8$$

Only the middle one at f_2 corresponds to an integer, and therefore, coincides with a DFT value. The other two are missed by the DFT. We may round k_1 and k_3 to their nearest integers and then compute the corresponding DFT frequencies. We find for $N = 32$:

$$k_1 = 6.4 \quad\Rightarrow\quad k_1 = 6 \quad\Rightarrow\quad \frac{f_1}{f_s} = \frac{k_1}{N} = 0.1875$$

$$k_3 = 9.6 \quad\Rightarrow\quad k_3 = 10 \quad\Rightarrow\quad \frac{f_3}{f_s} = \frac{k_3}{N} = 0.3125$$

and for $N = 64$:

$$k_1 = 12.8 \quad\Rightarrow\quad k_1 = 13 \quad\Rightarrow\quad \frac{f_1}{f_s} = \frac{k_1}{N} = 0.203125$$

$$k_3 = 19.2 \quad\Rightarrow\quad k_3 = 19 \quad\Rightarrow\quad \frac{f_3}{f_s} = \frac{k_3}{N} = 0.296875$$

The *rounding error* in the frequencies remains less than $f_s/2N$. It decreases with increasing DFT length N. The *biasing error*, on the other hand, can only be decreased by increasing the data length L.

Figure 10.3.2 shows the spectrum of the same signal of Example 9.1.4, but with length $L = 100$ samples. Biasing is virtually eliminated with the peak maxima at the correct frequencies. The spectrum is plotted versus the DFT index k, which is proportional to the frequency f via the mapping (10.1.3), or

$$\boxed{k = N\frac{f}{f_s}} \qquad \text{(frequency in units of the DFT index)} \tag{10.3.4}$$

The Nyquist interval $0 \le f \le f_s$ corresponds to the index interval $0 \le k \le N$. The N-point DFT is at the integer values $k = 0, 1, \dots, N-1$. For plotting purposes, the graph of the spectrum over the full interval $0 \le k \le N$ has been split into two side-by-side graphs covering the half-intervals: $0 \le k \le N/2$ and $N/2 \le k \le N$.

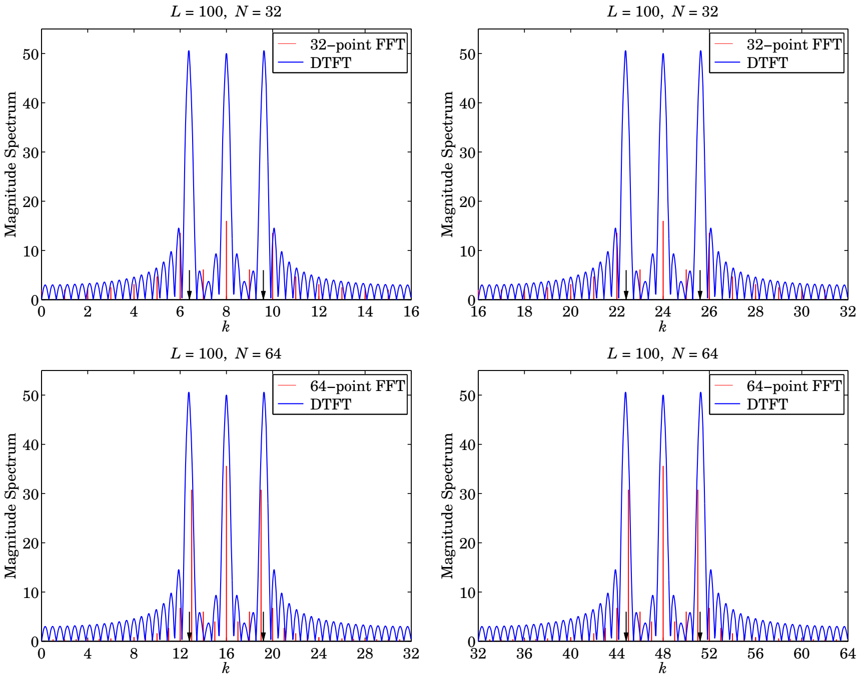


Fig. 10.3.2 DFT can miss peaks in the spectrum.

In the upper two graphs having $N = 32$, the DFT misses the f_1 and f_3 peaks completely (the peak positions are indicated by the arrows). The actual peaks are so narrow that they fit completely within the computational resolution width Δf_{bin}.

In the lower two graphs having $N = 64$, the DFT still misses these peaks, but less so. Further doubling of N will interpolate half-way between the frequencies of the 64-point case resulting in a better approximation.

Example 10.3.1: A 5 kHz sinusoidal signal is sampled at 40 kHz and 128 samples are collected and used to compute the 128-point DFT of the signal. What is the time duration in seconds of the collected samples? At what DFT indices do we expect to see any peaks in the spectrum?

Solution: The time duration is $T_N = NT = N/f_s = 128/40 = 3.2$ msec. Using Eq. (10.3.1), we calculate $k = Nf/f_s = 128 \cdot 5/40 = 16$. The negative frequency -5 kHz is represented by the DFT index $N - k = 128 - 16 = 112$. □

Example 10.3.2: A 10 msec segment of a signal is sampled at a rate of 10 kHz and the resulting samples are saved. It is desired to compute the spectrum of that segment at 128 equally spaced frequencies covering the range $2.5 \le f < 5$ kHz. We would like to use an off-the-shelf N-point FFT routine to perform this computation. The routine takes as input an N-dimensional vector $\mathbf{x}$ of time samples. Its output is an N-dimensional DFT vector $\mathbf{X}$. (a) What value of N should we use? (b) How is the routine's input vector $\mathbf{x}$ defined in terms of the time samples that we collected? (c) Exactly what DFT indices k and DFT values $X[k]$ correspond to the 128 spectral values that we wish to compute?

Solution: The interval $[2.5, 5]$ kHz is one-quarter the Nyquist interval $[0, 10]$ kHz. Thus, the DFT size should be $N = 4\times 128 = 512$. This choice places 128 frequencies over the $[2.5, 5)$ interval. Another way is to identify the bin width over the $[2.5, 5]$ subinterval with the bin width over the full interval:

$$\Delta f_{\text{bin}} = \frac{5 - 2.5}{128} = \frac{10}{N} \qquad \Rightarrow \qquad N = 512$$

The number of collected samples is $L = T_L f_s = (10 \text{ msec})\times(10 \text{ kHz}) = 100$. Thus, the subroutine's 512-dimensional input vector $\mathbf{x}$ will consist of the 100 input samples with 412 zeros padded at the end.

Because the range $[2.5, 5)$ is the second quarter of the Nyquist interval, it will be represented by the second quarter of DFT indices, that is, $128 \le k < 256$. □

10.4 Matrix Form of DFT

The N-point DFT (10.1.4) can be thought of as a linear *matrix transformation* of the L-dimensional vector of time data into an N-dimensional vector of frequency data:

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{L-1} \end{bmatrix} \xrightarrow{\text{DFT}} \mathbf{X} = \begin{bmatrix} X_0 \\ X_1 \\ \vdots \\ X_{N-1} \end{bmatrix}$$

where we denoted the DFT components by $X_k = X(\omega_k)$, $k = 0, 1, \dots, N-1$.

The linear transformation is implemented by an $N\times L$ matrix A, to be referred to as the *DFT matrix*, and can be written compactly as follows:

$$\boxed{\mathbf{X} = \text{DFT}(\mathbf{x}) = A\mathbf{x}} \qquad \text{(matrix form of DFT)} \tag{10.4.1}$$

or, component-wise:

$$\boxed{X_k = \sum_{n=0}^{L-1} A_{kn} x_n}, \qquad k = 0, 1, \dots, N-1 \tag{10.4.2}$$

The matrix elements A_{kn} are defined from Eq. (10.1.4):

$$\boxed{A_{kn} = e^{-j\omega_k n} = e^{-2\pi jkn/N} = W_N^{kn}} \qquad N\times L \text{ DFT matrix} \tag{10.4.3}$$

for $0 \le k \le N-1$ and $0 \le n \le L-1$. For convenience, we defined the so-called *twiddle factor* W_N as the complex number:

$$\boxed{W_N = e^{-2\pi j/N}} \tag{10.4.4}$$

Thus, the DFT matrix for an N-point DFT is built from the powers of W_N. Note that the first row $(k=0)$ and first column $(n=0)$ of A are always unity:

$$A_{0n} = 1, \quad 0 \le n \le L-1 \qquad \text{and} \qquad A_{k0} = 1, \quad 0 \le k \le N-1$$

The matrix A can be built from its second row $(k=1)$, consisting of the successive powers of W_N:

$$A_{1n} = W_N^n, \qquad n = 0, 1, \dots, L-1$$

It follows from the definition that the kth row is obtained by raising the second row to the kth power—element by element:

$$A_{kn} = W_N^{kn} = \left(W_N^n\right)^k = A_{1n}^k$$

Some examples of twiddle factors, DFT matrices, and DFTs are as follows: For $L = N$ and $N = 2, 4, 8$, we have:

$$\begin{aligned}
W_2 &= e^{-2\pi j/2} = e^{-\pi j} = -1 \\
W_4 &= e^{-2\pi j/4} = e^{-\pi j/2} = \cos(\pi/2) - j\sin(\pi/2) = -j \\
W_8 &= e^{-2\pi j/8} = e^{-\pi j/4} = \cos(\pi/4) - j\sin(\pi/4) = \frac{1-j}{\sqrt{2}}
\end{aligned} \tag{10.4.5}$$

The corresponding 2-point and 4-point DFT matrices are:

$$\begin{aligned}
A &= \begin{bmatrix} 1 & 1 \\ 1 & W_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \\
A &= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W_4 & W_4^2 & W_4^3 \\ 1 & W_4^2 & W_4^4 & W_4^6 \\ 1 & W_4^3 & W_4^6 & W_4^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}
\end{aligned} \tag{10.4.6}$$

And, the 2-point and 4-point DFTs of a length-2 and a length-4 signal will be:

$$\begin{aligned}
\begin{bmatrix} X_0 \\ X_1 \end{bmatrix} &= \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} x_0 + x_1 \\ x_0 - x_1 \end{bmatrix} \\
\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} &= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}
\end{aligned} \tag{10.4.7}$$

Thus, the 2-point DFT is formed by taking the sum and difference of the two time samples. We will see later that the 2-point DFT is a convenient starting point for the merging operation in performing the FFT by hand.

The twiddle factor W_N satisfies $W_N^N = 1$, and therefore it is one of the Nth roots of unity; indeed, in the notation of Eq. (10.1.10), it is the root $W_N = z_{N-1}$ and is shown in Fig. 10.1.3. Actually, all the successive powers W_N^k, $k = 0, 1, \dots, N-1$ are Nth roots of unity, but in *reverse order* (i.e., clockwise) than the z_k of Eq. (10.1.10):

$$W_N^k = e^{-2\pi jk/N} = z_{-k} = z_k^{-1}, \quad k = 0, 1, \dots, N-1 \tag{10.4.8}$$

Figure 10.4.1 shows W_N and its successive powers for the values $N = 2, 4, 8$. Because $W_N^N = 1$, the exponents in W_N^{kn} can be reduced modulo-N, that is, we may replace them by $W_N^{(nk)\,\mathrm{mod}\,(N)}$.

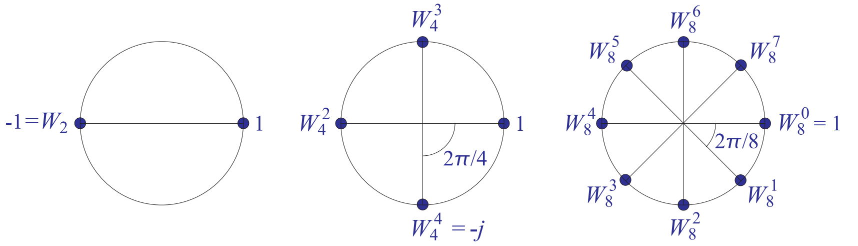


Fig. 10.4.1 Twiddle factor lookup tables for $N = 2, 4, 8$.

For example, using the property $W_4^4 = 1$, we may reduce all the powers of W_4 in the 4-point DFT matrix of Eq. (10.4.6) to one of the four powers W_4^k, $k = 0, 1, 2, 3$ and write it as

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W_4 & W_4^2 & W_4^3 \\ 1 & W_4^2 & W_4^4 & W_4^6 \\ 1 & W_4^3 & W_4^6 & W_4^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W_4 & W_4^2 & W_4^3 \\ 1 & W_4^2 & 1 & W_4^2 \\ 1 & W_4^3 & W_4^2 & W_4 \end{bmatrix}$$

The entries in A can be read off from the circular lookup table of powers of W_4 in Fig. 10.4.1, giving,

$$W_4 = -j, \quad W_4^2 = -1, \quad W_4^3 = j$$

and for $N = 8$,

$$W_8 = \frac{1-j}{\sqrt{2}}, \quad W_8^2 = -j, \quad W_8^3 = \frac{-1-j}{\sqrt{2}}, \quad W_8^4 = -1$$

$$W_8^5 = \frac{-1+j}{\sqrt{2}}, \quad W_8^6 = j, \quad W_8^7 = \frac{1+j}{\sqrt{2}}$$

The DFT matrix (10.4.3) can be implemented very simply in MATLAB by the anonymous function,

```
DFTmat = @(N,L) exp(-2*pi*j*(0:N-1)'/N * (0:L-1));   % Usage: A = DFTmat(N,L)
```

10.5 Modulo-*N* Reduction

The *modulo-N reduction* or *wrapping* of a signal plays a fundamental part in the theory of the DFT. It is defined by dividing the signal $\mathbf{x}$ into contiguous non-overlapping blocks of length N, wrapping the blocks around to be time-aligned with the first block, and adding them up. The process is illustrated in Fig. 10.5.1. The resulting wrapped block $\tilde{\mathbf{x}}$ has length N.

The length L of the signal $\mathbf{x}$ could be finite or infinite. If L is not an integral multiple of N, then the last sub-block will have length less than N; in this case, we may pad enough zeros at the end of the last block to increase its length to N.

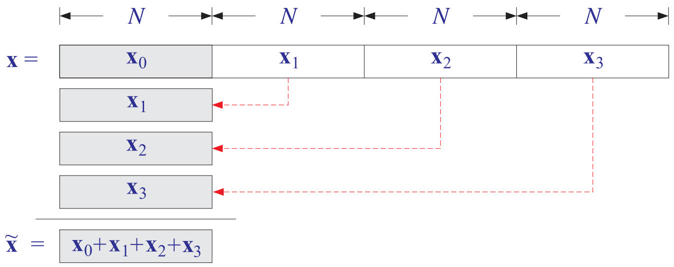


Fig. 10.5.1 Modulo-N reduction of a signal.

The wrapping process can also be thought of as *partitioning* the signal vector $\mathbf{x}$ into N-dimensional subvectors and adding them up. For example, if $L = 4N$, the signal $\mathbf{x}$ will consist of four length-N subvectors:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \end{bmatrix} \quad \Rightarrow \quad \tilde{\mathbf{x}} = \mathbf{x}_0 + \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 \tag{10.5.1}$$

Example 10.5.1: Determine the mod-4 and mod-3 reductions of the length-8 signal vector:

$$\mathbf{x} = [1,\, 2,\, -2,\, 3,\, 4,\, -2,\, -1,\, 1]^T$$

For the $N = 4$ case, we may divide $\mathbf{x}$ into two length-4 sub-blocks to get:

$$\tilde{\mathbf{x}} = \begin{bmatrix} 1 \\ 2 \\ -2 \\ 3 \end{bmatrix} + \begin{bmatrix} 4 \\ -2 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \\ -3 \\ 4 \end{bmatrix}$$

Similarly, for $N = 3$ we divide $\mathbf{x}$ into length-3 blocks:

$$\tilde{\mathbf{x}} = \begin{bmatrix} 1 \\ 2 \\ -2 \end{bmatrix} + \begin{bmatrix} 3 \\ 4 \\ -2 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 7 \\ -4 \end{bmatrix}$$

where we padded a zero at the end of the third sub-block. □

We may express the sub-block components in terms of the time samples of the signal $x(n)$, $0 \le n \le L-1$, as follows. For $m = 0, 1, \dots$

$$x_m(n) = x(mN + n), \quad n = 0, 1, \dots, N-1 \tag{10.5.2}$$

Thus, the mth sub-block occupies the time interval $[mN,\, (m+1)N)$. The wrapped vector $\tilde{\mathbf{x}}$ will be in this notation:

$$\begin{aligned} \tilde{x}(n) &= x_0(n) + x_1(n) + x_2(n) + x_3(n) + \cdots \\ &= x(n) + x(N+n) + x(2N+n) + x(3N+n) + \cdots \end{aligned} \tag{10.5.3}$$

for $n = 0, 1, \dots, N-1$, or, more compactly

$$\boxed{\tilde{x}(n) = \sum_{m=0}^{\infty} x(mN + n)}\,, \qquad n = 0, 1, \dots, N-1 \tag{10.5.4}$$

This expression can be used to define $\tilde{x}(n)$ for all n, not just $0 \le n \le N-1$. The resulting double-sided infinite signal is the so-called *periodic extension* of the signal $x(n)$ with period N. More generally, it is defined by

$$\tilde{x}(n) = \sum_{m=-\infty}^{\infty} x(mN + n), \qquad -\infty < n < \infty \tag{10.5.5}$$

The signal $\tilde{x}(n)$ is periodic in n with period N, that is, $\tilde{x}(n+N) = \tilde{x}(n)$. The definition (10.5.4) evaluates only one basic period $0 \le n \le N-1$ of $\tilde{x}(n)$, which is all that is needed in the DFT.

The periodic extension interpretation of mod-N reduction is shown in Fig. 10.5.2. The terms $x(n+N)$, $x(n+2N)$, and $x(n+3N)$ of Eq. (10.5.3) can be thought as the time-advanced or left-shifted versions of $x(n)$ by N, $2N$, and $3N$ time samples. The successive sub-blocks of $x(n)$ get time-aligned one under the other over the basic period $0 \le n \le N-1$, thus, their sum is the wrapped signal $\tilde{\mathbf{x}}$.

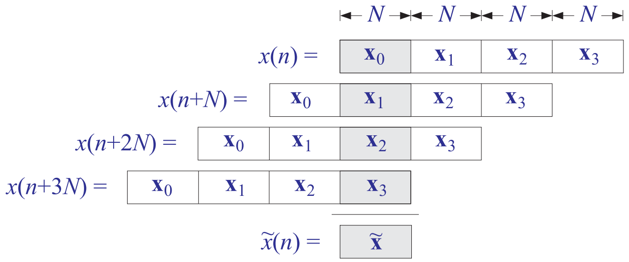


Fig. 10.5.2 Periodic extension interpretation of mod-N reduction of a signal.

The connection of the mod-N reduction to the DFT is the theorem that the length-N wrapped signal $\tilde{\mathbf{x}}$ has the *same* N-point DFT as the original unwrapped signal $\mathbf{x}$, that is,

$$\boxed{\tilde{X}_k = X_k \quad \text{or,} \quad \tilde{X}(\omega_k) = X(\omega_k)}\,, \qquad k = 0,1,\dots,N-1 \tag{10.5.6}$$

where $\tilde{X}_k = \tilde{X}(\omega_k)$ is the N-point DFT of the length-N signal $\tilde{x}(n)$:

$$\tilde{X}_k = \tilde{X}(\omega_k) = \sum_{n=0}^{N-1} \tilde{x}(n)e^{-j\omega_k n}\,, \qquad k = 0,1,\dots,N-1 \tag{10.5.7}$$

In the notation of Eq. (10.4.2), we may write:

$$\tilde{X}_k = \sum_{n=0}^{N-1} W_N^{kn}\tilde{x}(n) = \sum_{n=0}^{N-1} \tilde{A}_{kn}\tilde{x}(n) \tag{10.5.8}$$

where $\tilde{A}$ is the DFT matrix defined as in Eq. (10.4.3):

$$\tilde{A}_{kn} = W_N^{kn}\,, \qquad 0 \le k \le N-1\,, \quad 0 \le n \le N-1 \tag{10.5.9}$$

The DFT matrices A and $\tilde{A}$ have the same definition, except they differ in their dimensions, which are $N\times L$ and $N\times N$, respectively. We can write the DFT of $\tilde{\mathbf{x}}$ in the compact matrix form:

$$\tilde{\mathbf{X}} = \text{DFT}(\tilde{\mathbf{x}}) = \tilde{A}\tilde{\mathbf{x}} \tag{10.5.10}$$

Thus, the above theorem can be stated in vector form:

$$\tilde{\mathbf{X}} = \mathbf{X} = A\mathbf{x} = \tilde{A}\tilde{\mathbf{x}} \tag{10.5.11}$$

Symbolically, we will write $\text{DFT}(\tilde{\mathbf{x}}) = \text{DFT}(\mathbf{x})$ to denote Eqs. (10.5.6) or (10.5.11). The above theorem can be proved in many ways. In matrix form, it follows from the property that the $N\times N$ submatrices of the full $N\times L$ DFT matrix A are all *equal* to the DFT matrix $\tilde{A}$.

These submatrices are formed by grouping the first N columns of A into the first submatrix, the next N columns into the second submatrix, and so on. The matrix elements of the mth submatrix will be:

$$A_{k,mN+n} = W_N^{k(mN+n)} = W_N^{mkN}W_N^{kn}$$

Using the property $W_N^N = 1$, it follows that $W_N^{kmN} = 1$, and therefore:

$$A_{k,mN+n} = W_N^{kn} = A_{kn} = \tilde{A}_{kn}\,, \qquad 0 \le k,n \le N-1$$

Thus, in general, A is partitioned in the form:

$$A = [\tilde{A}, \tilde{A}, \tilde{A}, \dots] \tag{10.5.12}$$

As an example, consider the case $L = 8$, $N = 4$. The 4×8 DFT matrix A can be partitioned into two 4×4 identical submatrices, which are equal to $\tilde{A}$. Using $W_4 = e^{-2\pi j/4} = -j$, we have:

$$A = \left[\begin{array}{cccc|cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & W_4 & W_4^2 & W_4^3 & W_4^4 & W_4^5 & W_4^6 & W_4^7 \\ 1 & W_4^2 & W_4^4 & W_4^6 & W_4^8 & W_4^{10} & W_4^{12} & W_4^{14} \\ 1 & W_4^3 & W_4^6 & W_4^9 & W_4^{12} & W_4^{15} & W_4^{18} & W_4^{21} \end{array}\right]$$

$$= \left[\begin{array}{cccc|cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & W_4 & W_4^2 & W_4^3 & 1 & W_4 & W_4^2 & W_4^3 \\ 1 & W_4^2 & W_4^4 & W_4^6 & 1 & W_4^2 & W_4^4 & W_4^6 \\ 1 & W_4^3 & W_4^6 & W_4^9 & 1 & W_4^3 & W_4^6 & W_4^9 \end{array}\right]$$

$$= \left[\begin{array}{cccc|cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j & 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j & 1 & j & -1 & -j \end{array}\right] = [\tilde{A}, \tilde{A}]$$

where in the second submatrix, we partially reduced the powers of W_4 modulo-4.

The proof of the theorem follows now as a simple consequence of this partitioning property. For example, we have for the N-point DFT of Eq. (10.5.1):

$$\mathbf{X} = A\mathbf{x} = [\tilde{A}, \tilde{A}, \tilde{A}, \tilde{A}]\begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \end{bmatrix} = \tilde{A}\mathbf{x}_0 + \tilde{A}\mathbf{x}_1 + \tilde{A}\mathbf{x}_3 + \tilde{A}\mathbf{x}_3$$

$$= \tilde{A}(\mathbf{x}_0 + \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3) = \tilde{A}\tilde{\mathbf{x}} = \tilde{\mathbf{X}}$$

Figure 10.5.3 illustrates the relative dimensions of these operations. The DFT (10.5.10) of $\tilde{\mathbf{x}}$ requires N^2 complex multiplications, whereas that of $\mathbf{x}$ requires NL. Thus, if $L > N$, it is more efficient to first wrap the signal mod-N and then take its DFT.

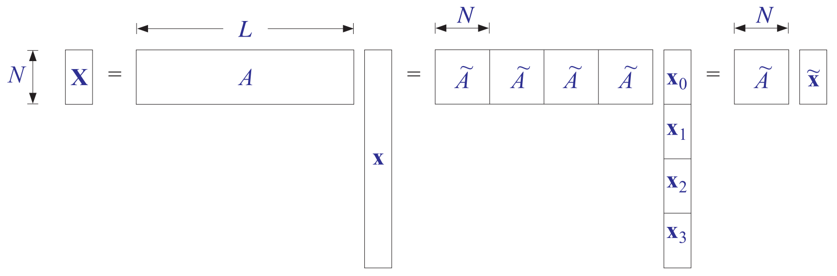


Fig. 10.5.3 N-point DFTs of the full and wrapped signals are equal.

Example 10.5.2: Compute the 4-point DFT of the length-8 signal of Example 10.5.1 in two ways: (a) working with the full unwrapped vector $\mathbf{x}$ and (b) computing the DFT of its mod-4 reduction.

Solution: The 4×8 DFT matrix was worked out above. The corresponding DFT is:

$$\mathbf{X}=A\mathbf{x}=\begin{bmatrix}1&1&1&1&1&1&1&1\\1&-j&-1&j&1&-j&-1&j\\1&-1&1&-1&1&-1&1&-1\\1&j&-1&-j&1&j&-1&-j\end{bmatrix}\begin{bmatrix}1\\2\\-2\\3\\4\\-2\\-1\\1\end{bmatrix}=\begin{bmatrix}6\\8+4j\\-2\\8-4j\end{bmatrix}$$

The same DFT can be computed by the DFT matrix $\widetilde{A}$ acting on the wrapped signal $\widetilde{\mathbf{x}}$, determined in Example 10.5.1:

$$\widetilde{\mathbf{X}}=\widetilde{A}\widetilde{\mathbf{x}}=\begin{bmatrix}1&1&1&1\\1&-j&-1&j\\1&-1&1&-1\\1&j&-1&-j\end{bmatrix}\begin{bmatrix}5\\0\\-3\\4\end{bmatrix}=\begin{bmatrix}6\\8+4j\\-2\\8-4j\end{bmatrix}$$

The two methods give identical results. □

Example 10.5.3: The length L of the signal $\mathbf{x}$ can be infinite, as long as the signal is stable, so that the sum (10.5.4) converges. To illustrate the theorem (10.5.6) or (10.5.11), consider the causal signal $x(n)=a^n u(n)$, where $|a|<1$.

To compute its N-point DFT, we determine its z-transform and evaluate it at the Nth root of unity points $z_k=e^{j\omega_k}=e^{2\pi jk/N}$. This gives:

$$X(z)=\frac{1}{1-az^{-1}}\quad\Rightarrow\quad X_k=X(z_k)=\frac{1}{1-az_k^{-1}},\quad k=0,1,\dots,N-1$$

Next, we compute its mod-N reduction by the sum (10.5.4):

$$\tilde{x}(n)=\sum_{m=0}^{\infty}x(mN+n)=\sum_{m=0}^{\infty}a^{mN}a^n=\frac{a^n}{1-a^N},\quad n=0,1,\dots,N-1$$

where we used the geometric series sum. Computing its z-transform, we find:

$$\widetilde{X}(z)=\sum_{n=0}^{N-1}\tilde{x}(n)z^{-n}=\frac{1}{1-a^N}\sum_{n=0}^{N-1}a^n z^{-n}=\frac{1-a^Nz^{-N}}{(1-a^N)(1-az^{-1})}$$

Evaluating it at $z=z_k$ and using the property that $z_k^N=1$, we find

$$\widetilde{X}_k=\widetilde{X}(z_k)=\frac{1-a^Nz_k^{-N}}{(1-a^N)(1-az_k^{-1})}=\frac{1-a^N}{(1-a^N)(1-az_k^{-1})}=\frac{1}{1-az_k^{-1}}=X_k$$

Thus, even though $x(n)$ and $\tilde{x}(n)$ are different and have different z-transforms and DTFTs, their N-point DFTs are the same. □

The following C routine `modwrap.c` implements the modulo-N reduction operation. If $L<N$, it pads $N-L$ zeros at the end of $\mathbf{x}$ so that $\widetilde{\mathbf{x}}$ will have length N. If $L>N$, it determines how many length-N blocks fit into L, and adds them up, also taking into account the few excess points at the end.

```
/* modwrap.c - modulo-N wrapping of length-L signal */

void modwrap(L, x, N, xtilde)                    usage: modwrap(L, x, N, xtilde);
int L, N;                                        x is L-dimensional
double *x, *xtilde;                              xtilde is N-dimensional
{
    int n, r, m, M;

    r = L % N;                                   remainder r = 0, 1, ..., N − 1
    M = (L-r) / N;                               quotient of division L/N

    for (n=0; n<N; n++) {
         if (n < r)                              non-zero part of last block
              xtilde[n] = x[M*N+n];              if L < N, this is the only block
         else
              xtilde[n] = 0;                     if L < N, pad N − L zeros at end

         for (m=M-1; m>=0; m--)                  remaining blocks
               xtilde[n] += x[m*N+n];            if L < N, this loop is skipped
         }
}
```

Using this routine, we may compute the N-point DFT of a length-L signal by first wrapping it modulo-N and then computing the N-point DFT of the wrapped signal:

```
modwrap(L, x, N, xtilde);                 wrap input modulo-N
dft(N, xtilde, N, X);                     DFT(x̃) = DFT(x)
```

Assuming L is a multiple of N, $L=MN$, the computational cost of the routine `modwrap` is $N(M-1)\simeq MN$ MACs, whereas that of the above `dft` is N^2 MACS. Thus, the total cost of computing the DFT is N^2+MN MACs. This is to be compared to $LN=MN^2$ MACs for the routine `dft` acting on the full length-L input. Replacing the above DFT by an FFT routine gives an even more efficient implementation, requiring $N\log_2(N)/2+MN$ operations.

The built-in MATLAB function **datawrap** also implements the wrapping process. The FFT function in MATLAB, $\mathbf{X}=\text{fft}(\mathbf{x},N)$, has the peculiar syntax that it truncates a length-L input signal vector $\mathbf{x}$ to length N, if $N<L$, discarding those parts of $\mathbf{x}$ that would be wrapped mod-N. Thus, the correct syntax for computing the N=point DFT of a signal using the FFT function is,

```
xtilde = datawrap(x,N);       % wrap signal mod-N
X = fft(xtilde, N);           % compute N-point DFT of wrapped signal
```

Example 10.5.4: Compare the cost of computing the 128-point DFT of a length-1024 signal, using a direct DFT, a prewrapped DFT, and a prewrapped FFT.

The number of length-N segments is $M=L/N=1024/128=8$. The cost of wrapping the signal to length 128 is $N(M-1)=896$. The cost of the three methods will be:

$$\begin{aligned}
&\text{(direct DFT)} && LN = 1024\cdot 128 = 131{,}072\\
&\text{(wrapped DFT)} && N^2 + N(M-1) = 128^2 + 128\cdot(8-1) = 17{,}280\\
&\text{(wrapped FFT)} && \frac{1}{2}N\log_2(N) + N(M-1) = \frac{1}{2}\cdot 128\cdot 7 + 128\cdot(8-1) = 1{,}344
\end{aligned}$$

where we assumed that all the MAC operations are complex-valued. We may also compare the above with the cost of a direct 1024-point FFT on the 1024-point input:

$$\text{(1024-point FFT)} \qquad \frac{1}{2}L\log_2(L) = \frac{1}{2}\cdot 1024\cdot 10 = 5{,}120$$

The DFT frequencies of the desired 128-point DFT are a *subset* of the DFT frequencies of the 1024-point DFT; indeed, we have:

$$\omega_k = \frac{2\pi k}{128} = \frac{2\pi(8k)}{1024}, \qquad k = 0, 1, \dots, 127$$

Thus, the 128-point DFT can be extracted from the 1024-point FFT by taking every eighth entry, that is, $X_{128}(k) = X_{1024}(8k)$. □

The two signals $\mathbf{x}$ and $\tilde{\mathbf{x}}$ are not the only ones that have a common DFT. Any other signal that has the *same* mod-N reduction as $\mathbf{x}$ will have the same DFT as $\mathbf{x}$. To see this, consider a length-L signal $\mathbf{y}$ such that $\tilde{\mathbf{y}} = \tilde{\mathbf{x}}$; then its N-point DFT can be obtained by applying Eq. (10.5.11):

$$\mathbf{Y} = A\mathbf{y} = \tilde{A}\tilde{\mathbf{y}} = \tilde{A}\tilde{\mathbf{x}} = A\mathbf{x} = \mathbf{X}$$

For example, the following length-8 signals all have the same 4-point DFT,

$$\begin{bmatrix} x_0\\ x_1\\ x_2\\ x_3\\ x_4\\ x_5\\ x_6\\ x_7 \end{bmatrix}, \quad
\begin{bmatrix} x_0+x_4\\ x_1\\ x_2\\ x_3\\ 0\\ x_5\\ x_6\\ x_7 \end{bmatrix}, \quad
\begin{bmatrix} x_0+x_4\\ x_1+x_5\\ x_2\\ x_3\\ 0\\ 0\\ x_6\\ x_7 \end{bmatrix}, \quad
\begin{bmatrix} x_0+x_4\\ x_1+x_5\\ x_2+x_6\\ x_3\\ 0\\ 0\\ 0\\ x_7 \end{bmatrix}, \quad
\begin{bmatrix} x_0+x_4\\ x_1+x_5\\ x_2+x_6\\ x_3+x_7\\ 0\\ 0\\ 0\\ 0 \end{bmatrix}$$

because all have the same mod-4 reduction:

$$\tilde{\mathbf{x}} = \begin{bmatrix} x_0+x_4\\ x_1+x_5\\ x_2+x_6\\ x_3+x_7 \end{bmatrix}$$

The above signals have a bottom half that becomes progressively zero, until the last vector which is recognized as the $\tilde{\mathbf{x}}$, viewed as a length-8 vector. In fact, the mod-N

wrapped signal $\tilde{\mathbf{x}}$ is *unique* in the above class of signals in the sense that it is *shortest* signal, that is, of length N, that has the same DFT as the signal $\mathbf{x}$.

An equivalent characterization of the class of signals that have a common DFT can be given in the z-domain. Suppose the length-L signals $\mathbf{y}$ and $\mathbf{x}$ have equal mod-N reductions, $\tilde{\mathbf{y}} = \tilde{\mathbf{x}}$ and, therefore, equal DFTs $X_k = Y_k$. We form the difference of their z-transforms:

$$F(z) = X(z) - Y(z) = \sum_{n=0}^{L-1} x(n)z^{-n} - \sum_{n=0}^{L-1} y(n)z^{-n}$$

Evaluating $F(z)$ at the Nth roots of unity and using the equality of their N-point DFTs, we find:

$$F(z_k) = X(z_k) - Y(z_k) = X_k - Y_k = 0, \qquad k = 0, 1, \dots, N-1$$

Thus, the N complex numbers z_k are roots of the difference polynomial $F(z)$. Therefore, $F(z)$ will be divisible by the Nth order product polynomial:

$$1 - z^{-N} = \prod_{k=0}^{N-1}(1 - z_k z^{-1})$$

which represents the factorization of $1 - z^{-N}$ into its Nth root-of-unity zeros. Therefore, we can write:

$$X(z) - Y(z) = F(z) = \big(1 - z^{-N}\big)Q(z) \quad \text{or,}$$

$$\boxed{X(z) = Y(z) + \big(1 - z^{-N}\big)Q(z)} \tag{10.5.13}$$

Because $X(z)$ and $Y(z)$ have degree $L-1$, it follows that $Q(z)$ is an *arbitrary* polynomial of degree $L-1-N$. Denoting the coefficients of $Q(z)$ by $q(n)$, $0 \le n \le L-1-N$, we may write Eq. (10.5.13) in the time domain:

$$\boxed{x(n) = y(n) + q(n) - q(n-N)}\,, \qquad n = 0, 1, \dots, L-1 \tag{10.5.14}$$

Thus, any two sequences $x(n)$ and $y(n)$ related by Eq. (10.5.14) will have the same N-point DFT. The mod-N reduction $\tilde{\mathbf{x}}$ and its z-transform $\tilde{X}(z)$ are also related by Eq. (10.5.13):

$$\boxed{X(z) = \big(1 - z^{-N}\big)Q(z) + \tilde{X}(z)} \tag{10.5.15}$$

Because $\tilde{X}(z)$ has degree $N-1$, Eq. (10.5.15) represents the division of the polynomial $X(z)$ by the DFT polynomial $1 - z^{-N}$, with $\tilde{X}(z)$ being the *remainder* polynomial and $Q(z)$ the *quotient* polynomial. The remainder $\tilde{X}(z)$ is the unique polynomial satisfying Eq. (10.5.15) that has *minimal* degree $N-1$.

10.6 Inverse DFT

The problem of inverting an N-point DFT is the problem of recovering the original length-L signal $\mathbf{x}$ from its N-point DFT $\mathbf{X}$, that is, inverting the relationship:

$$\mathbf{X} = A\mathbf{x} = \tilde{A}\tilde{\mathbf{x}} \tag{10.6.1}$$

When $L > N$, the matrix A is not invertible. As we saw, there are in this case several possible solutions $\mathbf{x}$, all satisfying Eq. (10.6.1) and having the same mod-N reduction $\tilde{\mathbf{x}}$.

Among these solutions, the only one that is uniquely obtainable from the knowledge of the DFT vector $\mathbf{X}$ is $\tilde{\mathbf{x}}$. The corresponding DFT matrix $\tilde{A}$ is an $N\times N$ square invertible matrix. Thus, we define the *inverse DFT* by

$$\boxed{\tilde{\mathbf{x}} = \text{IDFT}(\mathbf{X}) = \tilde{A}^{-1}\mathbf{X}} \qquad \text{(inverse DFT)} \tag{10.6.2}$$

or, component-wise,

$$\tilde{x}_n = \sum_{k=0}^{N-1} (\tilde{A}^{-1})_{nk} X_k\,, \qquad n = 0, 1, \dots, N-1 \tag{10.6.3}$$

The inverse $\tilde{A}^{-1}$ can be obtained *without* having to perform a matrix inversion by using the following *unitarity* property of the DFT matrix $\tilde{A}$:

$$\boxed{\frac{1}{N}\tilde{A}\tilde{A}^* = I_N} \tag{10.6.4}$$

where I_N is the N-dimensional identity matrix and $\tilde{A}^*$ is the complex conjugate of $\tilde{A}$, obtained by conjugating *every* matrix element of $\tilde{A}$. For example, for $N = 4$, we can verify easily:

$$\frac{1}{4}\tilde{A}\tilde{A}^* = \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Multiplying both sides of Eq. (10.6.4) by $\tilde{A}^{-1}$, we obtain for the matrix inverse:

$$\boxed{\tilde{A}^{-1} = \frac{1}{N}\tilde{A}^*} \tag{10.6.5}$$

Thus, the IDFT (10.6.2) can be written in the form:

$$\boxed{\tilde{\mathbf{x}} = \text{IDFT}(\mathbf{X}) = \frac{1}{N}\tilde{A}^*\mathbf{X}} \qquad \text{(inverse DFT)} \tag{10.6.6}$$

We note also that the IDFT can be thought of as a DFT in the following sense. Introducing a second conjugation instruction, we have:

$$\tilde{A}^*\mathbf{X} = (\tilde{A}\mathbf{X}^*)^* = \big[\text{DFT}(\mathbf{X}^*)\big]^*$$

where the matrix $\tilde{A}$ acting on the conjugated vector $\mathbf{X}^*$ is the DFT of that vector. Dividing by N, we have:

$$\boxed{\text{IDFT}(\mathbf{X}) = \frac{1}{N}\big[\text{DFT}(\mathbf{X}^*)\big]^*} \tag{10.6.7}$$

Replacing DFT by FFT, we get a convenient inverse FFT formula, which uses an FFT to perform the IFFT. It is used in most FFT routines.

$$\boxed{\text{IFFT}(\mathbf{X}) = \frac{1}{N}\big[\text{FFT}(\mathbf{X}^*)\big]^*} \tag{10.6.8}$$

Example 10.6.1: To illustrate Eqs. (10.6.6) and (10.6.7), we calculate the IDFT of the 4-point DFT of Example 10.5.2. We have:

$$\tilde{\mathbf{x}} = \text{IDFT}(\mathbf{X}) = \frac{1}{N}\tilde{A}^*\mathbf{X} = \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix}\begin{bmatrix} 6 \\ 8+4j \\ -2 \\ 8-4j \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \\ -3 \\ 4 \end{bmatrix}$$

and using Eq. (10.6.7), we conjugate $\mathbf{X}$ and transform it:

$$\frac{1}{N}(\tilde{A}\mathbf{X}^*)^* = \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}\begin{bmatrix} 6 \\ 8-4j \\ -2 \\ 8+4j \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \\ -3 \\ 4 \end{bmatrix}$$

where the final overall conjugation was omitted because $\tilde{\mathbf{x}}$ is real. □

Using Eq. (10.4.3) the matrix elements of $\tilde{A}^{-1}$ are:

$$(\tilde{A}^{-1})_{nk} = \frac{1}{N}\tilde{A}^*_{nk} = \frac{1}{N}(W_N^{nk})^* = \frac{1}{N}W_N^{-nk}$$

where we used the property $W_N^* = e^{2\pi j/N} = W_N^{-1}$. Then, Eq. (10.6.3) can be written in the form:

$$\text{(IDFT)} \qquad \boxed{\tilde{x}_n = \frac{1}{N}\sum_{k=0}^{N-1} W_N^{-nk} X_k}\,, \qquad n = 0, 1, \dots, N-1 \tag{10.6.9}$$

In terms of the DFT frequencies ω_k, we have $X_k = X(\omega_k)$ and

$$W_N^{-nk} = e^{2\pi jkn/N} = e^{j\omega_k n}$$

Therefore, the inverse DFT can be written in the alternative form:

$$\text{(IDFT)} \qquad \boxed{\tilde{x}(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(\omega_k)e^{j\omega_k n}}\,, \qquad n = 0, 1, \dots, N-1 \tag{10.6.10}$$

It expresses the signal $\tilde{x}(n)$ as a sum of N complex sinusoids of frequencies ω_k, whose *relative amplitudes and phases* are given by the DFT values $X(\omega_k)$.

The forward DFT of Eq. (10.1.4) is sometimes called an *analysis transform*, analyzing a signal $x(n)$ into N Fourier components. The inverse DFT (10.6.10) is called a *synthesis transform*, resynthesizing the signal $\tilde{x}(n)$ from those Fourier components. The forward and inverse N-point DFTs are akin to the more general forward and inverse DTFTs that use all frequencies, not just the N DFT frequencies:

$$X(\omega)=\sum_{n=0}^{L-1}x(n)e^{-j\omega n},\qquad x(n)=\int_0^{2\pi}X(\omega)e^{j\omega n}\frac{d\omega}{2\pi}\tag{10.6.11}$$

The difference between this inverse DTFT and (10.6.10) is that (10.6.11) reconstructs the full original signal $x(n)$, whereas (10.6.10) reconstructs only the wrapped signal $\tilde{x}(n)$. Eq. (10.6.10) can be thought of as a numerical approximation of the integral in (10.6.11), obtained by dividing the integration range into N equal bins:

$$\int_0^{2\pi}X(\omega)e^{j\omega n}\frac{d\omega}{2\pi}\simeq\sum_{k=0}^{N-1}X(\omega_k)e^{j\omega_k n}\frac{\Delta\omega_{\text{bin}}}{2\pi}$$

where from the definition (10.1.7), we have $\Delta\omega_{\text{bin}}/2\pi=1/N$.

In summary, the inverse of an N-point DFT reconstructs only the wrapped version of the original signal that was transformed. This property is shown in Fig. 10.6.1.

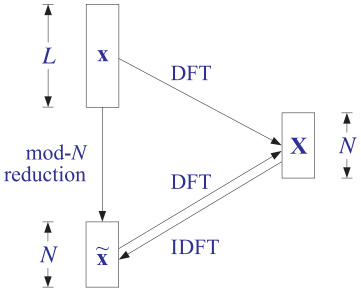


Fig. 10.6.1 Forward and inverse N-point DFTs.

In order for the IDFT to generate the original unwrapped signal $\mathbf{x}$, it is necessary to have $\tilde{\mathbf{x}}=\mathbf{x}$. This happens only if the DFT length N is at least L, so that there will be only one length-N sub-block in $\mathbf{x}$ and there will be nothing to wrap around. Thus, we have the condition:

$$\boxed{\tilde{\mathbf{x}}=\mathbf{x}\quad\text{only if}\quad N\geq L}\tag{10.6.12}$$

If $N=L$, then Eq. (10.6.12) is exact. If $N>L$, then we must pad $N-L$ zeros at the end of $\mathbf{x}$ so that the two sides of Eq. (10.6.12) have compatible lengths. If $N<L$, the wrapped and original signals will be different because there will be several length-N sub-blocks in $\mathbf{x}$ that get wrapped around. Thus, we also have the condition:

$$\boxed{\tilde{\mathbf{x}}\neq\mathbf{x}\quad\text{if}\quad N<L}\tag{10.6.13}$$

10.7 Sampling of Periodic Signals and the DFT

The inverse DFT (10.6.10) defines the signal $\tilde{x}(n)$ for $n=0,1,\dots,N-1$. However, the same expression can be used to define it for any value of n. The resulting $\tilde{x}(n)$ will be periodic in n with period N. This follows from the periodicity of the discrete-time sinusoids:

$$e^{j\omega_k(n+N)}=e^{2\pi jk(n+N)/N}=e^{2\pi jkn/N}e^{2\pi jk}=e^{2\pi jkn/N}=e^{j\omega_k n}$$

The periodic signal $\tilde{x}(n)$ is equivalent to the periodic extension of $x(n)$, as discussed in Section 10.5. Therefore, if the original signal $x(n)$ is also *periodic* with period N and we compute its N-point DFT over one period $L=N$, then we will have $\mathbf{x}=\tilde{\mathbf{x}}$, or, $x(n)=\tilde{x}(n)$. It follows that the *periodic* signal $x(n)$ may be represented by the *discrete Fourier series* (DFS):

$$\boxed{x(n)=\frac{1}{N}\sum_{k=0}^{N-1}X(\omega_k)e^{j\omega_k n}}\qquad\text{(DFS)}\tag{10.7.1}$$

with the DFT playing the role of Fourier series coefficients:

$$\boxed{X(\omega_k)=\sum_{n=0}^{N-1}x(n)e^{-j\omega_k n}}\qquad\text{(DFS coefficients)}\tag{10.7.2}$$

These relationships are helpful in the analysis of *analog* periodic signals. We saw in Example 1.4.6 and Section 16.1.2 that for a periodic signal to remain periodic after sampling, it is necessary that the sampling rate be a multiple of the fundamental frequency of the signal:

$$f_s=Nf_1$$

The periodic analog signal will have an ordinary Fourier series expansion into a sum of sinusoids at the harmonics of the fundamental, $f_m=mf_1$:

$$x(t)=\sum_{m=-\infty}^{\infty}c_m e^{2\pi jf_m t}$$

In general, an infinite number of harmonics are necessary to represent $x(t)$, and therefore, if the signal is sampled at a rate f_s, all harmonics that lie outside the Nyquist interval will be aliased with the harmonics inside the interval.

Taking the Nyquist interval to be the right-sided one $[0,f_s]$, we note that the harmonics within that interval are none other than the N DFT frequencies:

$$f_k=kf_1=k\frac{f_s}{N},\qquad k=0,1,\dots,N-1$$

Given an integer m, we determine its quotient and remainder of the division by N:

$$m = qN + k, \qquad 0 \le k \le N-1$$

and therefore, the corresponding harmonic will be:

$$f_m = mf_1 = qNf_1 + kf_1 = qf_s + f_k$$

which shows that f_m will be aliased with f_k. Therefore, if the signal $x(t)$ is sampled, it will give rise to the samples:

$$x(nT) = \sum_{m=-\infty}^{\infty} c_m e^{2\pi j f_m n/f_s} = \sum_{k=0}^{N-1}\sum_{q=-\infty}^{\infty} c_{qN+k} e^{2\pi j(qf_s+f_k)n/f_s}$$

where we wrote the summation over m as an equivalent double summation over q and k. Noting that,

$$e^{2\pi jqf_s n/f_s} = e^{2\pi jqn} = 1$$

and defining the aliased Fourier series amplitudes,

$$b_k = \sum_{q=-\infty}^{\infty} c_{qN+k}, \qquad k = 0, 1, \dots, N-1$$

we obtain:

$$x(nT) = \sum_{k=0}^{N-1} b_k e^{2\pi j f_k n/f_s} = \sum_{k=0}^{N-1} b_k e^{j\omega_k n} \tag{10.7.3}$$

Comparing it with Eq. (10.7.1), we may identify the aliased amplitudes:

$$\boxed{b_k = \frac{1}{N} X(\omega_k)}, \qquad k = 0, 1, \dots, N-1 \tag{10.7.4}$$

Thus, the aliased amplitudes b_k are computable by performing an N-point DFT on the N samples comprising *one period* of the signal $x(nT)$. If the samples $x(nT)$ were to be reconstructed back into analog form using an *ideal* reconstructor, the following aliased analog waveform would be obtained:

$$\boxed{x_{\text{al}}(t) = \sum_{k=0}^{N-1} b_k e^{2\pi j f_k t}} \tag{10.7.5}$$

with the proviso that those harmonics f_k that lie in the right half of the Nyquist interval, $f_s/2 < f_k \le f_s$, will be replaced by their negative selves, $f_k - f_s$.

Example 10.7.1: In Example 1.4.6, we determined the aliased signal $x_{\text{al}}(t)$ resulting by sampling a square wave of frequency $f_1 = 1$ Hz.

For a sampling rate of $f_s = 4$ Hz, we consider one period consisting of $N = 4$ samples and perform its 4-point DFT:

$$\mathbf{x} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ -1 \end{bmatrix} \quad \Rightarrow \quad \mathbf{X} = A\mathbf{x} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ -2j \\ 0 \\ 2j \end{bmatrix}$$

Thus, the Fourier coefficients are:

$$[b_0, b_1, b_2, b_3] = \frac{1}{4}[0,\ -2j,\ 0,\ 2j] = [0,\ \frac{1}{2j},\ 0,\ -\frac{1}{2j}]$$

corresponding to the harmonics:

$$[f_0, f_1, f_2, f_3] = [0, 1, 2, 3] \equiv [0, 1, 2, -1]$$

where $f_3 = 3$ was replaced by its negative version $f_3 - f_s = 3 - 4 = -1$. It follows that the aliased signal will be:

$$x_{\text{al}}(t) = b_1 e^{2\pi jt} + b_3 e^{-2\pi jt} = \frac{1}{2j} e^{2\pi jt} - \frac{1}{2j} e^{-2\pi jt} = \sin(2\pi t)$$

Similarly, for $N = 8$ corresponding to $f_s = 8$ Hz, we perform the 8-point DFT of one period of the square wave, and divide by 8 to get the aliased amplitudes:

$$\mathbf{x} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ -1 \\ -1 \\ -1 \end{bmatrix} \quad \xrightarrow{\text{DFT}} \quad \mathbf{X} = \begin{bmatrix} 0 \\ -2j(\sqrt{2}+1) \\ 0 \\ -2j(\sqrt{2}-1) \\ 0 \\ 2j(\sqrt{2}-1) \\ 0 \\ 2j(\sqrt{2}+1) \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 0 \\ (\sqrt{2}+1)/4j \\ 0 \\ (\sqrt{2}-1)/4j \\ 0 \\ -(\sqrt{2}-1)/4j \\ 0 \\ -(\sqrt{2}+1)/4j \end{bmatrix}$$

These amplitudes correspond to the frequencies $f_k = kf_1$:

$$[0, 1, 2, 3, 4, 5, 6, 7] \equiv [0, 1, 2, 3, 4, -3, -2, -1]$$

It follows that the aliased signal will be:

$$\begin{aligned} x_{\text{al}}(t) &= \frac{(\sqrt{2}+1)}{4j} e^{2\pi jt} + \frac{(\sqrt{2}-1)}{4j} e^{2\pi j3t} \\ &\quad - \frac{(\sqrt{2}-1)}{4j} e^{-2\pi j3t} - \frac{(\sqrt{2}+1)}{4j} e^{-2\pi jt} \\ &= \frac{\sqrt{2}+1}{2}\sin(2\pi t) + \frac{\sqrt{2}-1}{2}\sin(6\pi t) \end{aligned}$$

which agrees with Example 1.4.6. The above 8-point DFT can be done using the 8×8 DFT matrix, or, more quickly using an FFT by hand, as done in Example 10.8.3. □

Example 10.7.2: Without performing any DFT or FFT computations, determine the 16-point DFT of the signal:

$$x(n)=1+2\sin\Big(\frac{\pi n}{2}\Big)+2\cos\Big(\frac{3\pi n}{4}\Big)+\cos(\pi n),\quad n=0,1,\dots,15$$

Then, determine its 8-point DFT.

Solution: The signal $x(n)$ is already given as a sum of sinusoids at frequencies which are 16-point DFT frequencies. Thus, all we have to do is compare the given expression with the 16-point IDFT formula and identify the DFT coefficients X_k:

$$x(n)=\frac{1}{16}\sum_{k=0}^{15}X_k e^{j\omega_k n}$$

Using Euler's formula, we write the given signal as:

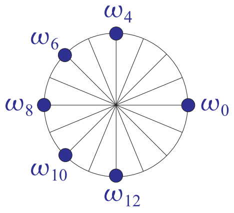
$$x(n)=1-je^{j\pi n/2}+je^{-j\pi n/2}+e^{3j\pi n/4}+e^{-3j\pi n/4}+e^{j\pi n}$$

Shifting the negative frequencies by 2π, noting that the 16-point DFT frequencies are $\omega_k=2\pi k/16=\pi k/8$, and writing the terms in increasing DFT index k, we have:

$$x(n)=\frac{1}{16}\big[16e^{j\omega_0 n}-16je^{j\omega_4 n}+16e^{j\omega_6 n}+16e^{j\omega_8 n}+16e^{j\omega_{10} n}+16je^{j\omega_{12} n}\big]$$

where the frequencies, their negatives, and their relative locations with respect to the 16 DFT roots of unity are as follows:

$$\begin{aligned}
\omega_4 &= \frac{2\pi\cdot 4}{16}=\frac{\pi}{2}\\
\omega_{12} &= \frac{2\pi\cdot 12}{16}=2\pi-\frac{2\pi\cdot 4}{16}=2\pi-\omega_4\\
\omega_6 &= \frac{2\pi\cdot 6}{16}=\frac{3\pi}{4}\\
\omega_{10} &= \frac{2\pi\cdot 10}{16}=2\pi-\frac{2\pi\cdot 6}{16}=2\pi-\omega_6\\
\omega_8 &= \frac{2\pi\cdot 8}{16}=\pi
\end{aligned}$$



Comparing with the IDFT, we identify the coefficients of the exponentials:

$$X_0=16,\ X_4=-16j,\ X_6=X_8=X_{10}=16,\ X_{12}=16j$$

Thus, the 16-point DFT vector will be:

$$\mathbf{X}=[16,\,0,\,0,\,0,\,-16j,\,0,\,16,\,0,\,16,\,0,\,16,\,0,\,16j,\,0,\,0,\,0]^T$$

The 8-point DFT is obtained by picking every other entry of $\mathbf{X}$, that is,

$$\mathbf{X}=[16,\,0,\,-16j,\,16,\,16,\,16,\,16j,\,0]^T \tag{10.7.6}$$

This follows because the 8-point DFT frequencies are a subset of the 16-point ones, that is, $\omega_k=2\pi k/8=2\pi(2k)/16$, $k=0,1,\dots,7$. □

Example 10.7.3: The 8-point DFT determined in the previous example was that of the 16-point signal. If the signal $x(n)$ is considered as a length-8 signal over $0\le n\le 7$, then its 8-point DFT will be different.

To find it, we follow the same method of writing $x(n)$ in its IDFT form, but now we identify the frequencies as 8-point DFT frequencies $\omega_k=2\pi k/8$. We have:

$$\omega_2=\frac{2\pi\cdot 2}{8}=\frac{\pi}{2},\quad \omega_3=\frac{2\pi\cdot 3}{8}=\frac{3\pi}{4},\quad \omega_4=\frac{2\pi\cdot 4}{8}=\pi$$

and $x(n)$ can be written as:

$$x(n)=\frac{1}{8}\big[8e^{j\omega_0 n}-8je^{j\omega_2 n}+8e^{j\omega_3 n}+8e^{j\omega_4 n}+8e^{j\omega_5 n}+8je^{j\omega_6 n}\big]$$

comparing with the 8-point IDFT,

$$x(n)=\frac{1}{8}\sum_{k=0}^{7}X_k e^{j\omega_k n}$$

we obtain:

$$\mathbf{X}=[8,\,0,\,-8j,\,8,\,8,\,8,\,8j,\,0]^T$$

The answer of Eq. (10.7.6) is doubled because the length-16 signal of the previous problem consists of two length-8 periods, which double when wrapped mod-8. □

10.8 FFT

The *fast Fourier transform* is a fast implementation of the DFT. It is based on a divide-and-conquer approach in which the DFT computation is divided into smaller, simpler, problems and the final DFT is rebuilt from the simpler DFTs. For a comprehensive review, history, and recent results, see [256]. For general references, see [242–263,642].

Another application of this divide-and-conquer approach is the computation of *very large FFTs*, in which the time data and their DFT are too large to be stored in main memory. In such cases the FFT is done in parts and the results are pieced together to form the overall FFT, and saved in secondary storage such as on hard disk [260–263,642].

In the simplest Cooley-Tukey version of the FFT, the dimension of the DFT is successively divided in half until it becomes unity. This requires the initial dimension N to be a power of two:

$$\boxed{N=2^B}\quad\Rightarrow\quad B=\log_2(N) \tag{10.8.1}$$

The problem of computing the N-point DFT is replaced by the simpler problems of computing two $(N/2)$-point DFTs. Each of these is replaced by two $(N/4)$-point DFTs, and so on.

We will see shortly that an N-point DFT can be rebuilt from two $(N/2)$-point DFTs by an *additional* cost of $N/2$ complex multiplications. This basic merging step is shown in Fig. 10.8.1.

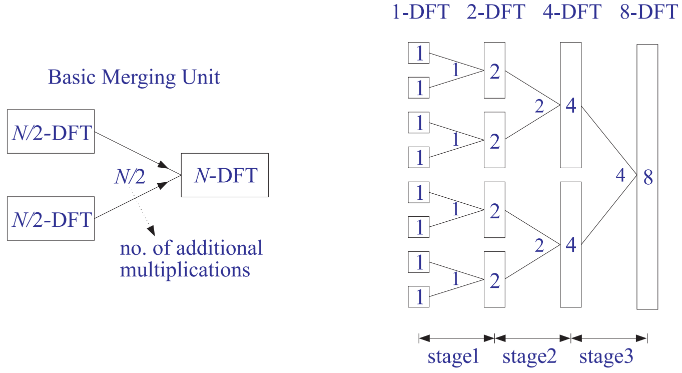


Fig. 10.8.1 Merging two $N/2$-DFTs into an N-DFT and its repeated application.

Thus, if we compute the two $(N/2)$-DFTs directly, at a cost of $(N/2)^2$ multiplications each, the total cost of rebuilding the full N-DFT will be:

$$2\left(\frac{N}{2}\right)^2 + \frac{N}{2} = \frac{N^2}{2} + \frac{N}{2} \simeq \frac{N^2}{2}$$

where for large N the quadratic term dominates. This amounts to 50 percent savings over computing the N-point DFT directly at a cost of N^2.

Similarly, if the two $(N/2)$-DFTs were computed indirectly by rebuilding each of them from two $(N/4)$-DFTs, the total cost for rebuilding an N-DFT would be:

$$4\left(\frac{N}{4}\right)^2 + 2\frac{N}{4} + \frac{N}{2} = \frac{N^2}{4} + 2\frac{N}{2} \simeq \frac{N^2}{4}$$

Thus, we gain another factor of two, or a factor of four in efficiency over the direct N-point DFT. In the above equation, there are 4 direct $(N/4)$-DFTs at a cost of $(N/4)^2$ each, requiring an additional cost of $N/4$ each to merge them into $(N/2)$-DFTs, which require another $N/2$ for the final merge.

Proceeding in a similar fashion, we can show that if we start with $(N/2^m)$-point DFTs and perform m successive merging steps, the total cost to rebuild the final N-DFT will be:

$$\frac{N^2}{2^m} + \frac{N}{2}m \qquad (10.8.2)$$

The first term, $N^2/2^m$, corresponds to performing the initial $(N/2^m)$-point DFTs directly. Because there are 2^m of them, they will require a total cost of $2^m(N/2^m)^2 = N^2/2^m$.

However, if the subdivision process is continued for $m = B$ stages, as shown in Fig. 10.8.1, the final dimension will be $N/2^m = N/2^B = 1$, which requires no computation at all because the 1-point DFT of a 1-point signal is itself.

In this case, the first term in Eq. (10.8.2) will be absent, and the total cost will arise from the second term. Thus, carrying out the subdivision/merging process to its logical extreme of $m = B = \log_2(N)$ stages, allows the computation to be done in:

$$\boxed{\frac{1}{2}NB = \frac{1}{2}N\log_2(N)} \qquad \text{(FFT computational cost)} \qquad (10.8.3)$$

It can be seen Fig. 10.8.1 that the total number of multiplications needed to perform all the mergings in each stage is $N/2$, and B is the number of stages. Thus, we may interpret Eq. (10.8.3) as

$$\text{(total multiplications)} = \text{(multiplications per stage)} \times \text{(no. stages)} = \frac{N}{2}B$$

For the $N = 8$ example shown in Fig. 10.8.1, we have $B = \log_2(8) = 3$ stages and $N/2 = 8/2 = 4$ multiplications per stage. Therefore, the total cost is $BN/2 = 3\cdot 4 = 12$ multiplications.

Next, we discuss the so-called *decimation-in-time radix-2 FFT algorithm*. There is also a decimation-in-frequency version, which is very similar. The term radix-2 refers to the choice of N as a power of 2, in Eq. (10.8.1).

Given a length-N sequence $x(n)$, $n = 0, 1, \dots, N-1$, its N-point DFT $X(k) = X(\omega_k)$ can be written in the component-form of Eq. (10.4.2):

$$X(k) = \sum_{n=0}^{N-1} W_N^{kn}x(n), \qquad k = 0, 1, \dots, N-1 \qquad (10.8.4)$$

The summation index n ranges over both even and odd values in the range $0 \le n \le N-1$. By grouping the even-indexed and odd-indexed terms, we may rewrite Eq. (10.8.4) as

$$X(k) = \sum_n W_N^{k(2n)}x(2n) + \sum_n W_N^{k(2n+1)}x(2n+1)$$

To determine the proper range of summations over n, we consider the two terms separately. For the even-indexed terms, the index $2n$ must be within the range $0 \le 2n \le N-1$. But, because N is even (a power of two), the upper limit $N-1$ will be odd. Therefore, the highest even index will be $N-2$. This gives the range:

$$0 \le 2n \le N-2 \quad \Rightarrow \quad 0 \le n \le \frac{N}{2} - 1$$

Similarly, for the odd-indexed terms, we must have $0 \le 2n+1 \le N-1$. Now the upper limit can be realized, but the lower one cannot; the smallest odd index is unity. Thus, we have:

$$1 \le 2n+1 \le N-1 \quad \Rightarrow \quad 0 \le 2n \le N-2 \quad \Rightarrow \quad 0 \le n \le \frac{N}{2} - 1$$

Therefore, the summation limits are the same for both terms:

$$X(k) = \sum_{n=0}^{N/2-1} W_N^{k(2n)}x(2n) + \sum_{n=0}^{N/2-1} W_N^{k(2n+1)}x(2n+1) \qquad (10.8.5)$$

This expression leads us to define the two length-$(N/2)$ subsequences:

$$\boxed{\begin{aligned} g(n) &= x(2n) \\ h(n) &= x(2n+1) \end{aligned}}\,, \qquad n = 0, 1, \dots, \frac{N}{2} - 1 \tag{10.8.6}$$

and their $(N/2)$-point DFTs:

$$\boxed{\begin{aligned} G(k) &= \sum_{n=0}^{N/2-1} W_{N/2}^{kn} g(n) \\ H(k) &= \sum_{n=0}^{N/2-1} W_{N/2}^{kn} h(n) \end{aligned}}\,, \qquad k = 0, 1, \dots, \frac{N}{2} - 1 \tag{10.8.7}$$

Then, the two terms of Eq. (10.8.5) can be expressed in terms of $G(k)$ and $H(k)$. We note that the twiddle factors W_N and $W_{N/2}$ of orders N and $N/2$ are related as follows:

$$W_{N/2} = e^{-2\pi j/(N/2)} = e^{-4\pi j/N} = W_N^2$$

Therefore, we may write:

$$W_N^{k(2n)} = (W_N^2)^{kn} = W_{N/2}^{kn}, \quad W_N^{k(2n+1)} = W_N^k W_N^{2kn} = W_N^k W_{N/2}^{kn}$$

Using the definitions (10.8.6), Eq. (10.8.5) can be written as:

$$X(k) = \sum_{n=0}^{N/2-1} W_{N/2}^{kn} g(n) + W_N^k \sum_{n=0}^{N/2-1} W_{N/2}^{kn} h(n)$$

and using Eq. (10.8.7),

$$\boxed{X(k) = G(k) + W_N^k H(k)}\,, \qquad k = 0, 1, \dots, N-1 \tag{10.8.8}$$

This is the basic merging result. It states that $X(k)$ can be rebuilt out of the two $(N/2)$-point DFTs $G(k)$ and $H(k)$. There are N additional multiplications, $W_N^k H(k)$. Using the periodicity of $G(k)$ and $H(k)$, the additional multiplications may be reduced by half to $N/2$. To see this, we split the full index range $0 \le k \le N-1$ into two half-ranges parametrized by the two indices k and $k + N/2$:

$$0 \le k \le \frac{N}{2} - 1 \quad \Rightarrow \quad \frac{N}{2} \le k + \frac{N}{2} \le N - 1$$

Therefore, we may write the N equations (10.8.8) as two groups of $N/2$ equations:

$$\begin{aligned} X(k) &= G(k) + W_N^k H(k) \\ X(k+N/2) &= G(k+N/2) + W_N^{(k+N/2)} H(k+N/2) \end{aligned} \qquad k = 0, 1, \dots, \frac{N}{2} - 1$$

Using the periodicity property that any DFT is periodic in k with period its length, we have $G(k+N/2) = G(k)$ and $H(k+N/2) = H(k)$. We also have the twiddle factor property:

$$W_N^{N/2} = (e^{-2\pi j/N})^{N/2} = e^{-j\pi} = -1$$

Then, the DFT merging equations become:

$$\boxed{\begin{aligned} X(k) &= G(k) + W_N^k H(k) \\ X(k+N/2) &= G(k) - W_N^k H(k) \end{aligned}}\,, \quad k = 0, 1, \dots, \frac{N}{2} - 1 \tag{10.8.9}$$

They are known as the *butterfly* merging equations. The upper group generates the upper half of the N-dimensional DFT vector $\mathbf{X}$, and the lower group generates the lower half. The $N/2$ multiplications $W_N^k H(k)$ may be used both in the upper and the lower equations, thus reducing the total extra merging cost to $N/2$. Vectorially, we may write them in the form:

$$\begin{aligned} \begin{bmatrix} X_0 \\ X_1 \\ \vdots \\ X_{N/2-1} \end{bmatrix} &= \begin{bmatrix} G_0 \\ G_1 \\ \vdots \\ G_{N/2-1} \end{bmatrix} + \begin{bmatrix} H_0 \\ H_1 \\ \vdots \\ H_{N/2-1} \end{bmatrix} \times \begin{bmatrix} W_N^0 \\ W_N^1 \\ \vdots \\ W_N^{N/2-1} \end{bmatrix} \\ \begin{bmatrix} X_{N/2} \\ X_{N/2+1} \\ \vdots \\ X_{N-1} \end{bmatrix} &= \begin{bmatrix} G_0 \\ G_1 \\ \vdots \\ G_{N/2-1} \end{bmatrix} - \begin{bmatrix} H_0 \\ H_1 \\ \vdots \\ H_{N/2-1} \end{bmatrix} \times \begin{bmatrix} W_N^0 \\ W_N^1 \\ \vdots \\ W_N^{N/2-1} \end{bmatrix} \end{aligned} \tag{10.8.10}$$

where the indicated multiplication is meant to be component-wise. Together, the two equations generate the full DFT vector $\mathbf{X}$. The operations are shown in Fig. 10.8.2.

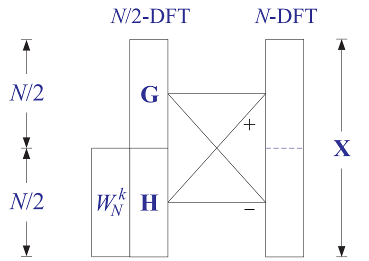


Fig. 10.8.2 Butterfly merging builds upper and lower halves of length-N DFT.

As an example, consider the case $N = 2$. The twiddle factor is now $W_2 = -1$, but only its zeroth power appears $W_2^0 = 1$. Thus, we get two 1-dimensional vectors, making up the final 2-dimensional DFT:

$$\begin{aligned} \begin{bmatrix} X_0 \end{bmatrix} &= \begin{bmatrix} G_0 \end{bmatrix} + \begin{bmatrix} H_0 W_2^0 \end{bmatrix} \\ \begin{bmatrix} X_1 \end{bmatrix} &= \begin{bmatrix} G_0 \end{bmatrix} - \begin{bmatrix} H_0 W_2^0 \end{bmatrix} \end{aligned}$$

For $N = 4$, we have $W_4 = -j$, and only the powers W_4^0, W_4^1 appear:

$$\begin{bmatrix} X_0 \\ X_1 \end{bmatrix} = \begin{bmatrix} G_0 \\ G_1 \end{bmatrix} + \begin{bmatrix} H_0 W_4^0 \\ H_1 W_4^1 \end{bmatrix}$$

$$\begin{bmatrix} X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} G_0 \\ G_1 \end{bmatrix} - \begin{bmatrix} H_0 W_4^0 \\ H_1 W_4^1 \end{bmatrix}$$

And, for $N = 8$, we have:

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} G_0 \\ G_1 \\ G_2 \\ G_3 \end{bmatrix} + \begin{bmatrix} H_0 W_8^0 \\ H_1 W_8^1 \\ H_2 W_8^2 \\ H_3 W_8^3 \end{bmatrix}$$

$$\begin{bmatrix} X_4 \\ X_5 \\ X_6 \\ X_7 \end{bmatrix} = \begin{bmatrix} G_0 \\ G_1 \\ G_2 \\ G_3 \end{bmatrix} - \begin{bmatrix} H_0 W_8^0 \\ H_1 W_8^1 \\ H_2 W_8^2 \\ H_3 W_8^3 \end{bmatrix}$$

To begin the merging process shown in Fig. 10.8.1, we need to know the starting one-dimensional DFTs. Once these are known, they may be merged into DFTs of dimension 2,4,8, and so on. The starting one-point DFTs are obtained by the so-called *shuffling* or *bit reversal* of the input time sequence. Thus, the typical FFT algorithm consists of three conceptual parts:

1. Shuffling the N-dimensional input into N one-dimensional signals.
2. Performing N one-point DFTs.
3. Merging the N one-point DFTs into one N-point DFT.

Performing the one-dimensional DFTs is only a conceptual part that lets us pass from the time to the frequency domain. Computationally, it is trivial because the one-point DFT $\mathbf{X} = [X_0]$ of a 1-point signal $\mathbf{x} = [x_0]$ is itself, that is, $X_0 = x_0$, as follows by setting $N = 1$ in Eq. (10.8.4).

The shuffling process is shown in Fig. 10.8.3 for $N = 8$. It has $B = \log_2(N)$ stages. During the first stage, the given length-N signal block $\mathbf{x}$ is divided into two length-$(N/2)$ blocks $\mathbf{g}$ and $\mathbf{h}$ by putting every other sample into $\mathbf{g}$ and the remaining samples into $\mathbf{h}$.

During the second stage, the same subdivision is applied to $\mathbf{g}$, resulting into the length-$(N/4)$ blocks $\{\mathbf{a}, \mathbf{b}\}$ and to $\mathbf{h}$ resulting into the blocks $\{\mathbf{c}, \mathbf{d}\}$, and so on. Eventually, the signal $\mathbf{x}$ is time-decimated down to N length-1 subsequences.

These subsequences form the starting point of the DFT merging process, which is depicted in Fig. 10.8.4 for $N = 8$. The butterfly merging operations are applied to each pair of DFTs to generate the next DFT of doubled dimension.

To summarize the operations, the shuffling process generates the smaller and smaller signals:

$$\mathbf{x} \to \{\mathbf{g}, \mathbf{h}\} \to \{\{\mathbf{a}, \mathbf{b}\}, \{\mathbf{c}, \mathbf{d}\}\} \to \cdots \to \{\text{1-point signals}\}$$

and the merging process rebuilds the corresponding DFTs:

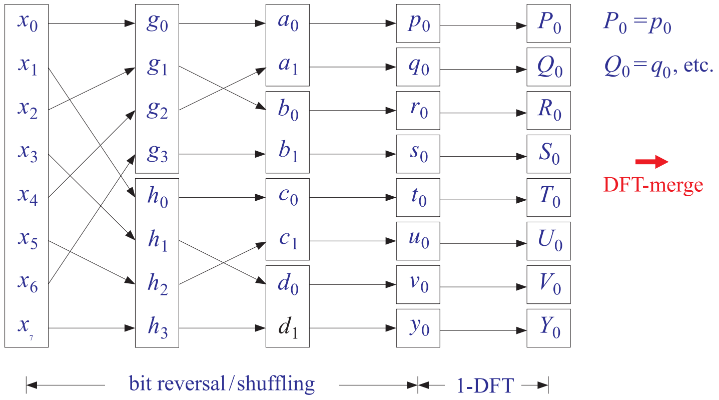


Fig. 10.8.3 Shuffling process generates N one-dimensional signals.

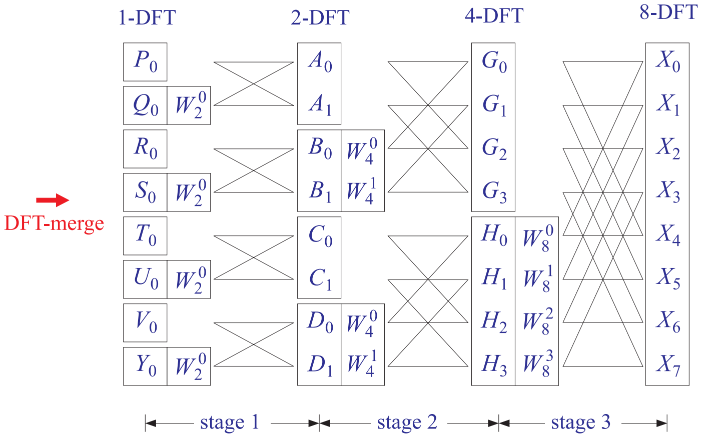


Fig. 10.8.4 DFT merging.

$$\{\text{1-point DFTs}\} \to \cdots \to \{\{\mathbf{A}, \mathbf{B}\}, \{\mathbf{C}, \mathbf{D}\}\} \to \{\mathbf{G}, \mathbf{H}\} \to \mathbf{X}$$

The shuffling process may also be understood as a bit-reversal process, shown in Fig. 10.8.5. Given a time index n in the range $0 \le n \le N-1$, it may be represented in binary by $B = \log_2(N)$ bits. For example, if $N = 8 = 2^3$, we may represent n by three bits $\{b_0, b_1, b_2\}$, which are zero or one:

$$n = (b_2\, b_1\, b_0) \equiv b_2 2^2 + b_1 2^1 + b_0 2^0$$

The binary representations of the time index n for x_n are indicated in Fig. 10.8.5, for both the input and the final shuffled output arrays. The bit-reversed version of n is obtained by reversing the order of the bits:

$$r = \text{bitrev}(n) = (b_0 b_1 b_2) \equiv b_0 2^2 + b_1 2^1 + b_2 2^0$$

We observe in Fig. 10.8.5 that the overall effect of the successive shuffling stages is to put the nth sample of the input array into the rth slot of the output array, that is, swap the locations of x_n with x_r, where r is the bit-reverse of n. Some slots are reverse-invariant so that $r = n$; those samples remain unmoved. All the others get swapped with the samples at the corresponding bit-reversed positions.

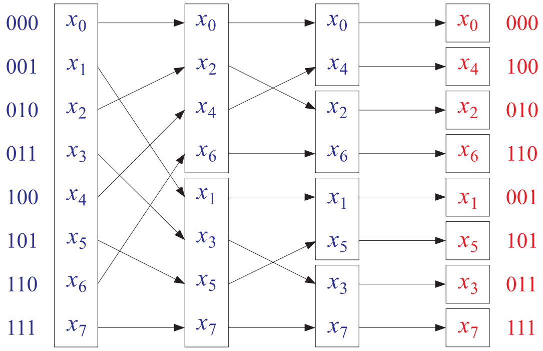

000
001
010
011
100
101
110
111
x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7
x_0 x_2 x_4 x_6 x_1 x_3 x_5 x_7
x_0 x_4 x_2 x_6 x_1 x_5 x_3 x_7
x_0 x_4 x_2 x_6 x_1 x_5 x_3 x_7
000
100
010
110
001
101
011
111

Fig. 10.8.5 Shuffling is equivalent to bit reversal.

The following C routine `fft.c` implements the FFT algorithm, as described above. It consists of two parts: bit-reversal and merging.

```
/* fft.c - decimation-in-time radix-2 FFT */

#include <cmplx.h>

void shuffle(), dftmerge();

void fft(N, X)                                    usage: fft(N, X);
complex *X;
int N;
{
       shuffle(N, X);                             bit-reversal
       dftmerge(N, X);                            merging of DFTs
}
```

The bit-reversal operation is implemented by the routine `shuffle.c`, which calls the routines `swap.c` and `bitrev.c` that implement the swapping of the bit-reversed locations:

```
/* shuffle.c - in-place shuffling (bit-reversal) of a complex array */
```

```
#include <cmplx.h>

void swap();
int bitrev();

void shuffle(N, X)
complex *X;
int N;                                        N must be a power of 2
{
       int n, r, B=1;

       while ( (N >> B) > 0 )                 B = number of bits
              B++;

       B--;                                   N = 2^B

       for (n = 0; n < N; n++) {
           r = bitrev(n, B);                  bit-reversed version of n
           if (r < n) continue;               swap only half of the ns
           swap(X+n, X+r);                    swap by addresses
           }
}


/* swap.c - swap two complex numbers (by their addresses) */

#include <cmplx.h>

void swap(a,b)
complex *a, *b;
{
       complex t;

        t = *a;
       *a = *b;
       *b =  t;
}


/* bitrev.c - bit reverse of a B-bit integer n */

#define two(x)       (1 << (x))                2^x by left-shifting

int bitrev(n, B)
int n, B;
{
       int m, r;

       for (r=0, m=B-1; m>=0; m--)
          if ((n >> m) == 1) {                 if 2^m term is present, then
             r += two(B-1-m);                  add 2^(B-1-m) to r, and
             n -= two(m);                      subtract 2^m from n
             }

       return(r);
}
```

A B-bit number n and its reverse can be expressed in terms of their bits as:

$$n = \sum_{m=0}^{B-1} b_m 2^m$$

$$r = \sum_{m=0}^{B-1} b_m 2^{B-1-m}$$

The routine `bitrev` builds r by determining if the mth bit b_m is one and adding the corresponding power 2^{B-1-m} to r.

The DFT merging operation is given by the routine `dftmerge.c`. It is basically a loop that runs over the successive merging stages of dimensions $M = 2, 4, \dots, N$.

```
/* dftmerge.c - DFT merging for radix 2 decimation-in-time FFT */

#include <cmplx.h>

void dftmerge(N, XF)
complex  *XF;
int N;
{
       double pi = 4. * atan(1.0);
       int k, i, p, q, M;
       complex  A, B, V, W;

       M = 2;
       while (M <= N) {                          two (M/2)-DFTs into one M-DFT
            W = cexp(cmplx(0.0, -2 * pi / M));  order-M twiddle factor
            V = cmplx(1., 0.);                  successive powers of W
            for (k = 0; k < M/2; k++) {         index for an (M/2)-DFT
                 for (i = 0; i < N; i += M) {   ith butterfly; increment by M
                      p = k + i;                absolute indices for
                      q = p + M / 2;            ith butterfly
                      A = XF[p];
                      B = cmul(XF[q], V);       V = W^k
                      XF[p] = cadd(A, B);       butterfly operations
                      XF[q] = csub(A, B);
                      }
                 V = cmul(V, W);                V = VW = W^(k+1)
                 }
            M = 2 * M;                          next stage
            }
}
```

The appropriate twiddle factors W_M^k are computed on the fly and updated from stage to stage. For each stage M and value of k, all the butterflies that use the power W_M^k are computed. For example, in Fig. 10.8.4 the butterflies filling the slots $\{G_0, G_2\}$ and $\{H_0, H_2\}$ are done together because they use the same power W_4^0. Then, the butterflies for the slots $\{G_1, G_3\}$ and $\{H_1, H_3\}$ are done, using the power W_4^1.

The routine performs the computations *in place*, that is, the input time data vector X is *overwritten* by its shuffled version, which is repeatedly overwritten by the higher and higher DFTs during the merging process. The final merge produces the desired DFT and stores it in X.

The following routine `ifft.c` implements the inverse FFT via Eq. (10.6.8). The routine conjugates the input DFT, performs its FFT, conjugates the answer, and divides by N.

```
/* ifft.c - inverse FFT */

#include <cmplx.h>

void fft();

void ifft(N, X)
complex *X;
int N;
{
    int k;

    for (k=0; k<N; k++)
         X[k] = conjg(X[k]);                          conjugate input

    fft(N, X);                                        compute FFT of conjugate

    for (k=0; k<N; k++)
         X[k] = rdiv(conjg(X[k]), (double)N);         conjugate and divide by N
}
```

Next, we present some FFT examples. In the merging operations from 2-point to 4-point DFTs and from to 4-DFTs to 8-DFTs, the following twiddle factors are used:

$$\begin{bmatrix} W_4^0 \\ W_4^1 \end{bmatrix} = \begin{bmatrix} 1 \\ -j \end{bmatrix}, \qquad \begin{bmatrix} W_8^0 \\ W_8^1 \\ W_8^2 \\ W_8^3 \end{bmatrix} = \begin{bmatrix} 1 \\ (1-j)/\sqrt{2} \\ -j \\ -(1+j)/\sqrt{2} \end{bmatrix}$$

MATLAB's built-in FFT function is very fast and flexible. Current versions are based on the FFTW library of FFT functions [264]. Its usage has some caveats, because as we mentioned earlier it truncates the inputs to the FFT length N instead of wrapping the input modulo-N. It can also calculate multiple FFTs with a single call, applied to each column of its input. Its usage is as follows:

```
X = fft(x,N);       % N-point FFT

% if N is omitted, it uses N = L = length(x)
% if N > L, it pads N-L zeros at the end of x before processing
% if N < L, it incorrectly truncates the signal to length N
%           without wrapping it mod-N
%           this can be fixed by using datawrap,
%
% X = fft(datawrap(x,N),N);
%
% x can be an LxK matrix of K columns of length-L
% the FFTs of all columns are returned into the NxK output X
```

Example 10.8.1: Using the FFT algorithm, compute the 4-point DFT of the 4-point wrapped signal of Example 10.5.2.

Solution: The sequence of FFT operations are shown in Fig. 10.8.6. The shuffling operation was stopped at dimension 2, and the corresponding 2-point DFTs were computed by taking the sum and difference of the time sequences, as in Eq. (10.4.7).

The DFT merging stage merges the two 2-DFTs into the final 4-DFT. □

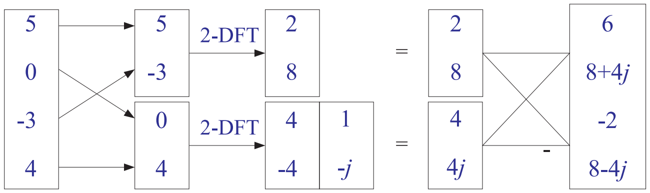


Fig. 10.8.6 4-point FFT of Example 10.8.1.

Example 10.8.2: Using the FFT algorithm, compute the 8-point DFT of the following 8-point signal:

$$\mathbf{x} = [4, -3, 2, 0, -1, -2, 3, 1]^T$$

Then, compute the inverse FFT of the result to recover the original time sequence.

Solution: The required FFT operations are shown in Fig. 10.8.7. Again, the shuffling stages stop with 2-dimensional signals which are transformed into their 2-point DFTs by forming sums and differences.

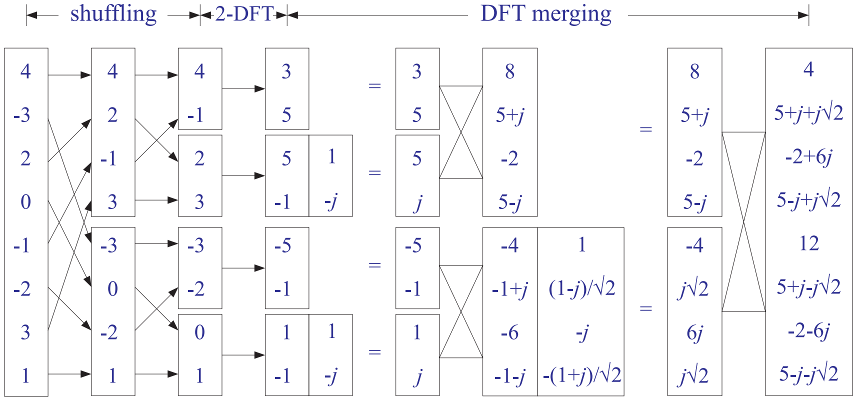


Fig. 10.8.7 8-point FFT of Example 10.8.2.

We find it more convenient to indicate the butterfly operations vectorially, that is, computing the sum and difference of the two 2-dimensional DFT vectors to form the upper and lower parts of the 4-dimensional DFTs, and computing the sum and difference of the two 4-DFT vectors to form the upper and lower parts of the final 8-DFT vector.

The inverse FFT is carried out by the expression (10.6.8). The calculations are shown in Fig. 10.8.8. First, the just computed DFT is complex conjugated. Then, its FFT is computed by carrying out the required shuffling and merging processes. The result must be conjugated (it is real already) and divided by $N = 8$ to recover the original sequence $\mathbf{x}$. □

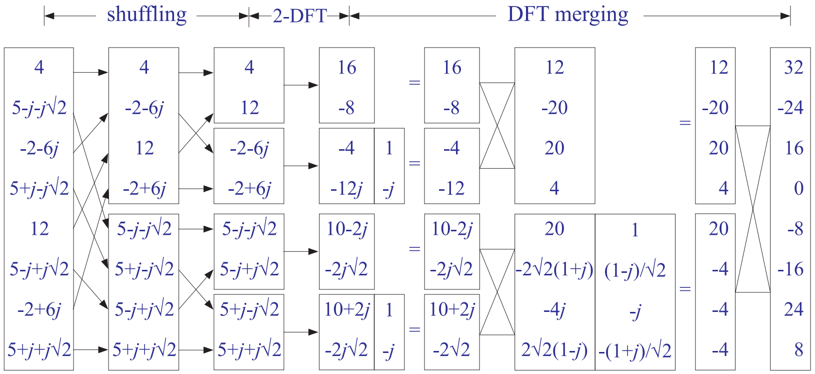


Fig. 10.8.8 8-point inverse FFT of the FFT in Fig. 10.8.7.

Example 10.8.3: The 8-point DFT of the square wave of Example 10.7.1 can be calculated easily using the FFT. Figure 10.8.9 shows the details. □

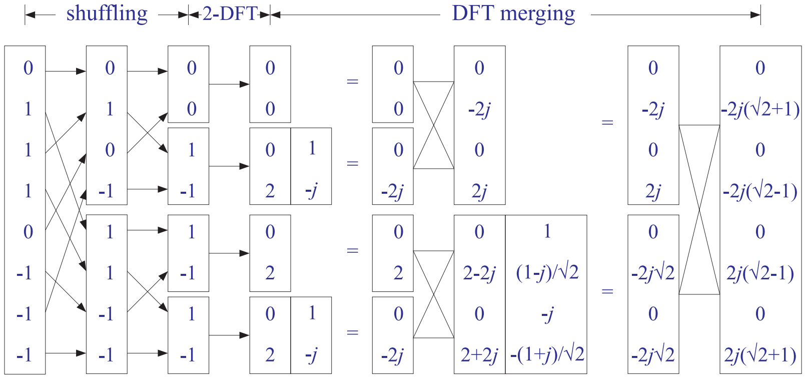


Fig. 10.8.9 8-point FFT of the square wave in Example 10.8.3.

10.9 Fast Convolution

Time-domain convolution for FIR filtering can be implemented in fast way using the FFT. Certain issues must be dealt with in this approach:

a. The wrap-around effects that arise in the inverse FFT lead to the concept of *circular convolution*, discussed in the next section.

b. Long input signals can be dealt with by dividing the input into blocks and applying the fast convolution method to each block. Depending how the input subdivision is done, there are two alternative versions, the *overlap-add* and the *overlap-save* methods, discussed in the following sections.

10.10 Circular Convolution

In the frequency domain, convolution of two sequences **h** and **x** is equivalent to multiplication of the respective DTFTs:

$$\mathbf{y}=\mathbf{h}*\mathbf{x}\qquad\Leftrightarrow\qquad Y(\omega)=H(\omega)X(\omega)\tag{10.10.1}$$

Therefore, $y(n)$ can be recovered by the inverse DTFT of the product of the two DTFTs:

$$y(n)=\int_{-\pi}^{\pi}Y(\omega)e^{j\omega n}\,\frac{d\omega}{2\pi}=\int_{-\pi}^{\pi}H(\omega)X(\omega)e^{j\omega n}\,\frac{d\omega}{2\pi}\tag{10.10.2}$$

Symbolically, we write Eq. (10.10.2) as:

$$\mathbf{y}=\text{IDTFT}\big(\text{DTFT}(\mathbf{h})\cdot\text{DTFT}(\mathbf{x})\big)\tag{10.10.3}$$

Equation (10.10.2) is not a practical method of computing $y(n)$ even in the case of finite-duration signals, because the ω-integration requires knowledge of $Y(\omega)$ at a continuous range of ω's.

A practical approach is to replace all the DTFTs by N-point DFTs. If Eq. (10.10.2) is replaced by an inverse DFT, we saw in Eq. (10.6.10) that it will reconstruct the wrapped signal $\tilde{y}(n)$ instead of the desired one:

$$\tilde{y}(n)=\frac{1}{N}\sum_{k=0}^{N-1}Y(\omega_k)e^{j\omega_k n}=\frac{1}{N}\sum_{k=0}^{N-1}H(\omega_k)X(\omega_k)e^{j\omega_k n}\tag{10.10.4}$$

for $n=0,1,\dots,N-1$, or, written symbolically:

$$\tilde{\mathbf{y}}=\text{IDFT}\big(\text{DFT}(\mathbf{h})\cdot\text{DFT}(\mathbf{x})\big)\tag{10.10.5}$$

Because the unwrapped **y** is the ordinary convolution $\mathbf{y}=\mathbf{h}*\mathbf{x}$, we can write the above as the wrapped convolution:

$$\boxed{\tilde{\mathbf{y}}=\widetilde{\mathbf{h}*\mathbf{x}}=\text{IDFT}\big(\text{DFT}(\mathbf{h})\cdot\text{DFT}(\mathbf{x})\big)}\quad\text{(mod-}N\text{ circular convolution)}\tag{10.10.6}$$

This expression is the definition of the length-N or modulo-N *circular convolution* of the two signals **h** and **x**. A fast version is obtained by replacing DFT by FFT resulting in:

$$\boxed{\tilde{\mathbf{y}}=\widetilde{\mathbf{h}*\mathbf{x}}=\text{IFFT}\big(\text{FFT}(\mathbf{h})\cdot\text{FFT}(\mathbf{x})\big)}\tag{10.10.7}$$

If **h** and **x** are length-N signals, the computational cost of Eq. (10.10.7) is the cost for three FFTs (i.e., of **x**, **h**, and the inverse FFT) plus the cost of the N complex multiplications $Y(\omega_k)=H(\omega_k)X(\omega_k)$, $k=0,1,\dots,N-1$. Thus, the total number of multiplications to implement Eq. (10.10.7) is:

$$3\frac{1}{2}N\log_2(N)+N\tag{10.10.8}$$

Some alternative ways of expressing $\tilde{\mathbf{y}}$ can be obtained by replacing **h** and/or **x** by their wrapped versions. This would not change the result because the wrapped signals have the same DFTs as the unwrapped ones, that is, $\text{DFT}(\mathbf{h})=\text{DFT}(\tilde{\mathbf{h}})$ and $\text{DFT}(\mathbf{x})=\text{DFT}(\tilde{\mathbf{x}})$. Thus, we can write:

$$\begin{aligned}\tilde{\mathbf{y}}=\widetilde{\mathbf{h}*\mathbf{x}}&=\text{IDFT}\big(\text{DFT}(\mathbf{h})\cdot\text{DFT}(\mathbf{x})\big)\\ =\widetilde{\tilde{\mathbf{h}}*\tilde{\mathbf{x}}}&=\text{IDFT}\big(\text{DFT}(\tilde{\mathbf{h}})\cdot\text{DFT}(\tilde{\mathbf{x}})\big)\\ =\widetilde{\tilde{\mathbf{h}}*\mathbf{x}}&=\text{IDFT}\big(\text{DFT}(\tilde{\mathbf{h}})\cdot\text{DFT}(\mathbf{x})\big)\\ =\widetilde{\mathbf{h}*\tilde{\mathbf{x}}}&=\text{IDFT}\big(\text{DFT}(\mathbf{h})\cdot\text{DFT}(\tilde{\mathbf{x}})\big)\end{aligned}\tag{10.10.9}$$

According to Eq. (10.6.12), in order for the circular convolution $\tilde{\mathbf{y}}$ to agree with the ordinary "linear" convolution **y**, the DFT length N must be chosen to be at least the length L_y of the sequence **y**. Recall from Eq. (4.1.12) that if a length-L signal **x** is convolved with an order-M filter **h**, the length of the resulting convolution will be $L_y=L+M$. Thus, we obtain the constraint on the choice of N:

$$\boxed{\tilde{\mathbf{y}}=\mathbf{y}\quad\text{only if}\quad N\geq L_y=L+M}\tag{10.10.10}$$

With this choice of N, Eq. (10.10.7) represents a fast way of computing linear convolution. Because both the filter and input vectors **h**, **x** have lengths less than N (because $L+M=L_y\leq N$), we must increase them to length N by *padding zeros* at their ends, before we actually compute their N-point FFTs.

If $N<L_y$, part of the tail of **y** gets wrapped around to ruin the beginning part of **y**. The following example illustrates the successive improvement of the circular convolution as the length N increases to the value required by (10.10.10).

Example 10.10.1: For the values $N=3,5,7,9,11$, compute the mod-N circular convolution of the two signals of Example 4.1.1:

$$\mathbf{h}=[1,2,-1,1],\qquad\mathbf{x}=[1,1,2,1,2,2,1,1]$$

Solution: For this example, we work exclusively in the time domain and perform ordinary convolution and wrap it modulo-N. The convolution table of Example 4.1.1, gives the output signal:

$$\mathbf{y} = \mathbf{x} * \mathbf{h} = [1,3,3,5,3,7,4,3,3,0,1]$$

The mod-3 circular convolution is obtained by dividing $\mathbf{y}$ into length-3 contiguous blocks, wrapping them around, and summing them to get:

$$\mathbf{y} = [1,3,3]\,[5,3,7]\,[4,3,3]\,[0,1,0] \quad\Rightarrow\quad \tilde{\mathbf{y}} = [10,10,13]$$

where we padded a 0 at the end to make the last block of length-3. In a similar fashion, we determine the other cases:

$$\begin{array}{lll}
\text{(mod-5):} & \mathbf{y} = [1,3,3,5,3]\,[7,4,3,3,0]\,[1] & \Rightarrow\quad \tilde{\mathbf{y}} = [9,7,6,8,3] \\
\text{(mod-7):} & \mathbf{y} = [1,3,3,5,3,7,4]\,[3,3,0,1] & \Rightarrow\quad \tilde{\mathbf{y}} = [4,6,3,6,3,7,4] \\
\text{(mod-9):} & \mathbf{y} = [1,3,3,5,3,7,4,3,3]\,[0,1] & \Rightarrow\quad \tilde{\mathbf{y}} = [1,4,3,5,3,7,4,3,3] \\
\text{(mod-11):} & \mathbf{y} = [1,3,3,5,3,7,4,3,3,0,1] & \Rightarrow\quad \tilde{\mathbf{y}} = [1,3,3,5,3,7,4,3,3,0,1]
\end{array}$$

As N increases to $L_y = L + M = 8 + 3 = 11$, the lengths of the parts that get wrapped around become less and less, making $\tilde{\mathbf{y}}$ resemble $\mathbf{y}$ more and more. □

Example 10.10.2: Recompute the length-3 circular convolution of the previous example by first wrapping mod-3 the signals $\mathbf{h}$ and $\mathbf{x}$, performing their linear convolution, and wrapping it mod-3.

Solution: We find for the mod-3 reductions:

$$\mathbf{h} = [1,2,-1]\,[1] \quad\Rightarrow\quad \tilde{\mathbf{h}} = [2,2,-1]$$

$$\mathbf{x} = [1,1,2]\,[1,2,2]\,[1,1] \quad\Rightarrow\quad \tilde{\mathbf{x}} = [3,4,4]$$

The convolution of the wrapped signals is:

$$\tilde{\mathbf{h}} * \tilde{\mathbf{x}} = [2,2,-1] * [3,4,4] = [6,14,13,4,-4]$$

and, its mod-3 reduction:

$$\tilde{\mathbf{h}} * \tilde{\mathbf{x}} = [6,14,13]\,[4,-4] \quad\Rightarrow\quad \widetilde{\tilde{\mathbf{h}} * \tilde{\mathbf{x}}} = [10,10,13]$$

which agrees with $\tilde{\mathbf{y}}$, in accordance with Eq. (10.10.9). □

Example 10.10.3: Compute the mod-4 circular convolution of the following signals in two ways: (a) working in the time domain, and (b) using DFTs.

$$\mathbf{h} = [1,2,2,1], \qquad \mathbf{x} = [1,3,3,1]$$

Solution: The linear convolution is:

$$\mathbf{y} = \mathbf{h} * \mathbf{x} = [1,2,2,1] * [1,3,3,1] = [1,5,11,14,11,5,1]$$

wrapping it mod-4, we get:

$$\mathbf{y} = [1,5,11,14]\,[11,5,1] \quad\Rightarrow\quad \tilde{\mathbf{y}} = [12,10,12,14]$$

Alternatively, we compute the 4-point DFTs of $\mathbf{h}$ and $\mathbf{x}$:

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 6 \\ -1-j \\ 0 \\ -1+j \end{bmatrix}$$

$$\mathbf{X} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 8 \\ -2-2j \\ 0 \\ -2+2j \end{bmatrix}$$

Multiplying them pointwise, we get:

$$\mathbf{Y} = \begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} = \begin{bmatrix} H_0X_0 \\ H_1X_1 \\ H_2X_2 \\ H_3X_3 \end{bmatrix} = \begin{bmatrix} 48 \\ 4j \\ 0 \\ -4j \end{bmatrix}$$

To take the inverse DFT, we conjugate, take the 4-point DFT, divide by 4, and conjugate the answer:

$$\tilde{\mathbf{y}} = \text{IDFT}(\mathbf{Y}) = \frac{1}{N}\big[\text{DFT}(\mathbf{Y}^*)\big]^*$$

$$\tilde{\mathbf{y}} = \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 48 \\ -4j \\ 0 \\ 4j \end{bmatrix} = \begin{bmatrix} 12 \\ 10 \\ 12 \\ 14 \end{bmatrix}$$

The final conjugation is not necessary because $\tilde{\mathbf{y}}$ is real. □

Besides the efficient computation of convolution, the FFT can also be used to determine the impulse response of an unknown system, such as the reverberation impulse response of a room. Given a length-N input and a corresponding length-N measured output, we may compute their N-point DFTs and solve for the DFT of the impulse response of the system:

$$Y(\omega_k) = H(\omega_k)X(\omega_k) \qquad\Rightarrow\qquad H(\omega_k) = \frac{Y(\omega_k)}{X(\omega_k)}, \quad k = 0,1,\dots,N-1$$

Then, taking the inverse DFT, we have:

$$\tilde{h}(n)=\frac{1}{N}\sum_{k=0}^{N-1}H(\omega_k)e^{j\omega_k n}=\frac{1}{N}\sum_{k=0}^{N-1}\frac{Y(\omega_k)}{X(\omega_k)}e^{j\omega_k n} \qquad (10.10.11)$$

or, symbolically,

$$\tilde{\mathbf{h}}=\text{IDFT}\left[\frac{\text{DFT}(\mathbf{y})}{\text{DFT}(\mathbf{x})}\right]=\text{IFFT}\left[\frac{\text{FFT}(\mathbf{y})}{\text{FFT}(\mathbf{x})}\right] \qquad (10.10.12)$$

The result is again the wrapped version $\tilde{h}(n)$ of the desired impulse response. For this type of application, the true impulse response $h(n)$ is typically infinite, and therefore, its wrapped version will be different from $h(n)$. However, if the wrapping length N is sufficiently large, such that the exponentially decaying tails of $h(n)$ can be ignored, then $\tilde{h}(n)$ may be an adequate approximation.

Example 10.10.4: The reverberation impulse response of a room is of the form

$$h(n)=Aa^n+Bb^n,\qquad n\ge 0$$

Determine the response $\tilde{h}(n)$ that might be measured by the above procedure of dividing the output DFT by the input DFT and taking the IDFT of the result.

Solution: The mod-N reduction of $h(n)$ can be computed as in Example 10.5.3:

$$\tilde{h}(n)=\sum_{m=0}^{\infty}h(mN+n)=\frac{A}{1-a^N}a^n+\frac{B}{1-b^N}b^n,\qquad 0\le n\le N-1$$

If N is large enough such that a^N and b^N are small enough to be ignored, then $\tilde{h}(n)\simeq h(n)$ for $n=0,1,\dots,N-1$. □

10.11 Overlap-Add and Overlap-Save Methods

When the length L of the input signal $\mathbf{x}$ is infinite or very long, the length $L_y=L+M$ of the output will be infinite and the condition (10.10.10) cannot be satisfied.

A practical approach is to divide the long input into *contiguous* non-overlapping blocks of manageable length, say L samples, then filter each block and piece the output blocks together to obtain the overall output, as shown in Fig. 10.11.1. Thus, processing is carried out on a block by block basis.

This is the *overlap-add* method of block convolution, which we introduced in Sec. 4.1.10. Each of the input sub-blocks $\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2$, ..., is convolved with the order-M filter $\mathbf{h}$ producing the outputs blocks:

$$\begin{aligned}\mathbf{y}_0&=\mathbf{h}*\mathbf{x}_0\\ \mathbf{y}_1&=\mathbf{h}*\mathbf{x}_1\\ \mathbf{y}_2&=\mathbf{h}*\mathbf{x}_2\end{aligned} \qquad (10.11.1)$$

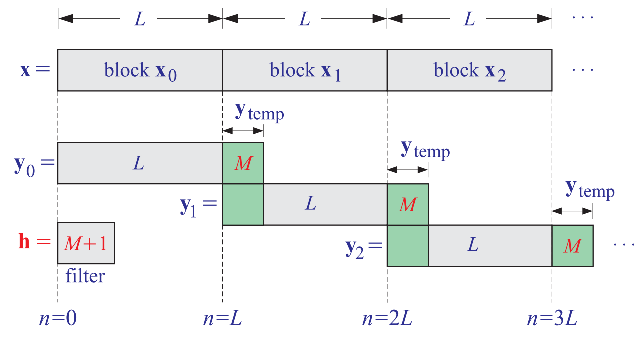


Fig. 10.11.1 Overlap-add block convolution method.

and so on. The resulting blocks are pieced together according to their absolute timing. Block $\mathbf{y}_0$ starts at absolute time $n=0$; block $\mathbf{y}_1$ starts at $n=L$ because the corresponding input block $\mathbf{x}_1$ starts then; block $\mathbf{y}_2$ starts at $n=2L$, and so forth.

Because each output block is longer than the corresponding input block by M samples, the *last* M samples of each output block will *overlap* with the first M outputs of the *next* block.

Note that only the next sub-block will be involved if we assume that $2L>L+M$, or, $L>M$. To get the correct output points, the overlapped portions must be added together (hence the name, overlap-add).

The method can be implemented by the following algorithm, which reads the input data in blocks $\mathbf{x}$ of length L and outputs the result also in blocks of length L:

```
for each length-$L$ input block $\mathbf{x}$, do:
    1. compute the length-$(L+M)$ output $\mathbf{y}$:
           $\mathbf{y}=\mathbf{h}*\mathbf{x}$,   or, alternatively,
           $\mathbf{y}=\text{IFFT}\big(\text{FFT}(\mathbf{h})\cdot\text{FFT}(\mathbf{x})\big)$
    2. for $i=0,1,\dots,M-1$:                                          (10.11.2)
           $y(i)=y(i)+y_{\text{temp}}(i)$        (overlap)
           $y_{\text{temp}}(i)=y(i+L)$           (save tail)
    3. for $i=0,1,\dots,L-1$:
           output $y(i)$
```

The algorithm uses a temporary M-dimensional vector $\mathbf{y}_{\text{temp}}$ to store the last M samples of each *previous* block. Before processing the first block, $\mathbf{y}_{\text{temp}}$ must be initialized to zero.

After computing the length-$(L+M)$ filter output, $\mathbf{y}=\mathbf{h}*\mathbf{x}$, the first M samples of $\mathbf{y}$ are added to the last M samples of the previous block held in $\mathbf{y}_{\text{temp}}$. Then, the last M samples of the currently computed block $\mathbf{y}$ are saved in $\mathbf{y}_{\text{temp}}$ for use in the next iteration. Only the first L corrected output samples of $\mathbf{y}$ are sent to the output.

A fast version of the method can be obtained by performing the convolutions of the input blocks using circular convolution and the FFT by Eq. (10.10.7). The FFT length N must satisfy Eq. (10.10.10) in order for the output blocks to be correct. Given a desired power of two for the FFT length N, we determine the length of the input segments via:

$$N = L + M \quad \Rightarrow \quad \boxed{L = N - M} \tag{10.11.3}$$

With this choice of N, there would be no wrap-around errors, and the outputs of the successive input blocks $\{\mathbf{x}_0, \mathbf{x}_1, \dots\}$, can be computed by:

$$\begin{aligned}
\mathbf{y}_0 &= \tilde{\mathbf{y}}_0 = \text{IFFT}\big(\text{FFT}(\mathbf{h})\cdot\text{FFT}(\mathbf{x}_0)\big)\\
\mathbf{y}_1 &= \tilde{\mathbf{y}}_1 = \text{IFFT}\big(\text{FFT}(\mathbf{h})\cdot\text{FFT}(\mathbf{x}_1)\big)\\
\mathbf{y}_2 &= \tilde{\mathbf{y}}_2 = \text{IFFT}\big(\text{FFT}(\mathbf{h})\cdot\text{FFT}(\mathbf{x}_2)\big)
\end{aligned} \tag{10.11.4}$$

and so on. In counting the computational cost of this method, the FFT of $\mathbf{h}$ need not be counted. It can be computed once, $\mathbf{H} = \text{FFT}(\mathbf{h})$, and used in all convolutions of Eq. (10.11.4). We must only count the cost of *two* FFTs plus the N pointwise multiplications. Thus, the number of multiplications per input block is:

$$2\frac{1}{2}N\log_2 N + N = N(\log_2 N + 1)$$

This must be compared with the cost of $(M+1)L = (M+1)(N-M)$ for performing the ordinary time-domain convolution of each block with the filter. The relative cost of the fast versus the conventional slow method is:

$$\frac{\text{fast}}{\text{slow}} = \frac{N(\log_2 N + 1)}{(M+1)(N-M)} \simeq \frac{\log_2 N}{M} \tag{10.11.5}$$

where the last equation follows in the limit $N \gg M \gg 1$.

The *overlap-save* fast convolution method is an alternative method that also involves partitioning the input into blocks and filtering each block by Eq. (10.10.7). The method is shown in Fig. 10.11.2.

In this method, the input blocks have length equal to the FFT length, $L = N$, but they are made to overlap each other by M points, where M is the filter order. The output blocks will have length $L_y = L + M = N + M$ and therefore, do not satisfy the condition Eq. (10.10.10).

If the output blocks are computed via Eq. (10.10.7), then the last M points of each output block will get wrapped around and be added to the first M output points, ruining them. This is shown in Fig. 10.11.3. Assuming $N > M$, the remaining output points will be correct.

As shown in Fig. 10.11.2, because the input blocks overlap by M points, when the wrapped output blocks are aligned according to their absolute timings, the first M points of each block can be ignored because the correct outputs have already been computed from the previous block.

There is only one exception, that is, the very first M points of the output sequence are not computed correctly. This can be corrected by delaying the input by M time units before commencing the filtering operation.

The computational cost of the method is essentially the same as that of the overlap-add method, with the relative performance over conventional convolution given by Eq. (10.11.5).

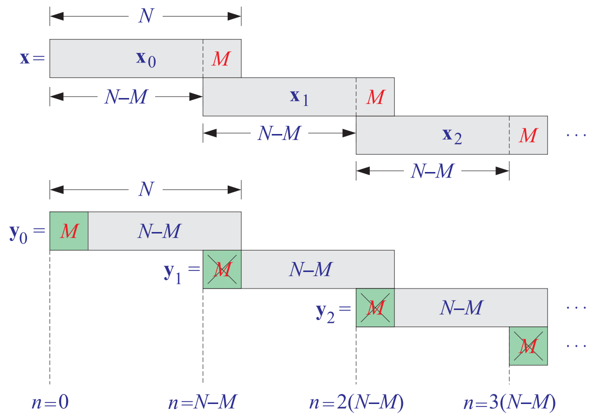


Fig. 10.11.2 Overlap-save method of fast convolution.

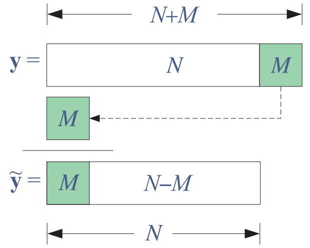


Fig. 10.11.3 Mod-N reduction of output block ruins first M output samples.

Example 10.11.1: Using the overlap-save method of fast convolution, implemented in the time domain by mod-8 circular convolutions, compute the linear convolution of the "long" input:

$$\mathbf{x} = [1,1,1,1,3,3,3,3,1,1,1,2,2,2,2,1,1,1,1]$$

with the "short" filter:

$$\mathbf{h} = [1,-1,-1,1]$$

Solution: For comparison, we compute the linear convolution using the convolution table:

$$\mathbf{y} = [1,0,-1,0,2,0,-2,0,-2,0,2,1,0,-1,0,-1,0,1,0,-1,0,1]$$

For the overlap-save method, we divide the input into length-8 blocks which overlap by $M = 3$ points. These blocks are:

$$\mathbf{x} = [1,1,1,1,3,\big(3,3,3],1,1,[1,2,2\big),2,2,\big(1,1,1],1,0,0,0,0\big)$$

Convolving these blocks with $\mathbf{h}$ gives:

$$\begin{aligned}
\mathbf{y}_0 &= \mathbf{h} * [1,1,1,1,3,3,3,3] = [1,0,-1,0,2,0,-2,0,-3,0,3]\\
\mathbf{y}_1 &= \mathbf{h} * [3,3,3,1,1,1,2,2] = [3,0,-3,-2,0,2,1,0,-3,0,2]\\
\mathbf{y}_2 &= \mathbf{h} * [1,2,2,2,2,1,1,1] = [1,1,-1,-1,0,-1,0,1,-1,0,1]\\
\mathbf{y}_3 &= \mathbf{h} * [1,1,1,1,0,0,0,0] = [1,0,-1,0,-1,0,1,0,0,0,0]
\end{aligned}$$

Reducing them modulo-8 and ignoring the first M points (indicated by $*$),

$$\begin{aligned}
\tilde{\mathbf{y}}_0 &= [*,*,*,0,2,0,-2,0]\\
\tilde{\mathbf{y}}_1 &= [*,*,*,-2,0,2,1,0]\\
\tilde{\mathbf{y}}_2 &= [*,*,*,-1,0,-1,0,1]\\
\tilde{\mathbf{y}}_3 &= [*,*,*,0,-1,0,1,0]
\end{aligned}$$

These would be the outputs computed via the FFT method. Putting them together, we obtain the overall output signal:

$$\mathbf{y} = [*,*,*,0,2,0,-2,0]\,[-2,0,2,1,0]\,[-1,0,-1,0,1]\,[0,-1,0,1,0]$$

With the exception of the first 3 points, the answer is correct. □

10.12 Computer Experiment – Fast Convolution

Write two MATLAB functions, **ovadd** and **ovsave**, that implement the overlap-add and overlap-save methods using both time-domain convolution or the FFT. They should have syntax,

```
y = ovadd(h,x,N,'t');       % using time-domain convolution
y = ovadd(h,x,N,'f');       % using the FFT

y = ovsave(h,x,N,'t');       % using time-domain convolution
y = ovsave(h,x,N,'f');       % using the FFT

% h = filter of order M
% x = input signal vector
% y = output signal vector

% N = FFT length
```

where in all cases, N should denote the FFT length. Thus, in the overlap-add case, the input block length should be chosen as, $L = N - M$, and moreover, we must have $N > 2M$. In all four cases, your functions should match *exactly* the output of the convolution function,

```
y = conv(h,x);
```

In implementing the FFT versions, you need to perform the FFT of the filter $\mathbf{h}$ only once, and then use it in filtering all input blocks. In order to facilitate the partitioning of the input into blocks (overlapping or not), you may use the built-in function **buffer**—even though its use would defeat the purpose of real-time block processing.

The following example MATLAB code could be used as a guide to constructing the required **ovadd** and **ovsave** functions.

```
h = [1 2 -1 1];                  % filter
x = [1 1 2 1 2 2 1 1 3 3 1 1];   % input
y = conv(h,x);                   % expected output

% y =
%     1  3  3  5  3  7  4  3  6  9  5  3  4  0  1

N = 8;                   % FFT length
M = length(h)-1;         % filter order
Lx = length(x);          % input length
Ly = Lx + M;             % length of expected result


% ------------------------------------------------------------
% overlap-add method
% ------------------------------------------------------------

L = N-M;                 % input block length, L=5

Xbuff = buffer(x,L);     % divide x into length-L contiguous blocks
                         % additional zeros are padded at the end
                         % to make complete columns
% Xbuff =
%       1   2   1
%       1   1   1
%       2   1   0
%       1   3   0
%       2   3   0

% H = fft(h(:),N);           % N-point FFT of filter

m = 1:M;                   % selects the last M outputs
ytemp = zeros(M,1);        % temporary length-M vector
Y = [];                    % collect outputs from the input blocks

for s = Xbuff              % loop over the columns of Xbuff

   ys = conv(h,s);                    % time-domain method
%  ys = real(ifft(H.*fft(s,N), N));   % FFT method

   ys(m) = ys(m) + ytemp;  % correct last M samples
   ytemp = ys(L+m);        % save last M samples to be used in next block
   Y = [Y,ys];             % corrected output blocks

end                        % end s-loop

% Y =                      % the first L entries of each column
%      1     7     5       % are correct, and their concatentation
%      3     4     3       % produces the overall output
%      3     3     4
%      5     6     0
%      3     9     1
%
%      5     4     0       % the last M rows of Y can be discarded
%     -1     0     0
%      2     3     0

Y = Y(1:L,:);             % keep only the first L entries from each output

y = [Y(:); ytemp];        % concatenate columns and append last M outputs

y = y(1:Ly);              % discard any extraneous zeros at end

% y’ =
%     1  3  3  5  3  7  4  3  6  9  5  3  4  0  1
% matches expected result


% alternative calculation using the OLA function
% ---------------------------------------------------------
%
% Y = [];                   % collect outputs from the input blocks
% for s = Xbuff             % loop over the columns of Xbuff
%    ys = conv(h,s);                   % time-domain method
% %  ys = real(ifft(H.*fft(s,N), N));   % FFT method
%    Y = [Y,ys];            % uncorrected output blocks
% end
%
% y = ola(Y,N-M);           % N-M = hop size here
% y = y(1:Ly);              % discard any extraneous zeros at end

% another FFT calculation, without using loops
% -------------------------------------------
%
% Nb = size(Xbuff,2);                    % no. of input frames
% Hb = H(:,ones(1,Nb));                  % replicate FFT Nb times
% Y = real(ifft(Hb.*fft(Xbuff,N), N));   % do all FFTs at once
%
% y = ola(Y,N-M);           % N-M = hop size here
% y = y(1:Ly);              % discard any extraneous zeros at end


% -----------------------------------------------------------
% overlap-save method
% -----------------------------------------------------------

% use same h,x, and FFT length N=8, as in the overlap-add case

Xbuff = buffer([x(:); zeros(M,1)], N, M);

% Xbuff =              % pads M zeros in front
%      0   2   1       % input blocks overlap by M samples
%      0   1   3       % pad extra M zeros at end in order
%      0   2   3       % to correctly account for the last block
%      1   2   1
%      1   1   1
%      2   1   0
%      1   3   0
```

```
%      2   3   0

% H = fft(h(:),N);                    % N-point FFT of filter

Y = [];                               % collect output blocks
for s = Xbuff                         % loop over columms of Xbuff
   y = datawrap(conv(h,s), N);        % t-domain, wrapped mod-N
%    y = real(ifft(H.*fft(s,N), N));  % FFT method
   Y = [Y, y(M+1:N)];                 % discard first M bad poits
end

y = Y(:);                             % concatenate output blocks

% Y =
%      1     7     5
%      3     4     3
%      3     3     4
%      5     6     0
%      3     9     1


%  Y’ =
%     1  3  3  5  3
%                    7  4  3  6  9
%                                   5  3  4  0  1
% matches expected result
% y’ =
%     1  3  3  5  3  7  4  3  6  9  5  3  4  0  1
```

Test your functions by applying them to an example of your own choosing that has total input length, $L_x = 19$, filter order $M = 3$, and FFT length $N = 8$.

10.13 Computer Experiment – Matched Filtering

A major application of fast convolution is in implementing the matched filtering operations required for signal detection in radar processing.

A typical radar transmits a finite-duration pulse, such as a chirped sinusoid, at regular intervals at the so-called pulse repetition frequency. If there is target at some distance d, then there will be reflected pulses arriving back at the radar each with a delay t_d corresponding to the time it takes for light to travel from radar to target and back to radar, that is, such that, $2d = ct_d$, or, $d = ct_d/2$. By measuring the time delay t_d, one can infer the distance d of the target.

To simplify the problem, consider a single transmitted pulse of duration of T seconds of some known shape, $s(t)$, $0 \le t \le T$, and a reflected, delayed and attenuated, version of $s(t)$, received in the presence of noise $\nu(t)$,

$$x_{\text{rec}}(t) = a\,s(t - t_d) + \nu(t) \qquad (10.13.1)$$

We will assume that the signals are sampled at a rate f_s with a sampling interval $T_s = 1/f_s$. In radar, the sampling is done typically at the down converted IF, intermediate frequency, level, and the sampling rate is of the order of 1 MHz.

From Communications theory, we know that there are two equivalent approaches to detecting the presence of the target and estimating the time delay t_d.

(i) A *correlator* approach, whereby one computes the cross-correlation between the sampled received signal and the known pulse, that is,

$$R_{xs}(k) = E\big[x_{\rm rec}(n+k)s(n)\big]$$

and determines the lag k_d at which there is a maximum, with the time delay being estimated by, $t_d = k_d T_s$.

(ii) A *matched filter* approach, whereby the received signal $x_{\rm rec}(t)$ is filtered by an FIR filter whose impulse response is the reverse of the transmitted pulse, that is,

$$h(t) = s(T-t)\,, \quad 0 \le t \le T$$

and one determines the time at which the filter output is maximum,

$$y(t) = h(t) * x_{\rm rec}(t) = \max$$

The matched filter approach is the more efficient of the two, and can be done with fast convolution using the FFT. In this part you will actually carry out both approaches and compare the results. Consider the following chirped sinusoid of duration of $T = 1$ msec,

$$s(t) = \sin(20\pi t + 10\pi t^2)\,, \quad 0 \le t \le 1$$

where t is in units of msec. The sinusoid is sampled at a rate of $f_s = 1$ MHz, and the sampled matched filter $h(t_n) = h(nT_s)$ is constructed, where $T_s = 1/f_s = 1\ \mu$sec.

The upper-left graph in Fig. 10.13.1 below shows this sinusoid and a noise-free version of it, received with a delay of $t_d = 3$ msec, and attenuated by an amplitude of $a = 0.5$, that is, $s_{\rm rec}(t) = 0.5\,s(t-3)$, and measured over a time period, $0 \le t \le 5$ msec.

The upper-right graph shows the same signal $s_{\rm rec}(t)$, deeply buried in noise, as in Eq. (10.13.1), such that the received pulse $s_{\rm rec}(t)$ is no longer visible.

The bottom-left graph shows the result of processing the noisy received signal both with the matched filter, and with the cross-correlator. Note that all signals have been normalized to unity maximum for display purposes.

The enclosed file, **pr05match.m**, provides some hints on how to generate the above three graphs. Please carry out the following similar tasks:

(a) Construct the matched filter impulse response, $h(n)$, based on the given chirped pulse.

(b) The bottom-right graph shows a different noisy received signal that contains a reflected pulse with an unknown delay t_d. This signal resides in the included file, **xrec.mat**, and can be loaded into MATLAB with the command,

```
load xrec.mat
```

Process this signal with the matched filter using either your overlap-add or overlap-save functions, Moreover, calculate the cross correlation between $x_{\rm rec}(t)$ and $s(t)$ as a function of the time lag. For this part, you may use the built-in **xcorr** function, but you will need first to extend $s(t)$ to have equal length as $x_{\rm rec}(t)$. Then, make a plot like the one in the bottom-left graph, and determine the signal delay t_d in msec.

(c) Explain why there seems to be a discrepancy between the correlator and the matched-filter approaches in estimating t_d, and how you can compensate for it.

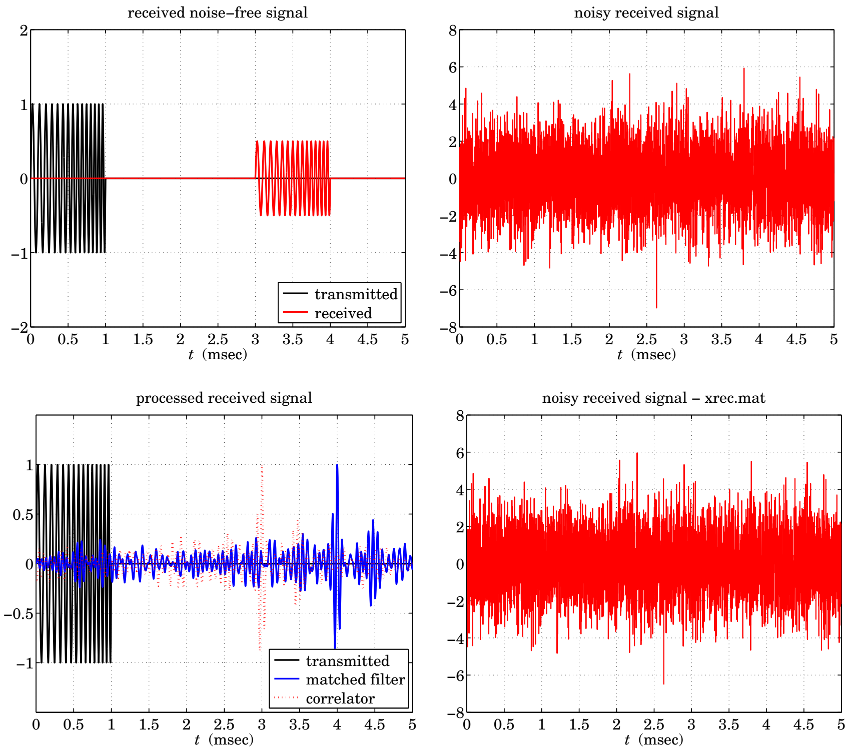


Fig. 10.13.1 Correlator and matched filter inputs and outputs.

10.14 Problems

10.1 Let $\mathbf{x} = [1,2,2,1,2,1,1,2]$. Compute the 4-point DFT of $\mathbf{x}$ using the definition in matrix form. Recompute it by first reducing $\mathbf{x}$ modulo 4 and then computing the 4-DFT of the result. Finally, compute the 4-point IDFT of the result and verify that you recover the mod-4 wrapped version of $\mathbf{x}$.

10.2 Compute the 8-point FFT of the length-8 signal $\mathbf{x} = [5,\ -1,\ -3,\ -1,\ 5,\ -1,\ -3,\ -1]$. Noting that these samples are the first 8 samples of $x(n) = 4\cos(\pi n/2) + \cos(\pi n)$, discuss whether the 8 computed FFT values accurately represent the expected spectrum of $x(n)$. What FFT indices correspond to the two frequencies of the cosinusoids?

10.3 The 8-point DFT $\mathbf{X}$ of an 8-point sequence $\mathbf{x}$ is given by

$$\mathbf{X} = [0, 4, -4j, 4, 0, 4, 4j, 4]$$

Using the FFT algorithm, compute the inverse DFT: $\mathbf{x} = \text{IFFT}(\mathbf{X})$. Using the given FFT $\mathbf{X}$, express $\mathbf{x}$ as a sum of real-valued (co)sinusoidal signals.

10.4 When a very large FFT of a very large data set is required (e.g., of size 2^{16} or larger), it may be computed in stages by partially decimating the time data down to several data sets of manageable dimension, computing their FFTs, and then rebuilding the desired FFT from the smaller ones. See [260–263,642] for a variety of approaches.

In this context, suppose you want to compute a $(4N)$-point FFT but your FFT hardware can only accommodate N-point FFTs. Explain how you might use this hardware to compute that FFT. Discuss how you must partition the time data, what FFTs must be computed, how they must be combined, and how the partial results must be shipped back and forth from secondary storage to the FFT processor in groups of no more than N samples. What is the total number of complex multiplications with your method? Compare this total to the cost of performing the $(4N)$-point FFT in a single pass? Do you observe anything interesting?

10.5 Compute the length-4 circular convolution of the two signals $\mathbf{h} = [1, 2, 1, 2, 1]$, $\mathbf{x} = [1, 1, 1, 1, 1]$ in two ways: (*a*) by computing their linear convolution and then reducing the result mod-4, (*b*) by first reducing $\mathbf{h}$ and $\mathbf{x}$ mod-4, computing the linear convolution of the reduced signals, and reducing the result mod-4.

10.6 Compute the 8-point FFT of $\mathbf{x} = [4, 2, 4, -6, 4, 2, 4, -6]$. Without performing any additional computations, determine the 4-point DFT and the 2-point DFT of the above signal. Explain your reasoning. Using the computed DFT and the inverse DFT formula, express the sequence $x(n)$, $n = 0, 1, \dots, 7$ as a linear combination of real-valued sinusoidal signals. Does your $x(n)$ agree with the given sequence?

10.7 Let $\mathbf{x} = [1, 2, 3, 4, 5]$. (a) Determine a length-6 signal that has the same 5-point DFT as $\mathbf{x}$. (b) Determine a length-7 signal that has the same 5-point DFT as $\mathbf{x}$. Your answers should be nontrivial, that is, do not increase the length of $\mathbf{x}$ by padding zeros at its end.

10.8 Show the property:

$$\frac{1}{N}\big[1 + W_N^k + W_N^{2k} + W_N^{3k} + \cdots + W_N^{(N-1)k}\big] = \delta(k), \qquad k = 0, 1, \dots, N-1$$

10.9 Show the following properties:

a. $W_N = W_{2N}^2 = W_{3N}^3 = \cdots = W_{pN}^p$

b. $X_N(k) = X_{pN}(pk), \qquad k = 0, 1, \dots, N-1$

where W_{pN} is the twiddle factor of order pN, p is any integer, $X_N(k)$ denotes the N-point DFT, and $X_{pN}(k)$ the (pN)-point DFT of a common signal $x(n)$ of length L.

10.10 Consider a 16-point signal x_n, $0 \le n \le 15$, with 16-point DFT X_k, $0 \le k \le 15$, namely,

$$[X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, X_{10}, X_{11}, X_{12}, X_{13}, X_{14}, X_{15}]$$

Show that the 8-point DFT of the given 16-point signal is:

$$[X_0, X_2, X_4, X_6, X_8, X_{10}, X_{12}, X_{14}]$$

10.11 The following analog signal $x(t)$, where t is in msec, is sampled at a rate of 8 kHz:

$$x(t) = \cos(24\pi t) + 2\sin(12\pi t)\cos(8\pi t)$$

a. Determine the signal $x_a(t)$ that is aliased with $x(t)$.

b. Eight consecutive samples of $x(t)$ are collected. *Without* performing any DFT or FFT operations, determine the 8-point DFT of these 8 samples.

10.12 Consider the following 8-point signal, defined for $n = 0, 1, \dots, 7$:

$$x(n) = 1 + 2\sin\Big(\frac{\pi n}{4}\Big) - 2\sin\Big(\frac{\pi n}{2}\Big) + 2\sin\Big(\frac{3\pi n}{4}\Big) + 3(-1)^n$$

Without performing any DFT or FFT computations, determine the 8-point DFT of this signal.

10.13 Let $x(n) = \cos(\pi n/2) + 2\cos(\pi n/8)$, $n = 0, 1, \dots, 15$. Without performing any actual DFT/FFT computations, determine the 16-point DFT of this 16-point signal. [*Hint*: Compare $x(n)$ with the 16-point inverse DFT formula.]

10.14 Let $x(n) = \cos(\pi n/2) + 2\cos(\pi n/8)$, $n = 0, 1, \dots, 31$. Without performing any actual DFT/FFT computations, determine the 32-point DFT of this 32-point signal.

10.15 Consider the following length-16 signal:

$$x(n) = 0.5 + 2\sin(0.5\pi n) + 1.5\cos(\pi n), \qquad n = 0, 1, \dots, 15$$

a. Determine the DTFT $X(\omega)$ of this finite sequence, and sketch it roughly versus ω in the range $0 \le \omega \le 2\pi$. [*Hint:* Remember that each spectral line gets replaced by the rectangular window's frequency response.]

b. Without performing any DFT or FFT computations, determine the 16-point DFT of this sequence. Then, determine the 8-point DFT of the *same* sequence.

c. Place the 16-point DFT values on the graph of $X(\omega)$ of part (a).

10.16 Let $\mathbf{X} = A\mathbf{x}$ be the N-point DFT of the length-N signal $\mathbf{x}$ expressed in matrix form, where A is the $N \times N$ DFT matrix defined by its matrix elements $A_{kn} = W_N^{kn}$, $k, n = 0, 1, \dots, N-1$. Show that the inverse of this matrix can be obtained essentially by conjugating A, that is,

$$A^{-1} = \frac{1}{N}A^*$$

Therefore the IDFT can be expressed by $\mathbf{x} = A^{-1}\mathbf{X} = A^*\mathbf{X}/N$. Explain how this result justifies the rule:

$$\text{IFFT}(\mathbf{X}) = \frac{1}{N}\big(\text{FFT}(\mathbf{X}^*)\big)^*$$

10.17 Let $X(k)$ be the N-point DFT of a length-N (complex-valued) signal $x(n)$. Use the results of Problem 10.16 to show the Parseval relation:

$$\sum_{n=0}^{N-1}|x(n)|^2 = \frac{1}{N}\sum_{k=0}^{N-1}|X(k)|^2$$

10.18 Compute the mod-4, mod-5, mod-6, mod-7, and mod-8 circular convolutions of the signals $\mathbf{x} = [2,\ 1,\ 1,\ 2]$ and $\mathbf{h} = [1,\ -1,\ -1,\ 1]$. For what value of N does the mod-N circular convolution agree with the ordinary linear convolution?

10.19 Compute the modulo-8 circular convolution of the two signals

$$\mathbf{h} = [2, 1, 1, 1, 2, 1, 1, 1], \qquad \mathbf{x} = [2, 1, 2, -3, 2, 1, 2, -3]$$

in two ways:

a. Working exclusively in the time domain.

b. Using the formula:

$$\tilde{\mathbf{y}} = \text{IFFT}\big(\text{FFT}(\mathbf{h})\cdot\text{FFT}(\mathbf{x})\big)$$

implemented via 8-point FFTs. All the computational details of the required FFTs must be shown explicitly.

10.20 a. Compute the 8-point FFT of the 8-point signal $\mathbf{x} = [6, 1, 0, 1, 6, 1, 0, 1]$.

b. Using the inverse DFT formula, express $\mathbf{x}$ as a linear combination of real-valued sinusoids.

c. Find two other signals, one of length-9 and one of length-10, that have the same 8-point DFT as $\mathbf{x}$. These signals must not begin or end with zeros.

d. Compute the 4-point FFT of $\mathbf{x}$ by carrying out a *single* 4-point FFT.

10.21 Let $A(k)$ be the N-point DFT of a *real-valued* signal $a(n)$, $n = 0, 1, \dots, N-1$. Prove the symmetry property:

$$A(k)^* = A(N-k), \qquad k = 0, 1, \dots, N-1$$

If we think of $A(k)$ as an N-dimensional array, then how can we state the above relationship at $k = 0$?

10.22 *Two Real-Valued Signals at a Time.* Let $x(n) = a(n) + jb(n)$ be a length-N complex-valued signal and let $X(k)$ be its N-point DFT. Let $A(k)$ and $B(k)$ denote the N-point DFTs of the real and imaginary parts $a(n)$ and $b(n)$ of $x(n)$. Show that they can be recovered from $X(k)$ by

$$A(k) = \frac{1}{2}\big[X(k) + X(N-k)^*\big], \qquad B(k) = \frac{1}{2j}\big[X(k) - X(N-k)^*\big]$$

for $k = 0, 1, \dots, N-1$. If we think of $X(k)$ as an N-dimensional array, then how can we state the above relationships at $k = 0$?

Thus, the DFTs of *real-valued signals* can be computed *two at a time* by computing the DFT of a *single* complex-valued signal.

10.23 *FFT of Real-Valued Signal.* Using the results of Problem 10.22, show that the N-point FFT $X(k)$ of an N-point real-valued signal $x(n)$, $n = 0, 1, \dots, N-1$ can be computed efficiently as follows: First, pack the even and odd parts of $x(n)$ into a complex-valued signal of length $N/2$, that is, define

$$y(n) = x(2n) + jx(2n+1) \equiv g(n) + jh(n), \qquad n = 0, 1, \dots, \frac{N}{2} - 1$$

Then, compute the $N/2$-point FFT of $y(n)$, say, $Y(k)$, $k = 0, 1, \dots, N/2 - 1$, and extract the $N/2$-point FFTs of $g(n)$ and $h(n)$ by

$$G(k) = \frac{1}{2}\big[Y(k) + Y(\frac{N}{2} - k)^*\big], \qquad H(k) = \frac{1}{2j}\big[Y(k) - Y(\frac{N}{2} - k)^*\big]$$

for $k = 0, 1, \dots, N/2 - 1$. And finally, construct the desired N-point FFT by

$$X(k) = G(k) + W_N^k H(k), \qquad X(k + \frac{N}{2}) = G(k) - W_N^k H(k)$$

for $k = 0, 1, \dots, N/2 - 1$. What happens at $k = 0$?

Determine the relative computational savings of this method versus performing the N-point FFT of $x(n)$ directly.

10.24 *Computer Experiment: FFT of Real-Valued Signal.* Write a C routine `fftreal.c` that implements the method of Problem 10.23. The routine must have inputs/output declarations:

```
void fftreal(N, x, X)
int N;                          must be a power of 2
double *x;                      real-valued N-dimensional time data
complex *X;                     complex N-dimensional FFT array
```

The routine must invoke the routine `fft.c` once on the time-decimated, complexified, input. In rebuilding the final DFT $X(k)$, special care must be exercised at $k = 0$.

Write a small main program to test the routine by comparing its output to the output of `fft` called on the full input array as usual.

10.25 Consider the following N-point signal and its reverse:

$$\begin{aligned}\mathbf{x} &= [x_0, x_1, \dots, x_{N-1}]\\ \mathbf{x}_R &= [x_{N-1}, \dots, x_1, x_0]\end{aligned}$$

Show that the z-transform and N-point DFT of the reversed signal can be expressed as:

$$\begin{aligned}X_R(z) &= z^{-(N-1)}X(z^{-1})\\ X_R(k) &= W_N^{-k}X(N-k), \qquad k = 0, 1, \dots, N-1\end{aligned}$$

Show that in the time domain the reversal process is equivalent to a two-step process of first reflecting the signal around the origin $n = 0$, and then delaying it by $N - 1$ units.

10.26 *Discrete Cosine Transform (DCT).* Consider a length-N real-valued signal $\mathbf{x}$ and its reverse as defined in Problem 10.25. Construct the concatenated signal of length $2N$:

$$\mathbf{y} = [\mathbf{x}, \mathbf{x}_R] = [x_0, x_1, \dots, x_{N-1}, x_{N-1}, \dots, x_1, x_0]$$

a. Show that its z-transform can be expressed in terms of the z-transform of $\mathbf{x}$:

$$Y(z) = X(z) + z^{-N}X_R(z) = X(z) + z^{-(2N-1)}X(z^{-1})$$

b. Let Y_k be the $(2N)$-point DFT of $\mathbf{y}$. Show that it can be expressed in the form:

$$Y_k = 2e^{j\omega_k/2}C_k, \qquad k = 0, 1, \dots, 2N-1$$

where $\omega_k = 2\pi k/(2N) = \pi k/N$ is the kth frequency for the $(2N)$-point DFT and C_k is one form of the discrete cosine transform of x_n given by:

$$C_k = \sum_{n=0}^{N-1} x_n \cos\big(\omega_k(n + 1/2)\big) \tag{10.14.1}$$

[*Hint:* Evaluate part (a) at the $(2N)$th roots of unity and multiply by $z^{-1/2}$.]

c. Using the results of Problem 10.21, show that C_k satisfies the symmetry property:

$$C_{2N-k} = -C_k, \qquad k = 0, 1, \dots, 2N-1$$

In particular, show $C_N = 0$.

d. Applying the inverse DFT equation on Y_k, show the inverse DCT:

$$x_n = \frac{1}{N}\sum_{k=0}^{2N-1} C_k e^{j\omega_k(n+1/2)}\,, \qquad n = 0,1,\dots,N-1$$

Using the symmetry property of part (c), show the alternative inverse DCT, which uses only the first N DCT coefficients C_k, $k = 0,1,\dots,N-1$:

$$x_n = \frac{1}{N}\left[C_0 + 2\sum_{k=1}^{N-1} C_k \cos\bigl(\omega_k(n+1/2)\bigr)\right], \qquad n = 0,1,\dots,N-1 \qquad (10.14.2)$$

Together, Eqs. (10.14.1) and (10.14.2) form a forward/inverse DCT pair. The relationship to the doubled signal $\mathbf{y}$ allows an efficient calculation using $(2N)$-point FFTs [257–259].

10.27 a. Let $X_N(k)$ denote the N-point DFT of a length-L sequence $x(n)$, $n = 0,1,\dots,L-1$. Show the relationships:

$$X_N(k) = X_{2N}(2k)\,, \qquad k = 0,1,\dots,N-1$$

b. In particular, we have $X_4(k) = X_8(2k)$, for $k = 0,1,2,3$. That is, the 4-point DFT of a sequence can be obtained by keeping every other entry of the 8-point DFT of that sequence.

10.28 Consider a length-5 sequence and its "circular shifts"

$$\begin{aligned}
\mathbf{x}_0 &= [x_0, x_1, x_2, x_3, x_4]\\
\mathbf{x}_1 &= [x_4, x_0, x_1, x_2, x_3]\\
\mathbf{x}_2 &= [x_3, x_4, x_0, x_1, x_2]\\
\mathbf{x}_3 &= [x_2, x_3, x_4, x_0, x_1]\\
\mathbf{x}_4 &= [x_1, x_2, x_3, x_4, x_0]
\end{aligned}$$

Show that the 5-point DFT $X_i(k)$ of $\mathbf{x}_i$ is related to the 5-point DFT $X_0(k)$ of $\mathbf{x}_0$ by

$$X_i(k) = W_5^{ik} X_0(k)\,, \qquad \text{for} \quad i = 1,2,3,4$$

Explain this result in terms of ordinary "linear" shifts of the original sequence $\mathbf{x}_0$.

10.29 Show that the following, successively shorter, signals all have the same 4-point DFT:

$$\begin{bmatrix} x_0\\ x_1\\ x_2\\ x_3\\ x_4\\ x_5\\ x_6\\ x_7 \end{bmatrix}, \quad
\begin{bmatrix} x_0\\ x_1\\ x_2\\ x_3+x_7\\ x_4\\ x_5\\ x_6\\ 0 \end{bmatrix}, \quad
\begin{bmatrix} x_0\\ x_1\\ x_2+x_6\\ x_3+x_7\\ x_4\\ x_5\\ 0\\ 0 \end{bmatrix}, \quad
\begin{bmatrix} x_0\\ x_1+x_5\\ x_2+x_6\\ x_3+x_7\\ x_4\\ 0\\ 0\\ 0 \end{bmatrix}, \quad
\begin{bmatrix} x_0+x_4\\ x_1+x_5\\ x_2+x_6\\ x_3+x_7\\ 0\\ 0\\ 0\\ 0 \end{bmatrix}$$

10.30 Using the *overlap-save* method of fast convolution implemented in the time domain using length-8 circular convolutions, compute the ordinary convolution of the "long" signal

$$\mathbf{x} = [1,1,1,1,3,3,3,3,1,1,1,2,2,2,2,1,1,1,1]$$

with the "short" filter

$$\mathbf{h} = [1,-1,-1,1]$$

and explain any discrepancies from the correct answer. Repeat using the *overlap-add* method.

10.31 A *periodic* triangular waveform of period $T_0 = 1$ sec is defined over one period $0 \le t \le 1$ sec as follows (see also Fig. 1.9.1):

$$x(t) = \begin{cases} t, & \text{if } \ 0 \le t \le 0.25\\ 0.5 - t, & \text{if } \ 0.25 \le t \le 0.75\\ t - 1, & \text{if } \ 0.75 \le t \le 1 \end{cases}$$

The signal $x(t)$ is sampled at a rate of 8 Hz and the sampled signal $x(nT)$ is immediately reconstructed into analog form using an *ideal* reconstructor. Because $x(t)$ is not bandlimited, aliasing effects will cause the reconstructed signal to be different from $x(t)$. Show that the aliased reconstructed signal will have the form:

$$x_{\text{al}}(t) = A\sin(2\pi f_1 t) + B\sin(2\pi f_2 t)$$

What are the frequencies f_1 and f_2? Determine the amplitudes A and B by performing an appropriate 8-point FFT by hand. Explain how the negative frequencies in $x_{\text{al}}(t)$ are represented in this FFT.

10.32 A length-L input signal is to be filtered by an order-M FIR filter using the overlap-save method of fast convolution, implemented via N-point FFTs. Assume that $L \gg N$ and $N > M$.

a. Derive an expression for the total number of multiplications required to compute the output, in terms of L, N, and M.

b. Repeat part (a) if the overlap-add method is used.

10.33 *Computer Experiment: Overlap-Save Method.* Write a stand-alone C or MATLAB program, say `ovsave.c`, that implements the overlap-save method of fast convolution. The program must have usage:

```
ovsave h.dat N < x.dat > y.dat
```

Like the program `firfilt.c` of Problem 4.10, it must read dynamically the impulse response coefficients from a file `h.dat`. It must keep reading the input samples in blocks of length N (overlapped by M points), processing each block, and writing the output block. The processing of each block must be implemented by N-point FFTs, that is,

$$\tilde{\mathbf{y}} = \text{IFFT}\bigl(\mathbf{H}\cdot\text{FFT}(\mathbf{x})\bigr)$$

where the FFT of the filter $\mathbf{H} = \text{FFT}(\mathbf{h})$ may be computed once and used in processing all the input blocks.

Care must be exercised in handling the first M inputs, where M zeros must be padded to the beginning of the input. When the end-of-file of the input is detected, the program must calculate correctly the input-off output transients. (The output of this program and that of `firfilt.c` must be identical, up to perhaps some last zeros.)

11

FIR Digital Filter Design

The *filter design problem* is the problem of constructing the transfer function of a filter that meets *prescribed* frequency response specifications.

The input to any filter design method is the set of desired specifications and the output is the finite impulse response coefficient vector $\mathbf{h} = [h_0, h_1, \dots, h_{N-1}]$ in the case of FIR filters, or the numerator and denominator coefficient vectors $\mathbf{b} = [b_0, b_1, \dots, b_M]$, $\mathbf{a} = [1, a_1, \dots, a_M]$ in the case of IIR filters.

The subject of FIR and IIR digital filter design is very extensive [2–8]. In this and the next chapter, we present only a small cross section of available design methods—our objective being to give the flavor of the subject, while at the same time presenting some practical methods.

The two main advantages of FIR filters are their *linear phase* property and their guaranteed *stability* because of the absence of poles. Their potential disadvantage is that the requirement of sharp filter specifications can lead to long filter lengths N, consequently increasing their computational cost. Recall from Chapter 4 that modern DSP chips require N MACs per output point computed.

The main advantages of IIR filters are their *low computational cost* and their *efficient implementation* in cascade of second-order sections. Their main disadvantage is the potential for instabilities introduced when the quantization of the coefficients pushes the poles outside the unit circle. For IIR filters, linear phase cannot be achieved exactly over the entire Nyquist interval, but it can be achieved *approximately* over the relevant passband of the filter, for example, using Bessel filter designs.

11.1 Window Method

11.1.1 Ideal Filters

The window method is one of the simplest methods of designing FIR digital filters. It is well suited for designing filters with *simple* frequency response shapes, such as ideal lowpass filters. Some typical filter shapes that can be designed are shown in Figs. 11.1.1 and 11.1.2. For arbitrary shapes, a variant of the method, known as *frequency sampling* method, may be used; it will be discussed in Section 11.4.

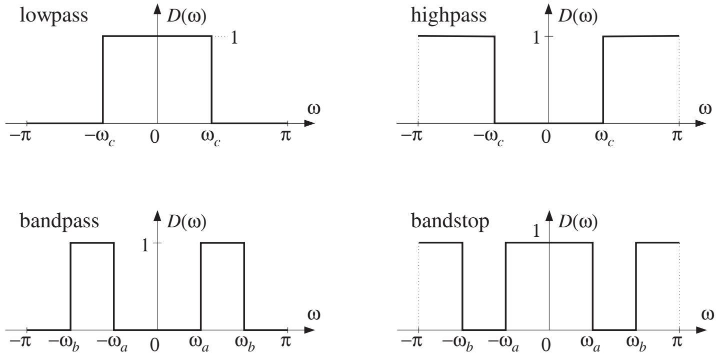


Fig. 11.1.1 Ideal lowpass, highpass, bandpass, and bandstop filters.

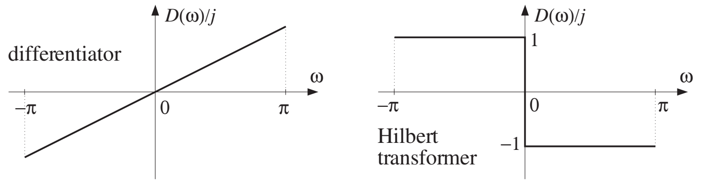


Fig. 11.1.2 Ideal differentiator and Hilbert transformer filters.

A given desired ideal frequency response, say $D(\omega)$, being periodic in ω with period 2π, need only be specified over one complete Nyquist interval $-\pi \le \omega \le \pi$. The corresponding impulse response, say $d(k)$, is related to $D(\omega)$ by the DTFT and inverse DTFT relationships:

$$D(\omega) = \sum_{k=-\infty}^{\infty} d(k)e^{-j\omega k} \quad \Leftrightarrow \quad d(k) = \int_{-\pi}^{\pi} D(\omega)e^{j\omega k}\,\frac{d\omega}{2\pi} \tag{11.1.1}$$

In general, the impulse response $d(k)$ will be *double-sided and infinite*. For many ideal filter shapes, the ω-integration in Eq. (11.1.1) can be done in closed form. For example, for the *lowpass filter* shown in Fig. 11.1.1, the quantity $D(\omega)$ is defined over the Nyquist interval by

$$D(\omega) = \begin{cases} 1, & \text{if } -\omega_c \le \omega \le \omega_c \\ 0, & \text{if } -\pi \le \omega < -\omega_c, \text{ or } \omega_c < \omega \le \pi \end{cases}$$

Therefore, Eq. (11.1.1) gives:

$$\begin{aligned} d(k) &= \int_{-\pi}^{\pi} D(\omega)e^{j\omega k}\,\frac{d\omega}{2\pi} = \int_{-\omega_c}^{\omega_c} 1\cdot e^{j\omega k}\,\frac{d\omega}{2\pi} \\ &= \left[\frac{e^{j\omega k}}{2\pi jk}\right]_{-\omega_c}^{\omega_c} = \frac{e^{j\omega_c k} - e^{-j\omega_c k}}{2\pi jk} \end{aligned}$$

which can be rewritten as

$$\text{(lowpass filter)}\qquad \boxed{d(k)=\frac{\sin(\omega_c k)}{\pi k}}\,,\qquad -\infty<k<\infty \tag{11.1.2}$$

For computational purposes, the case $k = 0$ must be handled separately. Taking the limit $k \to 0$, we find from Eq. (11.1.2):

$$d(0)=\frac{\omega_c}{\pi} \tag{11.1.3}$$

Similarly, we find for the highpass, bandpass, and bandstop filters of Fig. 11.1.1, defined over $-\infty < k < \infty$

$$\begin{aligned}
&\text{(highpass filter)} && d(k)=\delta(k)-\frac{\sin(\omega_c k)}{\pi k}\\
&\text{(bandpass filter)} && d(k)=\frac{\sin(\omega_b k)-\sin(\omega_a k)}{\pi k}\\
&\text{(bandstop filter)} && d(k)=\delta(k)-\frac{\sin(\omega_b k)-\sin(\omega_a k)}{\pi k}
\end{aligned} \tag{11.1.4}$$

Note that for the same values of the cutoff frequencies ω_c, ω_a, ω_b, the lowpass/highpass and bandpass/bandstop filters are *complementary*, that is, their impulse responses add up to a unit impulse $\delta(k)$ and their frequency responses add up to unity (as can also be seen by inspecting Fig. 11.1.1):

$$\begin{aligned}
d_{LP}(k)+d_{HP}(k)=\delta(k) \quad&\Leftrightarrow\quad D_{LP}(\omega)+D_{HP}(\omega)=1\\
d_{BP}(k)+d_{BS}(k)=\delta(k) \quad&\Leftrightarrow\quad D_{BP}(\omega)+D_{BS}(\omega)=1
\end{aligned} \tag{11.1.5}$$

As we see below, such complementarity properties can be exploited to simplify the implementation of loudspeaker cross-over networks and graphic equalizers.

The ideal *differentiator* filter of Fig. 11.1.2 has frequency response $D(\omega)=j\omega$, defined over the Nyquist interval. The ideal *Hilbert transformer* response can be expressed compactly as $D(\omega)=-j\text{sign}(\omega)$, where $\text{sign}(\omega)$ is the signum function which is equal to ± 1 depending on the algebraic sign of its argument. The ω-integrations in Eq. (11.1.1) give the impulse responses:

$$\begin{aligned}
&\text{(differentiator)} && d(k)=\frac{\cos(\pi k)}{k}-\frac{\sin(\pi k)}{\pi k^2}\\
&\text{(Hilbert transformer)} && d(k)=\frac{1-\cos(\pi k)}{\pi k}
\end{aligned} \tag{11.1.6}$$

Both filters have $d(0)=0$, as can be verified by carefully taking the limit $k \to 0$. Both impulse responses $d(k)$ are real-valued and *odd* (antisymmetric) functions of k. By contrast, the filters of Fig. 11.1.1 all have impulse responses that are real and *even* (symmetric) in k. We will refer to the two classes of filters of Figs. 11.1.1 and 11.1.2 as the *symmetric* and *antisymmetric* classes.

In the frequency domain, the symmetric types are characterized by a frequency response $D(\omega)$ which is *real and even* in ω; the antisymmetric ones have $D(\omega)$ which is *imaginary and odd* in ω. One of the main consequences of these frequency properties is the *linear phase* property of the window designs.

11.1.2 Rectangular Window

The window method consists of truncating, or rectangularly windowing, the double-sided $d(k)$ to a finite length. For example, we may keep only the coefficients:

$$\boxed{d(k)=\int_{-\pi}^{\pi}D(\omega)e^{j\omega k}\,\frac{d\omega}{2\pi}\,,\qquad -M\le k\le M} \tag{11.1.7}$$

Because the coefficients are taken equally for positive and negative k's, the total number of coefficients will be *odd*, that is, $N = 2M + 1$ (even values of N are also possible, but not discussed in this text). The resulting N-dimensional coefficient vector is the *FIR impulse response* approximating the infinite ideal response:

$$\mathbf{d} = [d_{-M},\dots,d_{-2},d_{-1},d_0,d_1,d_2,\dots,d_M] \tag{11.1.8}$$

The time origin $k = 0$ is at the middle d_0 of this vector. To make the filter causal we may shift the time origin to the left of the vector and re-index the entries accordingly:

$$\mathbf{h} = \mathbf{d} = [h_0,\dots,h_{M-2},h_{M-1},h_M,h_{M+1},h_{M+2},\dots,h_{2M}] \tag{11.1.9}$$

where we defined $h_0 = d_{-M}$, $h_1 = d_{-M+1}$, ..., $h_M = d_0$, ..., $h_{2M} = d_M$. Thus, the vectors $\mathbf{d}$ and $\mathbf{h}$ are the same, with the understanding that $\mathbf{d}$'s origin is in its middle and $\mathbf{h}$'s at its left. The definition of $\mathbf{h}$ may be thought of as time-delaying the double-sided sequence $d(k)$, $-M \le k \le M$, by M time units to make it causal:

$$\boxed{h(n)=d(n-M)\,,\qquad n=0,1,\dots,N-1} \tag{11.1.10}$$

The operations of windowing and delaying are shown in Fig. 11.1.3. To summarize, the steps of the *rectangular window method* are simply:

1. Pick an odd length $N = 2M + 1$, and let $M = (N - 1)/2$.
2. Calculate the N coefficients $d(k)$ from Eq. (11.1.7), and
3. Make them causal by the delay (11.1.10).

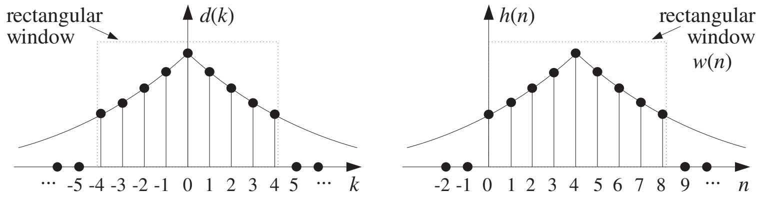

rectangular window
d(k)
··· -5 -4 -3 -2 -1 0 1 2 3 4 5 ··· k
h(n)
rectangular window
w(n)
-2 -1 0 1 2 3 4 5 6 7 8 9 ··· n

Fig. 11.1.3 Rectangularly windowed impulse response, with $N = 9$, $M = 4$.

For example, the N-dimensional approximation to the ideal lowpass filter of Eq. (11.1.2) will be:

$$h(n)=d(n-M)=\frac{\sin\big(\omega_c(n-M)\big)}{\pi(n-M)}\,,\quad n=0,\dots,M,\dots,N-1 \tag{11.1.11}$$

where we must calculate separately $h(M)=d(0)=\omega_c/\pi$. For other ideal filter shapes, we can use the functions $d(k)$ of Eqs. (11.1.4) or (11.1.6). Once the impulse response coefficients are calculated, the filter may be implemented by its FIR filtering equation, using the routines `fir` or `cfir` of Chapter 4:

$$y_n=\sum_{m=0}^{N-1}h_m x_{n-m} \tag{11.1.12}$$

Example 11.1.1: Determine the length-11, rectangularly windowed impulse response that approximates (a) an ideal lowpass filter of cutoff frequency $\omega_c=\pi/4$, (b) the ideal differentiator filter, and (c) the ideal Hilbert transformer filter.

Solution: With $N=11$, we have $M=(N-1)/2=5$. For the lowpass filter, we evaluate Eq. (11.1.2), that is,

$$d(k)=\frac{\sin(\pi k/4)}{\pi k}\,,\quad \text{for } -5\le k\le 5$$

We find the numerical values:

$$\mathbf{h}=\mathbf{d}=\left[-\frac{\sqrt{2}}{10\pi},\,0,\,\frac{\sqrt{2}}{6\pi},\,\frac{1}{2\pi},\,\frac{\sqrt{2}}{2\pi},\,\frac{1}{4},\,\frac{\sqrt{2}}{2\pi},\,\frac{1}{2\pi},\,\frac{\sqrt{2}}{6\pi},\,0,\,-\frac{\sqrt{2}}{10\pi}\right]$$

For the differentiator filter, the second term, $\sin(\pi k)/\pi k^2$, vanishes for all values $k\neq 0$. Therefore, we find:

$$\mathbf{h}=\mathbf{d}=\left[\frac{1}{5},\,-\frac{1}{4},\,\frac{1}{3},\,-\frac{1}{2},\,1,\,0,\,-1,\,\frac{1}{2},\,-\frac{1}{3},\,\frac{1}{4},\,-\frac{1}{5}\right]$$

And, for the Hilbert transformer:

$$\mathbf{h}=\mathbf{d}=\left[-\frac{2}{5\pi},\,0,\,-\frac{2}{3\pi},\,0,\,-\frac{2}{\pi},\,0,\,\frac{2}{\pi},\,0,\,\frac{2}{3\pi},\,0,\,\frac{2}{5\pi}\right]$$

Note that the lowpass filter's impulse response is symmetric about its middle, whereas the differentiator's and Hilbert transformer's are antisymmetric. Note also that because of the presence of the factor $1-\cos(\pi k)$, every other entry of the Hilbert transformer vanishes. This property can be exploited to reduce by half the total number of multiplications required in the convolutional equation Eq. (11.1.12). □

In the frequency domain, the FIR approximation to $D(\omega)$ is equivalent to truncating the DTFT Fourier series expansion (11.1.1) to the finite sum:

$$\boxed{\hat{D}_M(\omega)=\sum_{k=-M}^{M}d(k)e^{-j\omega k}} \tag{11.1.13}$$

Replacing $z=e^{j\omega}$, we may also write it as the double-sided z-transform:

$$\hat{D}_M(z)=\sum_{k=-M}^{M}d(k)z^{-k} \tag{11.1.14}$$

The final length-N filter obtained by Eq. (11.1.10) will have transfer function:

$$\boxed{H(z)=z^{-M}\hat{D}_M(z)=z^{-M}\sum_{k=-M}^{M}d(k)z^{-k}} \tag{11.1.15}$$

and frequency response:

$$\boxed{H(\omega)=e^{-j\omega M}\hat{D}_M(\omega)=e^{-j\omega M}\sum_{k=-M}^{M}d(k)e^{-j\omega k}} \tag{11.1.16}$$

Example 11.1.2: To illustrate the definition of $H(z)$, consider a case with $N=7$ and $M=(N-1)/2=3$. Let the FIR filter weights be $\mathbf{d}=[d_{-3},d_{-2},d_{-1},d_0,d_1,d_2,d_3]$ with truncated z-transform:

$$\hat{D}_M(z)=d_{-3}z^3+d_{-2}z^2+d_{-1}z+d_0+d_1z^{-1}+d_2z^{-2}+d_3z^{-3}$$

Delaying it by $M=3$, we get the causal transfer function:

$$\begin{aligned}H(z)=z^{-3}\hat{D}_M(z)&=z^{-3}\big(d_{-3}z^3+d_{-2}z^2+d_{-1}z+d_0+d_1z^{-1}+d_2z^{-2}+d_3z^{-3}\big)\\&=d_{-3}+d_{-2}z^{-1}+d_{-1}z^{-2}+d_0z^{-3}+d_1z^{-4}+d_2z^{-5}+d_3z^{-6}\\&=h_0+h_1z^{-1}+h_2z^{-2}+h_3z^{-3}+h_4z^{-4}+h_5z^{-5}+h_6z^{-6}\end{aligned}$$

where we defined $h(n)=d(n-3)$, $n=0,1,2,3,4,5,6$. □

The *linear phase property* of the window design is a direct consequence of Eq. (11.1.16). The truncated $\hat{D}_M(\omega)$ has the same symmetry/antisymmetry properties as $D(\omega)$. Thus, in the *symmetric* case, $\hat{D}_M(\omega)$ will be real and even in ω. It follows from Eq. (11.1.16) that the designed FIR filter will have linear phase, arising essentially from the delay factor $e^{-j\omega M}$. More precisely, we may write the real factor $\hat{D}_M(\omega)$ in terms of its positive magnitude and its sign:

$$\hat{D}_M(\omega)=\text{sign}\big(\hat{D}_M(\omega)\big)\,|\hat{D}_M(\omega)|=e^{j\pi\beta(\omega)}\,|\hat{D}_M(\omega)|$$

where $\beta(\omega)=\big[1-\text{sign}\big(\hat{D}_M(\omega)\big)\big]/2$, which is zero or one depending on the sign of $\hat{D}_M(\omega)$. It follows that $H(\omega)$ will be:

$$H(\omega)=e^{-j\omega M}\hat{D}_M(\omega)=e^{-j\omega M+j\pi\beta(\omega)}\,|\hat{D}_M(\omega)|$$

Thus, its magnitude and phase responses will be:

$$\boxed{|H(\omega)|=|\hat{D}_M(\omega)|\,,\quad \arg H(\omega)=-\omega M+\pi\,\beta(\omega)} \tag{11.1.17}$$

making the phase response piece-wise linear in ω with 180° jumps at those ω where $\hat{D}_M(\omega)$ changes sign. For the *antisymmetric* case, $\hat{D}_M(\omega)$ will be pure imaginary, that is, of the form $\hat{D}_M(\omega) = jA(\omega)$. The factor j may be made into a phase by writing it as $j = e^{j\pi/2}$. Thus, we have

$$H(\omega) = e^{-j\omega M}\hat{D}_M(\omega) = e^{-j\omega M}e^{j\pi/2}A(\omega) = e^{-j\omega M}e^{j\pi/2}e^{j\pi\alpha(\omega)}\,|A(\omega)|$$

where $\alpha(\omega) = \big[1 - \text{sign}\big(A(\omega)\big)\big]/2$, which gives for the magnitude and phase responses:

$$\boxed{|H(\omega)| = |A(\omega)|\,,\quad \arg H(\omega) = -\omega M + \frac{\pi}{2} + \pi\,\alpha(\omega)} \tag{11.1.18}$$

How good is the rectangular window design? How well does the truncated $\hat{D}_M(\omega)$ represent the desired response $D(\omega)$? In other words, how good is the approximation $\hat{D}_M(\omega) \simeq D(\omega)$?

Intuitively one would expect that $\hat{D}_M(\omega) \to D(\omega)$ as N increases. This is true for any ω which is a point of *continuity* of $D(\omega)$, but it fails at points of *discontinuity*, such as at the transition edges from passband to stopband. Around these edges one encounters the celebrated *Gibbs phenomenon* of Fourier series, which causes the approximation to be bad regardless of how large N is.

To illustrate the nature of the approximation $\hat{D}_M(\omega) \simeq D(\omega)$, we consider the design of an ideal lowpass filter of cutoff frequency $\omega_c = 0.3\pi$, approximated by a rectangularly windowed response of length $N = 41$ and then by another one of length $N = 121$. For the case $N = 41$, we have $M = (N-1)/2 = 20$. The designed impulse response is given by Eq. (11.1.10):

$$h(n) = d(n-20) = \frac{\sin\big(0.3\pi(n-20)\big)}{\pi(n-20)}\,, \qquad n = 0, 1, \dots, 40$$

and in particular, $h(20) = d(0) = \omega_c/\pi = 0.3$. The second design has $N = 121$ and $M = 60$. Its impulse response is, with $h(60) = d(0) = 0.3$:

$$h(n) = d(n-60) = \frac{\sin\big(0.3\pi(n-60)\big)}{\pi(n-60)}\,, \qquad n = 0, 1, \dots, 120$$

The two impulse responses are plotted in Fig. 11.1.4. Note that the portion of the second response extending ±20 samples around the central peak at $n = 60$ coincides numerically with the first response. The corresponding magnitude responses are shown in Fig. 11.1.5. An intermediate case having $N = 81$ is shown in Figs. 11.1.6 and 11.1.7. In Fig. 11.1.5, the magnitude responses were computed by evaluating:

$$H(\omega) = \sum_{n=0}^{N-1} h(n)e^{-j\omega n} \tag{11.1.19}$$

The length-N impulse response $h(n)$ defined in Eq. (11.1.10) may be thought of formally as the rectangularly windowed double-sided sequence defined by

$$\boxed{h(n) = w(n)d(n-M)}\,, \qquad -\infty < n < \infty \tag{11.1.20}$$

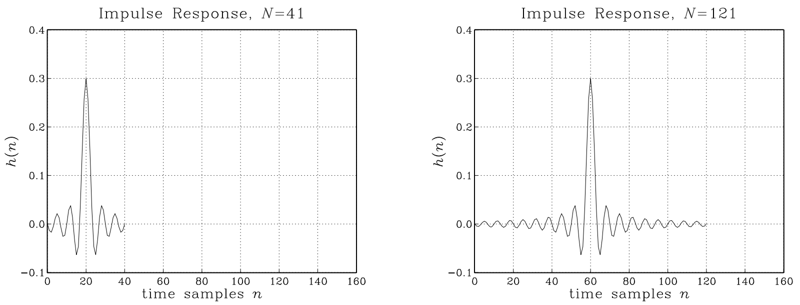


Fig. 11.1.4 Rectangularly windowed impulse responses for $N = 41$ and $N = 121$.

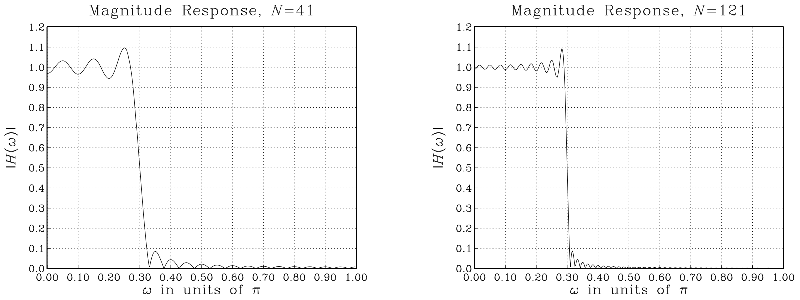


Fig. 11.1.5 Rectangularly windowed magnitude responses for $N = 41$ and $N = 121$.

where $w(n)$ is the length-N rectangular window. In the frequency domain, this translates to the convolution of the corresponding spectra, as in Eq. (9.1.8):

$$\boxed{H(\omega) = \int_{-\pi}^{\pi} W(\omega - \omega')e^{-j\omega' M}D(\omega')\,\frac{d\omega'}{2\pi}} \tag{11.1.21}$$

where the $e^{-j\omega' M}$ arises from the delay in $d(n-M)$.

The spectrum $W(\omega)$ of the rectangular window was given in Eq. (9.1.9) (with $L = N$). Thus, the designed filter $H(\omega)$ will be a smeared version of the desired shape $D(\omega)$. In particular, for the ideal lowpass case, because $D(\omega')$ is nonzero and unity only over the subinterval $-\omega_c \le \omega' \le \omega_c$, the frequency convolution integral becomes:

$$H(\omega) = \int_{-\omega_c}^{\omega_c} W(\omega - \omega')e^{-j\omega' M}\,\frac{d\omega'}{2\pi} \tag{11.1.22}$$

The ripples in the frequency response $H(\omega)$, observed in Fig. 11.1.5, arise from the (integrated) ripples of the rectangular window spectrum $W(\omega)$. As N increases, we observe three effects in Fig. 11.1.5:

1. For ω's that lie well within the passband or stopband (i.e., points of continuity), the ripple size decreases as N increases, resulting in flatter passband and stopband. For such ω, we have $\hat{D}_M(\omega)\to D(\omega)$ as $N\to\infty$.

2. The transition width decreases with increasing N. Note also that for any N, the windowed response $H(\omega)$ is always equal to 0.5 at the cutoff frequency $\omega=\omega_c$. (This is a standard property of Fourier series.)

3. The largest ripples tend to cluster near the passband-to-stopband discontinuity (from both sides) and do not get smaller with N. Instead, their size remains approximately *constant*, about 8.9 percent, independent of N. Eventually, as $N\to\infty$, these ripples get squeezed onto the discontinuity at $\omega=\omega_c$, occupying a set of measure zero. This behavior is the Gibbs phenomenon.

11.1.3 Hamming Window

To eliminate the 8.9% passband and stopband ripples, we may replace the rectangular window $w(n)$ in Eq. (11.1.20) by a non-rectangular one, which tapers off gradually at its endpoints, thus reducing the ripple effect. There exist dozens of windows [222–225] and among these the Hamming window is a popular choice; it is defined by:

$$\boxed{w(n)=0.54-0.46\cos\left(\frac{2\pi n}{N-1}\right),\qquad n=0,1,\dots,N-1} \tag{11.1.23}$$

In particular, the Hamming windowed impulse response for a length-N lowpass filter will be, where $N=2M+1$ and $n=0,1,\dots,N-1$:

$$h(n)=w(n)d(n-M)=\left[0.54-0.46\cos\left(\frac{2\pi n}{N-1}\right)\right]\cdot\frac{\sin\big(\omega_c(n-M)\big)}{\pi(n-M)} \tag{11.1.24}$$

As an example, consider the design of a length $N=81$ lowpass filter with cutoff frequency $\omega_c=0.3\pi$. Fig. 11.1.6 shows the rectangularly and Hamming windowed impulse responses. Note how the Hamming impulse response tapers off to zero more gradually. It was computed by Eq. (11.1.24) with $N=81$ and $M=40$. Fig. 11.1.7 shows the corresponding magnitude responses.

The passband/stopband ripples of the rectangular window design are virtually eliminated from the Hamming window design. Actually, there are small ripples with maximum overshoot of about 0.2%, but they are not visible in the scale of Fig. 11.1.7. The price for eliminating the ripples is loss of resolution, which is reflected into a *wider transition width.*

11.2 Gibbs Phenomenon

In this section we give an explanation of the Gibbs phenomenon and provide a justification for the 8.9% overshoot. The Gibbs phenomenenon was described by Gibbs in 1899, but was earlier observed by Wilbraham in 1848. See Refs. [277–282] for a more extensive discussion and reviews.

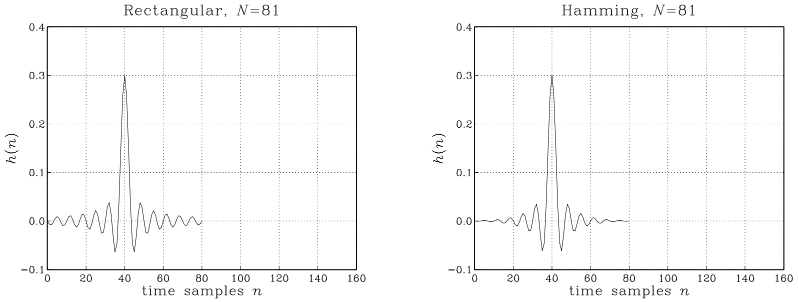


Fig. 11.1.6 Rectangular and Hamming windowed impulse responses for $N=81$.

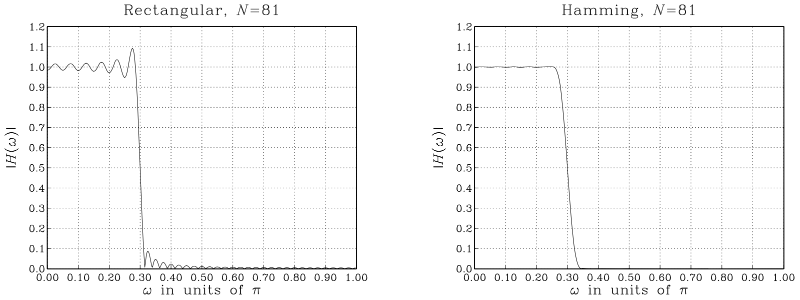


Fig. 11.1.7 Rectangular and Hamming windowed magnitude responses for $N=81$.

The DTFT/IDTFT relationships relating an ideal impulse reponse d_k to the corresponding frequency response $D(\omega)$ represent the Fourier series expansion of the peirodic function $D(\omega)$, with the d_k being the "Fourier series coefficients",

$$D(\omega)=\sum_{k=-\infty}^{\infty}d_k e^{-j\omega k}\quad\Leftrightarrow\quad d_k=\int_{-\pi}^{\pi}D(\omega)e^{j\omega k}\,\frac{d\omega}{2\pi} \tag{11.2.1}$$

Generally, the convergence of the infinite series (11.2.1) can be proved under the so-called *Dirichlet conditions*, which require that over one 2π-period, the periodic function $D(\omega)$ be absolutely integrable, and that it have a finite number of discontinuities, and a finite number of extrema. More precisely, the infinite Fourier series can be thought of as the limit of the following finite sum, as the number of terms tends to infinity, that is,

$$D(\omega)=\lim_{M\to\infty}\hat{D}_M(\omega)\,,\quad\text{where}\quad \hat{D}_M(\omega)=\sum_{k=-M}^{M}d_k e^{-j\omega k} \tag{11.2.2}$$

Under the Dirichlet conditions, it can be proved that $\hat{D}_M(\omega)$ converges to $D(\omega)$ in a *point-wise* sense but only at points of *continuity* of $D(\omega)$, that is, for each value of ω

which is not a point of discontinuity. At a discontinuity point, the finite series converges to the average of the function values from either side of the discontinuity, that is,

$$\lim_{M\to\infty}\hat{D}_M(\omega)=\frac{D(\omega^-)+D(\omega^+)}{2}$$

If the function $D(\omega)$ is *square-integrable* over one period, then it will also be absolutely integrable over one period, and if it meets the other Dirichlet conditions (discontinuities and extrema), the Fourier series representation will still be valid. In this case, one can show the following *Parseval identity* between the average "power" and the Fourier coefficients,

$$\boxed{\frac{1}{2\pi}\int_{-\pi}^{\pi}\big|D(\omega)\big|^2\,d\omega=\sum_{k=-\infty}^{\infty}|d_k|^2}\quad\text{(Parseval)}\tag{11.2.3}$$

Moreover, one can prove that $\hat{D}_M(\omega)$ converges to $D(\omega)$ in the following *mean-square-error* sense,

$$\lim_{M\to\infty}\frac{1}{2\pi}\int_{-\pi}^{\pi}\big|D(\omega)-\hat{D}_M(\omega)\big|^2\,d\omega=0\tag{11.2.4}$$

If $D(\omega)$ has a finite number of discontinuity points, they occupy a set of measure zero and therefore do not contribute to this integral.

Even though, under the Dirichlet conditions, the finite-term approximation $\hat{D}_M(\omega)$ converges to $D(\omega)$ at points of continuity, the function $\hat{D}_M(\omega)$ is not a good approximation to $D(\omega)$ near discontinuity points where it exhibits rapidly oscillating Gibbs ripples which do not diminish in size as M gets larger, but rather they tend to cluster more and more closely near the discontinuity with their overshoot remaining constant approximately equal to 8.95%.

To understand this effect, let us start with a symmetric rectangular window, w_k, $-M\le k\le M$. The windowed coefficients are,

$$\hat{d}_k=w_k d_k=\begin{cases}d_k\,, & -M\le k\le M\\ 0\,, & \text{otherwise}\end{cases}$$

with corresponding approximate and exact frequency responses,

$$\hat{D}_M(\omega)=\sum_{k=-M}^{M}d_k e^{-j\omega k}=\sum_{k=-\infty}^{\infty}w_k d_k e^{-j\omega k}$$

$$D(\omega)=\sum_{k=-\infty}^{\infty}d_k e^{-j\omega k}$$

which will be related by the convolution property,

$$\hat{D}_M(\omega)=\int_{-\pi}^{\pi}W(\omega-\omega')\,D(\omega')\,\frac{d\omega'}{2\pi}$$

where the DTFT $W(\omega)$ of the symmetric window is, with, $2M+1$,

$$W(\omega)=\frac{\sin(N\omega/2)}{\sin(\omega/2)}$$

For even small values of M or $N=2M+1$, such as $N=7$, we have the approximation,

$$W(\omega)=\frac{\sin(N\omega/2)}{\sin(\omega/2)}\approx\frac{\sin(N\omega/2)}{\omega/2}$$

so that the $\hat{D}_M(\omega)$ can be expressed as,

$$\hat{D}_M(\omega)=\int_{-\pi}^{\pi}\frac{\sin\big(N(\omega-\omega')/2\big)}{\pi(\omega-\omega')}\,D(\omega')\,d\omega'\tag{11.2.5}$$

Next, consider an ideal lowpass filter with cutoff frequency ω_c,

$$D(\omega)=\begin{cases}1\,, & |\omega|<\omega_c\\ 0.5\,, & \omega=\pm\omega_c\\ 0\,, & |\omega|>\omega_c\end{cases}$$

It can be expressed compactly either in terms of the Heaviside unit-step function, or in terms of the signum function, as follows,

$$\begin{aligned}D(\omega)&=u(\omega+\omega_c)-u(\omega-\omega_c)\\ D(\omega)&=\frac{1}{2}\,\text{sign}\,(\omega+\omega_c)-\frac{1}{2}\,\text{sign}\,(\omega-\omega_c)\end{aligned}$$

which is a consequence of the relationship,

$$u(x)=\frac{1}{2}+\frac{1}{2}\,\text{sign}(x)$$

For such ideal lowpass filter, the approximation of Eq. (11.2.5) becomes,

$$\hat{D}_M(\omega)=\int_{-\omega_c}^{\omega_c}\frac{\sin\big(N(\omega-\omega')/2\big)}{\pi(\omega-\omega')}\,d\omega'$$

Introducing the *sine-integral*, $\text{Si}(x)$, function,[†]

$$\boxed{\text{Si}(x)=\int_0^x\frac{\sin\nu}{\nu}\,d\nu}$$

we may express $\hat{D}_M(\omega)$ as the difference,

$$\hat{D}_M(\omega)=\frac{1}{\pi}\,\text{Si}\big(N(\omega+\omega_c)/2\big)-\frac{1}{\pi}\,\text{Si}\big(N(\omega-\omega_c)/2\big)$$

In fact, the two Si terms match the two sign terms of

$$D(\omega)=\frac{1}{2}\,\text{sign}\,(\omega+\omega_c)-\frac{1}{2}\,\text{sign}\,(\omega-\omega_c)$$

with the Gibbs overshoot arising from the properties of the Si function.

Fig. 11.2.1 below demonstrates how the function, $\text{Si}(x)/\pi$ approximates the function, $\text{sign}(x)/2$, and how, $\text{Si}(Nx/2)/\pi$, approximates, $\text{sign}(x)/2$, even better, being a compressed version of $\text{Si}(x)/\pi$.

[†] it can be evaluated by MATLAB's built-in function, **sinint**.

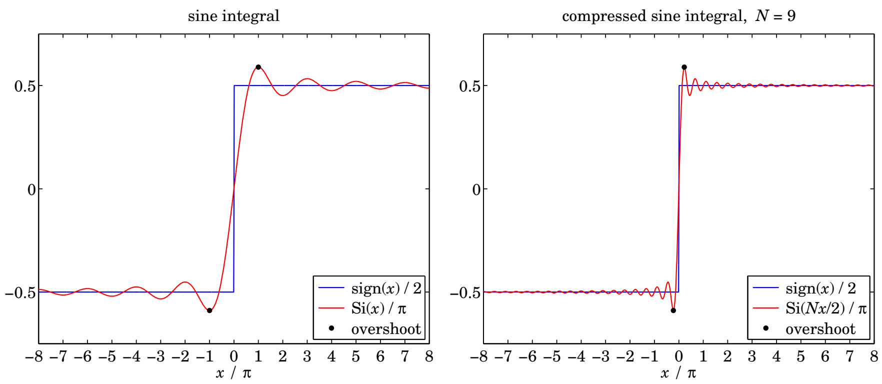


Fig. 11.2.1 Si and signum functions, and Gibbs overshoot.

The maximum of the Si function occurs at,

$$\frac{d}{dx}\,\mathrm{Si}(x) = \frac{\sin x}{x} = 0 \quad\Rightarrow\quad x = \pi$$

resulting in the maximum value, $\mathrm{Si}(\pi)/\pi = 0.58949$, shown on the left graph of the figure, whose deviation from the maximum value of the function $\mathrm{sign}(x)/2$, that is, from $1/2$, is the Gibbs overshoot,

$$\boxed{\text{Gibbs overshoot} = \frac{1}{\pi}\,\mathrm{Si}(\pi) - \frac{1}{2} = 0.58949 - 0.5 = 0.08949} \tag{11.2.6}$$

The maximum shown on right graph occurs at $Nx/2 = \pi$, or, $x = 2\pi/N$, and the maximum value is still the same, $\mathrm{Si}(\pi)/\pi = 0.58949$.

The Gibbs phenomenon is a peculiar property of Fourier series and it always appears in periodic waveforms that have discontinuities. Replacing the rectangular window with a tapered non-rectangular one, such as Hamming, tends to diminish the overshoot.

11.3 Kaiser Window

11.3.1 Kaiser Window for Filter Design

The rectangular and Hamming window designs are very simple, but do not provide good control over the filter design specifications. With these windows, the amount of overshoot is always fixed to 8.9% or 0.2% and cannot be changed to a smaller value if so desired.

A flexible set of specifications is shown in Fig. 11.3.1 in which the designer can arbitrarily specify the amount of passband and stopband overshoot δ_{pass}, δ_{stop}, as well as the transition width Δf.

The passband/stopband frequencies $\{f_{\text{pass}}, f_{\text{stop}}\}$ are related to the ideal cutoff frequency f_c and transition width Δf by

$$\boxed{f_c = \frac{1}{2}(f_{\text{pass}} + f_{\text{stop}})\,,\qquad \Delta f = f_{\text{stop}} - f_{\text{pass}}} \tag{11.3.1}$$

Thus, f_c is chosen to lie exactly in the middle between f_{pass} and f_{stop}. Eqs. (11.3.1) can be inverted to give:

$$\boxed{f_{\text{pass}} = f_c - \frac{1}{2}\Delta f\,,\qquad f_{\text{stop}} = f_c + \frac{1}{2}\Delta f} \tag{11.3.2}$$

The normalized versions of the frequencies are the digital frequencies:

$$\omega_{\text{pass}} = \frac{2\pi f_{\text{pass}}}{f_s}\,,\quad \omega_{\text{stop}} = \frac{2\pi f_{\text{stop}}}{f_s}\,,\quad \omega_c = \frac{2\pi f_c}{f_s}\,,\quad \Delta\omega = \frac{2\pi\Delta f}{f_s}$$

In practice, the passband and stopband overshoots are usually expressed in dB:

$$\boxed{A_{\text{pass}} = 20\log_{10}\left(\frac{1+\delta_{\text{pass}}}{1-\delta_{\text{pass}}}\right)\,,\qquad A_{\text{stop}} = -20\log_{10}\delta_{\text{stop}}} \tag{11.3.3}$$

A simplified version of the passband equation can be obtained by expanding it to first order in δ_{pass}, giving:

$$A_{\text{pass}} = 17.372\delta_{\text{pass}} \tag{11.3.4}$$

which is valid for small values of δ_{pass}. Eqs. (11.3.3) can be inverted to give:

$$\boxed{\delta_{\text{pass}} = \frac{10^{A_{\text{pass}}/20} - 1}{10^{A_{\text{pass}}/20} + 1}\,,\qquad \delta_{\text{stop}} = 10^{-A_{\text{stop}}/20}} \tag{11.3.5}$$

Thus, one can pass back and forth between the specification sets:

$$\{f_{\text{pass}}, f_{\text{stop}}, A_{\text{pass}}, A_{\text{stop}}\} \quad\Leftrightarrow\quad \{f_c, \Delta f, \delta_{\text{pass}}, \delta_{\text{stop}}\}$$

Although δ_{pass} and δ_{stop} can be specified independently of each other, it is a property of *all* window designs that the final designed filter will have *equal* passband and stopband ripples. Therefore, we must design the filter on the basis of the *smaller* of the two ripples, that is,

$$\boxed{\delta = \min(\delta_{\text{pass}}, \delta_{\text{stop}})} \tag{11.3.6}$$

The designed filter will have passband and stopband ripple equal to δ. The value of δ can also be expressed in dB:

$$\boxed{A = -20\log_{10}\delta\,,\qquad \delta = 10^{-A/20}} \tag{11.3.7}$$

In practice, the design is usually based on the stopband ripple δ_{stop}. This is so because any reasonably good choices for the passband and stopband attenuations (e.g., $A_{\text{pass}} = 0.1$ dB and $A_{\text{stop}} = 60$ dB) will almost always result into $\delta_{\text{stop}} < \delta_{\text{pass}}$, and

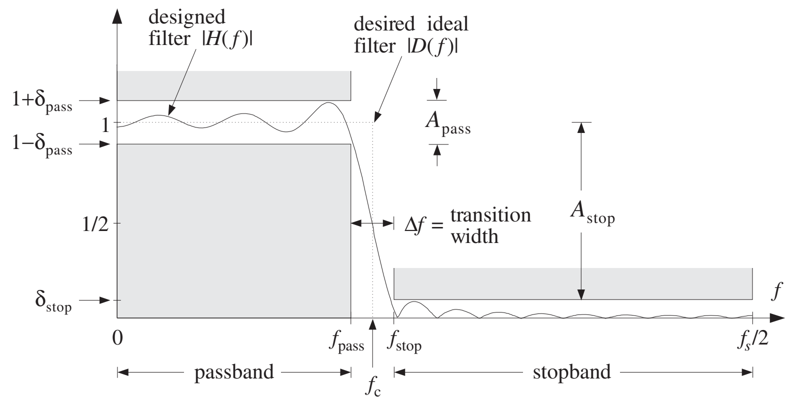


Fig. 11.3.1 Magnitude response specifications for a lowpass filter.

therefore, $\delta = \delta_{\text{stop}}$, and in dB, $A = A_{\text{stop}}$. Thus, it is useful to think of A as the stopband attenuation.

The main limitation of most windows is that they have a *fixed* value of δ, which depends on the particular window shape. Such windows limit the achievable passband and stopband attenuations $\{A_{\text{pass}}, A_{\text{stop}}\}$ to only certain specific values.

For example, Table 11.3.1 shows the attenuations achievable by the rectangular and Hamming windows, calculated from Eq. (11.3.3) with the values $\delta = \delta_{\text{pass}} = \delta_{\text{stop}} = 0.089$ and $\delta = \delta_{\text{pass}} = \delta_{\text{stop}} = 0.002$, respectively. The table also shows the corresponding value of the transition width parameter D of Eq. (11.3.11).

The only windows that do not suffer from the above limitation are the Kaiser window [265–267], the Dolph-Chebyshev window [268,269,228–230,270], and the Saramäki windows [271]. These windows have an *adjustable shape parameter* that allows the window to achieve any desired value of ripple δ or attenuation A.

Window	δ	A_{stop}	A_{pass}	D
Rectangular	8.9%	−21 dB	1.55 dB	0.92
Hamming	0.2%	−54 dB	0.03 dB	3.21
Kaiser	variable δ	$-20\log_{10}\delta$	17.372δ	$(A-7.95)/14.36$

Table 11.3.1 Specifications for rectangular, Hamming, and Kaiser windows.

The Kaiser window is unique in the above class in that it has near-optimum performance (in the sense of minimizing the sidelobe energy of the window), as well as having the simplest implementation. It depends on two parameters: its length N and the shape parameter α. Assuming odd length $N = 2M+1$, the window is defined, for $n = 0, 1, \dots, N-1$, as follows:

$$\text{(Kaiser window)} \qquad \boxed{w(n) = \frac{I_0\left(\alpha\sqrt{1-(n-M)^2/M^2}\right)}{I_0(\alpha)}} \tag{11.3.8}$$

where $I_0(x)$ is the *modified Bessel function of the first kind and 0th order.* This function and its evaluation by the routine `I0.c` are discussed at the end of this section. The numerator in Eq. (11.3.8) can be rewritten in the following form, which is more convenient for numerical evaluation:

$$w(n) = \frac{I_0\left(\alpha\sqrt{n(2M-n)}/M\right)}{I_0(\alpha)}, \quad n = 0, 1, \dots, N-1 \tag{11.3.9}$$

Like all window functions, the Kaiser window is *symmetric* about its middle, $n = M$, and has the value $w(M) = 1$ there. At the endpoints, $n = 0$ and $n = N-1$, it has the value $1/I_0(\alpha)$ because $I_0(0) = 1$.

Figure 11.3.2 compares a Hamming window of length $N = 51$ to the Kaiser windows of the same length and shape parameters $\alpha = 7$ and $\alpha = 5$. For $\alpha = 5$ the Kaiser and Hamming windows agree closely, except near their endpoints. For $\alpha = 0$ the Kaiser window reduces to the rectangular one.

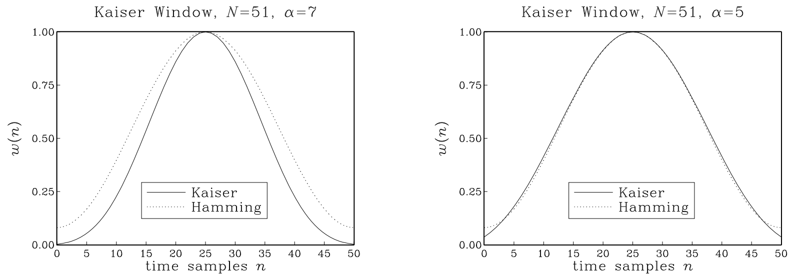


Fig. 11.3.2 Kaiser and Hamming windows for $N = 51$ and $\alpha = 5, 7$.

The window parameters $\{N, \alpha\}$ are computable in terms of the filter specifications, namely, the ripple δ and transition width Δf. The design equations developed by Kaiser [265–267] are as follows. The shape parameter α is calculated from:

$$\boxed{\alpha = \begin{cases} 0.1102(A-8.7), & \text{if } A \ge 50 \\ 0.5842(A-21)^{0.4} + 0.07886(A-21), & \text{if } 21 < A < 50 \\ 0, & \text{if } A \le 21 \end{cases}} \tag{11.3.10}$$

where A is the ripple in dB, given by Eq. (11.3.7). The filter length N is inversely related to the transition width:

$$\boxed{\Delta f = \frac{Df_s}{N-1} \quad \Leftrightarrow \quad N-1 = \frac{Df_s}{\Delta f}} \tag{11.3.11}$$

where the factor D is computed also in terms of A by

$$\boxed{D = \begin{cases} \dfrac{A - 7.95}{14.36}, & \text{if } A > 21 \\ 0.922, & \text{if } A \le 21 \end{cases}} \tag{11.3.12}$$

The most practical range of these formulas is for $A \ge 50$ dB, for which they simplify to:

$$\boxed{\alpha = 0.1102(A - 8.7)\,, \quad D = \frac{A - 7.95}{14.36}} \qquad (\text{for } A \ge 50 \text{ dB}) \tag{11.3.13}$$

To summarize, the steps for designing a lowpass filter are as follows. Given the specifications $\{f_{\text{pass}}, f_{\text{stop}}, A_{\text{pass}}, A_{\text{stop}}\}$:

1. Calculate f_c and Δf from Eq. (11.3.1). Then, calculate $\omega_c = 2\pi f_c/f_s$.
2. Calculate δ_{pass} and δ_{stop} from Eq. (11.3.5).
3. Calculate $\delta = \min(\delta_{\text{pass}}, \delta_{\text{stop}})$ and $A = -20\log_{10}\delta$ in dB.
4. Calculate α and D from Eqs. (11.3.10) and (11.3.12).
5. Calculate the filter length N from Eq. (11.3.11) and round it up to the next *odd* integer, $N = 2M + 1$, and set $M = (N - 1)/2$.
6. Calculate the window function $w(n)$, $n = 0, 1, \dots, N - 1$ from Eq. (11.3.8).
7. Calculate the windowed impulse response, for $n = 0, 1, \dots, N - 1$:

$$h(n) = w(n)d(n - M) = w(n)\cdot\frac{\sin\big(\omega_c(n - M)\big)}{\pi(n - M)} \tag{11.3.14}$$

In particular, we have $h(M) = w(M)\omega_c/\pi = \omega_c/\pi$, because $w(M) = 1$.

Note that the window parameters $\{N, \alpha\}$ depend only on the specifications $\{A, \Delta f\}$ and not on f_c. However, $h(n)$ does depend on f_c.

The design steps can be modified easily to design highpass and bandpass filters. For *highpass* filters, the role of f_{pass} and f_{stop} are interchanged; therefore, the only change in the steps is to define $\Delta f = f_{\text{pass}} - f_{\text{stop}}$ and to use the highpass response from Eq. (11.1.4). The highpass impulse response will be:

$$h(n) = w(n)d(n - M) = w(n)\cdot\left[\delta(n - M) - \frac{\sin\big(\omega_c(n - M)\big)}{\pi(n - M)}\right]$$

The first term can be simplified to $w(n)\delta(n - M) = w(M)\delta(n - M) = \delta(n - M)$ because $w(M) = 1$. Therefore, the designed filter will be:

$$h(n) = \delta(n - M) - w(n)\cdot\frac{\sin\big(\omega_c(n - M)\big)}{\pi(n - M)} \tag{11.3.15}$$

For the same value of ω_c, the lowpass and highpass filters are complementary. The sum of Eqs. (11.3.14) and (11.3.15) gives:

$$h_{LP}(n) + h_{HP}(n) = \delta(n - M)\,, \quad n = 0, 1, \dots, N - 1 \tag{11.3.16}$$

which becomes in the z-domain:

$$H_{LP}(z) + H_{HP}(z) = z^{-M} \tag{11.3.17}$$

For *bandpass* filters, the desired specifications may be given as in Fig. 11.3.3. There are now two stopbands and two transition widths. The final design will have equal transition widths, given by Eq. (11.3.11). Therefore, we must design the filter based on the smaller of the two widths, that is,

$$\Delta f = \min(\Delta f_a, \Delta f_b) \tag{11.3.18}$$

where the left and right transition widths are:

$$\Delta f_a = f_{pa} - f_{sa}\,, \qquad \Delta f_b = f_{sb} - f_{pb} \tag{11.3.19}$$

Figure 11.3.3 shows the case where the left transition width is the smaller one and, thus, defines Δf. The ideal cutoff frequencies f_a and f_b can be calculated by taking them to be $\Delta f/2$ away from the passband or from the stopbands. The *standard definition* is with respect to the passband:

$$f_a = f_{pa} - \frac{1}{2}\Delta f\,, \qquad f_b = f_{pb} + \frac{1}{2}\Delta f \tag{11.3.20}$$

This choice makes the passband just right and the stopband somewhat wider than required. The *alternative definition* makes the stopbands right and the passband wider:

$$f_a = f_{sa} + \frac{1}{2}\Delta f\,, \qquad f_b = f_{sb} - \frac{1}{2}\Delta f \tag{11.3.21}$$

Once the cutoff frequencies $\{f_a, f_b\}$ and the window parameters $\{N, \alpha\}$ are calculated, the bandpass impulse response may be defined, for $n = 0, 1, \dots, N - 1$:

$$h(n) = w(n)d(n - M) = w(n)\cdot\frac{\sin\big(\omega_b(n - M)\big) - \sin\big(\omega_a(n - M)\big)}{\pi(n - M)} \tag{11.3.22}$$

where $h(M) = (\omega_b - \omega_a)/\pi$, and $\omega_a = 2\pi f_a/f_s$, $\omega_b = 2\pi f_b/f_s$. Next, we present a lowpass and a bandpass design example.

Example 11.3.1: *Lowpass Design.* Using the Kaiser window, design a lowpass digital filter with the following specifications:

$$\begin{aligned} &f_s = 20 \text{ kHz} \\ &f_{\text{pass}} = 4 \text{ kHz}, \quad f_{\text{stop}} = 5 \text{ kHz} \\ &A_{\text{pass}} = 0.1 \text{ dB}, \quad A_{\text{stop}} = 80 \text{ dB} \end{aligned}$$

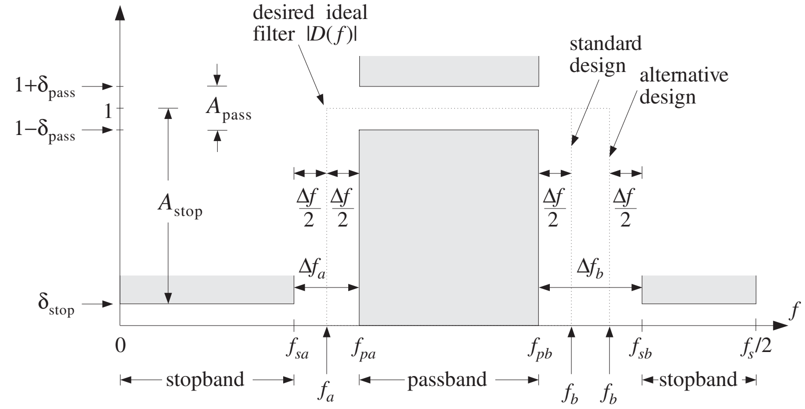


Fig. 11.3.3 Bandpass filter specifications.

Solution: First, we calculate δ_{pass} and δ_{stop} from Eq. (11.3.5):

$$\delta_{\text{pass}} = \frac{10^{0.1/20}-1}{10^{0.1/20}+1} = 0.0058, \qquad \delta_{\text{stop}} = 10^{-80/20} = 0.0001$$

Therefore, $\delta = \min(\delta_{\text{pass}}, \delta_{\text{stop}}) = \delta_{\text{stop}} = 0.0001$, which in dB is $A = -20\log_{10}\delta = A_{\text{stop}} = 80$. The D and α parameters are computed by:

$$\alpha = 0.1102(A-8.7) = 0.1102(80-8.7) = 7.857, \quad D = \frac{A-7.95}{14.36} = 5.017$$

The filter width and ideal cutoff frequency are:

$$\Delta f = f_{\text{stop}} - f_{\text{pass}} = 1 \text{ kHz}, \quad f_c = \frac{1}{2}(f_{\text{pass}} + f_{\text{stop}}) = 4.5 \text{ kHz}, \quad \omega_c = \frac{2\pi f_c}{f_s} = 0.45\pi$$

Eq. (11.3.11) gives for the filter length (rounded up to the nearest odd integer):

$$N = 1 + \frac{Df_s}{\Delta f} = 101.35 \quad \Rightarrow \quad N = 103, \quad M = \frac{1}{2}(N-1) = 51$$

The windowed impulse response will be, for $n = 0, 1, \dots, 102$:

$$h(n) = w(n)d(n-M) = \frac{I_0\big(7.857\sqrt{n(102-n)}/51\big)}{I_0(7.857)} \cdot \frac{\sin\big(0.45\pi(n-51)\big)}{\pi(n-51)}$$

with $h(51) = \omega_c/\pi = 0.45$. Figure 11.3.4 shows the magnitude response in dB of $h(n)$, that is, $20\log_{10}|H(\omega)|$, where $H(\omega)$ was evaluated by Eq. (11.1.19). Note the transition width extending from 4 to 5 kHz and the stopband specification defined by the horizontal grid line at −80 dB. The passband specification is more than satisfied. It is $A_{\text{pass}} \simeq 17.372\delta = 0.0017$ dB.

The figure also shows the corresponding Hamming and rectangularly windowed designs for the same length of $N = 103$. They both have a smaller transition width—the rectangular one even more so, but their stopband attenuations are limited to the standard values of 54 dB and 21 dB, respectively. □

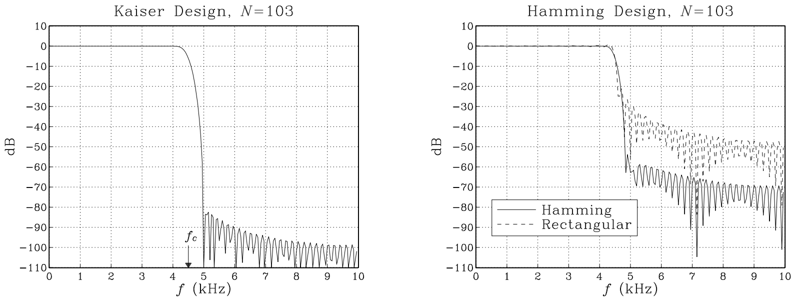


Fig. 11.3.4 Kaiser, Hamming, and rectangular window designs, $N = 103$.

Example 11.3.2: *Bandpass Design.* Using the Kaiser window, design a bandpass digital filter with the following specifications:

$$\begin{aligned} &f_s = 20 \text{ kHz} \\ &f_{sa} = 3 \text{ kHz}, \quad f_{pa} = 4 \text{ kHz}, \quad f_{pb} = 6 \text{ kHz}, \quad f_{sb} = 8 \text{ kHz} \\ &A_{\text{pass}} = 0.1 \text{ dB}, \quad A_{\text{stop}} = 80 \text{ dB} \end{aligned}$$

Solution: The parameters $\{\delta_{\text{pass}}, \delta_{\text{stop}}, \delta, A, \alpha, D\}$ are the same as in the previous example. The two transition widths are:

$$\Delta f_a = f_{pa} - f_{sa} = 4 - 3 = 1 \text{ kHz}, \quad \Delta f_b = f_{sb} - f_{pb} = 8 - 6 = 2 \text{ kHz}$$

Therefore, the minimum width is $\Delta f = \min(\Delta f_a, \Delta f_b) = 1$ kHz, and the filter length:

$$N = 1 + \frac{Df_s}{\Delta f} = 101.35 \quad \Rightarrow \quad N = 103, \quad M = \frac{1}{2}(N-1) = 51$$

Using the *standard definition* of Eq. (11.3.20), we find for the left and right ideal cutoff frequencies:

$$f_a = f_{pa} - \frac{1}{2}\Delta f = 4 - 0.5 = 3.5 \text{ kHz}, \quad f_b = f_{pb} + \frac{1}{2}\Delta f = 6 + 0.5 = 6.5 \text{ kHz}$$

with the normalized values $\omega_a = 2\pi f_a/f_s = 0.35\pi$, $\omega_b = 2\pi f_b/f_s = 0.65\pi$.

For the *alternative definition* of Eq. (11.3.21), we have $f_a = 3 + 0.5 = 3.5$ and $f_b = 8 - 0.5 = 7.5$ kHz, resulting in $\omega_a = 0.35\pi$ and $\omega_b = 0.75\pi$. Figure 11.3.5 shows the magnitude response of the designed filter in dB, both for the standard and the alternative definitions. The standard design has just the right passband extending over $[4, 6]$ kHz and a wider stopband that starts at 7 kHz. The alternative design has a wider passband extending over $[4, 7]$ kHz. □

Next, we discuss three more Kaiser design examples for *digital audio* applications, namely, two-way and three-way loudspeaker crossover filters and a five-band graphic equalizer.

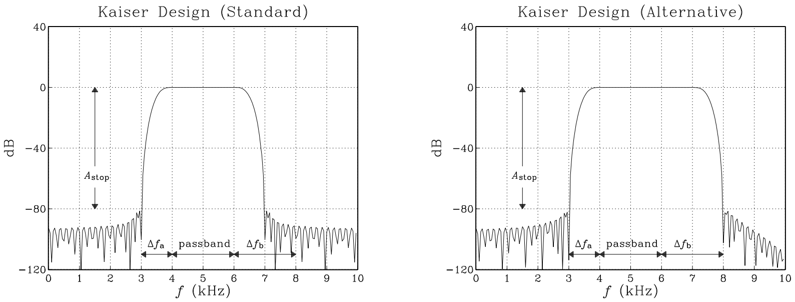


Fig. 11.3.5 Kaiser window design of a bandpass filter.

In all three cases, the sampling rate is $f_s = 44.1$ kHz, the stopband attenuation is $A_{\rm stop} = 65$ dB, and all the transition widths are taken to be equal to $\Delta f = 2$ kHz. This implies that all the filters will have the same length N and Kaiser parameters D and α. With $A = A_{\rm stop} = 65$ dB, we find

$$D = \frac{A - 7.95}{14.36} = 3.973, \qquad \alpha = 0.1102(A - 8.7) = 6.204,$$

$$N - 1 = \frac{Df_s}{\Delta f} = \frac{3.973 \times 44.1}{2} = 87.6 \quad \Rightarrow \quad N = 89, \quad M = \frac{1}{2}(N-1) = 44$$

Note that the given value of $A = 65$ dB corresponds to $\delta_{\rm pass} = \delta_{\rm stop} = 10^{-65/20} = 0.00056$, which results in the small passband attenuation $A_{\rm pass} = 0.0097$ dB.

Such filters are to within the capabilities of modern DSP chips. Assuming a typical instruction rate of 20 MIPS, which is 20 instructions per μsec, or, $T_{\rm instr} = 50$ nsec per instruction, and assuming the sample processing implementation requires N MACs per output sample computed, the total computational time for processing each input sample will be $NT_{\rm instr} = 89 \times 50 = 4.45\ \mu$sec, which fits well within the time separating each input sample, $T = 1/f_s = 1/44.1 = 22.68\ \mu$sec.

Several such filters can even be implemented simultaneously on the same DSP chip, namely, $22.68/4.45 = 5.1$, or, about five length-89 filters. Conversely, the longest single filter that can be implemented will have length such that $NT_{\rm instr} = T$, or, $N = T/T_{\rm instr} = f_{\rm instr}/f_s = 20000/44.1 \simeq 453$, resulting in the smallest implementable transition width of

$$\Delta f_{\rm min} \simeq \frac{Df_s}{N} = \frac{D{f_s}^2}{f_{\rm instr}} = 0.386 \text{ kHz}$$

Example 11.3.3: *Two-Way Crossover Filters.* All conventional loudspeakers contain an analog *crossover* network that splits the incoming analog audio signal into its low- and high-frequency components that drive the *woofer* and *tweeter* parts of the loudspeaker. More advanced loudspeakers may contain even a third component for the mid-frequency part of the input [272].

Digital loudspeaker systems operate on the digitized audio input and use (FIR or IIR) digital filters to split it into the appropriate frequency bands, which are then converted to analog format, amplified, and drive the corresponding parts of the loudspeaker [273,274]. Such "digital" loudspeakers have been available for a while in professional digital studios and are becoming commercially available for home use (where typically the digital output of a CD player is connected to the digital input of the loudspeaker).

In this example, we take the cutoff frequency of the lowpass and highpass filters, known as the *crossover frequency*, to be $f_c = 3$ kHz, which leads to the normalized frequency $\omega_c = 2\pi f_c/f_s = 0.136\pi$. (This numerical value of f_c is chosen only for plotting convenience—a more realistic value would be 1 kHz.) The designed low- and high-frequency driver filters are then: For $n = 0, 1, \cdots, N-1$

$$\begin{aligned} h_{\rm LP}(n) &= w(n)d_{\rm LP}(n-M) = w(n)\left[\frac{\sin\big(\omega_c(n-M)\big)}{\pi(n-M)}\right] \\ h_{\rm HP}(n) &= w(n)d_{\rm HP}(n-M) = w(n)\left[\delta(n-M) - \frac{\sin\big(\omega_c(n-M)\big)}{\pi(n-M)}\right] = \\ &= \delta(n-M) - h_{\rm LP}(n) \end{aligned}$$

where $w(n)$ is the Kaiser window given by Eq. (11.3.8). The magnitude responses of the designed filters are shown in Fig. 11.3.6, plotted both in absolute scales, $|H(\omega)|$, and in decibels, $20\log_{10}|H(\omega)|$.

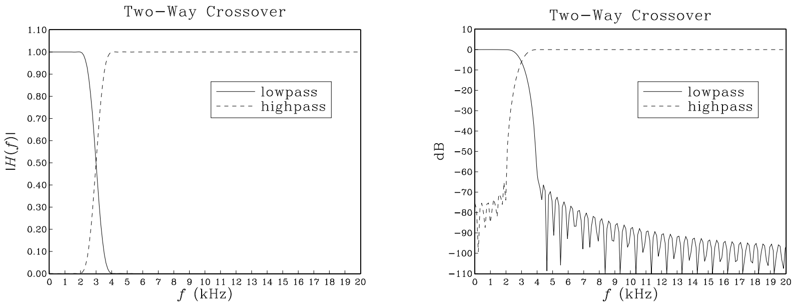


Fig. 11.3.6 Low- and high-frequency magnitude responses.

The complementarity relationship between the impulse responses implies in the z-domain:

$$H_{\rm HP}(z) = z^{-M} - H_{\rm LP}(z)$$

It leads to the realization of Fig. 11.3.7. Instead of realizing the lowpass and highpass filters separately, it requires only the lowpass filter and one multiple delay. □

Example 11.3.4: *Three-Way Crossover Filters.* In this example, the audio input must be split into its low-, mid-, and high-frequency components. The crossover frequencies are chosen to be $f_a = 3$ kHz and $f_b = 7$ kHz. The midpass filter will be a bandpass filter with these cutoff frequencies. The designed impulse responses will be:

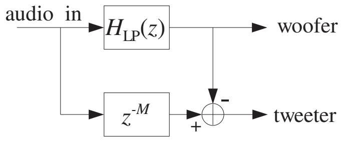


Fig. 11.3.7 Complementary implementation of two-way crossover filters.

$$
\begin{aligned}
h_{LP}(n) &= w(n)d_{LP}(n-M) = w(n)\left[\frac{\sin(\omega_a(n-M))}{\pi(n-M)}\right] \\
h_{MP}(n) &= w(n)d_{MP}(n-M) = w(n)\left[\frac{\sin(\omega_b(n-M)) - \sin(\omega_a(n-M))}{\pi(n-M)}\right] \\
h_{HP}(n) &= w(n)d_{HP}(n-M) = w(n)\left[\delta(n-M) - \frac{\sin(\omega_b(n-M))}{\pi(n-M)}\right]
\end{aligned}
$$

where, $\omega_a = 2\pi f_a/f_s = 0.136\pi$ and $\omega_b = 2\pi f_b/f_s = 0.317\pi$. Adding the three impulse responses, we find

$$h_{LP}(n) + h_{MP}(n) + h_{HP}(n) = \delta(n-M)$$

and, in the z-domain

$$H_{LP}(z) + H_{MP}(z) + H_{HP}(z) = z^{-M}$$

which allows us to express one of them in terms of the other two, for example

$$H_{HP}(z) = z^{-M} - H_{LP}(z) - H_{MP}(z)$$

The magnitude responses of the designed filters are shown in Fig. 11.3.8. A realization that uses the above complementarity property and requires only two filtering operations instead of three is shown in Fig. 11.3.9. □

Example 11.3.5: *Five-Band Graphic Equalizer.* Present-day graphic equalizers typically employ second-order IIR filters. However, there is no reason not to use FIR filters, if the computational cost is manageable. In this example, we choose the crossover frequencies of the five bands to be $f_a = 3$ kHz, $f_b = 7$ kHz, $f_c = 11$ kHz, $f_d = 15$ kHz, defining the five frequency bands:

$[0, f_a]$	band 1
$[f_a, f_b]$	band 2
$[f_b, f_c]$	band 3
$[f_c, f_d]$	band 4
$[f_d, f_s/2]$	band 5

The designed filter impulse responses will be:

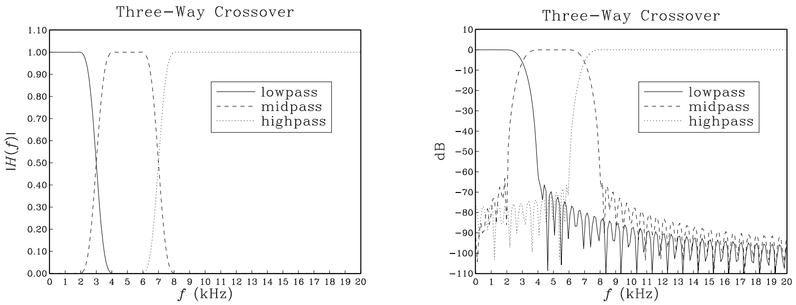


Fig. 11.3.8 Lowpass, midpass, and highpass magnitude responses.

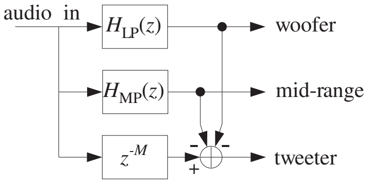


Fig. 11.3.9 Complementary implementation of three-way crossover filters.

$$
\begin{aligned}
h_1(n) &= w(n)\left[\frac{\sin(\omega_a(n-M))}{\pi(n-M)}\right] \\
h_2(n) &= w(n)\left[\frac{\sin(\omega_b(n-M)) - \sin(\omega_a(n-M))}{\pi(n-M)}\right] \\
h_3(n) &= w(n)\left[\frac{\sin(\omega_c(n-M)) - \sin(\omega_b(n-M))}{\pi(n-M)}\right] \\
h_4(n) &= w(n)\left[\frac{\sin(\omega_d(n-M)) - \sin(\omega_c(n-M))}{\pi(n-M)}\right] \\
h_5(n) &= w(n)\left[\delta(n-M) - \frac{\sin(\omega_d(n-M))}{\pi(n-M)}\right]
\end{aligned}
$$

where, $\omega_a = 2\pi f_a/f_s = 0.136\pi$, $\omega_b = 2\pi f_b/f_s = 0.317\pi$, $\omega_c = 2\pi f_c/f_s = 0.499\pi$, $\omega_d = 2\pi f_d/f_s = 0.680\pi$. Adding the five filters we find the relationship:

$$h_1(n) + h_2(n) + h_3(n) + h_4(n) + h_5(n) = \delta(n-M)$$

and, in the z-domain

$$H_1(z) + H_2(z) + H_3(z) + H_4(z) + H_5(z) = z^{-M}$$

It can be solved for one of the transfer functions in terms of the other ones:

$$H_5(z)=z^{-M}-H_1(z)-H_2(z)-H_3(z)-H_4(z)$$

The magnitude responses are shown in Fig. 11.3.10. A realization that uses the above complementarity property and requires only four filtering operations instead of five is shown in Fig. 11.3.11.

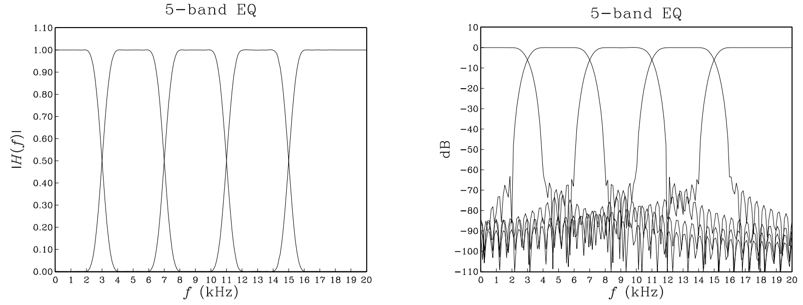


Fig. 11.3.10 Graphic equalizer magnitude responses.

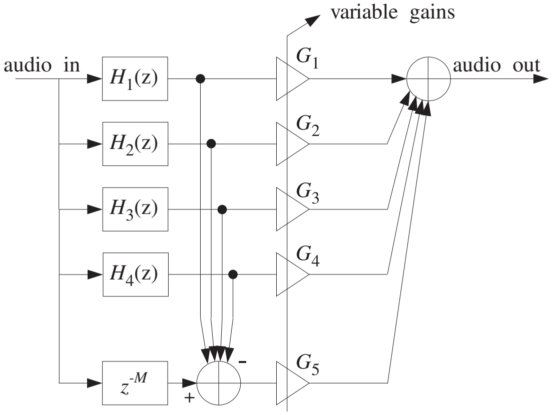


Fig. 11.3.11 Complementary implementation of graphic equalizer.

The outputs of the five filters are weighted by the user-selectable gains G_i and then summed up to form the "equalized" audio signal. The overall transfer function from the input to the overall output is:

$$H(z)=G_1H_1(z)+G_2H_2(z)+G_3H_3(z)+G_4H_4(z)+G_5H_5(z)$$

In practice, the crossover frequencies are chosen to follow standard ISO (International Standards Organization) frequencies, dividing the 20 kHz audio range into octaves or fractions of octaves. □

The above three examples are special cases of *parallel filter banks* in which the input is split into several non-overlapping frequency bands covering the Nyquist interval. Applications include multirate filter banks and subband coding of speech, audio, and picture signals in which the outputs of the bank filters are quantized with fewer number of bits and at the same time their sampling rates are dropped, such that the overall bit rate required for the digital transmission or storage of the signal is substantially reduced [349]. For example, in the recent DCC audio cassette system, the allocation of bits in each band is governed by *psychoacoustic perceptual criteria* in which fewer bits are assigned to bands that will be less audible [369–374]. Wavelets and the *discrete wavelet transform* are also examples of filter banks [349].

Finally, we consider the definition and computation of the Bessel function $I_0(x)$. It is defined by its Taylor series expansion:

$$I_0(x)=\sum_{k=0}^{\infty}\left[\frac{(x/2)^k}{k!}\right]^2 \tag{11.3.23}$$

The Kaiser window (11.3.8) requires the evaluation of $I_0(x)$ over the range of argument $0\le x\le\alpha$. The following C function `I0.c` evaluates $I_0(x)$ for any x. The routine is essentially a C version of the Fortran routine given by Kaiser in [267].

```
/* I0.c - Modified Bessel Function I0(x)
 *
 * $I_0(x) = \sum_{k=0}^{\infty} [(x/2)^k / k!]^2$
 *
 */

#include <math.h>

#define eps   (1.E-9)                          ε = 10^-9

double I0(x)                                   usage: y = I0(x)
double x;
{
       int n = 1;
       double S = 1, D = 1, T;

       while (D > eps * S) {
              T = x / (2 * n++);
              D *= T * T;
              S += D;
              }

       return S;
}
```

The routine is based on the following recursions, which evaluate the power series (11.3.23) by keeping enough terms in the expansion. We define the partial sum of the series (11.3.23):

$$S_n=\sum_{k=0}^{n}\left[\frac{(x/2)^k}{k!}\right]^2$$

It is initialized to $S_0=1$ and satisfies the recursion, for $n\ge 1$:

$$S_n = S_{n-1} + D_n\,, \qquad \text{where } D_n = \left[\frac{(x/2)^n}{n!}\right]^2$$

In turn, D_n itself satisfies a recursion, for $n \geq 1$:

$$D_n = \left[\frac{x}{2n}\right]^2 D_{n-1} = T_n^2 D_{n-1}\,, \qquad \text{where } T_n = \frac{x}{2n}$$

and it is initialized to $D_0 = 1$. The iteration stops when the successive D_n terms become much smaller than the accumulated terms S_n, that is, when

$$\frac{D_n}{S_n} = \frac{S_n - S_{n-1}}{S_n} < \epsilon$$

where ϵ is a small number, such as, $\epsilon = 10^{-9}$. The Kaiser window itself may be calculated by invoking the routine `I0.c`; for example:

```
I0a = I0(alpha);

for (n=0; n<N; n++)
       w[n]= I0(alpha * sqrt(n*(2*M-n)) / M) / I0a;
```

11.3.2 Kaiser Window for Spectral Analysis

We saw in Section 9.1 that one of the main issues in spectral analysis was the *tradeoff* between frequency resolution and leakage. The more one tries to suppress the sidelobes, the wider the mainlobe of the window becomes, reducing the amount of achievable resolution.

For example, the Hamming window provides about 40 dB sidelobe suppression at the expense of doubling the mainlobe width of the rectangular window. Recall from Eq. (9.1.18) that the mainlobe width Δf_w of a window depends inversely on the data record length L:

$$\boxed{\Delta f_w = \frac{cf_s}{L-1} \quad \Leftrightarrow \quad L - 1 = \frac{cf_s}{\Delta f_w}} \tag{11.3.24}$$

where the factor c depends on the window used. For the Kaiser window, we use the more accurate denominator $L - 1$, instead of L of Eq. (9.1.18); the difference is insignificant in spectral analysis where L is large.

The more the sidelobe suppression, the larger the factor c. Thus, to maintain a certain required value for the resolution width Δf_w, one must increase the data length L commensurately with c.

Most windows have fixed values for the amount of sidelobe suppression and width factor c. Table 11.3.2 shows these values for the rectangular and Hamming windows. Adjustable windows, like the Kaiser window, have a variable sidelobe level R that can be chosen as the application requires.

Kaiser and Schafer [224] have developed simple design equations for the use of the Kaiser window in spectral analysis—we discussed it briefly in Sec. 9.3. Given a desired relative sidelobe level R in dB and a desired amount of resolution Δf_w, the design equations determine the length L and shape parameter α of the window. The length is determined from Eq. (11.3.24), where c is given in terms of R by:

Window	R	c
Rectangular	-13 dB	1
Hamming	-40 dB	2
Kaiser	variable R	$6(R+12)/155$

Table 11.3.2 Relative sidelobe levels.

$$\boxed{c = \frac{6(R+12)}{155}} \tag{11.3.25}$$

Note that our values for c given in Eq. (11.3.25) and Table 11.3.2 are smaller by a factor of 2 than in most texts, because we define Δf_w to be *half the base* of the mainlobe, instead of the base itself. The window shape parameter α can be calculated also in terms of R by:

$$\alpha = \begin{cases} 0, & R < 13.26 \\ 0.76609(R-13.26)^{0.4} + 0.09834(R-13.26)\,, & 13.26 < R < 60 \\ 0.12438(R+6.3)\,, & 60 < R < 120 \end{cases} \tag{11.3.26}$$

Once the window parameters $\{L, \alpha\}$ have been determined, the window may be calculated by:

$$w(n) = \frac{I_0\left(\alpha\sqrt{1-(n-M)^2/M^2}\right)}{I_0(\alpha)}\,, \qquad n = 0, 1, \dots, L-1 \tag{11.3.27}$$

where $M = (L-1)/2$, and then applied to a length-L data record by

$$x_L(n) = w(n)x(n)\,, \qquad n = 0, 1, \dots, L-1 \tag{11.3.28}$$

The sidelobe level R must be distinguished from the attenuation A of the filter design case. There, the attenuation A and ripple δ arise from the integrated window spectrum $W(\omega)$, as in Eq. (11.1.22), whereas R arises from $W(\omega)$ itself.

Because of the adjustable sidelobe level R, the Kaiser window can be used to pull very weak sinusoids out of the DFT spectrum of a windowed signal, whereas another type of window, such as a Hamming window, might fail. The following example illustrates this remark.

Example 11.3.6: The following analog signal consisting of three sinusoids of frequencies $f_1 = 2$ kHz, $f_2 = 2.5$ kHz, and $f_3 = 3$ kHz is sampled at a rate of $f_s = 10$ kHz:

$$x(t) = A_1\cos(2\pi f_1 t) + A_2\cos(2\pi f_2 t) + A_3\cos(2\pi f_3 t)7$$

where t is in msec. The relative amplitudes are

$$A_1 = A_3 = 1, \qquad A_2 = 10^{-50/20} = 0.0032$$

so that the middle sinusoid is 50 dB below the other two. A finite duration portion of length L is measured and the DFT spectrum is computed for the purpose of detecting the presence of the sinusoids by their peaks.

If we use a length-L Hamming window on the time data, the f_2-component will be lost below the 40 dB sidelobes of the window. Thus, we must use a window whose sidelobe level is well below 50 dB. We choose a Kaiser window with $R = 70$ dB. Moreover, to make the peaks clearly visible, we choose the resolution width to be $\Delta f = (f_2 - f_1)/3 = 0.167$ kHz. The Kaiser window parameters are calculated as follows:

$$\alpha = 0.12438(R + 6.3) = 9.490, \quad c = \frac{6(R+12)}{155} = 3.174$$

$$L = 1 + \frac{cf_s}{\Delta f} = 191.45 \quad \Rightarrow \quad L = 193, \quad M = \frac{1}{2}(L-1) = 96$$

The length-L sampled signal is, for $n = 0, 1, \dots, L-1$:

$$x(n) = A_1\cos(2\pi f_1 n/f_s) + A_2\cos(2\pi f_2 n/f_s) + A_3\cos(2\pi f_3 n/f_s)$$

The Kaiser and Hamming windowed signals will be, for $n = 0, 1, \dots, L-1$:

$$x_K(n) = w(n)x(n) = \frac{I_0\big(\alpha\sqrt{n(2M-n)}/M\big)}{I_0(\alpha)} \cdot x(n)$$

$$x_H(n) = w(n)x(n) = \left[0.54 - 0.46\cos\left(\frac{2\pi n}{L-1}\right)\right] \cdot x(n)$$

The corresponding spectra are:

$$X_K(f) = \sum_{n=0}^{L-1} x_K(n)e^{-2\pi jfn/f_s}, \qquad X_H(f) = \sum_{n=0}^{L-1} x_H(n)e^{-2\pi jfn/f_s}$$

Figure 11.3.12 shows these spectra in dB, that is, $20\log_{10}|X_K(f)|$, computed at 256 equally spaced frequencies in the interval $[0, f_s/2]$. (This can be done by the MATLAB function `dtft.m` of Appendix C.)

Both spectra are normalized to 0 dB at their maximum value. The Kaiser spectrum shows three clearly separated peaks, with the middle one being 50 dB below the other two. The sidelobes are suppressed by at least 70 dB and do not swamp the middle peak, as they do in the Hamming spectrum. That spectrum, on the other hand, has narrower peaks because the length L is somewhat larger than required to resolve the given Δf. The width of the Hamming peaks is $\Delta f = cf_s/(L-1)$ with $c = 2$, or, $\Delta f = 0.104$ kHz. □

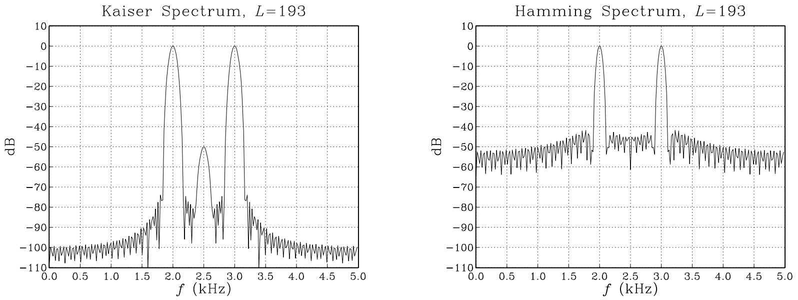


Fig. 11.3.12 Kaiser and Hamming spectra.

11.4 Frequency Sampling Method

The window method is very convenient for designing ideally shaped filters, primarily because the frequency integral in Eq. (11.1.7) can be carried out in closed form.

For arbitrary frequency responses $D(\omega)$, we may use the *frequency sampling* method, in which the integral (11.1.7) is replaced by the approximate sum:

$$\boxed{\tilde{d}(k) = \frac{1}{N}\sum_{i=-M}^{M} D(\omega_i)e^{j\omega_i k}, \qquad -M \le k \le M} \tag{11.4.1}$$

where $N = 2M + 1$. The approximation is essentially an inverse N-point DFT, with the DFT frequencies ω_i spanning equally the interval $[-\pi, \pi]$, instead of the standard DFT interval $[0, 2\pi]$:

$$\boxed{\omega_i = \frac{2\pi i}{N}, \qquad -M \le i \le M} \tag{11.4.2}$$

The forward DFT applied to Eq. (11.4.1) gives:

$$D(\omega_i) = \sum_{k=-M}^{M} \tilde{d}(k)e^{-j\omega_i k} \tag{11.4.3}$$

The rest of the window method may be applied as before, that is, given an appropriate length-N window $w(n)$, the final designed filter will be the delayed and windowed version of $\tilde{d}(k)$:

$$\boxed{h(n) = w(n)\tilde{d}(n-M), \qquad n = 0, 1, \dots, N-1} \tag{11.4.4}$$

We will discuss some examples of the frequency sampling method in Section 14.4.3, where we will design FIR filters for equalizing the slight passband droop of D/A converters and imperfect analog anti-image postfilters.

11.5 Other FIR Design Methods

The Kaiser window method is simple and flexible and can be applied to a variety of filter design problems. However, it does not always result in the smallest possible filter length N, which may be required in some very stringent applications.

The Parks-McClellan method [2–8] based on the so-called optimum *equiripple Chebyshev approximation* generally results in shorter filters. Kaiser [267] has shown that the filter length can be estimated in such cases by a variant of Eq. (11.3.12) that uses the *geometric mean* of the two ripples, $\delta_g = \sqrt{\delta_{\text{pass}}\delta_{\text{stop}}}$:

$$N - 1 = \frac{Df_s}{\Delta f}\,, \qquad D = \frac{A_g - 13}{14.6}\,, \quad A_g = -20\log_{10}(\delta_g) \tag{11.5.1}$$

Moreover, it may be desirable at times to design filters that have additional properties, such as convexity constraints, monotonicity constraints in the passband, or a certain degree of flatness at DC. A recent linear-programming-based filter design program called “meteor” by Steiglitz, Parks, and Kaiser [275,276] addresses such type of designs with constraints.

See Ref. [34] for a summary of filter design methods in MATLAB, both for FIR and IIR filters.

11.6 Problems

11.1 Consider the $d(k)$, $D(\omega)$ pair of Eq. (11.1.1). For the symmetric case, show that the condition that $d(k)$ be *real and even* in k is equivalent to $D(\omega)$ being *real and even* in ω. Similarly for the antisymmetric case, show that the condition that $d(k)$ be *real and odd* in k is equivalent to $D(\omega)$ being *imaginary and odd* in ω.

In both cases, use Euler’s formula to write $D(\omega)$ is a form that shows its symmetry properties explicitly.

If you only had the reality condition that $d(k)$ be real-valued (with no other symmetry constraints), what would be the equivalent condition on $D(\omega)$?

11.2 By performing the appropriate integrations in Eq. (11.1.1), verify the expressions for $d(k)$ of the five filters in Eqs. (11.1.4) and (11.1.6).

11.3 Consider the lowpass differentiator and Hilbert transformer filters with ideal frequency responses defined over one Nyquist interval:

$$D(\omega) = \begin{cases} j\omega, & \text{if } |\omega| \le \omega_c \\ 0, & \text{if } \omega_c < |\omega| \le \pi \end{cases}\,, \quad D(\omega) = \begin{cases} -j\text{sign}(\omega), & \text{if } |\omega| \le \omega_c \\ 0, & \text{if } \omega_c < |\omega| \le \pi \end{cases}$$

Show that the corresponding ideal impulse responses are given by:

$$\begin{aligned} d(k) &= \frac{\omega_c\cos(\omega_c k)}{\pi k} - \frac{\sin(\omega_c k)}{\pi k^2} \qquad &\text{(differentiator)} \\ d(k) &= \frac{1-\cos(\omega_c k)}{\pi k} \qquad &\text{(Hilbert transformer)} \end{aligned}$$

They reduce to those of Eq. (11.1.6) in the full-band case of $\omega_c = \pi$. Do they have the right value at $k = 0$?

11.4 Determine the ideal impulse response $d(k)$ of the bandpass differentiator defined over one Nyquist interval by:

$$D(\omega) = \begin{cases} j\omega, & \text{if } \omega_a \le |\omega| \le \omega_b \\ 0, & \text{if } 0 \le |\omega| < \omega_a, \text{ or } \omega_b < |\omega| \le \pi \end{cases}$$

11.5 Differentiation is an inherently noisy operation in the sense that it amplifies any noise in the data. To see this, calculate the NRR of the FIR differentiation filter $y(n) = x(n) - x(n-1)$. In what sense is this filter an approximation to the ideal differentiator?

Then, calculate the NRR of the ideal lowpass differentiator of Problem 11.3 and compare it with NRR of the full-band case. How does it vary with the cutoff frequency ω_c?

By choosing ω_c to be the bandwidth of the desired signal, lowpass differentiators strike a compromise between differentiating the data and keeping the noise amplification as low as possible.

Lowpass differentiators designed by the Kaiser window method (see Problem 11.20), perform better than the optimal Savitzky-Golay least-squares differentiators of Chap. 23; see Refs. [640,643].

11.6 Show that the mean-square approximation error between the desired and windowed frequency responses of Eqs. (11.1.1) and (11.1.13) can be expressed in the form:

$$\mathcal{E}_M = \int_{-\pi}^{\pi} |D(\omega) - \hat{D}_M(\omega)|^2\,\frac{d\omega}{2\pi} = \int_{-\pi}^{\pi} |D(\omega)|^2\,\frac{d\omega}{2\pi} - \sum_{k=-M}^{M} d(k)^2$$

Then, show the limit $\mathcal{E}_M \to 0$ as $M \to \infty$.

11.7 Using the differentiator and Hilbert transformer filters and the result of Problem 11.6, show the infinite series:

$$\sum_{k=1}^{\infty}\frac{1}{k^2} = \frac{\pi^2}{6}\,, \qquad \sum_{\substack{k=1\\k=\text{odd}}}^{\infty}\frac{1}{k^2} = \frac{\pi^2}{8}\,, \qquad \sum_{\substack{k=2\\k=\text{even}}}^{\infty}\frac{1}{k^2} = \frac{\pi^2}{24}$$

11.8 The ideal Hilbert transformer has frequency response $D(\omega) = -j\text{sign}(\omega)$, for $-\pi \le \omega \le \pi$. Show that it acts as 90° phase shifter converting a cosinusoidal input into a sinusoidal output and vice versa, that is, show the input/output pairs:

$$\cos(\omega n) \xrightarrow{D} \sin(\omega n)\,, \qquad \sin(\omega n) \xrightarrow{D} -\cos(\omega n)$$

11.9 Consider the length-$(2M+1)$ FIR filter of Eq. (11.1.14). Show that it satisfies $\hat{D}_M(z) = \hat{D}_M(z^{-1})$ in the symmetric case, and $\hat{D}_M(z) = -\hat{D}_M(z^{-1})$ in the antisymmetric one.

Then, assuming real coefficients $d(k)$, show that in both the symmetric and antisymmetric cases, if z_0 is a zero of $\hat{D}_M(z)$ not on the unit circle, then necessarily the complex numbers $\{z_0^{-1}, z_0^*, z_0^{-1*}\}$ are also zeros. Indicate the relative locations of these four zeros on the z-plane with respect to the unit circle.

Moreover, show that in the antisymmetric case, the points $z = \pm 1$ are always zeros of $\hat{D}_M(z)$. Can you also see this result from the expression of $\hat{D}_M(\omega)$?

Show that the results of this problem still hold if the impulse response $d(k)$ is windowed with a Hamming, Kaiser, or any other window.

11.10 It is desired to design a linear-phase, odd-length FIR filter having real-valued *symmetric* impulse response. The filter is required to have the *smallest* possible length and to have a zero at the complex location $z = 0.5 + 0.5j$. Determine the impulse response $\mathbf{h}$ of this filter. [*Hint:* Use the results of Problem 11.9.]

Determine expressions for the *magnitude and phase responses* of this filter. Is the phase response linear in ω?

Repeat the problem if the filter is to have an *antisymmetric* impulse response. Is your answer antisymmetric?

11.11 Determine the (a) symmetric and (b) antisymmetric linear-phase FIR filter that has the *shortest* possible length and has at least one zero at the location $z = 0.5j$.

11.12 Expanding Eq. (11.3.3) to first-order in δ_{pass}, show the approximation of (11.3.4).

11.13 It is desired to design a digital lowpass linear-phase FIR filter using the Kaiser window method. The design specifications are as follows: sampling rate of 10 kHz, passband frequency of 1.5 kHz, stopband frequency of 2 kHz, passband attenuation of 0.1 dB, and stopband attenuation of 80 dB. Determine the number of filter taps N.

11.14 It is desired to design a digital lowpass FIR linear phase filter using the Kaiser window method. The maximum passband attenuation is 0.1 dB and the minimum stopband attenuation is 80 dB. At a sampling rate of 10 kHz, the maximum filter length that can be accommodated by your DSP hardware is 251 taps. What is the narrowest transition width Δf in kHz that you can demand?

11.15 Your DSP chip can accommodate FIR filters of maximum length 129 at audio rates of 44.1 kHz. Suppose such a filter is designed by the Kaiser method.

a. What would be the *minimum* transition width Δf between passband and stopband that you can demand if the stopband attenuation is to be 80 dB?

b. If the minimum transition width Δf between passband and stopband is taken to be 2 kHz, then what would be the *maximum* stopband attenuation in dB that you can demand? What would be the corresponding passband attenuation in dB of the designed filter in this case?

c. Suppose your DSP chip could handle length-129 FIR filters at four times the audio rate, that is, $4 \times 44.1 = 176.4$ kHz. You wish to use such a filter as a four-times oversampling FIR interpolator filter for a CD player. The filter is required to have passband from 0 kHz to 19.55 kHz and stopband from 24.55 kHz up to the Nyquist frequency $176.4/2 = 88.2$ kHz. Using a Kaiser design, how much stopband attenuation in dB would you have for such a filter?

11.16 A lowpass FIR filter operating at a rate f_s is implemented on a DSP chip that has instruction rate f_{instr}. Suppose the filter is designed using the Kaiser method. Show that the maximum filter length and minimum transition width (in Hz) that can be implemented are given approximately by:

$$N_{\max} = \frac{f_{\text{instr}}}{f_s}, \qquad \Delta f_{\min} = \frac{Df_s{}^2}{f_{\text{instr}}}$$

where D is the Kaiser design parameter of Eq. (11.3.12). What assumptions were made about the DSP chip?

11.17 A lowpass FIR filter designed by the Kaiser method is required to have transition width Δf Hz and sampling rate f_s. If the filter is to be implemented on a DSP chip that has instruction rate f_{instr}, show that the maximum attainable stopband attenuation for the filter is given by:

$$A_{\max} = 14.36 F_{\text{instr}} \Delta F + 7.95$$

where we defined the normalized frequencies $F_{\text{instr}} = f_{\text{instr}}/f_s$, $\Delta F = \Delta f/f_s$.

11.18 *Computer Experiment: Rectangular and Hamming Windows.* For the lengths $N = 11$, 41, 81, and 121, and using a rectangular window, design a lowpass FIR filter of cutoff frequency $\omega_c = 0.3\pi$. Plot the impulse responses $h(n)$ and magnitude responses $|H(\omega)|$ of the designed filters. Repeat using a Hamming window. Compare the two windows.

11.19 *Computer Experiment: Kaiser Window Designs.* Reproduce the designs and graphs of Examples 11.3.1 and 11.3.2. Plot also the phase responses of the designed filters for $0 \le f \le f_s/2$.

You may find useful the MATLAB routines `klh.m` and `kbp.m`. Write versions of these routines for the Hamming window and use them in this experiment.

Finally, using a Kaiser window, design a highpass filter with specifications: $f_s = 20$ kHz, $f_{\text{pass}} = 5$ kHz, $f_{\text{stop}} = 4$ kHz, $A_{\text{pass}} = 0.1$ dB, and $A_{\text{stop}} = 80$ dB. Plot its magnitude (in dB) and phase response. Compare the Kaiser design with the rectangular and Hamming window designs of the same length.

11.20 *Computer Experiment: Kaiser Window Differentiator Design.* Using the MATLAB routine `kdiff`, design a lowpass FIR differentiator for the following values of the cutoff frequency, transition width, and stopband attenuation parameters $\{\omega_c, \Delta\omega, A\}$:

$$\begin{aligned} \omega_c &= 0.40\pi,\ 0.80\pi && \text{[rads/sample]} \\ \Delta\omega &= 0.10\pi,\ 0.05\pi && \text{[rads/sample]} \\ A &= 30,\ 60 && \text{[dB]} \end{aligned}$$

For each of the eight cases, plot the magnitude response $|H(\omega)|$ of the designed filter over the interval $0 \le \omega \le \pi$.

11.21 *Computer Experiment: Comparison of Kaiser and Savitzky-Golay Differentiators.* For the two cases of Problem 11.20 having $\omega_c = 0.4\pi$, $A = 60$ dB, and $\Delta\omega = \{0.1\pi, 0.05\pi\}$, you will find that the corresponding filter lengths are $N = 75$ and $N = 147$. Using the MATLAB routine `sg.m`, design the order-2 and order-3 Savitzky-Golay differentiator filters of lengths $N = 75$ and 147. This can be done by the MATLAB statements:

```
[B, S] = sg(d, N);  F = S' * S;  G = S * F^(-1);
```

and extract the second column of G. On the same graph, plot and compare the magnitude responses of the Kaiser design, the order-2, and order-3 SG designs for the two values of N. Use frequency scales $0 \le \omega \le \pi$. Then replot only over the range $0 \le \omega \le 0.1\pi$ and use vertical scales $[0, 0.1]$ to magnify the graphs.

Comment on the bandwidth of the SG designs versus the Kaiser design. See also the comments of Problem 11.5.

11.22 *Computer Experiment: Kaiser Window Hilbert Transformer Design.* Using the MATLAB routine `khilb`, design a lowpass FIR Hilbert transformer for the following values of the cutoff frequency, transition width, and stopband attenuation parameters $\{\omega_c, \Delta\omega, A\}$:

$$\begin{aligned} \omega_c &= 0.80\pi,\ 1.00\pi && \text{[rads/sample]} \\ \Delta\omega &= 0.10\pi,\ 0.05\pi && \text{[rads/sample]} \\ A &= 30,\ 60 && \text{[dB]} \end{aligned}$$

For each of the eight cases, plot the magnitude response $|H(\omega)|$ of the designed filter over the interval $0 \le \omega \le \pi$.

11.23 *Computer Experiment: Kaiser Window for Spectral Analysis.* (a) Reproduce all the results and graphs of Example 11.3.6. You may use the MATLAB routine `kparm2` to calculate the window parameters and the routine `dtft.m` to calculate the spectra. (b) Keeping the Kaiser sidelobe level at $R = 70$ dB, repeat part (a) when the middle sinusoid is 35 dB below the other two, and when it is 70 dB below. (c) Repeat parts (a,b) when the transition width is chosen to be $\Delta f = (f_2 - f_1)/6$, and when it is $\Delta f = (f_2 - f_1)/12$.

12

IIR Digital Filter Design

12.1 Bilinear Transformation

One of the simplest and effective methods of designing IIR digital filters with prescribed magnitude response specifications is the *bilinear transformation* method.

Instead of designing the digital filter directly, the method maps the digital filter into an *equivalent analog* filter, which can be designed by one of the well-developed analog filter design methods, such as Butterworth, Chebyshev, or elliptic filter designs. The designed analog filter is then mapped back into the desired digital filter. The procedure is illustrated in Fig. 12.1.1.

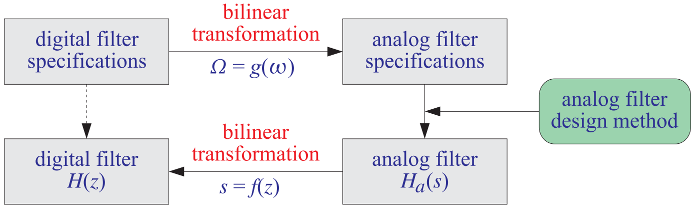


Fig. 12.1.1 Bilinear transformation method.

The z-plane design of the digital filter is replaced by an s-plane design of the equivalent analog filter. The mapping between the s and z planes is carried out by a transformation of the form:

$$\boxed{s = f(z)} \tag{12.1.1}$$

The corresponding mapping between the physical digital frequency $\omega = 2\pi f/f_s$ and the equivalent analog frequency[†] Ω is obtained by replacing $s = j\Omega$ and $z = e^{j\omega}$ into Eq. (12.1.1), giving $j\Omega = f(e^{j\omega})$, which can be written as:

[†] Here, Ω is the frequency of a fictitious equivalent analog filter. It has arbitrary units and should not be confused with the physical frequency $2\pi f$ in radians/sec.

$$\boxed{\Omega = g(\omega)} \tag{12.1.2}$$

The *bilinear transformation* is a particular case of Eq. (12.1.1) defined by:

$$\boxed{s = f(z) = \frac{1-z^{-1}}{1+z^{-1}}} \quad \text{(bilinear transformation)} \tag{12.1.3}$$

The corresponding mapping of frequencies is obtained as follows:

$$j\Omega = f(e^{j\omega}) = \frac{1-e^{-j\omega}}{1+e^{-j\omega}} = \frac{e^{j\omega/2}-e^{-j\omega/2}}{e^{j\omega/2}+e^{-j\omega/2}} = j\frac{\sin(\omega/2)}{\cos(\omega/2)} = j\tan\left(\frac{\omega}{2}\right)$$

which gives:

$$\boxed{\Omega = g(\omega) = \tan\left(\frac{\omega}{2}\right)} \quad \text{(bilinear transformation)} \tag{12.1.4}$$

Because of the nonlinear relationship between the physical frequency ω and the fictitious analog frequency Ω, Eq. (12.1.4) is sometimes referred to as a *frequency prewarping* transformation.

Other versions of the bilinear transformation, which are appropriate for designing highpass, bandpass, or bandstop digital filters by starting from an equivalent lowpass analog filter, are as follows:

$$\boxed{\begin{array}{ll} \text{(lowpass)} & s = f(z) = \dfrac{1-z^{-1}}{1+z^{-1}} \\[2ex] \text{(highpass)} & s = f(z) = \dfrac{1+z^{-1}}{1-z^{-1}} \\[2ex] \text{(bandpass)} & s = f(z) = \dfrac{1-2cz^{-1}+z^{-2}}{1-z^{-2}} \\[2ex] \text{(bandstop)} & s = f(z) = \dfrac{1-z^{-2}}{1-2c+z^{-2}} \end{array}} \tag{12.1.5}$$

with corresponding frequency maps:

$$\boxed{\begin{array}{ll} \text{(lowpass)} & \Omega = g(\omega) = \tan\left(\dfrac{\omega}{2}\right) \\[2ex] \text{(highpass)} & \Omega = g(\omega) = -\cot\left(\dfrac{\omega}{2}\right) \\[2ex] \text{(bandpass)} & \Omega = g(\omega) = \dfrac{c-\cos\omega}{\sin\omega} \\[2ex] \text{(bandstop)} & \Omega = g(\omega) = \dfrac{\sin\omega}{\cos\omega - c} \end{array}} \tag{12.1.6}$$

The overall design method can be summarized as follows: Starting with given *magnitude response* specifications for the *digital filter*, the specifications are transformed by the *appropriate* prewarping transformation, Eqs. (12.1.4) or (12.1.6), into the specifications of an equivalent analog filter. Using an analog filter design technique, the equivalent analog filter, say $H_a(s)$, is designed. Using the bilinear transformation, Eqs. (12.1.3) or (12.1.5), the analog filter is mapped back into the desired digital filter $H(z)$, by defining:

$$\boxed{H(z) = H_a(s)\Big|_{s=f(z)} = H_a\big(f(z)\big)} \tag{12.1.7}$$

The corresponding frequency responses also map in a similar fashion:

$$\boxed{H(\omega) = H_a(\Omega)\Big|_{\Omega=g(\omega)} = H_a\big(g(\omega)\big)} \tag{12.1.8}$$

A useful property of the bilinear transformation (12.1.3) is that it maps the left-hand s-plane into the inside of the unit circle on the z-plane. Figure 12.1.2 shows this property. Because all analog filter design methods give rise to stable and causal transfer functions $H_a(s)$, this property guarantees that the digital filter $H(z)$ obtained by Eq. (12.1.7) will also be *stable and causal.*

The alternative transformations of Eqs. (12.1.5) also share this property, where in the bandpass and bandstop cases it is required that $|c| \le 1$.

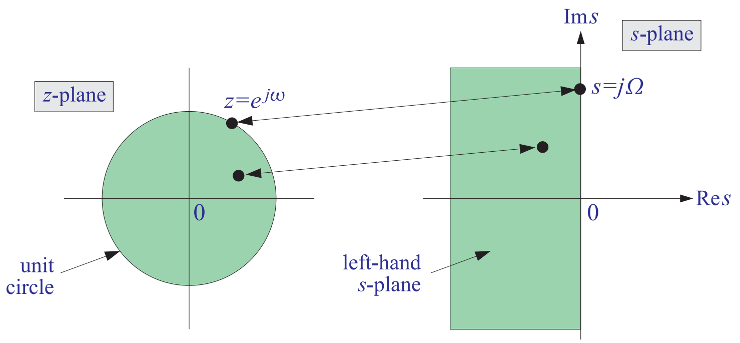


Fig. 12.1.2 Interior of unit z-circle gets mapped onto left-hand s-plane.

A related property of the bilinear transformation is that it maps the s-plane frequency axis, that is, the imaginary axis $s = j\Omega$ onto the z-plane frequency axis, that is, the periphery of the unit circle $z = e^{j\omega}$. The above properties can be proved easily by taking real parts of Eq. (12.1.3):

$$\text{Re}\,s = \frac{1}{2}(s+s^*) = \frac{1}{2}\left[\frac{z-1}{z+1} + \frac{z^*-1}{z^*+1}\right] = \frac{(z-1)(z^*+1)+(z+1)(z^*-1)}{2(z+1)(z^*+1)}$$

or,

$$\mathrm{Re}\, s = \frac{|z|^2 - 1}{|z+1|^2}$$

which shows that

$$\mathrm{Re}\, s < 0 \quad \Leftrightarrow \quad |z| < 1 \qquad \text{and} \qquad \mathrm{Re}\, s = 0 \quad \Leftrightarrow \quad |z| = 1$$

Next, we apply the bilinear transformation method to the design of simple first- and second-order filters, and then to higher-order filters based on Butterworth and Chebyshev analog designs.

12.2 First-Order Lowpass and Highpass Filters

Perhaps the simplest filter design problem is that of designing a first-order lowpass filter that has a prescribed cutoff frequency, say f_c, and operates at a given sampling rate f_s. Such a filter will have a transfer function of the form:

$$H(z) = \frac{b_0 + b_1 z^{-1}}{1 + a_1 z^{-1}}$$

The design problem is to determine the filter coefficients $\{b_0, b_1, a_1\}$ in terms of the cutoff frequency f_c and rate f_s. The definition of the cutoff frequency is a matter of convention. Roughly speaking, it defines the range of frequencies that pass through, that is, $0 \le f \le f_c$, and the range of frequencies that are filtered out, that is, $f_c \le f \le f_s/2$. The digital cutoff frequency ω_c in units of radians per sample is defined to be:

$$\boxed{\omega_c = \frac{2\pi f_c}{f_s}}$$

By convention, ω_c is usually taken to be the so-called 3-dB cutoff frequency, that is, the frequency at which the magnitude response squared drops by a factor of two (i.e., 3 dB) compared to its value at DC:

$$\frac{|H(\omega_c)|^2}{|H(0)|^2} = \frac{1}{2} \quad \Rightarrow \quad -10\log_{10}\left[\frac{|H(\omega_c)|^2}{|H(0)|^2}\right] = -10\log_{10}\left[\frac{1}{2}\right] = 3\text{ dB}$$

Assuming that $H(z)$ is normalized to unity gain at DC, $|H(0)| = 1$, this condition reads equivalently:

$$|H(\omega_c)|^2 = \frac{1}{2} \tag{12.2.1}$$

More generally, we may define ω_c or f_c to be the frequency at which $|H(\omega)|^2$ drops by a factor of $G_c^2 < 1$, or a drop in dB:

$$\boxed{A_c = -10\log_{10}(G_c^2) = -20\log_{10} G_c} \tag{12.2.2}$$

which can be inverted to give:

$$\boxed{G_c = 10^{-A_c/20}} \tag{12.2.3}$$

Thus, in this case, the defining condition for ω_c is:

$$\boxed{|H(\omega_c)|^2 = G_c^2 = 10^{-A_c/10}} \tag{12.2.4}$$

If $A_c = 3$ dB, we have $G_c^2 = 1/2$, and (12.2.4) reduces to (12.2.1). Figure 12.2.1 shows this type of magnitude response specification, both in the general and 3-dB cases. The design problem is then to determine the filter coefficients $\{b_0, b_1, a_1\}$ for given values of the cutoff specifications $\{f_c, A_c\}$.

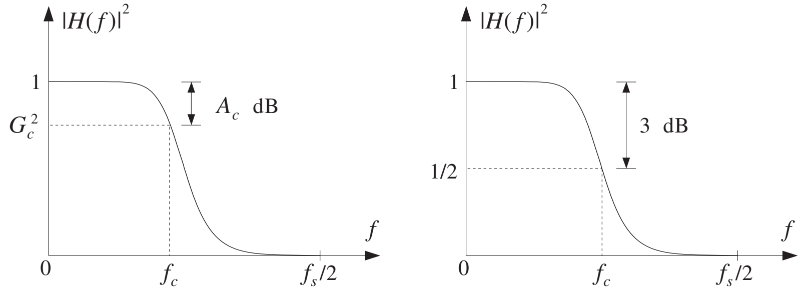


Fig. 12.2.1 Cutoff frequency specifications for lowpass digital filter.

The bilinear transformation method can be applied as follows. First, we prewarp the cutoff frequency to get the cutoff frequency of the equivalent analog filter:

$$\boxed{\Omega_c = \tan\left(\frac{\omega_c}{2}\right) = \tan\left(\frac{\pi f_c}{f_s}\right)}$$

Then, we design a first-order analog filter and adjust its parameters so that its cutoff frequency is Ω_c. Figure 12.2.2 shows the transformation of the specifications. The analog filter's transfer function is taken to be:

$$\boxed{H_a(s) = \frac{\alpha}{s + \alpha}} \tag{12.2.5}$$

Note that $H_a(s)$ has been normalized to unity gain at DC, or at $s = 0$. Its frequency and magnitude responses are obtained by setting $s = j\Omega$:

$$H_a(\Omega) = \frac{\alpha}{j\Omega + \alpha} \qquad \Rightarrow \qquad |H_a(\Omega)|^2 = \frac{\alpha^2}{\Omega^2 + \alpha^2} \tag{12.2.6}$$

Because the design method satisfies Eq. (12.1.8) for the frequency and magnitude responses, we can determine the filter parameter α by requiring the cutoff condition:

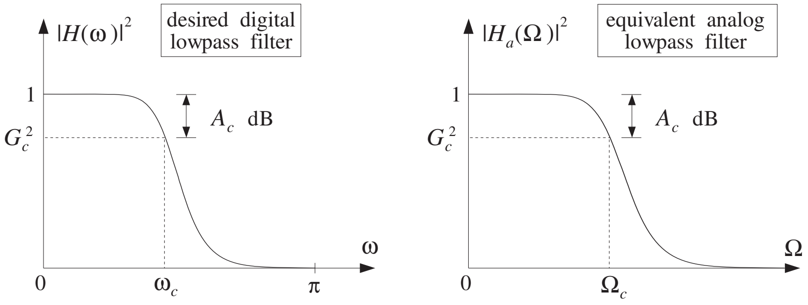


Fig. 12.2.2 Equivalent cutoff specifications of lowpass digital and analog filters.

$$|H(\omega_c)|^2 = |H_a(\Omega_c)|^2 = \frac{\alpha^2}{\Omega_c^2 + \alpha^2} = G_c^2 \tag{12.2.7}$$

which can be solved for α:

$$\boxed{\alpha = \frac{G_c}{\sqrt{1 - G_c^2}}\,\Omega_c = \frac{G_c}{\sqrt{1 - G_c^2}}\tan\left(\frac{\omega_c}{2}\right)} \tag{12.2.8}$$

Once the parameter α of the analog filter is fixed, we may transform the filter to the z-domain by the bilinear transformation (12.1.3):

$$H(z) = H_a(s) = \frac{\alpha}{s + \alpha}\bigg|_{s=\frac{1-z^{-1}}{1+z^{-1}}} = \frac{\alpha}{\dfrac{1 - z^{-1}}{1 + z^{-1}} + \alpha} = \frac{\alpha(1 + z^{-1})}{1 - z^{-1} + \alpha(1 + z^{-1})}$$

which gives after some algebra:

$$\boxed{H(z) = b\,\frac{1 + z^{-1}}{1 - az^{-1}}} \tag{12.2.9}$$

where its coefficients are computed in terms of α:

$$\boxed{a = \frac{1 - \alpha}{1 + \alpha}, \quad b = \frac{\alpha}{1 + \alpha}} \tag{12.2.10}$$

The overall design is summarized as follows: Given the cutoff frequency ω_c and corresponding gain A_c in dB, compute G_c using Eq. (12.2.3); then compute the analog parameter α and the digital filter coefficients $\{b, a\}$.

Note that because $H_a(s)$ is stable and causal, its pole $s = -\alpha$ lies in the left-hand s-plane. This follows from the fact that $\tan(\omega_c/2) > 0$ for any value of ω_c in the range $0 < \omega_c < \pi$. This pole gets mapped onto the z-plane pole $z = a$, which satisfies $|a| < 1$ for $\alpha > 0$. The zero of the digital filter at $z = -1$ corresponds to the Nyquist frequency $\omega = \pi$ or $f = f_s/2$. Also note that the normalizing gain b can be expressed directly in terms of a, as follows:

$$b = \frac{1 - a}{2} \tag{12.2.11}$$

If ω_c is taken to be the 3-dB cutoff frequency, then $G_c^2 = 1/2$ and Eq. (12.2.8) simplifies to:

$$\boxed{\alpha = \Omega_c = \tan\left(\frac{\omega_c}{2}\right)} \tag{12.2.12}$$

The frequency response of the digital filter can be obtained by setting $z = e^{j\omega}$ in Eq. (12.2.9), or more simply in terms of the frequency response of the transformed analog filter, that is, using Eq. (12.1.8):

$$H(\omega) = H_a(\Omega) = \frac{\alpha}{\alpha + j\Omega} = \frac{\alpha}{\alpha + j\tan(\omega/2)}$$

Thus, we have the two equivalent expressions:

$$H(\omega) = b\,\frac{1 + e^{-j\omega}}{1 - ae^{-j\omega}} = \frac{\alpha}{\alpha + j\tan(\omega/2)}$$

and similarly for the magnitude response:

$$|H(\omega)|^2 = b^2\,\frac{2(1 + \cos\omega)}{1 - 2a\cos\omega + a^2} = \frac{\alpha^2}{\alpha^2 + \tan^2(\omega/2)} \tag{12.2.13}$$

The bilinear transformation is not necessary. In fact, using the first of the two expressions in Eq. (12.2.13), the digital filter can be designed directly without the intermediate step of an analog filter. However, for higher-order filters the bilinear transformation approach is *algebraically simpler* than the direct design.

Example 12.2.1: Design a lowpass digital filter operating at a rate of 10 kHz, whose 3-dB frequency is 1 kHz. Then, redesign it such that at 1 kHz its attenuation is $G_c^2 = 0.9$, corresponding to $A_c = -10\log_{10}(0.9) = 0.46$ dB.

Then, redesign the above two filters such that their cutoff frequency is now 3.5 kHz.

Solution: The digital cutoff frequency is

$$\omega_c = \frac{2\pi f_c}{f_s} = \frac{2\pi \cdot 1\ \text{kHz}}{10\ \text{kHz}} = 0.2\pi \quad \text{rads/sample}$$

and its prewarped analog version:

$$\Omega_c = \tan\left(\frac{\omega_c}{2}\right) = \tan(0.1\pi) = 0.3249$$

For the first filter, we have $G_c^2 = 0.5$ corresponding to Eq. (12.2.12), which gives the filter parameters:

$$\alpha = \Omega_c = 0.3249, \quad a = \frac{1 - \alpha}{1 + \alpha} = 0.5095, \quad b = \frac{1 - a}{2} = 0.2453$$

and digital filter transfer function:

$$H(z)=0.2453\,\frac{1+z^{-1}}{1-0.5095z^{-1}}$$

For the second filter, ω_c corresponds to attenuation $G_c^2 = 0.9$. Using Eq. (12.2.8) we find:

$$\alpha=\Omega_c\,\frac{G_c}{\sqrt{1-G_c^2}}=0.3249\,\frac{\sqrt{0.9}}{\sqrt{1-0.9}}=0.9748$$

corresponding to filter coefficients and transfer function:

$$a=\frac{1-\alpha}{1+\alpha}=0.0128,\quad b=\frac{1-a}{2}=0.4936,\quad H(z)=0.4936\,\frac{1+z^{-1}}{1-0.0128z^{-1}}$$

The magnitude responses of the two designed filters and their specifications are shown in the left graph of Fig. 12.2.3.

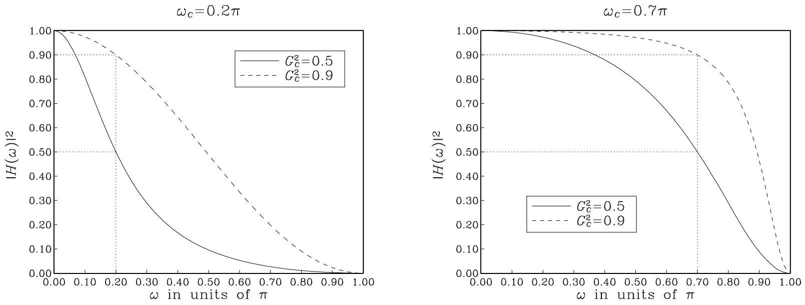


Fig. 12.2.3 First-order lowpass digital filters of Example 12.2.1.

For the next two cases, we have $f_c = 3.5$ kHz, resulting in the digital frequency $\omega_c = 2\pi f_c/f_s = 0.7\pi$, and corresponding prewarped version $\Omega_c = \tan(\omega_c/2)= \tan(0.35\pi)= 1.9626$.

For the 3-dB case, we have $\alpha = \Omega_c = 1.9626$, which gives the filter coefficients and transfer function:

$$a=\frac{1-\alpha}{1+\alpha}=-0.3249,\quad b=\frac{1-a}{2}=0.6625,\quad H(z)=0.6625\,\frac{1+z^{-1}}{1+0.3249z^{-1}}$$

For the 0.46-dB case having $G_c^2 = 0.9$, we calculate:

$$\alpha=\Omega_c\,\frac{G_c}{\sqrt{1-G_c^2}}=1.9626\,\frac{\sqrt{0.9}}{\sqrt{1-0.9}}=5.8878$$

which gives for the digital filter coefficients and transfer function:

$$a=\frac{1-\alpha}{1+\alpha}=-0.7096,\quad b=\frac{1-a}{2}=0.8548,\quad H(z)=0.8548\,\frac{1+z^{-1}}{1+0.7096z^{-1}}$$

The corresponding magnitude responses are shown in the right graph of Fig. 12.2.3. Note that in the last two cases the cutoff frequency is $\omega_c > \pi/2$ which results in a negative value of the filter pole a. □

Highpass digital filters can be designed just as easily. Starting with a highpass analog first-order filter of the form:

$$H_a(s)=\frac{s}{s+\alpha} \tag{12.2.14}$$

we can transform it into a highpass digital filter by the bilinear transformation:

$$H(z)=H_a(s)=\left.\frac{s}{s+\alpha}\right|_{s=\frac{1-z^{-1}}{1+z^{-1}}}=\frac{\dfrac{1-z^{-1}}{1+z^{-1}}}{\dfrac{1-z^{-1}}{1+z^{-1}}+\alpha}=\frac{1-z^{-1}}{1-z^{-1}+\alpha(1+z^{-1})}$$

which gives:

$$\boxed{H(z)=b\,\frac{1-z^{-1}}{1-az^{-1}}} \tag{12.2.15}$$

where its coefficients are computed in terms of α:

$$\boxed{a=\frac{1-\alpha}{1+\alpha},\quad b=\frac{1}{1+\alpha}=\frac{1+a}{2}} \tag{12.2.16}$$

To determine the parameter α, we require that at a given highpass cutoff frequency ω_c the attenuation is A_c dB. That is, we demand

$$|H(\omega_c)|^2=G_c^2=10^{-A_c/10} \tag{12.2.17}$$

Figure (12.2.4) depicts the mapping of the specifications of the digital filter to the equivalent analog filter. Setting $s = j\Omega$ into Eq. (12.2.14), we obtain for the frequency and magnitude responses:

$$H_a(\Omega)=\frac{j\Omega}{j\Omega+\alpha}\quad\Rightarrow\quad|H_a(\Omega)|^2=\frac{\Omega^2}{\Omega^2+\alpha^2} \tag{12.2.18}$$

The design condition (12.2.17) gives then:

$$|H(\omega_c)|^2=|H_a(\Omega_c)|^2=\frac{\Omega_c^2}{\Omega_c^2+\alpha^2}=G_c^2$$

which can be solved for α:

$$\boxed{\alpha=\frac{\sqrt{1-G_c^2}}{G_c}\,\Omega_c=\frac{\sqrt{1-G_c^2}}{G_c}\,\tan\left(\frac{\omega_c}{2}\right)} \tag{12.2.19}$$

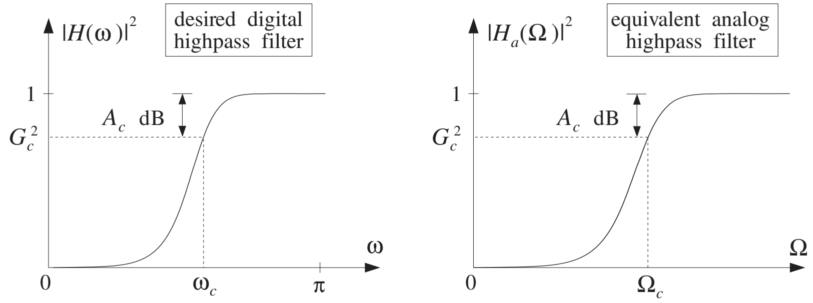


Fig. 12.2.4 Equivalent cutoff specifications of highpass digital and analog filters.

In summary, given the specifications $\{\omega_c, A_c\}$, we compute the analog parameter α and then the digital filter parameters $\{b, a\}$ which define the desired transfer function (12.2.15).

As in the lowpass case, we can replace $\Omega = \tan(\omega/2)$ and write the frequency and magnitude responses of the designed filter in the two equivalent forms:

$$H(\omega) = b\frac{1 - e^{-j\omega}}{1 - ae^{-j\omega}} = \frac{j\tan(\omega/2)}{\alpha + j\tan(\omega/2)}$$

$$|H(\omega)|^2 = b^2\frac{2(1 - \cos\omega)}{1 - 2a\cos\omega + a^2} = \frac{\tan^2(\omega/2)}{\alpha^2 + \tan^2(\omega/2)}$$

We note again that if ω_c represents the 3-dB cutoff frequency corresponding to $G_c^2 = 1/2$, then Eq. (12.2.19) simplifies to:

$$\boxed{\alpha = \Omega_c = \tan\left(\frac{\omega_c}{2}\right)} \tag{12.2.20}$$

This and Eq. (12.2.12) imply that if the lowpass and highpass filters have the *same* 3-dB cutoff frequency ω_c, then they will have the same analog filter parameter α, and hence the same z-plane pole parameter a. Thus, in this case, the analog filters will be:

$$H_{\text{LP}}(s) = \frac{\alpha}{s + \alpha}, \quad H_{\text{HP}}(s) = \frac{s}{s + \alpha} \tag{12.2.21}$$

and the corresponding digital filters:

$$H_{\text{LP}}(z) = \frac{1 - a}{2}\,\frac{1 + z^{-1}}{1 - az^{-1}}, \quad H_{\text{HP}}(z) = \frac{1 + a}{2}\,\frac{1 - z^{-1}}{1 - az^{-1}} \tag{12.2.22}$$

They are *complementary filters* in the sense that their transfer functions add up to unity:

$$H_{\text{LP}}(s) + H_{\text{HP}}(s) = \frac{\alpha}{s + \alpha} + \frac{s}{s + \alpha} = \frac{s + \alpha}{s + \alpha} = 1$$

and similarly,

$$\boxed{H_{\text{LP}}(z) + H_{\text{HP}}(z) = 1} \tag{12.2.23}$$

We have already encountered such complementary filters in the comb and notch filters of Section 15.11. The corresponding frequency and magnitude responses squared also add up to unity; for example,

$$|H_{\text{LP}}(\omega)|^2 + |H_{\text{HP}}(\omega)|^2 = \frac{\alpha^2}{\alpha^2 + \tan^2(\omega/2)} + \frac{\tan^2(\omega/2)}{\alpha^2 + \tan^2(\omega/2)} = 1$$

Example 12.2.2: Design a highpass digital filter operating at a rate of 10 kHz, whose 3-dB cutoff frequency is 1 kHz. Then, redesign it such that at 1 kHz its attenuation is $G_c^2 = 0.9$, corresponding to $A_c = -10\log_{10}(0.9) = 0.46$ dB.

Solution: The digital cutoff frequency is as in Example 12.2.1, $\omega_c = 0.2\pi$. Its prewarped analog version is $\Omega_c = \tan(\omega_c/2) = 0.3249$.

For the first filter, we have $G_c^2 = 0.5$ corresponding to Eq. (12.2.20), which gives the filter parameters:

$$\alpha = \Omega_c = 0.3249, \quad a = \frac{1 - \alpha}{1 + \alpha} = 0.5095, \quad b = \frac{1 + a}{2} = 0.7548$$

and transfer function:

$$H(z) = 0.7548\,\frac{1 - z^{-1}}{1 - 0.5095z^{-1}}$$

For the second filter, we set $G_c^2 = 0.9$ into Eq. (12.2.19) to get:

$$\alpha = \Omega_c\frac{\sqrt{1 - G_c^2}}{G_c} = 0.1083$$

corresponding to filter coefficients and transfer function:

$$a = \frac{1 - \alpha}{1 + \alpha} = 0.8046, \quad b = \frac{1 + a}{2} = 0.9023, \quad H(z) = 0.9023\,\frac{1 - z^{-1}}{1 - 0.8046z^{-1}}$$

The magnitude responses of the two designed filters and their specifications are shown in the left graph of Fig. 12.2.5.

The right graph shows the complementarity property of the highpass and lowpass filters, with 3-dB frequency of $\omega_c = 0.2\pi$. The magnitude responses intersect at precisely the 3-dB point. □

12.3 Second-Order Peaking and Notching Filters

In Section 6.4, we designed second-order resonator and notch filters using pole/zero placement. For narrow-width filters this technique is adequate. But, it becomes cumbersome for wider peak widths, such as those that might be used in graphic and parametric audio equalizers. The bilinear transformation method offers precise control over the desired specifications of such filters [283–292].

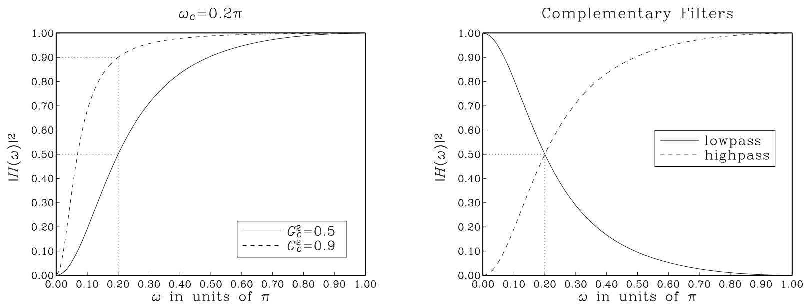


Fig. 12.2.5 Highpass and complementary lowpass digital filters.

Consider first the design of *notch* filters. The desired specifications are the sampling rate f_s, notch frequency f_0, and bandwidth Δf of the notch, or, equivalently, the corresponding digital frequencies:

$$\boxed{\omega_0 = \frac{2\pi f_0}{f_s}, \quad \Delta\omega = \frac{2\pi\Delta f}{f_s}}$$

Alternatively, we may specify ω_0 and the Q-factor, $Q = \omega_0/\Delta\omega = f_0/\Delta f$. The specifications together with their bilinear analog equivalents are shown in Fig. 12.3.1. The bandwidth $\Delta\omega$ is usually defined to be the 3-dB width, that is, the *full width at half maximum* of the magnitude squared response. More generally, it can be defined to be the full width at a level G_B^2, or in decibels:

$$A_B = -10\log_{10}(G_B^2) \qquad \Rightarrow \qquad G_B = 10^{-A_B/20} \tag{12.3.1}$$

The bandwidth $\Delta\omega$ is defined as the difference $\Delta\omega = \omega_2 - \omega_1$ of the left and right bandwidth frequencies ω_1 and ω_2 that are solutions of the equation $|H(\omega)|^2 = G_B^2$, as shown in Fig. 12.3.1. For the 3-dB width, we have the condition $|H(\omega)|^2 = 1/2$.

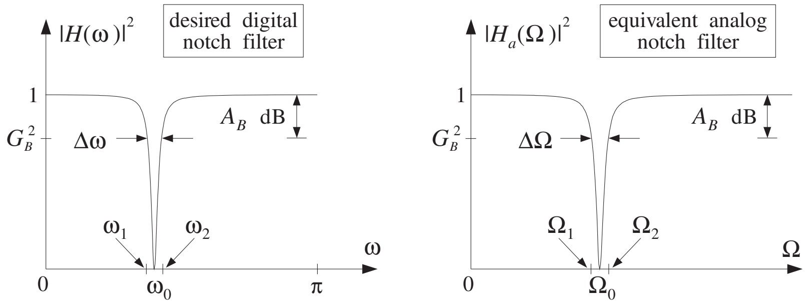


Fig. 12.3.1 Digital notch filter and its analog equivalent.

Given the desired specifications $\{\omega_0, \Delta\omega, G_B^2\}$, the design procedure begins with the following expression for the equivalent analog filter, which has a notch at frequency $\Omega = \Omega_0$:

$$\boxed{H_a(s) = \frac{s^2 + \Omega_0^2}{s^2 + \alpha s + \Omega_0^2}} \tag{12.3.2}$$

We will see below that the filter parameters $\{\alpha, \Omega_0\}$ can be calculated from the given specifications by:

$$\boxed{\Omega_0 = \tan\left(\frac{\omega_0}{2}\right), \quad \alpha = \frac{\sqrt{1-G_B^2}}{G_B}(1+\Omega_0^2)\tan\left(\frac{\Delta\omega}{2}\right)} \tag{12.3.3}$$

Then, using the bilinear transformation $s = (1-z^{-1})/(1+z^{-1})$, the filter $H_a(s)$ is transformed into the digital filter $H(z)$ as follows:

$$\begin{aligned}
H(z) = H_a(s) = \frac{s^2+\Omega_0^2}{s^2+\alpha s+\Omega_0^2} &= \frac{\left(\dfrac{1-z^{-1}}{1+z^{-1}}\right)^2 + \Omega_0^2}{\left(\dfrac{1-z^{-1}}{1+z^{-1}}\right)^2 + \alpha\left(\dfrac{1-z^{-1}}{1+z^{-1}}\right) + \Omega_0^2} \\
&= \frac{(1-z^{-1})^2 + \Omega_0^2(1+z^{-1})^2}{(1-z^{-1})^2 + \alpha(1-z^{-1})(1+z^{-1}) + \Omega_0^2(1+z^{-1})^2} \\
&= \left(\frac{1+\Omega_0^2}{1+\Omega_0^2+\alpha}\right)\frac{1 - 2\left(\dfrac{1-\Omega_0^2}{1+\Omega_0^2}\right)z^{-1} + z^{-2}}{1 - 2\left(\dfrac{1-\Omega_0^2}{1+\Omega_0^2+\alpha}\right)z^{-1} + \left(\dfrac{1+\Omega_0^2-\alpha}{1+\Omega_0^2+\alpha}\right)z^{-2}}
\end{aligned}$$

The coefficients of the digital filter can be simplified considerably by recognizing that α already has a factor $(1+\Omega_0^2)$ in its definition (12.3.3). Thus, we may replace it by

$$\alpha = (1+\Omega_0^2)\beta$$

where

$$\boxed{\beta = \frac{\sqrt{1-G_B^2}}{G_B}\tan\left(\frac{\Delta\omega}{2}\right)} \tag{12.3.4}$$

Using some trigonometry, we can write also

$$\frac{1-\Omega_0^2}{1+\Omega_0^2} = \frac{1-\tan^2(\omega_0/2)}{1+\tan^2(\omega_0/2)} = \cos\omega_0$$

Canceling several common factors of $(1+\Omega_0^2)$, we can write the transfer function $H(z)$ in the simplified form:

$$H(z)=\left(\frac{1}{1+\beta}\right)\frac{1-2\cos\omega_0\,z^{-1}+z^{-2}}{1-2\left(\dfrac{\cos\omega_0}{1+\beta}\right)z^{-1}+\left(\dfrac{1-\beta}{1+\beta}\right)z^{-2}} \tag{12.3.5}$$

Defining the overall normalization gain by

$$\boxed{b=\frac{1}{1+\beta}=\frac{1}{1+\dfrac{\sqrt{1-G_B^2}}{G_B}\tan\left(\dfrac{\Delta\omega}{2}\right)}} \tag{12.3.6}$$

we may write $(1-\beta)/(1+\beta)=2b-1$, and therefore, the coefficients of $H(z)$ can be expressed in terms of b as follows:

$$\boxed{H(z)=b\frac{1-2\cos\omega_0\,z^{-1}+z^{-2}}{1-2b\cos\omega_0\,z^{-1}+(2b-1)z^{-2}}}\qquad\text{(notch filter)} \tag{12.3.7}$$

This is the final design. It expresses the filter coefficients in terms of the design specifications $\{\omega_0,\Delta\omega,G_B^2\}$. Note that the numerator has a notch at the desired frequency ω_o and its conjugate $-\omega_0$, because it factors into:

$$1-2\cos\omega_0\,z^{-1}+z^{-2}=(1-e^{j\omega_0}z^{-1})(1-e^{-j\omega_0}z^{-1})$$

It remains to justify the design equations (12.3.3). The first one, $\Omega_0=\tan(\omega_0/2)$, is simply the bilinear transformation of ω_0 and makes the analog filter's notch correspond to the digital filter's notch. The equation for α can be derived as follows. Setting $s=j\Omega$ in Eq. (12.3.2), we obtain the frequency and magnitude responses:

$$H_a(\Omega)=\frac{-\Omega^2+\Omega_0^2}{-\Omega^2+j\alpha\Omega+\Omega_0^2}\quad\Rightarrow\quad |H_a(\Omega)|^2=\frac{(\Omega^2-\Omega_0^2)^2}{(\Omega^2-\Omega_0^2)^2+\alpha^2\Omega^2}$$

It is evident from these expressions that $H_a(\Omega)$ has a notch at $\Omega=\pm\Omega_0$. The analog bandwidth frequencies Ω_1 and Ω_2 are solutions of the equation $|H_a(\Omega)|^2=G_B^2$, that is,

$$\frac{(\Omega^2-\Omega_0^2)^2}{(\Omega^2-\Omega_0^2)^2+\alpha^2\Omega^2}=G_B^2 \tag{12.3.8}$$

Eliminating the denominator and rearranging terms, we can write it as the quartic equation in Ω:

$$\Omega^4-\left(2\Omega_0^2+\frac{G_B^2}{1-G_B^2}\alpha^2\right)\Omega^2+\Omega_0^4=0 \tag{12.3.9}$$

It may be thought of as a quadratic equation in the variable $x=\Omega^2$, that is,

$$x^2-\left(2\Omega_0^2+\frac{G_B^2}{1-G_B^2}\alpha^2\right)x+\Omega_0^4=0$$

Let $x_1=\Omega_1^2$ and $x_2=\Omega_2^2$ be its two solutions. Rather than solving it, we use the properties that the sum and product of the two solutions are related to the first and second coefficients of the quadratic by:

$$\begin{aligned}\Omega_1^2+\Omega_2^2&=x_1+x_2=2\Omega_0^2+\frac{G_B^2}{1-G_B^2}\alpha^2\\ \Omega_1^2\Omega_2^2&=x_1x_2=\Omega_0^4\end{aligned} \tag{12.3.10}$$

From the second equation, we obtain:

$$\boxed{\Omega_1\Omega_2=\Omega_0^2} \tag{12.3.11}$$

which states that Ω_0 is the *geometric mean* of the left and right bandwidth frequencies. Using this result in the first of (12.3.10), we obtain:

$$\Omega_1^2+\Omega_2^2=2\Omega_1\Omega_2+\frac{G_B^2}{1-G_B^2}\alpha^2$$

which allows us to solve for the analog bandwidth:

$$\Delta\Omega^2=(\Omega_2-\Omega_1)^2=\Omega_1^2+\Omega_2^2-2\Omega_1\Omega_2=\frac{G_B^2}{1-G_B^2}\alpha^2$$

or,

$$\boxed{\Delta\Omega=\Omega_2-\Omega_1=\frac{G_B}{\sqrt{1-G_B^2}}\,\alpha}\qquad (A_B\text{-dB width}) \tag{12.3.12}$$

Solving for α, we have:

$$\boxed{\alpha=\frac{\sqrt{1-G_B^2}}{G_B}\,\Delta\Omega} \tag{12.3.13}$$

Note that for the 3-dB case, $G_B^2=1/2$, the parameter α is equal to the 3-dB bandwidth:

$$\boxed{\alpha=\Delta\Omega}\qquad\text{(3-dB width)} \tag{12.3.14}$$

Finally, we must relate the analog bandwidth $\Delta\Omega$ to the physical bandwidth $\Delta\omega=\omega_2-\omega_1$. Using the bilinear transformations $\Omega_1=\tan(\omega_1/2)$, $\Omega_2=\tan(\omega_2/2)$, and some trigonometry, we find:

$$\begin{aligned}\tan\left(\frac{\Delta\omega}{2}\right)=\tan\left(\frac{\omega_2-\omega_1}{2}\right)&=\frac{\tan(\omega_2/2)-\tan(\omega_1/2)}{1+\tan(\omega_2/2)\tan(\omega_1/2)}\\ &=\frac{\Omega_2-\Omega_1}{1+\Omega_2\Omega_1}=\frac{\Delta\Omega}{1+\Omega_0^2}\end{aligned}$$

where we used $\Omega_1\Omega_2=\Omega_0^2$. Solving for $\Delta\Omega$, we have:

$$\boxed{\Delta\Omega = (1+\Omega_0^2)\tan\left(\frac{\Delta\omega}{2}\right)} \tag{12.3.15}$$

Thus, combining Eqs. (12.3.13) and (12.3.15), we obtain Eq. (12.3.3). The design equations (12.3.6) and (12.3.7) and some design examples were discussed also in Sections 16.2.2 and 15.11. For example, see Fig. 16.2.14.

In the limit as ω_0 or Ω_0 tend to zero, the notch filter will behave as a highpass filter. The transfer functions $H(z)$ and $H_a(s)$ become in this case the highpass transfer functions of the previous section. For example, setting $\Omega_0 = 0$ in Eq. (12.3.2), we have:

$$H_a(s) = \left.\frac{s^2+\Omega_0^2}{s^2+\alpha s+\Omega_0^2}\right|_{\Omega_0=0} = \frac{s^2}{s^2+\alpha s} = \frac{s}{s+\alpha}$$

Peaking or resonator filters can be designed in a similar fashion. The desired specifications are shown in Fig. 12.3.2. The design procedure starts with the second-order analog resonator filter:

$$\boxed{H_a(s) = \frac{\alpha s}{s^2+\alpha s+\Omega_0^2}} \tag{12.3.16}$$

which has frequency and magnitude responses:

$$H_a(\Omega) = \frac{j\alpha\Omega}{-\Omega^2+j\alpha\Omega+\Omega_0^2} \quad\Rightarrow\quad |H_a(\Omega)|^2 = \frac{\alpha^2\Omega^2}{(\Omega^2-\Omega_0^2)^2+\alpha^2\Omega^2}$$

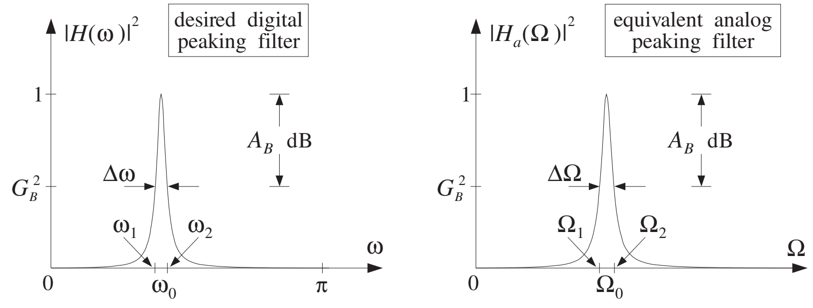


Fig. 12.3.2 Digital peaking filter and its analog equivalent.

Note that $H_a(\Omega)$ is normalized to unity gain at the peak frequencies $\Omega = \pm\Omega_0$. The bandwidth frequencies Ω_1 and Ω_2 will satisfy the bandwidth condition:

$$|H_a(\Omega)|^2 = \frac{\alpha^2\Omega^2}{(\Omega^2-\Omega_0^2)^2+\alpha^2\Omega^2} = G_B^2$$

It can be written as the quartic:

$$\Omega^4 - \left(2\Omega_0^2 + \frac{1-G_B^2}{G_B^2}\alpha^2\right)\Omega^2 + \Omega_0^4 = 0$$

which is similar to Eq. (12.3.9). Its two solutions Ω_1^2 and Ω_2^2 satisfy the conditions:

$$\Omega_1^2 + \Omega_2^2 = 2\Omega_0^2 + \frac{1-G_B^2}{G_B^2}\alpha^2$$

$$\Omega_1^2\Omega_2^2 = \Omega_0^4$$

from which we obtain $\Omega_1\Omega_2 = \Omega_0^2$ and

$$\Delta\Omega = \Omega_2 - \Omega_1 = \frac{\sqrt{1-G_B^2}}{G_B}\alpha \quad\Rightarrow\quad \boxed{\alpha = \frac{G_B}{\sqrt{1-G_B^2}}\Delta\Omega}$$

The relationship (12.3.15) between the analog and digital bandwidth remains the same. Therefore, we obtain the analog filter parameters $\{\alpha, \Omega_0\}$ by equations similar to (12.3.3):

$$\boxed{\Omega_0 = \tan\left(\frac{\omega_0}{2}\right), \quad \alpha = \frac{G_B}{\sqrt{1-G_B^2}}(1+\Omega_0^2)\tan\left(\frac{\Delta\omega}{2}\right)} \tag{12.3.17}$$

The digital filter is obtained by the bilinear transformation:

$$H(z) = H_a(s) = \left.\frac{\alpha s}{s^2+\alpha s+\Omega_0^2}\right|_{s=\frac{1-z^{-1}}{1+z^{-1}}}$$

which can be written in the form:

$$H(z) = \left(\frac{\beta}{1+\beta}\right)\frac{1-z^{-2}}{1-2\left(\frac{\cos\omega_0}{1+\beta}\right)z^{-1}+\left(\frac{1-\beta}{1+\beta}\right)z^{-2}} \tag{12.3.18}$$

where β is similar, but not identical, to that in Eq. (12.3.4):

$$\boxed{\beta = \frac{G_B}{\sqrt{1-G_B^2}}\tan\left(\frac{\Delta\omega}{2}\right)} \tag{12.3.19}$$

Defining the gain b as in Eq. (12.3.6)

$$\boxed{b = \frac{1}{1+\beta} = \frac{1}{1+\dfrac{G_B}{\sqrt{1-G_B^2}}\tan\left(\dfrac{\Delta\omega}{2}\right)}} \tag{12.3.20}$$

we may write $(1-\beta)/(1+\beta) = 2b-1$, and $\beta/(1+\beta) = 1-b$, and therefore, the coefficients of $H(z)$ can be expressed in terms of b as follows:

$$\boxed{H(z)=(1-b)\frac{1-z^{-2}}{1-2b\cos\omega_0\,z^{-1}+(2b-1)z^{-2}}}\qquad\text{(peak filter)}\qquad(12.3.21)$$

Note that the numerator vanishes at $z=\pm1$, that is, at DC and the Nyquist frequency. For the 3-dB widths, we have $G_B^2=1/2$, and the parameters β or b are the same as those of the notch filter:

$$\boxed{\beta=\tan\left(\frac{\Delta\omega}{2}\right),\quad b=\frac{1}{1+\beta}=\frac{1}{1+\tan(\Delta\omega/2)}}\qquad(12.3.22)$$

In this case, the notch and peak filters are *complementary* with transfer functions, frequency responses, and magnitude responses squared that add up to unity. For example, adding Eqs. (12.3.7) and (12.3.21), we have:

$$\boxed{H_{\text{notch}}(z)+H_{\text{peak}}(z)=1}\qquad(12.3.23)$$

Note, finally, that in the limit as ω_0 or Ω_0 tend to zero, the peaking filter will behave as a lowpass filter. The transfer functions $H(z)$ and $H_a(s)$ become in this case the lowpass transfer functions of the previous section. For example, setting $\Omega_0=0$ in Eq. (12.3.16), we have:

$$H_a(s)=\left.\frac{\alpha s}{s^2+\alpha s+\Omega_0^2}\right|_{\Omega_0=0}=\frac{\alpha s}{s^2+\alpha s}=\frac{\alpha}{s+\alpha}$$

Example 12.3.1: Design a peaking digital filter operating at a rate of 10 kHz that has a peak at 1.75 kHz and 3-dB width of 500 Hz. Then, redesign it such that 500 Hz represents its 10-dB width.

For the 3-dB width case, determine also the corresponding complementary notch filter.

Solution: The digital frequencies in radians per sample are:

$$\omega_0=\frac{2\pi f_0}{f_s}=\frac{2\pi\cdot1.75}{10}=0.35\pi,\quad\Delta\omega=\frac{2\pi\Delta f}{f_s}=\frac{2\pi\cdot0.5}{10}=0.1\pi$$

For the 3-dB case, we calculate the parameter, $\cos\omega_0=0.4540$, and:

$$\beta=\tan\left(\frac{\Delta\omega}{2}\right)=0.1584,\quad b=\frac{1}{1+\beta}=0.8633,\quad 1-b=0.1367$$

For the case $A_B=10$ dB, we have bandwidth gain $G_B^2=10^{-A_B/10}=0.1$. Then, we calculate:

$$\beta=\frac{G_B}{\sqrt{1-G_B^2}}\tan\left(\frac{\Delta\omega}{2}\right)=0.0528,\quad b=\frac{1}{1+\beta}=0.9499,\quad 1-b=0.0501$$

Inserting the above two sets of parameter values into Eq. (12.3.21), we obtain the transfer functions:

$$H(z)=\frac{0.1367(1-z^{-2})}{1-0.7838z^{-1}+0.7265z^{-2}},\qquad H(z)=\frac{0.0501(1-z^{-2})}{1-0.8624z^{-1}+0.8997z^{-2}}$$

The squared magnitude responses are shown in Fig. 12.3.3. They were calculated using the simpler analog expressions:

$$|H(\omega)|^2=|H_a(\Omega)|^2=\frac{\alpha^2\Omega^2}{(\Omega^2-\Omega_0^2)^2+\alpha^2\Omega^2}$$

Replacing $\Omega=\tan(\omega/2)=\tan(\pi f/f_s)$, we have in terms of the physical frequency f in Hz:

$$|H(f)|^2=\frac{\alpha^2\tan^2(\pi f/f_s)}{\left(\tan^2(\pi f/f_s)-\Omega_0^2\right)^2+\alpha^2\tan^2(\pi f/f_s)}$$

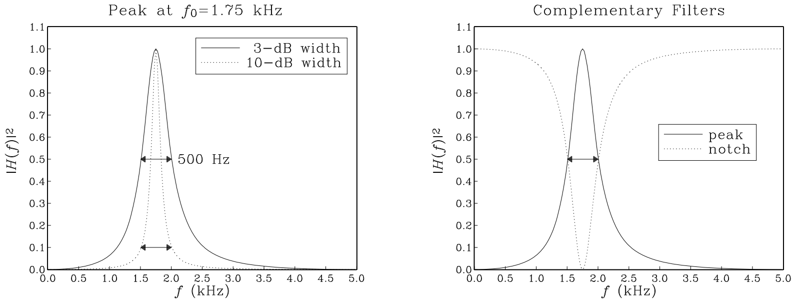


Fig. 12.3.3 Peaking and complementary notch filters.

The right graph of Fig. 12.3.3 shows the complementary peak and notch filters. The parameters β and b of the notch filter were already calculated above. Using Eq. (12.3.7), we find its transfer function:

$$H(z)=0.8633\frac{1-0.9080z^{-1}+z^{-2}}{1-0.7838z^{-1}+0.7265z^{-2}}$$

The zeros of the denominator, $1-0.7838z^{-1}+0.7265z^{-2}=0$, determine the poles of the transfer function. They are:

$$p,\,p^*=0.3919\pm j0.7569=0.8524e^{\pm j0.3479\pi}$$

The poles are not exactly at the desired frequency $\omega_0=0.35\pi$. Naive pole placement would have placed them there. For example, choosing the same radius $R=0.8524$, we would have in that case:

$$p,\,p^*=0.8524e^{\pm j0.35\pi}=0.3870\pm j0.7597$$

corresponding to the denominator polynomial $1-0.7739z^{-1}+0.7265z^{-2}$. The bilinear transformation method places the poles at appropriate locations to achieve the desired peak and width. □

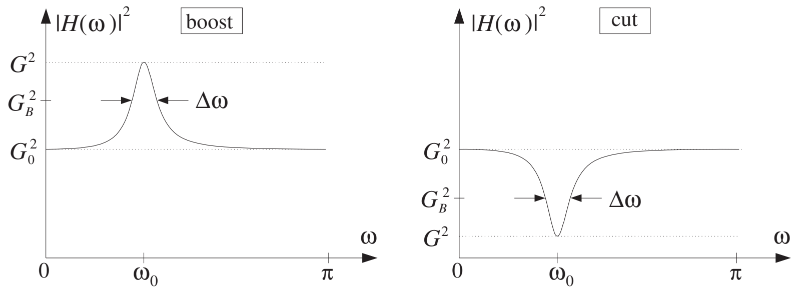


Fig. 12.4.1 Parametric EQ filter with boost or cut.

12.4 Parametric Equalizer Filters

Frequency equalization (EQ) is a common requirement in audio systems—analog, digital, home, car, public, or studio recording/mixing systems [272].

Graphic equalizers are the more common type, in which the audio band is divided into a *fixed* number of frequency bands, and the amount of equalization in each band is controlled by a bandpass filter whose gain can be varied up and down. The center frequencies of the bands and the filter 3-dB widths are fixed, and the user can vary only the overall gain in each band. Usually, second-order bandpass filters are adequate for audio applications.

A more flexible equalizer type is the *parametric equalizer*, in which all three filter parameters—gain, center frequency, and bandwidth—can be varied. Cascading four or five such filters together can achieve almost any desired equalization effect.

Figure 12.4.1 shows the frequency response of a typical second-order parametric equalizer. The specification parameters are: a *reference* gain G_0 (typically taken to be unity for cascadable filters), the center frequency ω_0 of the boost or cut, the filter gain G at ω_0, and a desired width $\Delta\omega$ at an appropriate bandwidth level G_B that lies between G_0 and G. As shown in Fig. 12.4.1, the relative gains must be chosen as follows, depending on whether we have a boost or a cut:

$$\boxed{\begin{array}{ll} G_0^2 < G_B^2 < G^2 & \text{(boost)} \\[1ex] G^2 < G_B^2 < G_0^2 & \text{(cut)} \end{array}} \tag{12.4.1}$$

The notch and peak filters of the previous section can be thought of as special cases of such a filter. The peaking filter corresponds to $G_0 = 0$, $G = 1$ and the notching filter to $G_0 = 1$, $G = 0$.

The definition of $\Delta\omega$ is arbitrary, and not without ambiguity. For example, we can define it to be the 3-dB width. But, what exactly do we mean by “3 dB”?

For the boosting case, we can take it to mean 3 dB *below the peak*, that is, choose $G_B^2 = G^2/2$; alternatively, we can take it to mean 3 dB *above the reference*, that is, $G_B^2 = 2G_0^2$. Moreover, because G_B^2 must lie between G_0^2 and G^2, the first alternative implies that $G_0^2 < G_B^2 = G^2/2$, or $2G_0^2 < G^2$, and the second $2G_0^2 = G_B^2 < G^2$. Thus, either alternative requires that $G^2 > 2G_0^2$, that is, the boost gain must be at least 3 dB higher than the reference. So, what do we do when $G_0^2 < G^2 < 2G_0^2$? In that case, any G_B^2 that lies in $G_0^2 < G_B^2 < G^2$ will do. A particularly interesting choice is to take it to be the arithmetic mean of the end values:

$$G_B^2 = \frac{G_0^2 + G^2}{2} \tag{12.4.2}$$

Another good choice is the geometric mean, $G_B^2 = GG_0$, corresponding to the arithmetic mean of the dB values of the endpoints [289,292] (see Problem 12.4.)

Similar ambiguities arise in the cutting case: we can take 3 dB to mean 3 dB *above the dip*, that is, $G_B^2 = 2G^2$, or, alternatively, we can take it to mean 3 dB *below the reference*, $G_B^2 = G_0^2/2$. Either alternative requires that $G^2 < G_0^2/2$, that is, the cut gain must be at least 3 dB below the reference. If $G_0^2/2 < G^2 < G_0^2$, we may again use the average (12.4.2). To summarize, some possible (but not necessary) choices for G_B^2 are as follows:

$$G_B^2 = \begin{cases} G^2/2, & \text{if } G^2 > 2G_0^2 \quad \text{(boost, alternative 1)} \\ 2G_0^2, & \text{if } G^2 > 2G_0^2 \quad \text{(boost, alternative 2)} \\ (G_0^2 + G^2)/2, & \text{if } G_0^2 < G^2 < 2G_0^2 \quad \text{(boost)} \\ & \\ 2G^2, & \text{if } G^2 < G_0^2/2 \quad \text{(cut, alternative 1)} \\ G_0^2/2, & \text{if } G^2 < G_0^2/2 \quad \text{(cut, alternative 2)} \\ (G_0^2 + G^2)/2, & \text{if } G_0^2/2 < G^2 < G_0^2 \quad \text{(cut)} \end{cases} \tag{12.4.3}$$

The filter design problem is to determine the filter’s transfer function in terms of the specification parameters: $\{G_0, G, G_B, \omega_0, \Delta\omega\}$. In this section, we present a simple design method based on the bilinear transformation, which is a variation of the methods in [283–292].

We define the parametric equalizer filter as the following *linear combination* of the notching and peaking filters of the previous section:

$$\boxed{H(z) = G_0 H_{\text{notch}}(z) + G H_{\text{peak}}(z)} \tag{12.4.4}$$

At ω_0 the gain is G, because the notch filter vanishes and the peak filter has unity gain. Similarly, at DC and the Nyquist frequency, the gain is equal to the reference G_0, because the notch is unity and the peak vanishes. From the complementarity property (12.3.23) it follows that when $G = G_0$ we have $H(z) = G_0$, that is, no equalization. Inserting the expressions (12.3.5) and (12.3.18) into Eq. (12.4.4), we obtain:

$$\boxed{H(z) = \frac{\left(\dfrac{G_0 + G\beta}{1+\beta}\right) - 2\left(\dfrac{G_0\cos\omega_0}{1+\beta}\right)z^{-1} + \left(\dfrac{G_0 - G\beta}{1+\beta}\right)z^{-2}}{1 - 2\left(\dfrac{\cos\omega_0}{1+\beta}\right)z^{-1} + \left(\dfrac{1-\beta}{1+\beta}\right)z^{-2}}} \tag{12.4.5}$$

The parameter β is a generalization of Eqs. (12.3.4) and (12.3.19) and is given by:

$$\boxed{\beta = \sqrt{\frac{G_B^2 - G_0^2}{G^2 - G_B^2}}\,\tan\left(\frac{\Delta\omega}{2}\right)} \tag{12.4.6}$$

Note that because of the assumed inequalities (12.4.1), the quantity under the square root is always positive. Also, for the special choice of G_B^2 of Eq. (12.4.2), the square root factor is *unity*. This choice (and those of Problem 12.4) allows a smooth transition to the no-equalization limit $G \to G_0$. Indeed, because β does not depend on the G's, setting $G = G_0$ in the numerator of Eq. (12.4.5) gives $H(z) = G_0$.

The design equations (12.4.5) and (12.4.6) can be justified as follows. Starting with the same linear combination of the analog versions of the notching and peaking filters given by Eqs. (12.3.2) and (12.3.16), we obtain the analog version of $H(z)$:

$$\boxed{H_a(s) = G_0 H_{\text{notch}}(s) + G H_{\text{peak}}(s) = \frac{G_0(s^2 + \Omega_0^2) + G\alpha s}{s^2 + \alpha s + \Omega_0^2}} \tag{12.4.7}$$

Then, the bandwidth condition $|H_a(\Omega)|^2 = G_B^2$ can be stated as:

$$|H_a(\Omega)|^2 = \frac{G_0^2(\Omega^2 - \Omega_0^2)^2 + G^2\alpha^2\Omega^2}{(\Omega^2 - \Omega_0^2)^2 + \alpha^2\Omega^2} = G_B^2 \tag{12.4.8}$$

It can be cast as the quartic equation:

$$\Omega^4 - \left(2\Omega_0^2 + \frac{G^2 - G_B^2}{G_B^2 - G_0^2}\alpha^2\right)\Omega^2 + \Omega_0^4 = 0$$

Proceeding as in the previous section and using the geometric-mean property $\Omega_1\Omega_2 = \Omega_0^2$ and Eq. (12.3.15), we find the relationship between the parameter α and the analog bandwidth $\Delta\Omega = \Omega_2 - \Omega_1$:

$$\alpha = \sqrt{\frac{G_B^2 - G_0^2}{G^2 - G_B^2}}\,\Delta\Omega = \sqrt{\frac{G_B^2 - G_0^2}{G^2 - G_B^2}}\,(1 + \Omega_0^2)\tan\left(\frac{\Delta\omega}{2}\right) \equiv (1 + \Omega_0^2)\beta$$

This defines β. Then, the bilinear transformation of Eq. (12.4.7) leads to Eq. (12.4.5).

Example 12.4.1: Design the following six parametric EQ filters operating at 10 kHz rate that satisfy the specifications: $G_0 = 1$ and

(a) center frequency of 1.75 kHz, 9-dB boost gain, and 3-dB width of 500 Hz defined to be 3 dB below the peak (alternative 1).

(b) same as (a), except the width is 3 dB above the reference (alternative 2).

(c) center frequency of 3 kHz, 9-dB cut gain, and 3-dB width of 1 kHz defined to be 3 dB above the dip (alternative 1).

(d) same as (c), except the width is 3 dB below the reference (alternative 2).

(e) center frequency of 1.75 kHz, 2-dB boost, and 500 Hz width defined by Eq. (12.4.2).

(f) center frequency of 3 kHz, 2-dB cut, and 1 kHz width defined by Eq. (12.4.2).

Solution: The boost examples (a), (b), and (e) have digital frequency and width:

$$\omega_0 = \frac{2\pi\cdot 1.75}{10} = 0.35\pi, \quad \Delta\omega = \frac{2\pi\cdot 0.5}{10} = 0.1\pi$$

and the cut examples (c), (d), and (f) have:

$$\omega_0 = \frac{2\pi\cdot 3}{10} = 0.6\pi, \quad \Delta\omega = \frac{2\pi\cdot 1}{10} = 0.2\pi$$

Normally, a "3-dB" change means a change by a factor of 2 in the magnitude square. Here, for plotting purposes, we take "3 dB" to mean literally 3 dB, which corresponds to changes by $10^{3/10} = 1.9953 \simeq 2$. Therefore, in case (a), a boost gain of 9 dB above the reference G_0 corresponds to the value:

$$G = 10^{9/20}G_0 = 2.8184, \qquad G^2 = 7.9433 \quad \text{(instead of 8)}$$

The bandwidth level is defined to be 3 dB below the peak, that is, $A_B = 9 - 3 = 6$ dB, and therefore:

$$G_B = 10^{-3/20}G = 10^{6/20}G_0 = 1.9953, \quad G_B^2 = 3.9811$$

With these values of $\{G_0, G, G_B, \omega_0, \Delta\omega\}$, we calculate the value of β from Eq. (12.4.6):

$$\beta = \sqrt{\frac{G_B^2 - G_0^2}{G^2 - G_B^2}}\,\tan\left(\frac{\Delta\omega}{2}\right) = \sqrt{\frac{3.9811 - 1}{7.9433 - 3.9811}}\,\tan\left(\frac{0.1\pi}{2}\right) = 0.1374$$

We calculate also $\cos\omega_0 = \cos(0.35\pi) = 0.4540$. The transfer function of filter (a), obtained from Eq. (12.4.5), is then:

$$H_a(z) = \frac{1.2196 - 0.7983z^{-1} + 0.5388z^{-2}}{1 - 0.7983z^{-1} + 0.7584z^{-2}}$$

For filter (b), the width is defined to be 3 dB above the reference, that is, $A_B = 3$ dB:

$$G_B = 10^{3/20}G_0 = 10^{3/20} = 1.4125, \quad G_B^2 = 1.9953$$

From Eq. (12.4.6), we calculate $\beta = 0.0648$, and from Eq. (12.4.5) the filter:

$$H_b(z) = \frac{1.1106 - 0.8527z^{-1} + 0.7677z^{-2}}{1 - 0.8527z^{-1} + 0.8783z^{-2}}$$

For filter (c), we have a 9-dB cut gain, that is, 9 dB below the reference:

$$G = 10^{-9/20}G_0 = 0.3548, \quad G^2 = 0.1259$$

and the bandwidth level is 3 dB above this dip, that is, $A_B = -9 + 3 = -6$ dB:

$$G_B = 10^{3/20}G = 10^{-6/20}G_0 = 0.5012, \quad G_B^2 = 0.2512$$

Then, we calculate $\cos\omega_0 = \cos(0.6\pi) = -0.3090$ and $\beta = 0.7943$, and the transfer function:

$$H_c(z) = \frac{0.7144 + 0.3444z^{-1} + 0.4002z^{-2}}{1 + 0.3444z^{-1} + 0.1146z^{-2}}$$

For filter (d), the width is 3 dB below the reference, that is, $A_B = 0 - 3 = -3$ dB:

$$G_B = 10^{-3/20}G_0 = 0.7079, \quad G_B^2 = 0.5012$$

We calculate $\beta = 0.3746$ and the transfer function:

$$H_d(z) = \frac{0.8242 + 0.4496z^{-1} + 0.6308z^{-2}}{1 + 0.4496z^{-1} + 0.4550z^{-2}}$$

The four filters (a)–(d) are shown in the left graph of Fig. 12.4.2. The magnitude responses are plotted in dB, that is, $20\log_{10}|H(\omega)|$. The reference level $G_0 = 1$ corresponds to 0 dB. Notice the horizontal grid lines at 6 dB, 3 dB, −3 dB, and −6 dB, whose intersections with the magnitude responses define the corresponding bandwidths $\Delta\omega$.

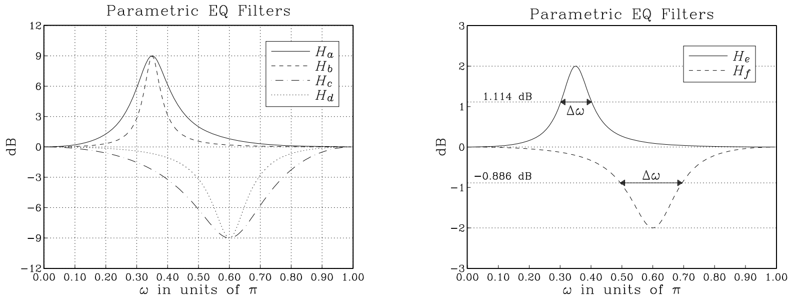


Fig. 12.4.2 Parametric EQ filters of Example 12.4.1.

For filter (e), the boost gain is 2 dB and therefore, the bandwidth level cannot be chosen to be 3 dB below the peak or 3 dB above the reference. We must use an intermediate level between 0 and 2 dB. In particular, we may use Eq. (12.4.2). Thus, we calculate the parameters:

$$G = 10^{2/20}G_0 = 1.2589, \quad G_B^2 = \frac{G_0^2 + G^2}{2} = 1.2924$$

corresponding to $A_B = 10\log_{10}(G_B^2) = 1.114$ dB. The square root factor in the definition of β is unity, therefore, we calculate:

$$\beta = \tan\left(\frac{\Delta\omega}{2}\right) = \tan\left(\frac{0.1\pi}{2}\right) = 0.1584$$

and the transfer function:

$$H_e(z) = \frac{1.0354 - 0.7838z^{-1} + 0.6911z^{-2}}{1 - 0.7838z^{-1} + 0.7265z^{-2}}$$

Finally, in case (f), we have a 2-dB cut, giving the values:

$$G = 10^{-2/20}G_0 = 0.7943, \quad G_B^2 = \frac{G_0^2 + G^2}{2} = 0.8155$$

corresponding to $A_B = 10\log_{10}(G_B^2) = -0.886$ dB. The parameter β is now $\beta = \tan(\Delta\omega/2) = \tan(0.2\pi/2) = 0.3249$, resulting in the transfer function:

$$H_f(z) = \frac{0.9496 + 0.4665z^{-1} + 0.5600z^{-2}}{1 + 0.4665z^{-1} + 0.5095z^{-2}}$$

Filters (e) and (f) are shown in the right graph of Fig. 12.4.2. The vertical scales are expanded compared to those of the left graph. The horizontal lines defining the bandwidth levels $A_B = 1.114$ dB and $A_B = -0.886$ dB are also shown.

In practice, parametric EQ filters for audio have cut and boost gains that vary typically from −18 dB to 18 dB with respect to the reference gain. □

Example 12.4.2: Instead of specifying the parameters $\{\omega_0, \Delta\omega\}$, it is often convenient to specify either one or both of the corner frequencies $\{\omega_1, \omega_2\}$ that define the width $\Delta\omega = \omega_2 - \omega_1$.

Design four parametric EQ filters that have a 2.5-dB cut and bandwidth defined at 1 dB below the reference, and have center or corner frequencies as follows:

(a) Center frequency $\omega_0 = 0.6\pi$ and right corner $\omega_2 = 0.7\pi$. Determine also the left corner ω_1 and the bandwidth $\Delta\omega$.

(b) Center frequency $\omega_0 = 0.6\pi$ and left corner $\omega_1 = 0.5\pi$. Determine also the right corner ω_2 and the bandwidth $\Delta\omega$.

(c) Left and right corner frequencies $\omega_1 = 0.5\pi$ and $\omega_2 = 0.7\pi$. Determine also the center frequency ω_0.

(d) Compare the above to the standard design that has $\omega_0 = 0.6\pi$, and $\Delta\omega = 0.2\pi$. Determine the values of ω_1, ω_2.

Solution: Assuming $G_0 = 1$, the cut and bandwidth gains are:

$$G = 10^{-2.5/20} = 0.7499, \quad G_B = 10^{-1/20} = 0.8913$$

Note that G_B was chosen arbitrarily in this example and not according to Eq. (12.4.2). For case (a), we are given ω_0 and ω_2. Under the bilinear transformation they map to the values:

$$\Omega_0 = \tan(\omega_0/2) = 1.3764, \quad \Omega_2 = \tan(\omega_2/2) = 1.9626$$

Using the geometric-mean property (12.3.11), we may solve for ω_1:

$$\tan\left(\frac{\omega_1}{2}\right) = \Omega_1 = \frac{\Omega_0^2}{\Omega_2} = 0.9653 \quad \Rightarrow \quad \omega_1 = 0.4887\pi$$

Thus, the bandwidth is $\Delta\omega = \omega_2 - \omega_1 = 0.2113\pi$. The design equations (12.4.5) and (12.4.6) give then $\beta = 0.3244$ and the transfer function:

$$H_a(z) = \frac{0.9387 + 0.4666z^{-1} + 0.5713z^{-2}}{1 + 0.4666z^{-1} + 0.5101z^{-2}}$$

For case (b), we are given ω_0 and ω_1 and calculate ω_2:

$$\Omega_1 = \tan\left(\frac{\omega_1}{2}\right) = 1, \quad \tan\left(\frac{\omega_2}{2}\right) = \Omega_2 = \frac{\Omega_0^2}{\Omega_1} = 1.8944 \;\Rightarrow\; \omega_2 = 0.6908\pi$$

where Ω_0 was as in case (a). The width is $\Delta\omega = \omega_2 - \omega_1 = 0.1908\pi$. Then, we find $\beta = 0.2910$ and the transfer function:

$$H_b(z) = \frac{0.9436 + 0.4787z^{-1} + 0.6056z^{-2}}{1 + 0.4787z^{-1} + 0.5492z^{-2}}$$

The magnitude responses (in dB) of cases (a) and (b) are shown in the left graph of Fig. 12.4.3. The bandwidths are defined by the intersection of the horizontal grid line at −1 dB and the curves.

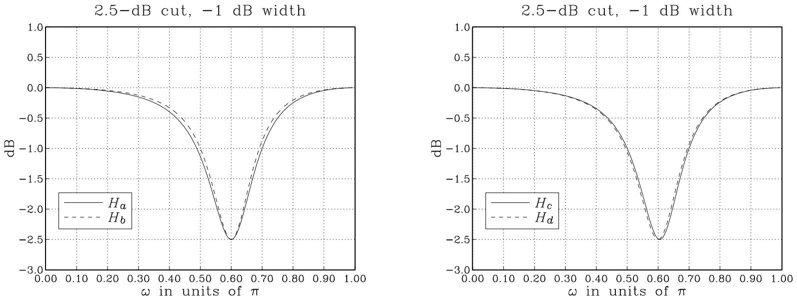


Fig. 12.4.3 Parametric EQ filters.

For case (c), we are given ω_1 and ω_2. Their bilinear transformations are:

$$\Omega_1 = \tan(\omega_1/2) = 1, \quad \Omega_2 = \tan(\omega_2/2) = 1.9626$$

The center frequency is computed from:

$$\tan(\omega_0/2) = \Omega_0 = \sqrt{\Omega_1\Omega_2} = 1.4009 \quad\Rightarrow\quad \omega_0 = 0.6053\pi$$

Using the calculated ω_0 and the width $\Delta\omega = \omega_2 - \omega_1 = 0.2\pi$, we find $\cos\omega_0 = -0.3249$, $\beta = 0.3059$, and the transfer function:

$$H_c(z) = \frac{0.9414 + 0.4976z^{-1} + 0.5901z^{-2}}{1 + 0.4976z^{-1} + 0.5315z^{-2}}$$

Finally, in the standard case (d), we start with ω_0 and $\Delta\omega$. We find $\cos\omega_0 = -0.3090$, $\beta = 0.3059$, and the transfer function:

$$H_d(z) = \frac{0.9414 + 0.4732z^{-1} + 0.5901z^{-2}}{1 + 0.4732z^{-1} + 0.5315z^{-2}}$$

With $\Omega_0 = \tan(\omega_0/2) = 1.3764$, the exact values of ω_1 and ω_2 are obtained by solving the system of equations:

$$\Omega_1\Omega_2 = \Omega_0^2 = 1.8944, \quad \Omega_2 - \Omega_1 = \Delta\Omega = (1 + \Omega_0^2)\tan(\Delta\omega/2) = 0.9404$$

which have positive solutions $\Omega_1 = 0.9843$, $\Omega_2 = 1.9247$. It follows that

$$\omega_1 = 2\arctan(\Omega_1) = 0.494951\pi, \quad \omega_2 = 2\arctan(\Omega_2) = 0.694951\pi$$

where as expected $\Delta\omega = \omega_2 - \omega_1 = 0.2\pi$. The magnitude responses are shown in the right graph of Fig. 12.4.3. Note that cases (c) and (d) have the same β because their widths $\Delta\omega$ are the same. But, the values of $\cos\omega_0$ are different, resulting in different values for the coefficients of z^{-1}; the other coefficients are the same. □

12.4.1 Shelving Equalizers

In addition to parametric equalizers with variable center frequencies ω_0, in audio applications we also need lowpass and highpass filters, referred to as “shelving” filters, with adjustable gains and cutoff frequencies. Such filters can be obtained from Eq. (12.4.5) by replacing $\omega_0 = 0$ for the lowpass case and $\omega_0 = \pi$ for the highpass one.

In the lowpass limit, $\omega_0 = 0$, we have $\cos\omega_0 = 1$ and the numerator and denominator of Eq. (12.4.5) develop a common factor $(1 - z^{-1})$. Canceling this factor, we obtain the *lowpass shelving filter*:

$$\boxed{H_{\text{LP}}(z) = \frac{\left(\dfrac{G_0 + G\beta}{1+\beta}\right) - \left(\dfrac{G_0 - G\beta}{1+\beta}\right)z^{-1}}{1 - \left(\dfrac{1-\beta}{1+\beta}\right)z^{-1}}} \tag{12.4.9}$$

where β is still given by Eq. (12.4.6), but with $\Delta\omega$ replaced by the filter's cutoff frequency ω_c and with G_B replaced by the defining level G_c of the cutoff frequency:

$$\boxed{\beta = \sqrt{\frac{G_c^2 - G_0^2}{G^2 - G_c^2}}\,\tan\left(\frac{\omega_c}{2}\right)} \tag{12.4.10}$$

In the highpass limit, $\omega_0 = \pi$, we have $\cos\omega_0 = -1$ and the numerator and denominator of Eq. (12.4.5) have a common factor $(1 + z^{-1})$. Canceling it, we obtain the *highpass shelving filter*:

$$\boxed{H_{\mathrm{HP}}(z)=\frac{\left(\dfrac{G_0+G\beta}{1+\beta}\right)+\left(\dfrac{G_0-G\beta}{1+\beta}\right)z^{-1}}{1+\left(\dfrac{1-\beta}{1+\beta}\right)z^{-1}}} \tag{12.4.11}$$

It can also be obtained from Eq. (12.4.9) by the replacement $z\to -z$. The parameter β is obtained from Eq. (12.4.6) by the replacements $G_B\to G_c$ and $\Delta\omega\to\pi-\omega_c$. The latter is necessary because $\Delta\omega$ is measured from the center frequency $\omega_0=\pi$, whereas ω_c is measured from the origin $\omega=0$. Noting that $\tan\big((\pi-\omega_c)/2\big)=\cot(\omega_c/2)$, we have:

$$\boxed{\beta=\sqrt{\frac{G_c^2-G_0^2}{G^2-G_c^2}}\,\cot\left(\frac{\omega_c}{2}\right)} \tag{12.4.12}$$

For both the lowpass and highpass cases, the filter specifications are the parameters $\{G_0,G,G_c,\omega_c\}$. They must satisfy Eq. (12.4.1) for boosting or cutting. Figure 12.4.4 depicts these specifications. Some possible choices for G_c^2 are still given by Eq. (12.4.3).

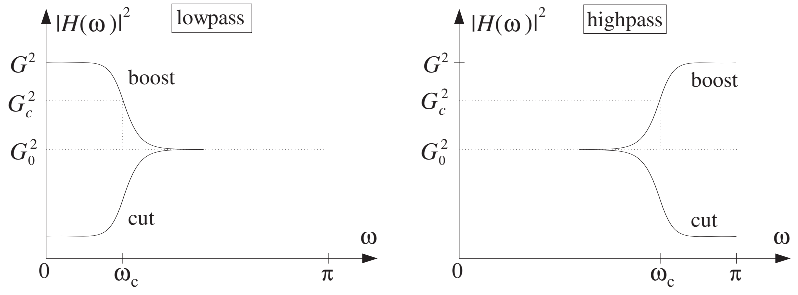


Fig. 12.4.4 Lowpass and highpass shelving filters with boost or cut.

The limiting forms of the corresponding analog filter (12.4.7) can be obtained by taking the appropriate limits in the variable $\Omega_0=\tan(\omega_0/2)$. For the lowpass case, we have the limit $\Omega_0\to 0$ and for the highpass case, the limit $\Omega_0\to\infty$. We must also replace $\alpha=(1+\Omega_0^2)\beta$ before taking these limits.

Taking the limits, we obtain the analog filters whose bilinear transformations are the shelving filters Eq. (12.4.9) and (12.4.11):

$$H_{\mathrm{LP}}(s)=\frac{G_0 s+G\beta}{s+\beta}\,,\quad H_{\mathrm{HP}}(s)=\frac{G_0+G\beta s}{1+\beta s} \tag{12.4.13}$$

The special case $G_0=0$, $G=1$ was considered in Section 12.2. In that section, the lowpass case corresponded to $\alpha=\beta$ and the highpass to $\alpha=1/\beta$.

Setting $\Omega=\Omega_c=\tan(\omega_c/2)$, the following bandwidth conditions may be solved for β, resulting into the design equations (12.4.10) and (12.4.12):

$$|H_{\mathrm{LP}}(\Omega)|^2=\frac{G_0^2\Omega^2+G^2\beta^2}{\Omega^2+\beta^2}=G_c^2\,,\quad |H_{\mathrm{HP}}(\Omega)|^2=\frac{G_0^2+G^2\beta^2\Omega^2}{1+\beta^2\Omega^2}=G_c^2$$

12.4.2 Equalizers with Prescribed Nyquist-Frequency Gain

An alternative type of second-order digital parametric equalizer has been proposed in [296], whose frequency response matches closely that of its analog counterpart throughout the Nyquist interval and does not suffer from the prewarping effect of the bilinear transformation near the Nyquist frequency. Fig. 12.4.5 compares the analog and digital equalizers for the conventional and the alternative designs.

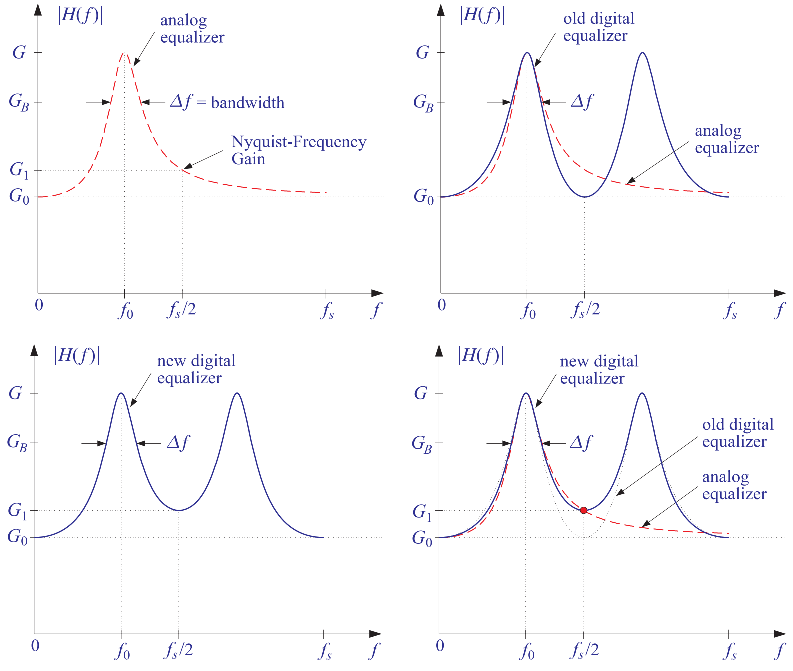


Fig. 12.4.5 Top figures: conventional analog and digital equalizers, digital design has $G_1=G_0$ at $f_s/2$. Bottom figures: New digital equalizer matches the Nyquist-frequency gain of the corresponding analog equalizer.

It is defined by the following expressions, where G_1 is the prescribed gain at the Nyquist frequency $f_1=f_s/2$, and G_B is the desired bandwidth gain,

$$
\begin{aligned}
\Omega_0 &= \tan\left(\frac{\omega_0}{2}\right) \\
\Delta\Omega &= \left(1 + \sqrt{\frac{G_B^2 - G_0^2}{G_B^2 - G_1^2}}\sqrt{\frac{G^2 - G_1^2}{G^2 - G_0^2}}\,\Omega_0{}^2\right)\tan\left(\frac{\Delta\omega}{2}\right)
\end{aligned}
\tag{12.4.14}
$$

$$
W^2 = \sqrt{\frac{G^2 - G_1^2}{G^2 - G_0^2}}\,\Omega_0{}^2\,, \qquad A = \sqrt{\frac{C + D}{|G^2 - G_B^2|}}\,, \qquad B = \sqrt{\frac{G^2 C + G_B^2 D}{|G^2 - G_B^2|}}
\tag{12.4.15}
$$

$$
\begin{aligned}
C &= (\Delta\Omega)^2 \,|G_B^2 - G_1^2| - 2W^2\left(|G_B^2 - G_0 G_1| - \sqrt{(G_B^2 - G_0^2)(G_B^2 - G_1^2)}\right) \\
D &= 2W^2\left(|G^2 - G_0 G_1| - \sqrt{(G^2 - G_0^2)(G^2 - G_1^2)}\right)
\end{aligned}
\tag{12.4.16}
$$

$$
\boxed{H(z) = \frac{\left(\dfrac{G_1 + G_0 W^2 + B}{1 + W^2 + A}\right) - 2\left(\dfrac{G_1 - G_0 W^2}{1 + W^2 + A}\right)z^{-1} + \left(\dfrac{G_1 + G_0 W^2 - B}{1 + W^2 + A}\right)z^{-2}}{1 - 2\left(\dfrac{1 - W^2}{1 + W^2 + A}\right)z^{-1} + \left(\dfrac{1 + W^2 - A}{1 + W^2 + A}\right)z^{-2}}}
\tag{12.4.17}
$$

To summarize, given the set of digital filter specifications $\{\omega_0, \Delta\omega, G_0, G_1, G, G_B\}$, use Eqs. (12.4.14) to calculate the prewarped analog frequencies. Then, use Eqs. (12.4.15) and (12.4.16) to calculate the parameters $\{A, B, W^2\}$, from which the digital filter coefficients of Eq. (12.4.17) are determined.

In the special case $G_1 = G_0$, we recover the conventional parametrric equalizer of Sec. 12.4. The Nyquist-frequency gain G_1 can be chosen arbitrarily. However, for the digital filter to match the corresponding (physical) analog filter as much as possible, the gain G_1 must match the analog filter's gain at $f_s/2$. This leads [296] to the following definition of G_1,

$$
G_1^2 = \frac{G_0^2(\omega_0^2 - \pi^2)^2 + G^2\pi^2(\Delta\omega)^2\,\dfrac{G_B^2 - G_0^2}{G^2 - G_B^2}}{(\omega_0^2 - \pi^2)^2 + \pi^2(\Delta\omega)^2\,\dfrac{G_B^2 - G_0^2}{G^2 - G_B^2}}
\tag{12.4.18}
$$

The design can be implemented by the following MATLAB function `peq.m`,

```
% peq.m - Parametric EQ with matching gain at Nyquist frequency
%
% Usage:  [b, a, G1] = peq(G0, G, GB, w0, Dw)
%
% G0 = reference gain at DC
% G  = boost/cut gain
% GB = bandwidth gain
```

```
%
% w0 = center frequency in rads/sample
% Dw = bandwidth in rads/sample
%
% b  = [b0, b1, b2] = numerator coefficients
% a  = [1,  a1, a2] = denominator coefficients
% G1 = Nyquist-frequency gain
```

Its inputs are the gains G_0, G, G_B in absolute units, and the digital frequencies ω_0, $\Delta\omega$ in units of rads/sample. Its outputs are the Nyquist-frequency gain G_1 given by Eq. (12.4.18), and the numerator and denominator coefficient vectors $\mathbf{b} = [b_0, b_1, b_2]$, $\mathbf{a} = [1, a_1, a_2]$, defining the transfer function of Eq. (12.4.17).

Additional details on this method, including derivations, bandwidth definitions, and several design examples, can be found in [296].

12.5 Comb Filters

The lowpass and highpass shelving filters of the previous section can be turned into *periodic comb or notch filters* with adjustable gains and peak widths. This can be accomplished by the replicating transformation of Eq. (15.11.7), that is, $z \to z^D$, which shrinks the frequency response by a factor of D and replicates it D times such that D copies of it fit into the Nyquist interval. Under this transformation, the lowpass filter $H_{\text{LP}}(z)$ of Eq. (12.4.9) becomes the comb filter:

$$
\boxed{H(z) = \frac{b - cz^{-D}}{1 - az^{-D}}}\,, \quad a = \frac{1 - \beta}{1 + \beta}\,,\; b = \frac{G_0 + G\beta}{1 + \beta}\,,\; c = \frac{G_0 - G\beta}{1 + \beta}
\tag{12.5.1}
$$

This transfer function can also be obtained from the analog lowpass shelving filter $H_{\text{LP}}(s)$ of Eq. (12.4.13) by the generalized bilinear transformation:

$$
s = \frac{1 - z^{-D}}{1 + z^{-D}}\,, \quad \Omega = \tan\left(\frac{\omega D}{2}\right) = \tan\left(\frac{\pi f D}{f_s}\right)
\tag{12.5.2}
$$

The DC peak of the lowpass filter $H_{\text{LP}}(z)$ has full width $2\omega_c$, counting also its symmetric negative-frequency side. Under D-fold replication, the symmetric DC peak will be shrunk in width by a factor of D and replicated D times, with replicas centered at the Dth root-of-unity frequencies $\omega_k = 2\pi k/D$, $k = 0, 1, \dots, D-1$. Thus, the full width $\Delta\omega$ of the individual replicas can be obtained by the substitution $2\omega_c/D \to \Delta\omega$, or $\omega_c \to D\Delta\omega/2$.

Making this substitution in Eq. (12.4.10) and replacing the cutoff frequency gain G_c by the bandwidth gain G_B, the filter parameter β can be expressed as:

$$
\boxed{\beta = \sqrt{\frac{G_B^2 - G_0^2}{G^2 - G_B^2}}\,\tan\left(\frac{D\,\Delta\omega}{4}\right)}
\tag{12.5.3}
$$

The resulting comb filter can be thought of as a “comb equalizer” with variable gain and peak width. The boost or cut choices of the gain G given in Eq. (12.4.1) will correspond to either a peaking or a notching periodic comb filter.

The periodic notch and comb filters of Section 15.11 are special cases of the general comb filter (12.5.1). The notch filter of Eq. (15.11.3) is obtained in the limit $G_0 = 1$, $G = 0$, and the comb filter of Eq. (15.11.10) in the limit $G_0 = 0$, $G = 1$. In both cases, the bandwidth gain can be chosen to be $G_B^2 = 1/2$, that is, 3 dB.

The replicating transformation $z \to z^D$ can also be applied to the highpass shelving filter $H_{\text{HP}}(z)$ of Eq. (12.4.11), resulting in the comb filter:

$$H(z) = \frac{b + cz^{-D}}{1 + az^{-D}} \tag{12.5.4}$$

The sign change of the coefficients causes the peaks to shift by π/D, placing them between the peaks of the comb (12.5.1), that is, at the odd-multiple frequencies $\omega_k = (2k+1)\pi/D$, $k = 0, 1, \dots, D-1$. The parameter β is still calculated from Eq. (12.5.3).

Example 12.5.1: Design a peaking comb filter of period 10, reference gain of 0 dB, peak gain of 9 dB, bandwidth gain of 3 dB (above the reference), and bandwidth $\Delta\omega = 0.025\pi$ rads/sample.

Then, design a notching comb filter with dip gain of -12 dB and having the same period, reference gain, and bandwidth as the peaking filer. The bandwidth is defined to be 3 dB below the reference.

Then, redesign both of the above peaking and notching comb filters such that their peaks or dips are shifted to lie exactly between the peaks of the previous filters.

Solution: For the peaking filter, we have:

$$G_0 = 1, \quad G = 10^{9/20}G_0 = 2.8184, \quad G_B = 10^{3/20}G_0 = 1.4125$$

and for the notching filter:

$$G_0 = 1, \quad G = 10^{-12/20}G_0 = 0.2512, \quad G_B = 10^{-3/20}G_0 = 0.7079$$

With $D = 10$ and $\Delta\omega = 0.025\pi$, we find the values: $\beta = 0.0814$ for the peaking filter and $\beta = 0.2123$ for the notching filter. Then, the transfer functions are obtained from Eq. (12.5.1):

$$H_{\text{peak}}(z) = \frac{1.1368 - 0.7127z^{-10}}{1 - 0.8495z^{-10}}, \quad H_{\text{notch}}(z) = \frac{0.8689 - 0.7809z^{-10}}{1 - 0.6498z^{-10}}$$

Their peaks/dips are at the multiples $\omega_k = 2\pi k/D = 0.2\pi k$, $k = 0, 1, \dots, 9$. The filters with the shifted peaks are obtained by changing the sign of z^{-10}:

$$H_{\text{peak}}(z) = \frac{1.1368 + 0.7127z^{-10}}{1 + 0.8495z^{-10}}, \quad H_{\text{notch}}(z) = \frac{0.8689 + 0.7809z^{-10}}{1 + 0.6498z^{-10}}$$

The magnitude responses of the four filters are shown in Fig. 12.5.1, plotted in dB, that is, $20\log_{10}|H(\omega)|$, over one complete Nyquist interval, $0 \le \omega \le 2\pi$. □

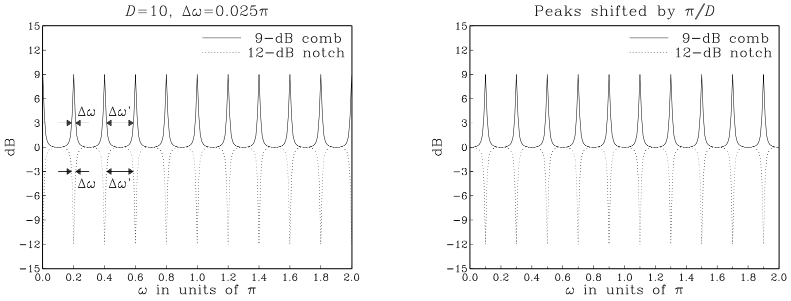


Fig. 12.5.1 Periodic peaking and notching comb filters.

Figure 12.5.1 also shows the peak width $\Delta\omega$ as well as the quantity $\Delta\omega'$, which is a measure of the *separation* of two successive peaks at the bandwidth level. Because of symmetry, the D equal peak widths and the D equal separations must make up the full 2π Nyquist interval. Thus, we have the condition:

$$D\Delta\omega + D\Delta\omega' = 2\pi \quad \Rightarrow \quad \Delta\omega + \Delta\omega' = \frac{2\pi}{D} \tag{12.5.5}$$

Decreasing the peak width increases the separation and vice versa. The maximum possible value of $\Delta\omega$ corresponds to the case when $\Delta\omega' = 0$, that is, zero separation between peaks. This gives $\Delta\omega_{\text{max}} = 2\pi/D$. However, a more useful practical limit is when $\Delta\omega \le \Delta\omega'$, which causes the peaks to be narrower than their separation. This condition requires that $2\Delta\omega \le \Delta\omega + \Delta\omega' = 2\pi/D$, and gives the maximum for $\Delta\omega$:

$$\Delta\omega \le \frac{\pi}{D} \quad \Rightarrow \quad \Delta f \le \frac{f_s}{2D} \tag{12.5.6}$$

This maximum was also discussed in Section 15.11.

12.6 Higher-Order Filters

The first- and second-order designs of the above sections are adequate in some applications such as audio equalization, but are too limited when we need filters with very sharp cutoff specifications. Higher-order filters can achieve such sharp cutoffs, but at the price of increasing the filter complexity, that is, the filter order.

Figure 12.6.1 shows the specifications of a typical lowpass filter and its analog equivalent obtained by the bilinear transformation. The specification parameters are the four numbers $\{f_{\text{pass}}, f_{\text{stop}}, A_{\text{pass}}, A_{\text{stop}}\}$, that is, the *passband and stopband frequencies* and the desired *passband and stopband attenuations* in dB.

Within the passband range $0 \le f \le f_{\text{pass}}$, the filter's attenuation is required to be *less* than A_{pass} decibels. And, within the stopband $f_{\text{stop}} \le f \le f_s/2$, it is required to be *greater* than A_{stop} decibels. Thus, the quantity A_{pass} is the *maximum* attenuation

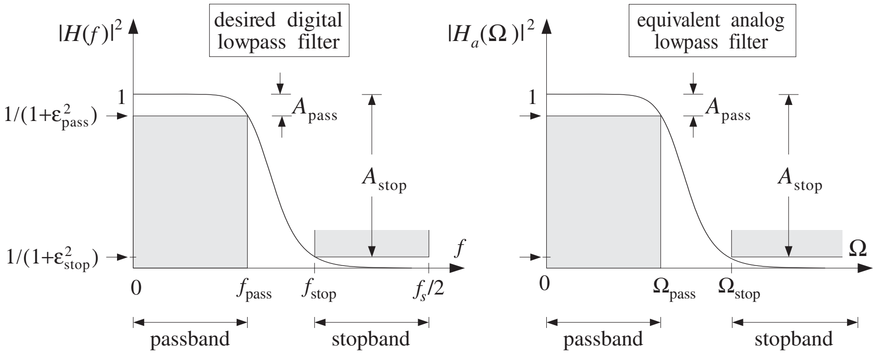


Fig. 12.6.1 Lowpass digital filter and its analog equivalent.

that can be tolerated in the passband and A_{stop} the *minimum* attenuation that must be achieved in the stopband.

The filter can be made into a *better* lowpass filter in three ways: (1) decreasing A_{pass} so that the passband becomes flatter, (2) increasing A_{stop} so that the stopband becomes deeper, and (3) moving f_{stop} closer to f_{pass} so that the *transition region* between passband and stopband becomes narrower. Thus, by appropriate choice of the specification parameters, the filter can be made as close to an ideal lowpass filter as desired.

Assuming the filter's magnitude response squared $|H(f)|^2$ is normalized to unity at DC, we can express the specification requirements as the following conditions on the filter's attenuation response in dB, defined as $A(f) = -10\log_{10}|H(f)|^2$:

$$\boxed{\begin{aligned} 0 \le A(f) \le A_{\text{pass}}\,, &\quad \text{for} \quad 0 \le f \le f_{\text{pass}} \\ A(f) \ge A_{\text{stop}}\,, &\quad \text{for} \quad f_{\text{stop}} \le f \le f_s/2 \end{aligned}} \tag{12.6.1}$$

Equivalently, in absolute units, the design specifications are:

$$\begin{aligned} 1 \ge |H(f)|^2 \ge \frac{1}{1+\varepsilon^2_{\text{pass}}}\,, &\quad \text{for} \quad 0 \le f \le f_{\text{pass}} \\ |H(f)|^2 \le \frac{1}{1+\varepsilon^2_{\text{stop}}}\,, &\quad \text{for} \quad f_{\text{stop}} \le f \le f_s/2 \end{aligned} \tag{12.6.2}$$

where $\{\varepsilon_{\text{pass}}, \varepsilon_{\text{stop}}\}$ are defined in terms of $\{A_{\text{pass}}, A_{\text{stop}}\}$ as follows:

$$\begin{aligned} |H(f_{\text{pass}})|^2 &= \frac{1}{1+\varepsilon^2_{\text{pass}}} = 10^{-A_{\text{pass}}/10}\,, \\ |H(f_{\text{stop}})|^2 &= \frac{1}{1+\varepsilon^2_{\text{stop}}} = 10^{-A_{\text{stop}}/10}\,, \end{aligned} \tag{12.6.3}$$

The quantities $\{\varepsilon_{\text{pass}}, \varepsilon_{\text{stop}}\}$ control the depths of the passband and stopband. They can be written in the equivalent forms:

$$\boxed{\begin{aligned} \varepsilon_{\text{pass}} &= \sqrt{10^{A_{\text{pass}}/10} - 1} \\ \varepsilon_{\text{stop}} &= \sqrt{10^{A_{\text{stop}}/10} - 1} \end{aligned}} \quad \Leftrightarrow \quad \boxed{\begin{aligned} A_{\text{pass}} &= 10\log_{10}(1+\varepsilon^2_{\text{pass}}) \\ A_{\text{stop}} &= 10\log_{10}(1+\varepsilon^2_{\text{stop}}) \end{aligned}} \tag{12.6.4}$$

The specifications of the equivalent analog filter are $\{\Omega_{\text{pass}}, \Omega_{\text{stop}}, A_{\text{pass}}, A_{\text{stop}}\}$, or, $\{\Omega_{\text{pass}}, \Omega_{\text{stop}}, \varepsilon_{\text{pass}}, \varepsilon_{\text{stop}}\}$, where the analog frequencies are obtained by prewarping the digital frequencies:

$$\boxed{\Omega_{\text{pass}} = \tan\left(\frac{\omega_{\text{pass}}}{2}\right), \quad \Omega_{\text{stop}} = \tan\left(\frac{\omega_{\text{stop}}}{2}\right)} \tag{12.6.5}$$

where

$$\boxed{\omega_{\text{pass}} = \frac{2\pi f_{\text{pass}}}{f_s}, \quad \omega_{\text{stop}} = \frac{2\pi f_{\text{stop}}}{f_s}} \tag{12.6.6}$$

The parameters $\{\varepsilon_{\text{pass}}, \varepsilon_{\text{stop}}\}$ are useful in the design of both Butterworth and Chebyshev filters. In the next section, we begin with the Butterworth case.

12.7 Analog Lowpass Butterworth Filters

Analog lowpass Butterworth filters are characterized by just *two* parameters: the filter order N and the 3-dB normalization frequency Ω_0. Their magnitude response is simply:

$$\boxed{|H(\Omega)|^2 = \frac{1}{1+\left(\dfrac{\Omega}{\Omega_0}\right)^{2N}}} \tag{12.7.1}$$

and the corresponding attenuation in decibels:

$$\boxed{A(\Omega) = -10\log_{10}|H(\Omega)|^2 = 10\log_{10}\left[1+\left(\frac{\Omega}{\Omega_0}\right)^{2N}\right]} \tag{12.7.2}$$

Note that, as N increases for fixed Ω_0, the filter becomes a better lowpass filter. At $\Omega = \Omega_0$, the magnitude response is $|H(\Omega_0)|^2 = 1/2$, or, 3-dB attenuation $A(\Omega_0) = 3$ dB. The two filter parameters $\{N, \Omega_0\}$ can be determined from the given specifications $\{\Omega_{\text{pass}}, \Omega_{\text{stop}}, A_{\text{pass}}, A_{\text{stop}}\}$ by requiring the conditions:

$$\begin{aligned} A(\Omega_{\text{pass}}) &= 10\log_{10}\left[1+\left(\frac{\Omega_{\text{pass}}}{\Omega_0}\right)^{2N}\right] = A_{\text{pass}} = 10\log_{10}(1+\varepsilon^2_{\text{pass}}) \\ A(\Omega_{\text{stop}}) &= 10\log_{10}\left[1+\left(\frac{\Omega_{\text{stop}}}{\Omega_0}\right)^{2N}\right] = A_{\text{stop}} = 10\log_{10}(1+\varepsilon^2_{\text{stop}}) \end{aligned}$$

Because of the monotonicity of the magnitude response, these conditions are equivalent to the passband/stopband range conditions (12.6.1). To solve them for N and Ω_0, we rewrite them in the form:

$$\begin{aligned}\left(\frac{\Omega_{\text{pass}}}{\Omega_0}\right)^{2N} &= 10^{A_{\text{pass}}/10}-1=\varepsilon_{\text{pass}}^2\\ \left(\frac{\Omega_{\text{stop}}}{\Omega_0}\right)^{2N} &= 10^{A_{\text{stop}}/10}-1=\varepsilon_{\text{stop}}^2\end{aligned}\tag{12.7.3}$$

Taking square roots and dividing, we get an equation for N:

$$\left(\frac{\Omega_{\text{stop}}}{\Omega_{\text{pass}}}\right)^{N}=\frac{\varepsilon_{\text{stop}}}{\varepsilon_{\text{pass}}}=\sqrt{\frac{10^{A_{\text{stop}}/10}-1}{10^{A_{\text{pass}}/10}-1}}$$

with exact solution:

$$\boxed{N_{\text{exact}}=\frac{\ln(\varepsilon_{\text{stop}}/\varepsilon_{\text{pass}})}{\ln(\Omega_{\text{stop}}/\Omega_{\text{pass}})}=\frac{\ln(e)}{\ln(w)}}\tag{12.7.4}$$

where we defined the stopband to passband ratios:

$$\boxed{e=\frac{\varepsilon_{\text{stop}}}{\varepsilon_{\text{pass}}}=\sqrt{\frac{10^{A_{\text{stop}}/10}-1}{10^{A_{\text{pass}}/10}-1}},\quad w=\frac{\Omega_{\text{stop}}}{\Omega_{\text{pass}}}}\tag{12.7.5}$$

Since N must be an integer, we choose it to be the *next* integer above N_{exact}, that is,

$$N=\lceil N_{\text{exact}}\rceil\tag{12.7.6}$$

Because N is slightly increased from its exact value, the resulting filter will be slightly better than required. But, because N is different from N_{exact}, we can no longer satisfy simultaneously both of Eqs. (12.7.3). So we choose to satisfy the first one exactly. This determines Ω_0 as follows:

$$\boxed{\Omega_0=\frac{\Omega_{\text{pass}}}{(10^{A_{\text{pass}}/10}-1)^{1/2N}}=\frac{\Omega_{\text{pass}}}{\varepsilon_{\text{pass}}^{1/N}}}\tag{12.7.7}$$

With these values of N and Ω_0, the stopband specification is more than satisfied, that is, the actual stopband attenuation will be now $A(\Omega_{\text{stop}})>A_{\text{stop}}$. In summary, given $\{\Omega_{\text{pass}},\Omega_{\text{stop}},A_{\text{pass}},A_{\text{stop}}\}$, we solve Eqs. (12.7.4)–(12.7.7) to get the filter parameters N and Ω_0. We note also that we may rewrite Eq. (12.7.1) in terms of the passband parameters; replacing Ω_0 by Eq. (12.7.7), we have

$$|H(\Omega)|^2=\frac{1}{1+\left(\dfrac{\Omega}{\Omega_0}\right)^{2N}}=\frac{1}{1+\varepsilon_{\text{pass}}^2\left(\dfrac{\Omega}{\Omega_{\text{pass}}}\right)^{2N}}\tag{12.7.8}$$

An alternative design can be obtained by matching the stopband specification exactly, resulting in a slightly better passband, that is, $A(\Omega_{\text{stop}})=A_{\text{stop}}$ and $A(\Omega_{\text{pass}})<A_{\text{pass}}$. The 3-dB frequency Ω_0 is now computed from the second of Eqs. (12.7.3):

$$\Omega_0=\frac{\Omega_{\text{stop}}}{(10^{A_{\text{stop}}/10}-1)^{1/2N}}=\frac{\Omega_{\text{stop}}}{\varepsilon_{\text{stop}}^{1/N}}$$

In this case, Eq. (12.7.1) can be written in terms of the stopband parameters:

$$|H(\Omega)|^2=\frac{1}{1+\varepsilon_{\text{stop}}^2\left(\dfrac{\Omega}{\Omega_{\text{stop}}}\right)^{2N}}=\frac{\left(\dfrac{\Omega_{\text{stop}}}{\Omega}\right)^{2N}}{\left(\dfrac{\Omega_{\text{stop}}}{\Omega}\right)^{2N}+\varepsilon_{\text{stop}}^2}\tag{12.7.9}$$

We will see in Section 12.12 that the expressions (12.7.8) and (12.7.9) generalize to the Chebyshev type 1 and type 2 filters.

The analog Butterworth transfer function $H(s)$ can be constructed from the knowledge of $\{N,\Omega_0\}$ by the method of *spectral factorization*, as described below. Using $s=j\Omega$ and noting that $\big(H(\Omega)\big)^*=H^*(-\Omega)$, we may write Eq. (12.7.1) in terms of the variable s[†]

$$H(s)H^*(-s)=\frac{1}{1+\left(\dfrac{s}{j\Omega_0}\right)^{2N}}=\frac{1}{1+(-1)^N\left(\dfrac{s}{\Omega_0}\right)^{2N}}$$

Setting $H(s)=\dfrac{1}{D(s)}$, we have

$$\boxed{D(s)D^*(-s)=1+(-1)^N\left(\frac{s}{\Omega_0}\right)^{2N}}\tag{12.7.10}$$

Because the right-hand side is a polynomial of degree $2N$ in s, $D(s)$ will be a polynomial of degree N. There exist 2^N *different* polynomials $D(s)$ of degree N satisfying Eq. (12.7.10). But, among them, there is a *unique* one that has *all* its zeros in the *left-hand* s-plane. This is the one we want, because then the transfer function $H(s)=1/D(s)$ will be *stable and causal*. To find $D(s)$, we first determine all the $2N$ roots of Eq. (12.7.10) and then choose those that lie in the left-hand s-plane. The $2N$ solutions of

$$1+(-1)^N\left(\frac{s}{\Omega_0}\right)^{2N}=0\qquad\Rightarrow\qquad s^{2N}=(-1)^{N-1}\Omega_0{}^{2N}$$

are given by

$$\boxed{s_i=\Omega_0e^{j\theta_i},\quad \theta_i=\frac{\pi}{2N}(N-1+2i)},\quad i=1,2,\dots,N,\dots,2N\tag{12.7.11}$$

The index i is chosen such that the first N of the s_i lie in the left-hand s-plane, that is, $\pi/2<\theta_i<3\pi/2$ for $i=1,2,\dots,N$. Because $|s_i|=\Omega_0$, all of the zeros lie on a circle of radius Ω_0, called the *Butterworth circle* and shown in Fig. 12.7.1.

It is evident from Eq. (12.7.11) that the s_i can be paired in complex conjugate pairs; that is, $s_N=s_1^*$, $s_{N-1}=s_2^*$, and so on. If N is *even*, say $N=2K$, then there are exactly K conjugate pairs, namely, $\{s_i,s_i^*\}$, $i=1,2,\dots,K$. In this case, $D(s)$ will factor into second-order sections as follows:

[†] The notation $H^*(-s)$ denotes complex conjugation of the filter coefficients and replacement of s by $-s$, for example, $H^*(-s)=\sum a_n^*(-s)^n$ if $H(s)=\sum a_ns^n$.

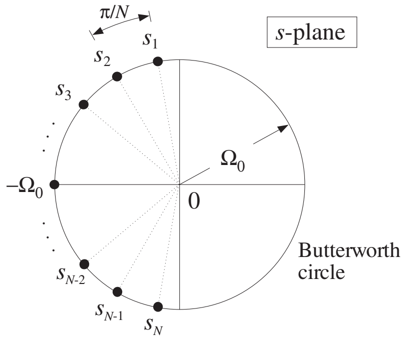


Fig. 12.7.1 Butterworth filter poles lie on Butterworth circle.

$$D(s)=D_1(s)D_2(s)\cdots D_K(s) \tag{12.7.12}$$

where

$$D_i(s)=\left(1-\frac{s}{s_i}\right)\left(1-\frac{s}{s_i^*}\right),\qquad i=1,2,\dots,K \tag{12.7.13}$$

On the other hand, if N is *odd*, say $N=2K+1$, there will be K conjugate pairs and one additional zero that cannot be paired and must necessarily be real-valued. That zero must lie in the left-hand s-plane and on the Butterworth circle; thus, it must be the point $s=-\Omega_0$. The polynomial $D(s)$ factors now as:

$$D(s)=D_0(s)D_1(s)D_2(s)\cdots D_K(s),\quad \text{where}\quad D_0(s)=\left(1+\frac{s}{\Omega_0}\right)$$

The remaining factors $D_i(s)$ are the same as in Eq. (12.7.13). They can be rewritten as factors with *real* coefficients as follows. Inserting $s_i=\Omega_0 e^{j\theta_i}$ into Eq. (12.7.13), we find for $i=1,2,\dots,K$:

$$D_i(s)=\left(1-\frac{s}{s_i}\right)\left(1-\frac{s}{s_i^*}\right)=1-2\frac{s}{\Omega_0}\cos\theta_i+\frac{s^2}{\Omega_0^2} \tag{12.7.14}$$

Inserting these factors into the Butterworth analog transfer function $H(s)=1/D(s)$, we can express it as a cascade of second-order sections:

$$\boxed{H(s)=H_0(s)H_1(s)H_2(s)\cdots H_K(s)} \tag{12.7.15}$$

where

$$\begin{aligned} H_0(s)&=\begin{cases} 1, & \text{if } N=2K\\ \dfrac{1}{1+\dfrac{s}{\Omega_0}}, & \text{if } N=2K+1\end{cases}\\ H_i(s)&=\frac{1}{1-2\dfrac{s}{\Omega_0}\cos\theta_i+\dfrac{s^2}{\Omega_0^2}},\qquad i=1,2,\dots,K\end{aligned} \tag{12.7.16}$$

Example 12.7.1: The Butterworth polynomials $D(s)$ of orders 1–7 and unity 3-dB normalization frequency $\Omega_0=1$ are shown in Table 12.7.1. For other values of Ω_0, s must be replaced by s/Ω_0 in each table entry.

The coefficients of s of the second-order sections are the cosine factors, $-2\cos\theta_i$, of Eq. (12.7.14). For example, in the case $N=7$, we have $K=3$ and the three θ's are calculated from Eq. (12.7.11):

$$\begin{aligned}\theta_i&=\frac{\pi}{14}(6+2i)=\frac{8\pi}{14},\ \frac{10\pi}{14},\ \frac{12\pi}{14},\quad \text{for}\quad i=1,2,3\\ -2\cos\theta_i&=0.4450,\ 1.2470,\ 1.8019\end{aligned}$$

The corresponding Butterworth filters $H(s)$ of orders 1–7 are obtained as the inverses of the table entries. □

N	K	$\theta_1,\theta_2,\dots,\theta_K$	$D(s)$
1	0		$(1+s)$
2	1	$\frac{3\pi}{4}$	$(1+1.4142s+s^2)$
3	1	$\frac{4\pi}{6}$	$(1+s)(1+s+s^2)$
4	2	$\frac{5\pi}{8},\frac{7\pi}{8}$	$(1+0.7654s+s^2)(1+1.8478s+s^2)$
5	2	$\frac{6\pi}{10},\frac{8\pi}{10}$	$(1+s)(1+0.6180s+s^2)(1+1.6180s+s^2)$
6	3	$\frac{7\pi}{12},\frac{9\pi}{12},\frac{11\pi}{12}$	$(1+0.5176s+s^2)(1+1.4142s+s^2)(1+1.9319s+s^2)$
7	3	$\frac{8\pi}{14},\frac{10\pi}{14},\frac{12\pi}{14}$	$(1+s)(1+0.4450s+s^2)(1+1.2470s+s^2)(1+1.8019s+s^2)$

Table 12.7.1 Butterworth polynomials.

Example 12.7.2: Determine the 2^N possible Nth degree polynomials $D(s)$ satisfying Eq. (12.7.10), for the cases $N=2$ and $N=3$. Take $\Omega_0=1$.

Solution: For $N=2$, we must find all second-degree polynomials that satisfy Eq. (12.7.10), $D(s)D^*(-s)=1+(-1)^2s^4$. They are:

$$
\begin{array}{lcl}
D(s) = 1+\sqrt{2}s+s^2 & & D^*(-s) = 1-\sqrt{2}s+s^2 \\
D(s) = 1-\sqrt{2}s+s^2 & \Rightarrow & D^*(-s) = 1+\sqrt{2}s+s^2 \\
D(s) = 1+js^2 & & D^*(-s) = 1-js^2 \\
D(s) = 1-js^2 & & D^*(-s) = 1+js^2
\end{array}
$$

Only the first one has all of its zeros in the left-hand s-plane. Similarly, for $N = 3$ the $2^3 = 8$ different third-degree polynomials $D(s)$ are:

$$
\begin{array}{lcl}
D(s) = (1+s)(1+s+s^2) & & D^*(-s) = (1-s)(1-s+s^2) \\
D(s) = (1+s)(1-s+s^2) & & D^*(-s) = (1-s)(1+s+s^2) \\
D(s) = (1+s)(1-s^2e^{2j\pi/3}) & & D^*(-s) = (1-s)(1-s^2e^{-2j\pi/3}) \\
D(s) = (1+s)(1-s^2e^{-2j\pi/3}) & \Rightarrow & D^*(-s) = (1-s)(1-s^2e^{2j\pi/3}) \\
D(s) = (1-s)(1-s^2e^{-2j\pi/3}) & & D^*(-s) = (1+s)(1-s^2e^{2j\pi/3}) \\
D(s) = (1-s)(1-s^2e^{2j\pi/3}) & & D^*(-s) = (1+s)(1-s^2e^{-2j\pi/3}) \\
D(s) = (1-s)(1+s+s^2) & & D^*(-s) = (1+s)(1-s+s^2) \\
D(s) = (1-s)(1-s+s^2) & & D^*(-s) = (1+s)(1+s+s^2)
\end{array}
$$

They all satisfy $D(s)D^*(-s) = 1 + (-1)^3 s^6$ but, only the first one has its zeros in the left-hand s-plane.

Note also that not all solutions of Eq. (12.7.10) have real coefficients. If we restrict our search to those with real coefficients (pairing the zeros in conjugate pairs), then there are 2^K such polynomials $D(s)$ if $N = 2K$, and 2^{K+1} if $N = 2K+1$. □

12.8 Digital Lowpass Filters

Under the bilinear transformation, the lowpass analog filter will be transformed into a lowpass digital filter. Each analog second-order section will be transformed into a second-order section of the digital filter, as follows:

$$
H_i(z) = \frac{1}{1 - 2\dfrac{s}{\Omega_0}\cos\theta_i + \dfrac{s^2}{\Omega_0^2}}\Bigg|_{s=\frac{1-z^{-1}}{1+z^{-1}}} = \frac{G_i(1+z^{-1})^2}{1+a_{i1}z^{-1}+a_{i2}z^{-2}} \tag{12.8.1}
$$

where the filter coefficients G_i, a_{i1}, a_{i2} are easily found to be:

$$
\begin{aligned}
G_i &= \frac{\Omega_0^2}{1 - 2\Omega_0\cos\theta_i + \Omega_0{}^2} \\
a_{i1} &= \frac{2(\Omega_0^2 - 1)}{1 - 2\Omega_0\cos\theta_i + \Omega_0{}^2} \\
a_{i2} &= \frac{1 + 2\Omega_0\cos\theta_i + \Omega_0^2}{1 - 2\Omega_0\cos\theta_i + \Omega_0{}^2}
\end{aligned} \tag{12.8.2}
$$

for $i = 1, 2, \dots, K$. If N is odd, then there is also a first-order section:

$$
H_0(z) = \frac{1}{1 + \dfrac{s}{\Omega_0}}\Bigg|_{s=\frac{1-z^{-1}}{1+z^{-1}}} = \frac{G_0(1+z^{-1})}{1+a_{01}z^{-1}} \tag{12.8.3}
$$

where

$$
G_0 = \frac{\Omega_0}{\Omega_0+1}, \qquad a_{01} = \frac{\Omega_0-1}{\Omega_0+1} \tag{12.8.4}
$$

If N is even, we may set $H_0(z) = 1$. The overall transfer function of the designed lowpass digital filter is given by:

$$
\boxed{H(z) = H_0(z)H_1(z)H_2(z)\cdots H_K(z)} \tag{12.8.5}
$$

with the factors given by Eqs. (12.8.1)–(12.8.4). Note that the 3-dB frequency f_0 in Hz is related to the Butterworth parameter Ω_0 by

$$
\Omega_0 = \tan\left(\frac{\omega_0}{2}\right) = \tan\left(\frac{\pi f_0}{f_s}\right) \quad \Rightarrow \quad \boxed{f_0 = \frac{f_s}{\pi}\arctan(\Omega_0)} \tag{12.8.6}
$$

Note that the filter sections have zeros at $z = -1$, that is, the Nyquist frequency $\omega = \pi$. Setting $\Omega = \tan(\omega/2)$, the magnitude response of the designed digital filter can be expressed simply via Eq. (12.7.1), as follows:

$$
|H(\omega)|^2 = \frac{1}{1 + (\Omega/\Omega_0)^{2N}} = \frac{1}{1 + \big(\tan(\omega/2)/\Omega_0\big)^{2N}} \tag{12.8.7}
$$

Note also that each second-order section has unity gain at zero frequency, $f = 0$, $\omega = 0$, or $z = 1$. Indeed, setting $z = 1$ in Eq. (12.8.1), we obtain the following condition, which can be verified from the definitions (12.8.2):

$$
\frac{4G_i}{1 + a_{i1} + a_{i2}} = 1 \qquad \text{and} \qquad \frac{2G_0}{1 + a_{01}} = 1
$$

In summary, the design steps for a lowpass digital filter with given specifications $\{f_{\text{pass}}, f_{\text{stop}}, A_{\text{pass}}, A_{\text{stop}}\}$ are:

1. Calculate the digital frequencies $\{\omega_{\text{pass}}, \omega_{\text{stop}}\}$ and the corresponding prewarped versions $\{\Omega_{\text{pass}}, \Omega_{\text{stop}}\}$ from Eqs. (12.6.6) and (12.6.5).

2. Calculate the order N and 3-dB frequency Ω_0 of the equivalent lowpass analog Butterworth filter based on the transformed specifications $\{\Omega_{\text{pass}}, \Omega_{\text{stop}}, A_{\text{pass}}, A_{\text{stop}}\}$ by Eqs. (12.7.4)–(12.7.7).

3. The transfer function of the desired lowpass digital filter is then obtained from Eq. (12.8.5), where the SOS coefficients are calculated from Eqs. (12.8.2) and (12.8.4).

Example 12.8.1: Using the bilinear transformation and a lowpass analog Butterworth prototype, design a lowpass digital filter operating at a rate of 20 kHz and having passband extending to 4 kHz with maximum passband attenuation of 0.5 dB, and stopband starting at 5 kHz with a minimum stopband attenuation of 10 dB.

Then, redesign it such that its magnitude response satisfies $1 \geq |H(f)|^2 \geq 0.98$ in the passband, and $|H(f)|^2 \leq 0.02$ in the stopband.

Solution: The digital frequencies in radians per sample are:

$$\omega_{\text{pass}} = \frac{2\pi f_{\text{pass}}}{f_s} = \frac{2\pi \cdot 4}{20} = 0.4\pi, \quad \omega_{\text{stop}} = \frac{2\pi f_{\text{stop}}}{f_s} = \frac{2\pi \cdot 5}{20} = 0.5\pi$$

and their prewarped versions:

$$\Omega_{\text{pass}} = \tan\left(\frac{\omega_{\text{pass}}}{2}\right) = 0.7265, \quad \Omega_{\text{stop}} = \tan\left(\frac{\omega_{\text{stop}}}{2}\right) = 1$$

Eq. (12.6.4) can be used with $A_{\text{pass}} = 0.5$ dB and $A_{\text{stop}} = 10$ dB to calculate the parameters $\{\varepsilon_{\text{pass}}, \varepsilon_{\text{stop}}\}$:

$$\varepsilon_{\text{pass}} = \sqrt{10^{A_{\text{pass}}/10} - 1} = \sqrt{10^{0.5/10} - 1} = 0.3493$$

$$\varepsilon_{\text{stop}} = \sqrt{10^{A_{\text{stop}}/10} - 1} = \sqrt{10^{10/10} - 1} = 3$$

Then, Eq. (12.7.4) gives:

$$N_{\text{exact}} = \frac{\ln(e)}{\ln(w)} = \frac{\ln(\varepsilon_{\text{stop}}/\varepsilon_{\text{pass}})}{\ln(\Omega_{\text{stop}}/\Omega_{\text{pass}})} = \frac{\ln(3/0.3493)}{\ln(1/0.7265)} = 6.73 \quad \Rightarrow \quad N = 7$$

Thus, there is one first-order section $H_0(z)$ and three second-order sections. Eq. (12.7.7) gives for Ω_0 and its value in Hz:

$$\Omega_0 = \frac{\Omega_{\text{pass}}}{\varepsilon_{\text{pass}}^{1/N}} = \frac{0.7265}{(0.3493)^{1/7}} = 0.8443$$

$$f_0 = \frac{f_s}{\pi}\arctan(\Omega_0) = \frac{20}{\pi}\arctan(0.8443) = 4.4640 \text{ kHz}$$

The Butterworth angles θ_1, θ_2, θ_3 were calculated in Example 12.7.1. The SOS coefficients are calculated from Eqs. (12.8.4) and (12.8.2):

i	G_i	a_{i1}	a_{i2}
0	0.4578	−0.0844	
1	0.3413	−0.2749	0.6402
2	0.2578	−0.2076	0.2386
3	0.2204	−0.1775	0.0592

resulting in the transfer function:

$$\begin{aligned} H(z) &= H_0(z)H_1(z)H_2(z)H_3(z) \\ &= \frac{0.4578(1+z^{-1})}{1-0.0844z^{-1}} \cdot \frac{0.3413(1+z^{-1})^2}{1-0.2749z^{-1}+0.6402z^{-2}} \\ &\quad \cdot \frac{0.2578(1+z^{-1})^2}{1-0.2076z^{-1}+0.2386z^{-2}} \cdot \frac{0.2204(1+z^{-1})^2}{1-0.1775z^{-1}+0.0592z^{-2}} \end{aligned}$$

It can be implemented in the time domain by the routines `cas` or `ccas`. The left graph of Fig. 12.8.1 shows the magnitude response squared, $|H(f)|^2$. The brick-wall specifications and the 3-dB line intersecting the response at $f = f_0$ are shown on the graph. The magnitude response was calculated using the simpler formula Eq. (12.8.7), with ω expressed in kHz, $\omega = 2\pi f/f_s$:

$$|H(f)|^2 = \frac{1}{1 + \big(\tan(\pi f/f_s)/\Omega_0\big)^{2N}} = \frac{1}{1 + \big(\tan(\pi f/20)/0.8443\big)^{14}}$$

The passband attenuation in absolute units is $10^{-0.5/10} = 0.89125$ and the stopband attenuation $10^{-10/10} = 0.1$. Note that the actual stopband attenuation at $f = f_{\text{stop}} = 5$ kHz is slightly better than required, that is, $A(f_{\text{stop}}) = 10.68$ dB.

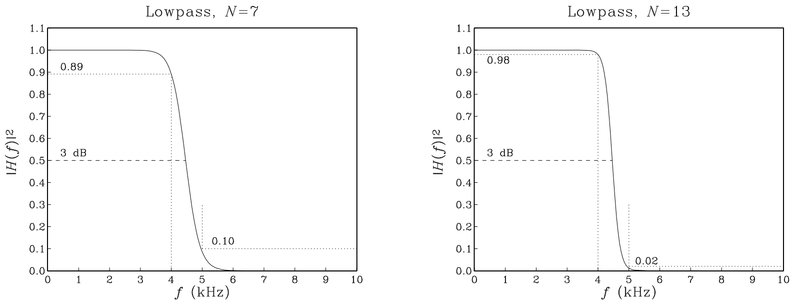


Fig. 12.8.1 Digital lowpass Butterworth filters.

The second filter has more stringent specifications. The desired passband attenuation is $A_{\text{pass}} = -10\log_{10}(0.98) = 0.0877$ dB, and the stopband attenuation $A_{\text{stop}} = -10\log_{10}(0.02) = 16.9897$ dB. With these values, we find the design parameters $\{\varepsilon_{\text{pass}}, \varepsilon_{\text{stop}}\} = \{0.1429, 7\}$ and:

$$N_{\text{exact}} = 12.18, \quad N = 13, \quad \Omega_0 = 0.8439, \quad f_0 = 4.4622 \text{ kHz}$$

The digital filter will have one first-order and six second-order sections. The SOS coefficients were calculated with the MATLAB function `lhbutt.m` of Appendix C:

i	G_i	a_{i1}	a_{i2}
0	0.4577	−0.0847	
1	0.3717	−0.3006	0.7876
2	0.3082	−0.2492	0.4820
3	0.2666	−0.2156	0.2821
4	0.2393	−0.1935	0.1508
5	0.2221	−0.1796	0.0679
6	0.2125	−0.1718	0.0219

Its magnitude response is shown in the right graph of Fig. 12.8.1. As is always the case, making the specifications more stringent results in higher order N. □

12.9 Digital Highpass Filters

There are two possible approaches one can follow to design a highpass digital filter with the bilinear transformation: One is to use the transformation (12.1.3) to map the given specifications onto the specifications of an equivalent *highpass* analog filter. The other is to use the *highpass* version of the bilinear transformation given in Eq. (12.1.5) to map the given highpass specifications onto equivalent analog *lowpass* specifications.

The first approach was used in the design of the first-order highpass filters of Section 12.2. Here, we will follow the second method, which is more convenient for high-order designs because we can use the lowpass Butterworth design we developed already. The mapping of the highpass specifications to the equivalent analog lowpass ones is depicted in Fig. 12.9.1.

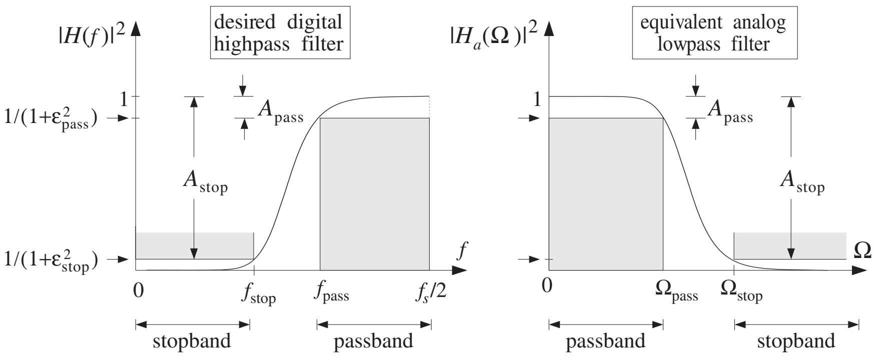


Fig. 12.9.1 Highpass digital filter and its analog lowpass equivalent.

The mapping is accomplished by the highpass version of the bilinear transformation, given in Eqs. (12.1.5) and (12.1.6):

$$s=\frac{1+z^{-1}}{1-z^{-1}}\,,\qquad \Omega=-\cot\left(\frac{\omega}{2}\right),\qquad \omega=\frac{2\pi f}{f_s} \tag{12.9.1}$$

It maps the point $z=-1$ to $s=0$, or equivalently, the center of the passband of the highpass filter at $\omega=\pi$ to the center of the passband of the lowpass filter at $\Omega=0$. The prewarped versions of the passband and stopband frequencies are computed as follows:

$$\begin{aligned}\Omega_{\rm pass}&=\cot\left(\frac{\omega_{\rm pass}}{2}\right)=\cot\left(\frac{\pi f_{\rm pass}}{f_s}\right)\\ \Omega_{\rm stop}&=\cot\left(\frac{\omega_{\rm stop}}{2}\right)=\cot\left(\frac{\pi f_{\rm stop}}{f_s}\right)\end{aligned} \tag{12.9.2}$$

According to Eq. (12.9.1), we should have used $\Omega_{\rm pass}=-\cot(\omega_{\rm pass}/2)$. However, as far as the determination of the parameters N and Ω_0 is concerned, it does not matter whether we use positive or negative signs because we are working only with the magnitude response of the analog filter, which is even as a function of Ω.

Using Eqs. (12.7.4)–(12.7.7), we determine the parameters N and Ω_0, and the corresponding analog filter sections given by Eq. (12.7.15). Under the highpass bilinear transformation of Eq. (12.9.1), each SOS of the analog filter will be transformed into an SOS of the digital filter, as follows:

$$H_i(z)=\frac{1}{1-2\dfrac{s}{\Omega_0}\cos\theta_i+\dfrac{s^2}{\Omega_0^2}}\Bigg|_{s=\frac{1+z^{-1}}{1-z^{-1}}}=\frac{G_i(1-z^{-1})^2}{1+a_{i1}z^{-1}+a_{i2}z^{-2}} \tag{12.9.3}$$

where the filter coefficients G_i, a_{i1}, a_{i2} are easily found to be

$$\begin{aligned}G_i&=\frac{\Omega_0^2}{1-2\Omega_0\cos\theta_i+{\Omega_0}^2}\\ a_{i1}&=-\frac{2(\Omega_0^2-1)}{1-2\Omega_0\cos\theta_i+{\Omega_0}^2}\\ a_{i2}&=\frac{1+2\Omega_0\cos\theta_i+\Omega_0^2}{1-2\Omega_0\cos\theta_i+{\Omega_0}^2}\end{aligned} \tag{12.9.4}$$

for $i=1,2,\dots,K$. If N is odd, then there is also a first-order section given by

$$H_0(z)=\frac{1}{1+\dfrac{s}{\Omega_0}}\Bigg|_{s=\frac{1+z^{-1}}{1-z^{-1}}}=\frac{G_0(1-z^{-1})}{1+a_{01}z^{-1}} \tag{12.9.5}$$

where

$$G_0=\frac{\Omega_0}{\Omega_0+1}\,,\qquad a_{01}=-\frac{\Omega_0-1}{\Omega_0+1} \tag{12.9.6}$$

If N is even, we may set $H_0(z)=1$. The overall transfer function of the designed highpass digital filter will be given by

$$\boxed{H(z)=H_0(z)H_1(z)H_2(z)\cdots H_K(z)} \tag{12.9.7}$$

with the factors given by Eqs. (12.9.3–12.9.6). The 3-dB frequency f_0 of the designed filter may be calculated from:

$$\Omega_0 = \cot\left(\frac{\omega_0}{2}\right) = \cot\left(\frac{\pi f_0}{f_s}\right) \quad\Rightarrow\quad \boxed{f_0 = \frac{f_s}{\pi}\arctan\left(\frac{1}{\Omega_0}\right)}$$

and the magnitude response from:

$$|H(\omega)|^2 = \frac{1}{1 + \left(\cot(\omega/2)/\Omega_0\right)^{2N}}$$

Note the similarities and differences between the highpass and lowpass cases: The coefficients G_i and a_{i2} are the same, but a_{i1} has reverse sign. Also, the numerator of the SOS is now $(1 - z^{-1})^2$ instead of $(1 + z^{-1})^2$, resulting in a zero at $z = 1$ or $\omega = 0$. These changes are easily understood by noting that the lowpass bilinear transformation (12.1.3) becomes the highpass one given by Eq. (12.9.1) under the substitution $z \to -z$.

Example 12.9.1: Using the bilinear transformation and a lowpass analog Butterworth prototype, design a highpass digital filter operating at a rate of 20 kHz and having passband starting at 5 kHz with maximum passband attenuation of 0.5 dB, and stopband ending at 4 kHz with a minimum stopband attenuation of 10 dB.

Then, redesign it such that its magnitude response satisfies $1 \ge |H(f)|^2 \ge 0.98$ in the passband, and $|H(f)|^2 \le 0.02$ in the stopband.

Solution: The digital frequencies and their prewarped versions are:

$$\begin{aligned} \omega_{\text{pass}} &= \frac{2\pi f_{\text{pass}}}{f_s} = \frac{2\pi\cdot 5}{20} = 0.5\pi, & & \Omega_{\text{pass}} = \cot\left(\frac{\omega_{\text{pass}}}{2}\right) = 1 \\ & & \Rightarrow & \\ \omega_{\text{stop}} &= \frac{2\pi f_{\text{stop}}}{f_s} = \frac{2\pi\cdot 4}{20} = 0.4\pi, & & \Omega_{\text{stop}} = \cot\left(\frac{\omega_{\text{stop}}}{2}\right) = 1.3764 \end{aligned}$$

The dB attenuations $\{A_{\text{pass}}, A_{\text{stop}}\} = \{0.5, 10\}$ correspond to $\{\varepsilon_{\text{pass}}, \varepsilon_{\text{stop}}\} = \{0.3493, 3\}$. Then, Eq. (12.7.4) can be solved for the filter order:

$$N_{\text{exact}} = \frac{\ln(\varepsilon_{\text{stop}}/\varepsilon_{\text{pass}})}{\ln(\Omega_{\text{stop}}/\Omega_{\text{pass}})} = \frac{\ln(3/0.3493)}{\ln(1.3764/1)} = 6.73 \quad\Rightarrow\quad N = 7$$

Thus, there is one first-order section $H_0(z)$ and three second-order sections. Eq. (12.7.7) gives for Ω_0:

$$\Omega_0 = \frac{\Omega_{\text{pass}}}{\left(10^{A_{\text{pass}}/10} - 1\right)^{1/2N}} = \frac{\Omega_{\text{pass}}}{\varepsilon_{\text{pass}}^{1/N}} = \frac{1}{(0.3493)^{1/7}} = 1.1621$$

The SOS coefficients are calculated from Eqs. (12.9.4) and (12.9.6):

i	G_i	a_{i1}	a_{i2}
0	0.5375	−0.0750	
1	0.4709	−0.2445	0.6393
2	0.3554	−0.1845	0.2372
3	0.3039	−0.1577	0.0577

resulting in the transfer function:

$$\begin{aligned} H(z) &= H_0(z)H_1(z)H_2(z)H_3(z) \\ &= \frac{0.5375(1 - z^{-1})}{1 - 0.0750z^{-1}} \cdot \frac{0.4709(1 - z^{-1})^2}{1 - 0.2445z^{-1} + 0.6393z^{-2}} \\ &\quad \cdot \frac{0.3554(1 - z^{-1})^2}{1 - 0.1845z^{-1} + 0.2372z^{-2}} \cdot \frac{0.3039(1 - z^{-1})^2}{1 - 0.1577z^{-1} + 0.0577z^{-2}} \end{aligned}$$

As in Example 12.8.1, the second filter has passband and stopband attenuations: $A_{\text{pass}} = -10\log_{10}(0.98) = 0.0877$ dB and $A_{\text{stop}} = -10\log_{10}(0.02) = 16.9897$ dB. With these values, we find the design parameters $\{\varepsilon_{\text{pass}}, \varepsilon_{\text{stop}}\} = \{0.1429, 7\}$ and:

$$N_{\text{exact}} = 12.18, \quad N = 13, \quad \Omega_0 = 1.1615, \quad f_0 = 4.5253 \text{ kHz}$$

The coefficients of the first- and second-order sections are:

i	G_i	a_{i1}	a_{i2}
0	0.5374	−0.0747	
1	0.5131	−0.2655	0.7870
2	0.4252	−0.2200	0.4807
3	0.3677	−0.1903	0.2806
4	0.3300	−0.1708	0.1493
5	0.3062	−0.1584	0.0663
6	0.2930	−0.1516	0.0203

The magnitude responses of the two designs are shown in Fig. 12.9.2. □

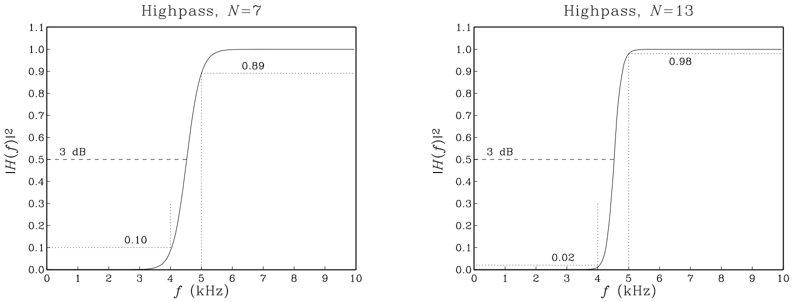


Fig. 12.9.2 Digital highpass Butterworth filters.

12.10 Digital Bandpass Filters

As in the highpass case, we can follow two possible approaches to the design of a digital bandpass filter. We can map the digital bandpass filter onto an equivalent analog bandpass filter using the transformation (12.1.3). Alternatively, we can use the bandpass version of the transformation (12.1.5) to map the bandpass digital filter onto an equivalent *lowpass* analog filter.

The first method was used in Sections 12.3 to design bandpass peaking filters. The second method is, however, more convenient because it reduces the bandpass design problem to a standard lowpass analog design problem. Figure 12.10.1 shows the bandpass specifications and their analog equivalents.

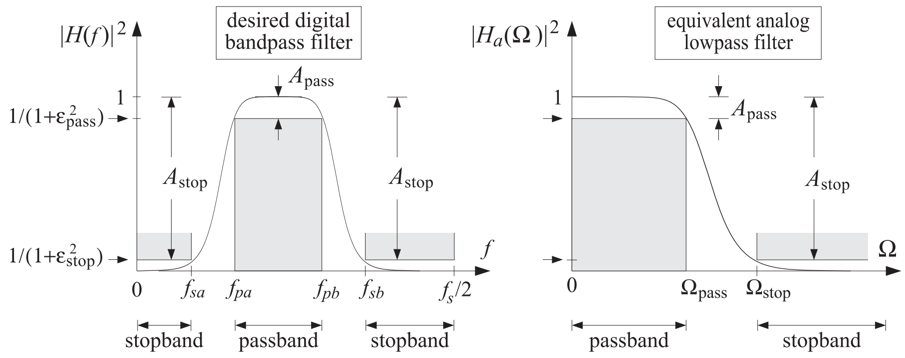


Fig. 12.10.1 Bandpass digital filter and its analog lowpass equivalent.

The specifications are the quantities $\{f_{pa}, f_{pb}, f_{sa}, f_{sb}, A_{\text{pass}}, A_{\text{stop}}\}$, defining the passband range $f_{pa} \le f \le f_{pb}$, the left stopband $0 \le f \le f_{sa}$, and the right stopband $f_{sb} \le f \le f_s/2$. The stopband attenuations were assumed to be equal in the two stopbands; if they are not, we may design the filter based on the maximum of the two.

The bandpass version of the bilinear[†] transformation and the corresponding frequency mapping are in this case:

$$s = \frac{1 - 2cz^{-1} + z^{-2}}{1 - z^{-2}}, \qquad \Omega = \frac{c - \cos\omega}{\sin\omega}, \qquad \omega = \frac{2\pi f}{f_s} \tag{12.10.1}$$

A new parameter c has been introduced. Note that $c = 1$ recovers the lowpass case, and $c = -1$ the highpass one. The parameter c is required to be $|c| \le 1$ in order to map the left-hand s-plane into the inside of the unit circle in the z-plane.

Therefore, we may set $c = \cos\omega_c$, for some value of ω_c. The center of the analog passband $\Omega = 0$ corresponds to $\cos\omega = c = \cos\omega_c$, or, $\omega = \omega_c$. Therefore, ω_c may be thought of as the "center" frequency of the bandpass filter (although it need not be exactly at the center of the passband).

The given bandpass specifications, must be mapped onto the specifications of the equivalent analog filter, $\{\Omega_{\text{pass}}, \Omega_{\text{stop}}, A_{\text{pass}}, A_{\text{stop}}\}$. This can be done as follows. We require that the passband $[f_{pa}, f_{pb}]$ of the digital filter be mapped onto the entire passband $[-\Omega_{\text{pass}}, \Omega_{\text{pass}}]$ of the analog filter. This requires that:

[†] It should really be called "biquadratic" in this case.

$$-\Omega_{\text{pass}} = \frac{c - \cos\omega_{pa}}{\sin\omega_{pa}}$$

$$\Omega_{\text{pass}} = \frac{c - \cos\omega_{pb}}{\sin\omega_{pb}}$$

where $\omega_{pa} = 2\pi f_{pa}/f_s$ and $\omega_{pb} = 2\pi f_{pb}/f_s$. By adding them, we solve for c. Then, inserting the computed value of c into one or the other we find Ω_{pass}. The resulting solution is:

$$c = \frac{\sin(\omega_{pa} + \omega_{pb})}{\sin\omega_{pa} + \sin\omega_{pb}}, \qquad \Omega_{\text{pass}} = \left|\frac{c - \cos\omega_{pb}}{\sin\omega_{pb}}\right| \tag{12.10.2}$$

Note that for ω_{pa}, ω_{pb} in the interval $[0, \pi]$, the above expression for c implies $|c| \le 1$, as required for stability. Next, we compute the two numbers:

$$\Omega_{sa} = \frac{c - \cos\omega_{sa}}{\sin\omega_{sa}}, \qquad \Omega_{sb} = \frac{c - \cos\omega_{sb}}{\sin\omega_{sb}}$$

where $\omega_{sa} = 2\pi f_{sa}/f_s$ and $\omega_{sb} = 2\pi f_{sb}/f_s$.

Ideally, the stopband of the digital filter should map exactly onto the stopband of the analog filter so that we should have $\Omega_{sb} = \Omega_{\text{stop}}$ and $\Omega_{sa} = -\Omega_{\text{stop}}$. But this is impossible because c has already been determined from Eq. (12.10.2).

Because the Butterworth magnitude response is a *monotonically decreasing* function of Ω, it is enough to choose the smallest of the two stopbands defined above. Thus, we define:

$$\Omega_{\text{stop}} = \min\big(|\Omega_{sa}|, |\Omega_{sb}|\big) \tag{12.10.3}$$

With the computed values of Ω_{pass} and Ω_{stop}, we proceed to compute the Butterworth parameters N and Ω_0 and the corresponding SOSs of Eq. (12.7.15). Because s is quadratic in z, the substitution of s into these SOSs will give rise to *fourth-order* sections in z:

$$\begin{aligned} H_i(z) &= \left.\frac{1}{1 - 2\dfrac{s}{\Omega_0}\cos\theta_i + \dfrac{s^2}{\Omega_0^2}}\right|_{s=\frac{1-2cz^{-1}+z^{-2}}{1-z^{-2}}} \\ &= \frac{G_i(1 - z^{-2})^2}{1 + a_{i1}z^{-1} + a_{i2}z^{-2} + a_{i3}z^{-3} + a_{i4}z^{-4}} \end{aligned} \tag{12.10.4}$$

where, for $i = 1, 2, \dots, K$:

$$G_i = \frac{\Omega_0^2}{1 - 2\Omega_0\cos\theta_i + {\Omega_0}^2}$$

$$\begin{aligned} a_{i1} &= \frac{4c(\Omega_0\cos\theta_i - 1)}{1 - 2\Omega_0\cos\theta_i + {\Omega_0}^2} & a_{i2} &= \frac{2(2c^2 + 1 - \Omega_0^2)}{1 - 2\Omega_0\cos\theta_i + {\Omega_0}^2} \\ a_{i3} &= -\frac{4c(\Omega_0\cos\theta_i + 1)}{1 - 2\Omega_0\cos\theta_i + {\Omega_0}^2} & a_{i4} &= \frac{1 + 2\Omega_0\cos\theta_i + \Omega_0^2}{1 - 2\Omega_0\cos\theta_i + {\Omega_0}^2} \end{aligned} \tag{12.10.5}$$

If N is odd, then there is also a first-order section in s which becomes a second-order section in z:

$$H_0(z)= \frac{1}{1+\dfrac{s}{\Omega_0}}\Bigg|_{s=\frac{1-2cz^{-1}+z^{-2}}{1-z^{-2}}} = \frac{G_0(1-z^{-2})}{1+a_{01}z^{-1}+a_{02}z^{-2}} \tag{12.10.6}$$

where

$$G_0=\frac{\Omega_0}{1+\Omega_0},\qquad a_{01}=-\frac{2c}{1+\Omega_0},\qquad a_{02}=\frac{1-\Omega_0}{1+\Omega_0} \tag{12.10.7}$$

The overall transfer function of the designed bandpass digital filter will be given as the cascade of fourth-order sections with the possibility of one SOS:

$$H(z)=H_0(z)H_1(z)H_2(z)\cdots H_K(z)$$

The order of the digital filter is $2N$, because s is quadratic in z. The filter sections have zeros at $z=\pm 1$, that is, $\omega=0$ and $\omega=\pi$. The left and right 3-dB frequencies can be calculated from the equations:

$$\frac{c-\cos\omega_0}{\sin\omega_0}=\mp\Omega_0$$

They can be solved by writing $\cos\omega_0$ and $\sin\omega_0$ in terms of $\tan(\omega_0/2)$, solving the resulting quadratic equation, and picking the positive solutions:

$$\begin{aligned}\tan\left(\frac{\omega_{0a}}{2}\right)&=\tan\left(\frac{\pi f_{0a}}{f_s}\right)=\frac{\sqrt{\Omega_0^2+1-c^2}-\Omega_0}{1+c}\\ \tan\left(\frac{\omega_{0b}}{2}\right)&=\tan\left(\frac{\pi f_{0b}}{f_s}\right)=\frac{\sqrt{\Omega_0^2+1-c^2}+\Omega_0}{1+c}\end{aligned} \tag{12.10.8}$$

Example 12.10.1: Using the bilinear transformation and a lowpass analog Butterworth prototype, design a bandpass digital filter operating at a rate of 20 kHz and having left and right passband frequencies of 2 and 4 kHz, and left and right stopband frequencies of 1.5 and 4.5 kHz. The maximum passband attenuation is required to be 0.5 dB, and the minimum stopband attenuation 10 dB.

Then, redesign it such that its magnitude response satisfies $1\ge |H(f)|^2\ge 0.98$ in the passband, and $|H(f)|^2\le 0.02$ in the stopbands.

Solution: The digital passband frequencies are:

$$\omega_{pa}=\frac{2\pi f_{pa}}{f_s}=\frac{2\pi\cdot 2}{20}=0.2\pi,\quad \omega_{pb}=\frac{2\pi f_{pb}}{f_s}=\frac{2\pi\cdot 4}{20}=0.4\pi$$

Then, we calculate c and Ω_{pass}:

$$c=\frac{\sin(\omega_{pa}+\omega_{pb})}{\sin\omega_{pa}+\sin\omega_{pb}}=0.6180,\qquad \Omega_{\text{pass}}=\left|\frac{c-\cos\omega_{pb}}{\sin\omega_{pb}}\right|=0.3249$$

With the stopband digital frequencies:

$$\omega_{sa}=\frac{2\pi f_{sa}}{f_s}=\frac{2\pi\cdot 1.5}{20}=0.15\pi,\quad \omega_{sb}=\frac{2\pi f_{sb}}{f_s}=\frac{2\pi\cdot 4.5}{20}=0.45\pi$$

we calculate:

$$\Omega_{sa}=\frac{c-\cos\omega_{sa}}{\sin\omega_{sa}}=-0.6013,\quad \Omega_{sb}=\frac{c-\cos\omega_{sb}}{\sin\omega_{sb}}=0.4674$$

and $\Omega_{\text{stop}}=\min(|\Omega_{sa}|,|\Omega_{sb}|)=0.4674$. The analog filter with the specifications $\{\Omega_{\text{pass}},\Omega_{\text{stop}},A_{\text{pass}},A_{\text{sto}}$ has parameters $\{\varepsilon_{\text{pass}},\varepsilon_{\text{stop}}\}=\{0.3493,3\}$ and:

$$N_{\text{exact}}=5.92,\quad N=6,\quad \Omega_0=0.3872$$

The left-right 3-dB frequencies are calculated from Eq. (12.10.8) to be: $f_{0a}=1.8689$ kHz, $f_{0b}=4.2206$ kHz. The coefficients of the three fourth-order sections of the digital filter are (computed by the MATLAB function `bpsbutt.m`):

i	G_i	a_{i1}	a_{i2}	a_{i3}	a_{i4}
1	0.1110	−2.0142	2.3906	−1.6473	0.7032
2	0.0883	−1.8551	1.9017	−1.0577	0.3549
3	0.0790	−1.7897	1.7009	−0.8154	0.2118

The magnitude response can be calculated from:

$$|H(\omega)|^2=\frac{1}{1+\left(\dfrac{\Omega}{\Omega_0}\right)^{2N}}=\frac{1}{1+\left(\dfrac{c-\cos\omega}{\Omega_0\sin\omega}\right)^{2N}}$$

The magnitude response is shown in the left graph of Fig. 12.10.2. The passband specifications are met exactly by design. Because the maximum stopband frequency was on the right, $\Omega_{\text{stop}}=|\Omega_{sb}|$, the right stopband specification is met stringently. The left stopband specification is more than required.

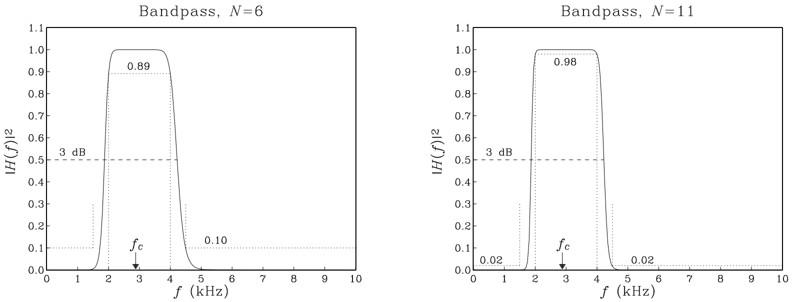


Fig. 12.10.2 Digital bandpass Butterworth filters.

For the second set of specifications, we have $A_{\text{pass}} = -10\log_{10}(0.98) = 0.0877$ dB, and $A_{\text{stop}} = -10\log_{10}(0.02) = 16.9897$ dB and $\{\varepsilon_{\text{pass}}, \varepsilon_{\text{stop}}\} = \{0.1429, 7\}$. The design has the same c, Ω_{pass}, and Ω_{stop}, which lead to the Butterworth parameters:

$$N_{\text{exact}} = 10.71, \quad N = 11, \quad \Omega_0 = 0.3878$$

The left and right 3-dB frequencies are now $f_{0a} = 1.8677$ kHz, $f_{0b} = 4.2228$ kHz. The digital filter coefficients of the second- and fourth-order sections are:

i	G_i	a_{i1}	a_{i2}	a_{i3}	a_{i4}
0	0.2794	−0.8907	0.4411		
1	0.1193	−2.0690	2.5596	−1.8526	0.8249
2	0.1021	−1.9492	2.1915	−1.4083	0.5624
3	0.0907	−1.8694	1.9460	−1.1122	0.3874
4	0.0834	−1.8186	1.7900	−0.9239	0.2762
5	0.0794	−1.7904	1.7033	−0.8193	0.2144

Again, the right stopband specification is more stringently met than the left one. The "center" frequency of the passband is the same for both filters and can be obtained by inverting $\cos\omega_c = c$. In Hz, we have $f_c = f_s \arccos(c)/(2\pi) = 2.8793$ kHz. The magnitude response is normalized to unity at f_c. □

12.11 Digital Bandstop Filters

The specifications of a *bandstop* digital filter are shown in Fig. 12.11.1, together with their analog equivalents. There are now two passbands, that is, $0 \le f \le f_{pa}$ and $f_{pb} \le f \le f_s/2$, and one stopband $f_{sa} \le f \le f_{sb}$.

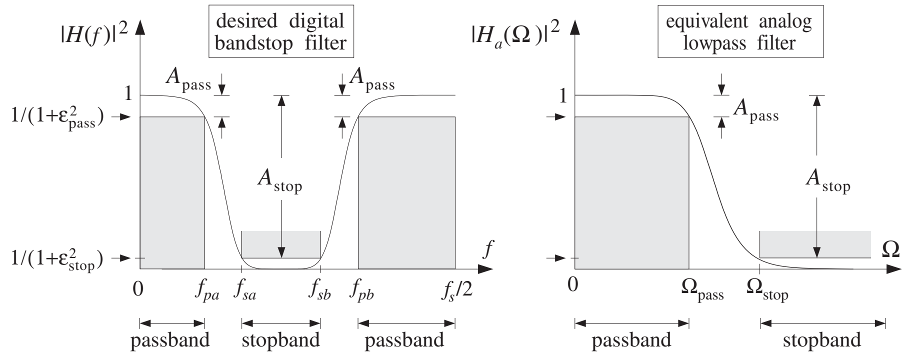


Fig. 12.11.1 Bandstop digital filter and its analog lowpass equivalent.

The bandstop version of the bilinear transformation and the corresponding frequency mapping were given in Eqs. (12.1.5) and (12.1.6):

$$s = \frac{1 - z^{-2}}{1 - 2cz^{-1} + z^{-2}}, \qquad \Omega = \frac{\sin\omega}{\cos\omega - c}, \qquad \omega = \frac{2\pi f}{f_s} \tag{12.11.1}$$

The design steps are summarized as follows. First, we compute the digital frequencies in radians per sample:

$$\omega_{pa} = \frac{2\pi f_{pa}}{f_s}, \quad \omega_{pb} = \frac{2\pi f_{pb}}{f_s}, \quad \omega_{sa} = \frac{2\pi f_{sa}}{f_s}, \quad \omega_{sb} = \frac{2\pi f_{sb}}{f_s}$$

Then, we calculate c and Ω_{pass} by requiring:

$$-\Omega_{\text{pass}} = \frac{\sin\omega_{pa}}{\cos\omega_{pa} - c}, \quad \Omega_{\text{pass}} = \frac{\sin\omega_{pb}}{\cos\omega_{pb} - c}$$

which may be solved as follows:

$$c = \frac{\sin(\omega_{pa} + \omega_{pb})}{\sin\omega_{pa} + \sin\omega_{pb}}, \qquad \Omega_{\text{pass}} = \left|\frac{\sin\omega_{pb}}{\cos\omega_{pb} - c}\right|$$

Next, we compute the two possible stopbands:

$$\Omega_{sa} = \frac{\sin\omega_{sa}}{\cos\omega_{sa} - c}, \qquad \Omega_{sb} = \frac{\sin\omega_{sb}}{\cos\omega_{sb} - c}$$

and define:

$$\Omega_{\text{stop}} = \min\big(|\Omega_{sa}|, |\Omega_{sb}|\big)$$

Then, use the analog specifications $\{\Omega_{\text{pass}}, \Omega_{\text{stop}}, A_{\text{pass}}, A_{\text{stop}}\}$ to compute the Butterworth parameters $\{N, \Omega_0\}$. And finally, transform the analog filter sections into fourth-order sections by Eq. (12.11.1):

$$\begin{aligned} H_i(z) &= \left.\frac{1}{1 - 2\dfrac{s}{\Omega_0}\cos\theta_i + \dfrac{s^2}{\Omega_0^2}}\right|_{s=\frac{1-z^{-2}}{1-2cz^{-1}+z^{-2}}} \\ &= \frac{G_i(1 - 2cz^{-1} + z^{-2})^2}{1 + a_{i1}z^{-1} + a_{i2}z^{-2} + a_{i3}z^{-3} + a_{i4}z^{-4}} \end{aligned}$$

where the coefficients are given for $i = 1, 2, \dots, K$:

$$G_i = \frac{\Omega_0^2}{1 - 2\Omega_0\cos\theta_i + {\Omega_0}^2}$$

$$a_{i1} = \frac{4c\Omega_0(\cos\theta_i - \Omega_0)}{1 - 2\Omega_0\cos\theta_i + {\Omega_0}^2} \qquad a_{i2} = \frac{2(2c^2\Omega_0^2 + \Omega_0^2 - 1)}{1 - 2\Omega_0\cos\theta_i + {\Omega_0}^2}$$

$$a_{i3} = -\frac{4c\Omega_0(\cos\theta_i + \Omega_0)}{1 - 2\Omega_0\cos\theta_i + {\Omega_0}^2} \qquad a_{i4} = \frac{1 + 2\Omega_0\cos\theta_i + \Omega_0^2}{1 - 2\Omega_0\cos\theta_i + {\Omega_0}^2}$$

If N is odd, we also have a second-order section in z:

$$H_0(z) = \left.\frac{1}{1 + \dfrac{s}{\Omega_0}}\right|_{s=\frac{1-z^{-2}}{1-2cz^{-1}+z^{-2}}} = \frac{G_0(1 - 2cz^{-1} + z^{-2})}{1 + a_{01}z^{-1} + a_{02}z^{-2}}$$

where

$$G_0 = \frac{\Omega_0}{1+\Omega_0}, \qquad a_{01} = -\frac{2c\Omega_0}{1+\Omega_0}, \qquad a_{02} = -\frac{1-\Omega_0}{1+\Omega_0}$$

Note that each section has zeros at $1 - 2cz^{-1} + z^{-2} = 0$, which correspond to the angles $\omega = \pm\omega_c$, where $\cos\omega_c = c$. The 3-dB frequencies at the edges of the passbands can be determined by solving for the positive solutions of the equations:

$$\frac{\sin\omega_0}{\cos\omega_0 - c} = \pm\Omega_0$$

which give:

$$\tan\left(\frac{\omega_{0a}}{2}\right) = \tan\left(\frac{\pi f_{0a}}{f_s}\right) = \frac{\sqrt{1+\Omega_0^2(1-c^2)} - 1}{\Omega_0(1+c)}$$

$$\tan\left(\frac{\omega_{0b}}{2}\right) = \tan\left(\frac{\pi f_{0b}}{f_s}\right) = \frac{\sqrt{1+\Omega_0^2(1-c^2)} + 1}{\Omega_0(1+c)}$$

Example 12.11.1: Using the bilinear transformation and a lowpass analog Butterworth prototype, design a bandstop digital filter operating at a rate of 20 kHz and having left and right passband frequencies of 1.5 and 4.5 kHz, and left and right stopband frequencies of 2 and 4 kHz. The maximum passband attenuation is required to be 0.5 dB, and the minimum stopband attenuation 10 dB.

Then, redesign it such that its magnitude response satisfies $1 \geq |H(f)|^2 \geq 0.98$ in the passbands, and $|H(f)|^2 \leq 0.02$ in the stopband.

Solution: The digital passband and stopband frequencies are:

$$\omega_{pa} = \frac{2\pi f_{pa}}{f_s} = \frac{2\pi\cdot 1.5}{20} = 0.15\pi, \qquad \omega_{pb} = \frac{2\pi f_{pb}}{f_s} = \frac{2\pi\cdot 4.5}{20} = 0.45\pi$$

$$\omega_{sa} = \frac{2\pi f_{sa}}{f_s} = \frac{2\pi\cdot 2}{20} = 0.2\pi, \qquad \omega_{sb} = \frac{2\pi f_{sb}}{f_s} = \frac{2\pi\cdot 4}{20} = 0.4\pi$$

Then, we calculate c and Ω_{pass}:

$$c = \frac{\sin(\omega_{pa}+\omega_{pb})}{\sin\omega_{pa} + \sin\omega_{pb}} = 0.6597, \qquad \Omega_{\text{pass}} = \left|\frac{\sin\omega_{pb}}{\cos\omega_{pb} - c}\right| = 1.9626$$

Then, we calculate the stopband frequencies:

$$\Omega_{sa} = \frac{\sin\omega_{sa}}{\cos\omega_{sa} - c} = 3.9361, \qquad \Omega_{sb} = \frac{\sin\omega_{sb}}{\cos\omega_{sb} - c} = -2.7121$$

and define $\Omega_{\text{stop}} = \min(|\Omega_{sa}|, |\Omega_{sb}|) = 2.7121$. The analog filter parameters are:

$$N_{\text{exact}} = 6.65, \quad N = 7, \quad \Omega_0 = 2.2808$$

The left-right 3-dB frequencies are calculated to be $f_{0a} = 1.6198$ kHz, $f_{0b} = 4.2503$ kHz. The coefficients of the SOS and the three fourth-order sections of the digital filter are:

i	G_i	a_{i1}	a_{i2}	a_{i3}	a_{i4}
0	0.6952	−0.9172	0.3904		
1	0.7208	−2.0876	2.4192	−1.7164	0.7187
2	0.5751	−1.9322	1.9301	−1.1026	0.3712
3	0.5045	−1.8570	1.6932	−0.8053	0.2029

The magnitude response of the designed filter is shown in the left graph of Fig. 12.11.2.

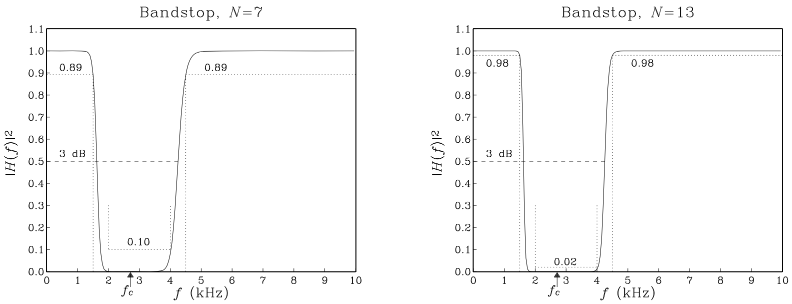


Fig. 12.11.2 Digital bandstop Butterworth filters.

For the second set of specifications, we have $A_{\text{pass}} = -10\log_{10}(0.98) = 0.0877$ dB, and $A_{\text{stop}} = -10\log_{10}(0.02) = 16.9897$ dB. The design has the same c, Ω_{pass}, and Ω_{stop}, which lead to the Butterworth parameters:

$$N_{\text{exact}} = 12.03, \quad N = 13, \quad \Omega_0 = 2.2795$$

The left-right 3-dB frequencies are now $f_{0a} = 1.6194$ kHz, $f_{0b} = 4.2512$ kHz. The digital filter coefficients of the second- and fourth-order sections are:

i	G_i	a_{i1}	a_{i2}	a_{i3}	a_{i4}
0	0.6951	−0.9171	−0.3902		
1	0.7703	−2.1401	2.5850	−1.9251	0.8371
2	0.6651	−2.0280	2.2319	−1.4820	0.5862
3	0.5914	−1.9495	1.9847	−1.1717	0.4105
4	0.5408	−1.8956	1.8148	−0.9584	0.2897
5	0.5078	−1.8604	1.7041	−0.8194	0.2110
6	0.4892	−1.8406	1.6415	−0.7410	0.1666

The magnitude response is shown on the right graph of Fig. 12.11.2. The rounding of the exact N of 12.03 to 13 is perhaps overkill in this case. It causes the actual stopband attenuation at the right edge of the stopband to be $A(\Omega_{\text{stop}}) = 19.67$ dB, corresponding to a magnitude square of 0.011 instead of the required 0.02.

For both designs, the "center" notch frequency of the stopband can be obtained by inverting $\cos\omega_c = c$. In Hz, we have $f_c = f_s \arccos(c)/(2\pi) = 2.7069$ kHz. □

12.12 Chebyshev Filter Design

In designing the equivalent analog lowpass filters one can use alternative filter prototypes, such as Chebyshev or elliptic filters [298–300]. For the same set of specifications, they provide steeper transition widths and lead to smaller filter orders than the Butterworth filters.

Chebyshev filters come in two varieties. Type 1 has equiripple passband and monotonic stopband, and type 2, also known as inverse Chebyshev, has equiripple stopband and monotonic passband. A typical Chebyshev magnitude response is shown in Fig. 12.12.1.

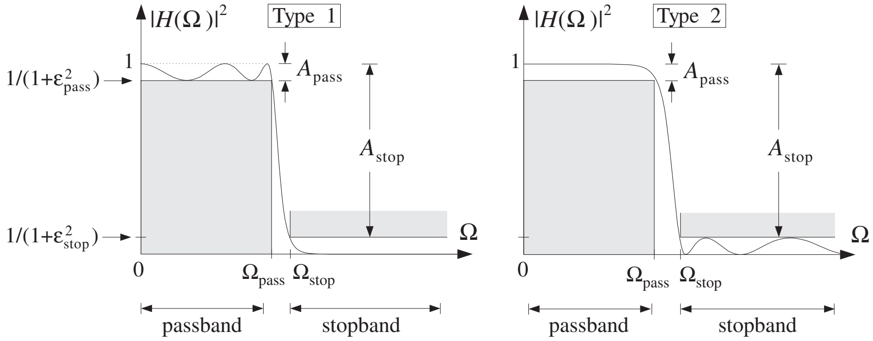


Fig. 12.12.1 Magnitude square responses of type 1 and type 2 Chebyshev filters.

It is the equiripple property that is responsible for the narrower transition widths of these filters. For example, for the type 1 case, because the passband response is allowed to go slightly up near the edge of the passband, it can fall off more steeply.

The specifications of the filter are $\{\Omega_{\text{pass}}, \Omega_{\text{stop}}, A_{\text{pass}}, A_{\text{stop}}\}$ and are obtained by prewarping the desired digital filter specifications using the appropriate bilinear transformation (lowpass, highpass, bandpass, or bandstop). Two important design parameters are the quantities $\{\varepsilon_{\text{pass}}, \varepsilon_{\text{stop}}\}$ that were defined in Eq. (12.6.4), and are shown in Figs. 12.6.1 and 12.12.1.

The magnitude response squared of an Nth order Chebyshev filter is expressible in terms of these parameters as follows. For the type 1 case:

$$\boxed{|H(\Omega)|^2 = \frac{1}{1 + \varepsilon_{\text{pass}}^2 C_N^2\left(\dfrac{\Omega}{\Omega_{\text{pass}}}\right)}} \tag{12.12.1}$$

and, for the type 2 case:

$$\boxed{|H(\Omega)|^2 = \frac{C_N^2\left(\dfrac{\Omega_{\text{stop}}}{\Omega}\right)}{C_N^2\left(\dfrac{\Omega_{\text{stop}}}{\Omega}\right) + \varepsilon_{\text{stop}}^2}} \tag{12.12.2}$$

where $C_N(x)$ is the Chebyshev polynomial [301] of degree N, defined by

$$C_N(x) = \begin{cases} \cos\big(N\cos^{-1}(x)\big), & \text{if } |x| \le 1 \\ \cosh\big(N\cosh^{-1}(x)\big), & \text{if } |x| > 1 \end{cases} \tag{12.12.3}$$

Chebyshev polynomials can be understood by defining the angle $\theta = \cos^{-1} x$, so that $x = \cos\theta$ and $C_N(x) = \cos(N\theta)$. When $|x| > 1$, the equation $x = \cos\theta$ requires θ to be imaginary, say $\theta = j\beta$, so that $x = \cos(j\beta) = \cosh(\beta)$ and

$$C_N(x) = \cos(N\theta) = \cos(Nj\beta) = \cosh(N\beta) = \cosh(N\cosh^{-1} x)$$

Using trigonometric identities, it can be shown that $\cos(N\theta)$ is expressible as an Nth order polynomial in $\cos\theta$, that is,

$$\cos(N\theta) = \sum_{i=0}^{N} c_i (\cos\theta)^i$$

The c_i are the coefficients of the Chebyshev polynomials:

$$C_N(x) = \sum_{i=0}^{N} c_i x^i$$

For example, we have $C_1(x) = \cos\theta = x$, and

$$\begin{aligned} \cos(2\theta) &= 2\cos^2\theta - 1 & & C_2(x) = 2x^2 - 1 \\ \cos(3\theta) &= 4\cos^3\theta - 3\cos\theta & \Rightarrow\quad & C_3(x) = 4x^3 - 3x \\ \cos(4\theta) &= 8\cos^4\theta - 8\cos^2\theta + 1 & & C_4(x) = 8x^4 - 8x^2 + 1 \end{aligned}$$

The following routine `cheby.c` returns the value of the Nth order Chebyshev polynomial for non-negative values of x and can be used in the numerical evaluation of the magnitude responses:

```
/* cheby.c - Chebyshev polynomial C_N(x) */

#include <math.h>

double cheby(N, x)                                       usage: y = cheby(N, x);
int N;                                                   N = polynomial order
double x;                                                x must be non-negative
{
       if (x <= 1)
              return cos(N * acos(x));
       else
              return cosh(N * log(x + sqrt(x*x-1)));
}
```

For $x > 1$, the values are computed by the alternative expression:

$$\cosh(N\cosh^{-1} x) = \cosh\left(N\ln\big(x + \sqrt{x^2 - 1}\big)\right)$$

Next, we consider the details of the type 1 case. The argument of $C_N(x)$ in Eq. (12.12.1) is $x = \Omega/\Omega_{\text{pass}}$. Therefore, within the passband range $0 \le \Omega \le \Omega_{\text{pass}}$ we have $0 \le x \le 1$, which makes $C_N(x)$ oscillatory and results in the passband ripples.

Within the passband, the magnitude response remains bounded between the values 1 and $1/(1+\varepsilon_{\text{pass}}^2)$. At the edge of the passband, corresponding to $x = \Omega/\Omega_{\text{pass}} = 1$, we have $C_N(x) = 1$, giving the value $|H(\Omega_{\text{pass}})|^2 = 1/(1+\varepsilon_{\text{pass}}^2)$. The value at $\Omega = 0$ depends on N. Because $C_N(0)$ equals zero for odd N and unity for even N, we have:

$$|H(0)|^2 = 1 \quad (\text{odd } N), \qquad |H(0)|^2 = \frac{1}{1+\varepsilon_{\text{pass}}^2} \quad (\text{even } N) \tag{12.12.4}$$

The order N can be determined by imposing the stopband specification, that is, $|H(\Omega)|^2 \le 1/(1+\varepsilon_{\text{stop}}^2)$ for $\Omega \ge \Omega_{\text{stop}}$. Because of the monotonicity of the stopband, this condition is equivalent to the stopband edge condition:

$$|H(\Omega_{\text{stop}})|^2 = \frac{1}{1+\varepsilon_{\text{stop}}^2}$$

Using Eq. (12.12.1), we obtain:

$$\frac{1}{1+\varepsilon_{\text{pass}}^2 \cosh^2\big(N\cosh^{-1}(\Omega_{\text{stop}}/\Omega_{\text{pass}})\big)} = \frac{1}{1+\varepsilon_{\text{stop}}^2}$$

which gives:

$$\cosh\big(N\cosh^{-1}(\Omega_{\text{stop}}/\Omega_{\text{pass}})\big) = \varepsilon_{\text{stop}}/\varepsilon_{\text{pass}} \quad \Rightarrow \quad \cosh(N\cosh^{-1} w) = e$$

where, as in Eq. (12.7.5), we used the stopband to passband ratios:

$$e = \frac{\varepsilon_{\text{stop}}}{\varepsilon_{\text{pass}}} = \sqrt{\frac{10^{A_{\text{stop}}/10}-1}{10^{A_{\text{pass}}/10}-1}}, \quad w = \frac{\Omega_{\text{stop}}}{\Omega_{\text{pass}}} \tag{12.12.5}$$

Thus, solving for N, we find:

$$\boxed{N_{\text{exact}} = \frac{\cosh^{-1} e}{\cosh^{-1} w} = \frac{\ln(e+\sqrt{e^2-1})}{\ln(w+\sqrt{w^2-1})}} \tag{12.12.6}$$

The final value of N is obtained by rounding N_{exact} up to the next integer, that is, $N = \lceil N_{\text{exact}} \rceil$. As in the Butterworth case, increasing N slightly from its exact value results in a slightly better stopband than required, that is, $|H(\Omega_{\text{stop}})|^2 < 1/(1+\varepsilon_{\text{stop}}^2)$. The 3-dB frequency can be calculated by requiring $|H(\Omega)|^2 = 1/2$, which can be solved to give:

$$\frac{1}{1+\varepsilon_{\text{pass}}^2 C_N^2(\Omega_{\text{3dB}}/\Omega_{\text{pass}})} = \frac{1}{2} \quad \Rightarrow \quad \cosh\big(N\cosh^{-1}(\Omega_{\text{3dB}}/\Omega_{\text{pass}})\big) = \frac{1}{\varepsilon_{\text{pass}}}$$

or,

$$\tan\left(\frac{\pi f_{\text{3dB}}}{f_s}\right) = \Omega_{\text{3dB}} = \Omega_{\text{pass}} \cosh\left(\frac{1}{N}\cosh^{-1}\big(\frac{1}{\varepsilon_{\text{pass}}}\big)\right) \tag{12.12.7}$$

The transfer function $H(s)$ of the Chebyshev filter can be constructed by determining the left-hand-plane poles of Eq. (12.12.1) and pairing them in conjugate pairs to form second-order sections. These conjugate pairs are $\{s_i, s_i^*\}$, where

$$s_i = \Omega_{\text{pass}} \sinh a \, \cos\theta_i + j\Omega_{\text{pass}} \cosh a \, \sin\theta_i, \quad i = 1, 2, \dots, K \tag{12.12.8}$$

where $N = 2K$ or $N = 2K+1$. In the odd case, there is also a real pole at

$$s_0 = -\Omega_{\text{pass}} \sinh a \tag{12.12.9}$$

where the parameter a is the solution of

$$\sinh(Na) = \frac{1}{\varepsilon_{\text{pass}}} \tag{12.12.10}$$

that is,

$$a = \frac{1}{N}\sinh^{-1}\left(\frac{1}{\varepsilon_{\text{pass}}}\right) = \frac{1}{N}\ln\left(\frac{1}{\varepsilon_{\text{pass}}} + \sqrt{\frac{1}{\varepsilon_{\text{pass}}^2}+1}\right) \tag{12.12.11}$$

The angles θ_i are the *same* as the Butterworth angles of Eq. (12.7.11):

$$\theta_i = \frac{\pi}{2N}(N-1+2i), \quad i = 1, 2, \dots, K \tag{12.12.12}$$

The second-quadrant values of these angles place the s_i in the left-hand s-plane. The second-order sections are then:

$$H_i(s) = \frac{1}{\left(1-\dfrac{s}{s_i}\right)\left(1-\dfrac{s}{s_i^*}\right)} = \frac{|s_i|^2}{s^2 - (2\text{Re}s_i)s + |s_i|^2}$$

For convenience, we define the parameters:

$$\Omega_0 = \Omega_{\text{pass}} \sinh a, \qquad \Omega_i = \Omega_{\text{pass}} \sin\theta_i, \quad i = 1, 2, \dots, K \tag{12.12.13}$$

Then, we may express the second-order sections in the form:

$$H_i(s) = \frac{\Omega_0^2 + \Omega_i{}^2}{s^2 - 2\Omega_0\cos\theta_i \, s + \Omega_0^2 + \Omega_i{}^2}, \quad i = 1, 2, \dots, K \tag{12.12.14}$$

The first-order factor $H_0(s)$ is defined by

$$H_0(s) = \begin{cases} \sqrt{\dfrac{1}{1+\varepsilon_{\text{pass}}^2}} & \text{if } N \text{ is even, } N = 2K \\[2ex] \dfrac{\Omega_0}{s+\Omega_0} & \text{if } N \text{ is odd, } N = 2K+1 \end{cases} \tag{12.12.15}$$

If N is odd, all filter sections are normalized to unity gain at DC, as required by Eq. (12.12.4). If N is even, the overall gain is $1/(1+\varepsilon_{\text{pass}}^2)^{1/2}$. It follows that the overall transfer function will be the cascade:

$$\boxed{H(s) = H_0(s)H_1(s)H_2(s)\cdots H_K(s)} \tag{12.12.16}$$

Next, we verify that the poles are properly given by Eq. (12.12.8). Replacing $s = j\Omega$ or $\Omega = -js$ into Eq. (12.12.1), we see that the zeros of the denominator are the solutions of the equation:

$$1 + \varepsilon_{\text{pass}}^2 \cosh^2\big(N\cosh^{-1}(-js/\Omega_{\text{pass}})\big) = 0\,, \qquad \text{or}$$

$$\cosh\big(N\cosh^{-1}(-js/\Omega_{\text{pass}})\big) = \frac{\pm j}{\varepsilon_{\text{pass}}} \tag{12.12.17}$$

Replacing $\theta_i = \phi_i + \pi/2$, where $\phi_i = (2i-1)\pi/(2N)$, into Eq. (12.12.8), we find

$$-js_i/\Omega_{\text{pass}} = \cosh a\,\sin\theta_i - j\sinh a\,\cos\theta_i = \cosh a\,\cos\phi_i + j\sinh a\,\sin\phi_i$$

Using the trigonometric identity

$$\cosh(x + jy) = \cosh x\,\cos y + j\sinh x\,\sin y$$

we find

$$-js_i/\Omega_{\text{pass}} = \cosh(a + j\phi_i) \quad\Rightarrow\quad \cosh^{-1}(-js_i/\Omega_{\text{pass}}) = a + j\phi_i$$

and therefore,

$$\begin{aligned}\cosh\big(N\cosh^{-1}(-js_i/\Omega_{\text{pass}})\big) &= \cosh(Na + jN\phi_i)\\ &= \cosh(Na)\cos(N\phi_i) + j\sinh(Na)\sin(N\phi_i) = \frac{\pm j}{\varepsilon_{\text{pass}}}\end{aligned}$$

where we used $\cos(N\phi_i) = \cos\big((2i-1)\pi/2\big) = 0$, $\sin(N\phi_i) = \sin\big((2i-1)\pi/2\big) = \pm 1$, and Eq. (12.12.10). Thus, the s_i are solutions of the root equation Eq. (12.12.17).

Once the analog transfer function is constructed, each second-order section may be transformed into a digital second-order section by the appropriate bilinear transformation. For example, applying the lowpass version of the bilinear transformation $s = (1 - z^{-1})/(1 + z^{-1})$, we find the digital transfer function:

$$\boxed{H(z) = H_0(z)H_1(z)H_2(z)\cdots H_K(z)} \tag{12.12.18}$$

where $H_i(z)$ are the transformations of Eq. (12.12.14):

$$H_i(z) = \frac{G_i(1 + z^{-1})^2}{1 + a_{i1}z^{-1} + a_{i2}z^{-2}}\,, \quad i = 1, 2, \dots, K \tag{12.12.19}$$

where the coefficients are computed in terms of the definitions Eq. (12.12.13):

$$\begin{aligned} G_i &= \frac{\Omega_0^2 + {\Omega_i}^2}{1 - 2\Omega_0\cos\theta_i + \Omega_0^2 + {\Omega_i}^2}\\ a_{i1} &= \frac{2(\Omega_0^2 + {\Omega_i}^2 - 1)}{1 - 2\Omega_0\cos\theta_i + \Omega_0^2 + {\Omega_i}^2}\\ a_{i2} &= \frac{1 + 2\Omega_0\cos\theta_i + \Omega_0^2 + {\Omega_i}^2}{1 - 2\Omega_0\cos\theta_i + \Omega_0^2 + {\Omega_i}^2}\end{aligned} \tag{12.12.20}$$

The first-order factor is given by

$$H_0(z) = \begin{cases} \sqrt{\dfrac{1}{1 + \varepsilon_{\text{pass}}^2}} & \text{if } N \text{ is even}\\[2ex] \dfrac{G_0(1 + z^{-1})}{1 + a_{01}z^{-1}} & \text{if } N \text{ is odd}\end{cases} \tag{12.12.21}$$

where

$$G_0 = \frac{\Omega_0}{\Omega_0 + 1}\,, \quad a_{01} = \frac{\Omega_0 - 1}{\Omega_0 + 1} \tag{12.12.22}$$

Example 12.12.1: Using the bilinear transformation and a lowpass analog Chebyshev type 1 prototype, design a lowpass digital filter operating at a rate of 20 kHz and having passband extending to 4 kHz with maximum passband attenuation of 0.5 dB, and stopband starting at 5 kHz with a minimum stopband attenuation of 10 dB.

Then, redesign it such that its magnitude response satisfies $1 \geq |H(f)|^2 \geq 0.98$ in the passband, and $|H(f)|^2 \leq 0.02$ in the stopband.

Solution: The specifications and the prewarped digital frequencies are the same as in Example 12.8.1, that is, $\Omega_{\text{pass}} = 0.7265$ and $\Omega_{\text{stop}} = 1$.

We calculate $\varepsilon_{\text{pass}} = 0.3493$ and $\varepsilon_{\text{stop}} = 3$ and the quantities in Eq. (12.12.5) $e = \varepsilon_{\text{stop}}/\varepsilon_{\text{pass}} = 8.5883$, $w = \Omega_{\text{stop}}/\Omega_{\text{pass}} = 1.3764$. Then, Eq. (12.12.6) gives $N_{\text{exact}} = 3.37$, which is rounded up to $N = 4$. Thus, there are $K = 2$ second-order sections. The 3-dB frequency can be calculated by inverting the bilinear transformation and Eq. (12.12.7):

$$\tan\left(\frac{\pi f_{3\text{dB}}}{f_s}\right) = \Omega_{3\text{dB}} = \Omega_{\text{pass}}\cosh\left(\frac{1}{N}\cosh^{-1}\Big(\frac{1}{\varepsilon_{\text{pass}}}\Big)\right)$$

which gives $f_{3\text{dB}} = 4.2729$ kHz. The actual stopband attenuation is larger than A_{stop} because of the increased value of N. We calculate:

$$A(\Omega_{\text{stop}}) = 10\log_{10}\big(1 + \varepsilon_{\text{pass}}^2 C_N^2(\Omega_{\text{stop}}/\Omega_{\text{pass}})\big) = 14.29 \text{ dB}$$

The parameter a is calculated from Eq. (12.12.10) to be $a = 0.4435$. Then, the coefficients of the digital filter are calculated from Eqs. (12.12.20) and (12.12.22), resulting in the transfer function:

$$H(z) = 0.9441 \cdot \frac{0.3091(1 + z^{-1})^2}{1 - 0.4830z^{-1} + 0.7194z^{-2}} \cdot \frac{0.1043(1 + z^{-1})^2}{1 - 0.9004z^{-1} + 0.3177z^{-2}}$$

The magnitude response squared is shown in the left graph of Fig. 12.12.2. It was computed by inserting the bilinear transformation into Eq. (12.12.1) and evaluating it with `cheby`, that is,

$$|H(f)|^2 = \frac{1}{1 + \varepsilon_{\text{pass}}^2\big(\text{cheby}(N, \tan(\pi f/f_s)/\Omega_{\text{pass}})\big)^2}$$

For the more stringent specifications, we have $A_{\text{pass}} = 0.08774$ dB and $A_{\text{stop}} = 16.9897$ dB. We calculate the parameters:

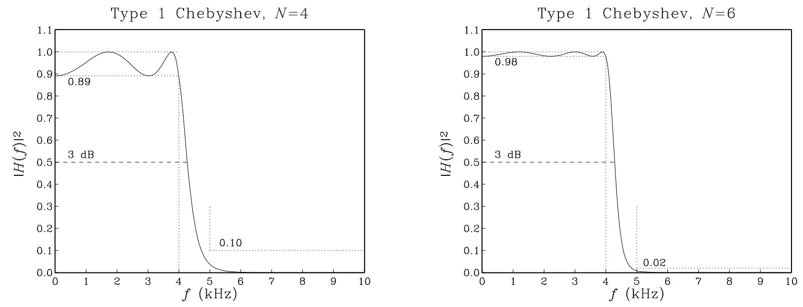


Fig. 12.12.2 Digital lowpass Chebyshev type 1 filters.

$$\varepsilon_{\text{pass}} = 0.1429, \quad \varepsilon_{\text{stop}} = 7, \quad N_{\text{exact}} = 5.44, \quad N = 6, \quad a = 0.4407$$

The 3-dB frequency is found to be $f_{3\text{dB}} = 4.2865$ kHz. The actual stopband attenuation is $A(\Omega_{\text{stop}}) = 21.02$ dB, instead of the nominal one of $A_{\text{stop}} = 16.9897$ dB. The digital filter coefficients are then:

i	G_i	a_{i1}	a_{i2}
0	0.9899		
1	0.3394	−0.4492	0.8069
2	0.2028	−0.6809	0.4920
3	0.0811	−0.9592	0.2837

The gain factor G_0 represents here the overall gain $1/\sqrt{1+\varepsilon_{\text{pass}}^2} = \sqrt{0.98} = 0.9899$. The magnitude response is shown in the right graph of Fig. 12.12.2. By comparison, recall that the two Butterworth filters of Example 12.8.1 had filter orders of 7 and 13, respectively. □

Example 12.12.2: Redesign the highpass digital filters of Example 12.9.1 using a Chebyshev type 1 analog lowpass prototype filter.

Solution: The equivalent analog lowpass specifications $\{\Omega_{\text{pass}}, \Omega_{\text{stop}}, A_{\text{pass}}, A_{\text{stop}}\}$ are the same as in Example 12.9.1. We have $\Omega_{\text{pass}} = 1$ and $\Omega_{\text{stop}} = 1.3764$. Based on the first set of specifications, we find the Chebyshev design parameters:

$$\varepsilon_{\text{pass}} = 0.3493, \quad \varepsilon_{\text{stop}} = 3, \quad N_{\text{exact}} = 3.37, \quad N = 4, \quad a = 0.4435$$

and based on the second set:

$$\varepsilon_{\text{pass}} = 0.1429, \quad \varepsilon_{\text{stop}} = 7, \quad N_{\text{exact}} = 5.44, \quad N = 6, \quad a = 0.4407$$

The 3-dB frequencies can be calculated by inverting:

$$\cot\left(\frac{\pi f_{3\text{dB}}}{f_s}\right) = \Omega_{3\text{dB}} = \Omega_{\text{pass}} \cosh\left(\frac{1}{N}\cosh^{-1}\left(\frac{1}{\varepsilon_{\text{pass}}}\right)\right)$$

which gives for the two specification sets: $f_0 = 4.7170$ and 4.7031 kHz. The actual stopband attenuations attained by the designed filter are in the two cases: $A(\Omega_{\text{stop}}) = 14.29$ and 21.02 dB.

The digital transfer functions are obtained by transforming the analog filter sections by the highpass bilinear transformation $s = (1+z^{-1})/(1-z^{-1})$.

The digital filter coefficients are obtained from Eqs. (12.12.20) and (12.12.22) by changing the sign of the a_{i1} coefficients. We have for the two specification sets:

i	G_i	a_{i1}	a_{i2}
0	0.9441		
1	0.4405	−0.0526	0.7095
2	0.1618	0.5843	0.2314

i	G_i	a_{i1}	a_{i2}
0	0.9899		
1	0.4799	−0.1180	0.8017
2	0.3008	0.2492	0.4524
3	0.1273	0.6742	0.1834

Thus, for example, the first transfer function will be:

$$H(z) = 0.9441 \cdot \frac{0.4405(1-z^{-1})^2}{1-0.0526z^{-1}+0.7095z^{-2}} \cdot \frac{0.1618(1-z^{-1})^2}{1+0.5843z^{-1}+0.2314z^{-2}}$$

The designed magnitude responses are shown in Fig. 12.12.3. □

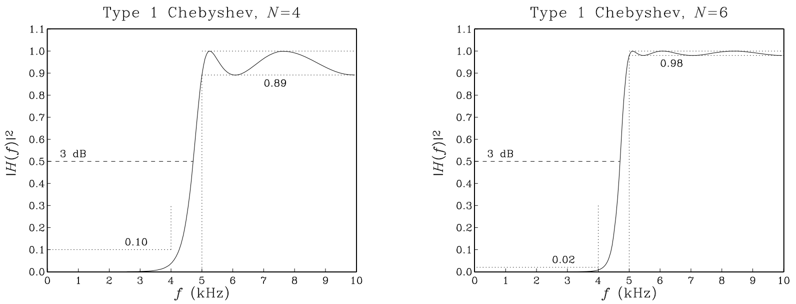


Fig. 12.12.3 Digital highpass Chebyshev type 1 filters.

Next, we discuss type 2 Chebyshev filters. The argument of the Chebyshev polynomials in Eq. (12.12.2) is now $x = \Omega_{\text{stop}}/\Omega$. Therefore, the stopband range $\Omega \geq \Omega_{\text{stop}}$ maps to $0 \leq x \leq 1$ where the Chebyshev polynomial is oscillatory resulting in the stopband ripples.

At the edge of the stopband, $\Omega = \Omega_{\text{stop}}$, we have $x = 1$, $C_N(x) = 1$, and magnitude response equal to $|H(\Omega_{\text{stop}})|^2 = 1/\sqrt{1+\varepsilon_{\text{stop}}^2}$. At large frequencies, $\Omega \to \infty$, we have $x \to 0$. Because the value of $C_N(0)$ depends on N being zero for odd N and unity for even N, it follows that the magnitude response will either tend to zero for odd N or to $1/\sqrt{1+\varepsilon_{\text{stop}}^2}$ for even N.

The zero frequency $\Omega = 0$ corresponds to the limit $x \to \infty$ which causes the Chebyshev polynomials to grow like a power x^N. It follows that the numerator and denominator of the magnitude response (12.12.2) will both diverge but in such a way that

$|H(0)|^2 = 1$. Thus, type 2 filters are always normalized to unity at DC. The filter order N can be determined by requiring the passband specification:

$$|H(\Omega_{\rm pass})|^2 = \frac{C_N^2\left(\dfrac{\Omega_{\rm stop}}{\Omega_{\rm pass}}\right)}{C_N^2\left(\dfrac{\Omega_{\rm stop}}{\Omega_{\rm pass}}\right) + \varepsilon_{\rm stop}^2} = \frac{1}{1+\varepsilon_{\rm pass}^2}$$

It has the *same* solution Eq. (12.12.6) as the type 1 case. Once N is fixed, the 3-dB frequency can be obtained by solving $|H(\Omega_{\rm 3dB})|^2 = 1/2$ which gives the solution:

$$\Omega_{\rm 3dB} = \frac{\Omega_{\rm stop}}{\cosh\big(\dfrac{1}{N}\cosh^{-1}(\varepsilon_{\rm stop})\big)} \tag{12.12.23}$$

Because of the non-trivial numerator in Eq. (12.12.2), the filter will have both zeros and poles. They are solutions of the following equations obtained by replacing $\Omega = -js$:

$$\cosh^2\left(N\cosh^{-1}\big(\frac{\Omega_{\rm stop}}{-js}\big)\right) = 0, \quad \cosh^2\left(N\cosh^{-1}\big(\frac{j\Omega_{\rm stop}}{-js}\big)\right) + \varepsilon_{\rm stop}^2 = 0$$

The zeros are the conjugate pairs $\{z_i, z_i^*\}$:

$$z_i = \frac{j\Omega_{\rm stop}}{\sin\theta_i}, \quad i = 1, 2, \dots, K \tag{12.12.24}$$

where $N = 2K$ or $N = 2K+1$. The poles are essentially the reciprocals of the type 1 poles, that is, the pairs $\{s_i, s_i^*\}$:

$$s_i = \frac{\Omega_{\rm stop}}{\sinh a\,\cos\theta_i + j\cosh a\,\sin\theta_i}, \quad i = 1, 2, \dots, K \tag{12.12.25}$$

In the odd-N case, there is also a real pole at

$$s_0 = -\frac{\Omega_{\rm stop}}{\sinh a} \tag{12.12.26}$$

where the parameter a is the solution of

$$\sinh(Na) = \varepsilon_{\rm stop} \tag{12.12.27}$$

that is,

$$a = \frac{1}{N}\sinh^{-1}(\varepsilon_{\rm stop}) = \frac{1}{N}\ln\left(\varepsilon_{\rm stop} + \sqrt{\varepsilon_{\rm stop}^2 + 1}\right) \tag{12.12.28}$$

The angles θ_i are the *same* as in the type 1 case and given by Eq. (12.12.12). The second-order sections are formed by pairing the zeros and poles in conjugate pairs:

$$H_i(s) = \frac{\left(1 - \dfrac{s}{z_i}\right)\left(1 - \dfrac{s}{z_i^*}\right)}{\left(1 - \dfrac{s}{s_i}\right)\left(1 - \dfrac{s}{s_i^*}\right)} \tag{12.12.29}$$

For convenience, we define the parameters:

$$\Omega_0 = \frac{\Omega_{\rm stop}}{\sinh a}, \qquad \Omega_i = \frac{\Omega_{\rm stop}}{\sin\theta_i}, \quad i = 1, 2, \dots, K \tag{12.12.30}$$

Then, we may express the second-order sections in the form:

$$H_i(s) = \frac{1 + \Omega_i^{-2}s^2}{1 - 2\Omega_0^{-1}\cos\theta_i\, s + (\Omega_0^{-2} + \Omega_i^{-2})s^2}, \quad i = 1, 2, \dots, K \tag{12.12.31}$$

The first-order factor $H_0(s)$ is defined by

$$H_0(s) = \begin{cases} 1, & \text{if } N \text{ is even} \\ \dfrac{\Omega_0}{\Omega_0 + s}, & \text{if } N \text{ is odd} \end{cases} \tag{12.12.32}$$

Again, all filter sections are normalized to unity gain at DC. Under the bilinear transformation $s = (1 - z^{-1})/(1 + z^{-1})$, the analog sections transform to digital versions of the form:

$$H_i(z) = \frac{G_i(1 + b_{i1}z^{-1} + z^{-2})}{1 + a_{i1}z^{-1} + a_{i2}z^{-2}}, \quad i = 1, 2, \dots, K \tag{12.12.33}$$

where the coefficients are computed in terms of the definitions in Eq. (12.12.30):

$$\begin{aligned} G_i &= \frac{1 + \Omega_i^{-2}}{1 - 2\Omega_0^{-1}\cos\theta_i + \Omega_0^{-2} + \Omega_i^{-2}}, \quad b_{i1} = 2\frac{1 - \Omega_i^{-2}}{1 + \Omega_i^{-2}} \\ a_{i1} &= \frac{2(1 - \Omega_0^{-2} - \Omega_i^{-2})}{1 - 2\Omega_0^{-1}\cos\theta_i + \Omega_0^{-2} + \Omega_i^{-2}} \\ a_{i2} &= \frac{1 + 2\Omega_0^{-1}\cos\theta_i + \Omega_0^{-2} + \Omega_i^{-2}}{1 - 2\Omega_0^{-1}\cos\theta_i + \Omega_0^{-2} + \Omega_i^{-2}} \end{aligned} \tag{12.12.34}$$

The first-order factor, if present, is given by

$$H_0(z) = \frac{G_0(1 + z^{-1})}{1 + a_{01}z^{-1}} \tag{12.12.35}$$

where

$$G_0 = \frac{\Omega_0}{\Omega_0 + 1}, \quad a_{01} = \frac{\Omega_0 - 1}{\Omega_0 + 1} \tag{12.12.36}$$

The overall digital transfer function is then:

$$\boxed{H(z) = H_0(z)H_1(z)H_2(z)\cdots H_K(z)} \tag{12.12.37}$$

Example 12.12.3: Redesign the two filters of Example 12.12.1 using a type 2 Chebyshev design.

Solution: The values of $\varepsilon_{\text{pass}}$, $\varepsilon_{\text{stop}}$, N_{exact}, N remain the same as in Example 12.12.1. The parameter a, calculated from Eq. (12.12.28), is for the two specification sets: $a = 0.4546$ and 0.4407.

The actual passband attenuations are slightly higher than the specified ones. Evaluating Eq. (12.12.2) for the two designs, we find the values $A(\Omega_{\text{pass}}) = 0.18$ and 0.03 dB, instead of the required ones 0.5 and 0.08774. The 3-dB frequencies are obtained by inverting:

$$\tan\left(\frac{\pi f_{3\text{dB}}}{f_s}\right) = \Omega_{3\text{dB}} = \frac{\Omega_{\text{stop}}}{\cosh\left(\frac{1}{N}\cosh^{-1}(\varepsilon_{\text{stop}})\right)}$$

which gives the values $f_{3\text{dB}} = 4.7009$ and 4.7031 kHz. For the first specification set, we have two second-order sections whose coefficients are calculated from Eq. (12.12.34), resulting in the transfer function:

$$H(z) = \frac{0.7612(1 + 0.1580z^{-1} + z^{-2})}{1 - 0.0615z^{-1} + 0.7043z^{-2}} \cdot \frac{0.5125(1 + 1.4890z^{-1} + z^{-2})}{1 + 0.5653z^{-1} + 0.2228z^{-2}}$$

For the second specification set, we have three sections with coefficients:

i	G_i	b_{i1}	a_{i1}	a_{i2}
1	0.8137	0.0693	−0.1180	0.8017
2	0.6381	0.6667	0.2492	0.4524
3	0.4955	1.7489	0.6742	0.1834

The designed magnitude responses are shown in Fig. 12.12.4. □

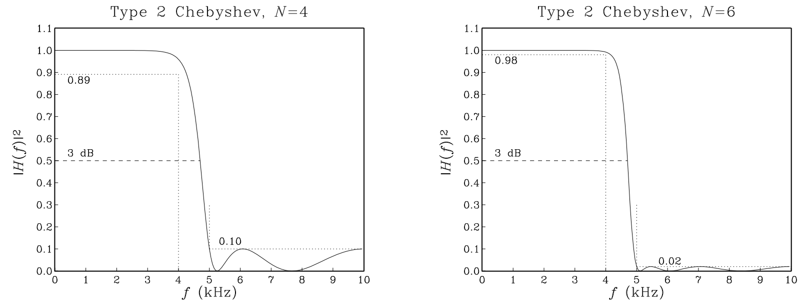


Fig. 12.12.4 Digital lowpass Chebyshev type 2 filters.

The Chebyshev filter can also be transformed by the bandpass or bandstop versions of the bilinear transformation to design bandpass or bandstop digital filters. For example, transforming a type 2 filter by the bandpass transformation in Eq. (12.1.5) gives rise to fourth-order sections of the form:

$$H_i(z) = G_i \frac{1 + b_{i1}z^{-1} + b_{i2}z^{-2} + b_{i1}z^{-3} + z^{-4}}{1 + a_{i1}z^{-1} + a_{i2}z^{-2} + a_{i3}z^{-3} + a_{i4}z^{-4}}, \quad i = 1, 2, \dots, K \qquad (12.12.38)$$

where by symmetry, the numerator coefficients of z^{-1} and z^{-3} are the same. The coefficients are given by:

$$\begin{aligned}
G_i &= \frac{1 + \Omega_i^{-2}}{1 - 2\Omega_0^{-1}\cos\theta_i + \Omega_0^{-2} + \Omega_i^{-2}} \\
b_{i1} &= -\frac{4c\Omega_i^{-2}}{1 + \Omega_i^{-2}}, \quad b_{i2} = \frac{2\left(\Omega_i^{-2}(2c^2 + 1) - 1\right)}{1 + \Omega_i^{-2}} \\
a_{i1} &= \frac{4c\left(\Omega_0^{-1}\cos\theta_i - \Omega_0^{-2} - \Omega_i^{-2}\right)}{1 - 2\Omega_0^{-1}\cos\theta_i + \Omega_0^{-2} + \Omega_i^{-2}}, \quad a_{i2} = \frac{2\left((\Omega_0^{-2} + \Omega_i^{-2})(2c^2 + 1) - 1\right)}{1 - 2\Omega_0^{-1}\cos\theta_i + \Omega_0^{-2} + \Omega_i^{-2}} \\
a_{i3} &= -\frac{4c\left(\Omega_0^{-1}\cos\theta_i + \Omega_0^{-2} + \Omega_i^{-2}\right)}{1 - 2\Omega_0^{-1}\cos\theta_i + \Omega_0^{-2} + \Omega_i^{-2}}, \quad a_{i4} = \frac{1 + 2\Omega_0^{-1}\cos\theta_i + \Omega_0^{-2} + \Omega_i^{-2}}{1 - 2\Omega_0^{-1}\cos\theta_i + \Omega_0^{-2} + \Omega_i^{-2}}
\end{aligned}$$

If N is odd, the first-order analog section $H_0(s)$ is transformed to the same quadratic section $H_0(z)$ given in Section 12.10. Similarly, applying the bandstop transformation of Eq. (12.11.1), the type 2 Chebyshev second-order sections in s are transformed into fourth-order sections in z in the same form of Eq. (12.12.38), but with coefficients given by:

$$\begin{aligned}
G_i &= \frac{1 + \Omega_i^{-2}}{1 - 2\Omega_0^{-1}\cos\theta_i + \Omega_0^{-2} + \Omega_i^{-2}} \\
b_{i1} &= -\frac{4c}{1 + \Omega_i^{-2}}, \quad b_{i2} = \frac{2\left(2c^2 + 1 - \Omega_i^{-2}\right)}{1 + \Omega_i^{-2}} \\
a_{i1} &= -\frac{4c\left(1 - \Omega_0^{-1}\cos\theta_i\right)}{1 - 2\Omega_0^{-1}\cos\theta_i + \Omega_0^{-2} + \Omega_i^{-2}}, \quad a_{i2} = \frac{2\left(2c^2 + 1 - \Omega_0^{-2} - \Omega_i^{-2}\right)}{1 - 2\Omega_0^{-1}\cos\theta_i + \Omega_0^{-2} + \Omega_i^{-2}} \\
a_{i3} &= -\frac{4c\left(1 + \Omega_0^{-1}\cos\theta_i\right)}{1 - 2\Omega_0^{-1}\cos\theta_i + \Omega_0^{-2} + \Omega_i^{-2}}, \quad a_{i4} = \frac{1 + 2\Omega_0^{-1}\cos\theta_i + \Omega_0^{-2} + \Omega_i^{-2}}{1 - 2\Omega_0^{-1}\cos\theta_i + \Omega_0^{-2} + \Omega_i^{-2}}
\end{aligned}$$

The first-order section $H_0(s)$ transforms to the same $H_0(z)$ as that of Section 12.11, but with Ω_0 given by Eq. (12.12.30).

Example 12.12.4: Redesign the bandpass digital filters of Example 12.10.1 using a type 2 Chebyshev analog prototype.

Solution: The prewarped frequencies and bilinear transformation parameter c are as in that example: $\Omega_{\text{pass}} = 0.3249$, $\Omega_{\text{stop}} = 0.4674$, $c = 0.6180$.

The Chebyshev design parameters are for the two specification sets:

$$\varepsilon_{\text{pass}} = 0.3493, \quad \varepsilon_{\text{stop}} = 3, \quad N_{\text{exact}} = 3.14, \quad N = 4, \quad a = 0.4546$$

$$\varepsilon_{\text{pass}} = 0.1429, \quad \varepsilon_{\text{stop}} = 7, \quad N_{\text{exact}} = 5.07, \quad N = 6, \quad a = 0.4407$$

For the first set, there are two fourth-order sections with coefficients:

i	G_i	b_{i1}	b_{i2}	a_{i1}	a_{i2}	a_{i3}	a_{i4}
1	0.7334	−1.9684	2.4015	−1.9604	2.2956	−1.6758	0.7697
2	0.3677	−0.9922	0.2187	−1.4221	0.8671	−0.4101	0.1813

For the second set, there are three fourth-order sections:

i	G_i	b_{i1}	b_{i2}	a_{i1}	a_{i2}	a_{i3}	a_{i4}
1	0.7840	−2.0032	2.4793	−2.0118	2.4413	−1.8265	0.8501
2	0.5858	−1.7205	1.8472	−1.7286	1.6780	−1.1223	0.5094
3	0.3159	−0.5802	−0.7026	−1.3122	0.5868	−0.1879	0.0904

The designed magnitude responses are shown in Fig. 12.12.5. □

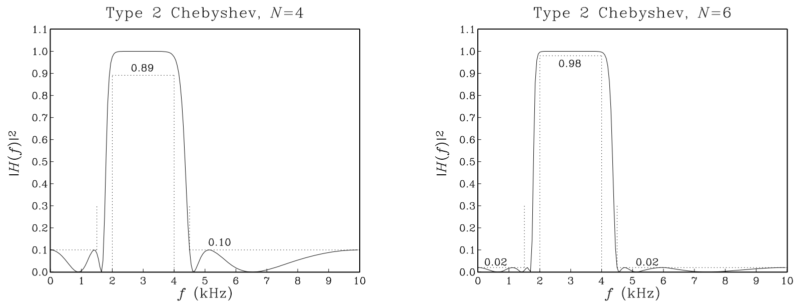


Fig. 12.12.5 Digital bandpass Chebyshev type 2 filters.

Example 12.12.5: Redesign the bandstop digital filters of Example 12.11.1 using a type 2 Chebyshev analog prototype.

Solution: The prewarped frequencies and bilinear transformation parameter c are as in that example: $\Omega_{\text{pass}} = 1.9626$, $\Omega_{\text{stop}} = 2.7121$, $c = 0.6597$.

For the two specification sets, the exact Chebyshev orders are $N_{\text{exact}} = 3.35$ and 5.40, which round up to $N = 4$ and 6, respectively. The other Chebyshev parameters remain the same as in Example 12.12.4. For the first set, the fourth-order sections have coefficients:

i	G_i	b_{i1}	b_{i2}	a_{i1}	a_{i2}	a_{i3}	a_{i4}
1	0.8727	−2.3644	3.1438	−2.2003	2.6965	−1.9264	0.7924
2	0.7442	−2.5872	3.6287	−2.2339	2.6565	−1.6168	0.5323

For the second set, there are three fourth-order sections:

i	G_i	b_{i1}	b_{i2}	a_{i1}	a_{i2}	a_{i3}	a_{i4}
1	0.9074	−2.3417	3.0945	−2.2171	2.7626	−2.0326	0.8601
2	0.8009	−2.4708	3.3754	−2.2137	2.6612	−1.7441	0.6441
3	0.7412	−2.6149	3.6889	−2.2524	2.6929	−1.6241	0.5238

The designed magnitude responses are shown in Fig. 12.12.6. □

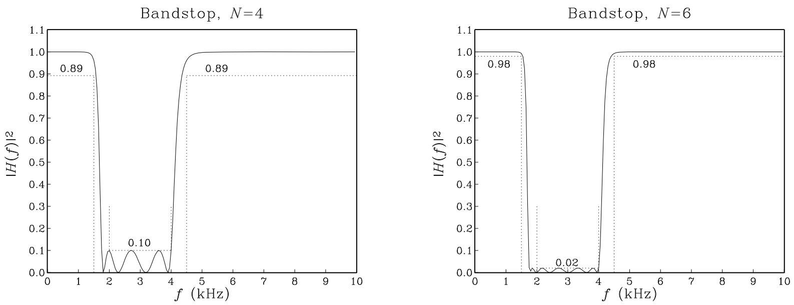


Fig. 12.12.6 Digital bandstop Chebyshev type 2 filters.

12.13 Problems

12.1 Consider the peaking filter of Eq. (12.3.21). Derive an analytical expression for its impulse response $h(n)$ in terms of the parameters ω_0 and $\Delta\omega$. Show that its transient (ϵ-level) time constant is given by:

$$n_{\text{eff}} = \frac{\ln\epsilon}{\ln\rho}, \qquad \rho = \sqrt{\frac{1-\beta}{1+\beta}}$$

12.2 Consider the peaking filter of Eq. (12.3.21). Taking the limit $\Delta\omega \to 0$ and keeping only the lowest-order terms in $\Delta\omega$, show that the pole radius R is related approximately to $\Delta\omega$ by Eq. (6.4.4), that is, $\Delta\omega \simeq 2(1-R)$. Show also that $n_{\text{eff}} \simeq -2\ln\epsilon/\Delta\omega$.

12.3 Verify the complementarity properties for the filters of Eqs. (12.3.7) and (12.3.21):

$$H_{\text{notch}}(z)+H_{\text{peak}}(z)=1, \quad |H_{\text{notch}}(\omega)|^2+|H_{\text{peak}}(\omega)|^2=1, \qquad (\text{when } G_B^2 = 1/2)$$

12.4 Consider the three choices for the bandwidth reference gain in parametric equalizer filters: $G_B^2 = (G_0^2+G^2)/2$ (arithmetic mean), $G_B^2 = G_0G$ (geometric mean), and $G_B^2 = 2G_0^2G^2/(G_0^2+G^2)$ (harmonic mean). For each case, discuss how the design parameter β of Eq. (12.4.6) simplifies. For the geometric mean case [292], show that if you design two boost and cut digital filters centered at the same frequency and having equal bandwidths and equal and opposite boost/cut gains (in dB), then their magnitude responses will be related by $|H_{\text{boost}}(\omega)|^2|H_{\text{cut}}(\omega)|^2 = G_0^4$. Show that the more general weighted geometric mean choice $G_B^2 = G_0^{1-c}G^{1+c}$, $0 \le c < 1$ also satisfies this property.

12.5 *Computer Experiment: Peaking and Notching Filters.* Reproduce the results and graphs of Example 12.3.1. Plot also the phase responses of all the filters. For each filter, draw its canonical realization and write the corresponding sample processing algorithm. (You may use the MATLAB function `parmeq.m` to design them.)

Calculate the 5% time constants of the filters. Send a unit impulse $\delta(n)$ into the input and calculate and plot the impulse responses $h(n)$ of these filters. You must compute $h(n)$ for a period of at least two 5% time constants. (You may use the routines `sos.c` or `sos.m` to implement them.)

12.6 *Computer Experiment: Parametric EQ Filter Design.* Reproduce the results and graphs of Examples 12.4.1 and 12.4.2. Plot also the phase responses of all the filters. (You may use the MATLAB function `parmeq.m` to design them.)

For the filters of Example 12.4.1, compute their 5% time constants and compute and plot their impulse responses $h(n)$ versus n.

12.7 *Computer Experiment: Lowpass/Highpass EQ Filters.* Write a MATLAB function (similar to `parmeq.m`) to design the lowpass or highpass shelving equalizer filters defined by Eqs. (12.4.9) and (12.4.11).

Use the function to reproduce the results and graphs of Examples 12.2.1 and 12.2.2.

12.8 *Computer Experiment: Periodic Comb/Notch Parametric EQ Design.* Reproduce the results and graphs of Example 12.5.1. Plot also the phase responses of all the filters. (You may use the MATLAB function `combeq.m` to design them.)

Write a C or MATLAB function, say `combfilt`, that implements the sample processing algorithm of the time domain operation of the comb filter. It must have usage:

```
     y = combfilt(a, b, c, D, w, x);        (C version)
[y, w] = combfilt(a, b, c, D, w, x);        (MATLAB version)
```

where $\mathbf{w}$ is the $(D+1)$-dimensional delay-line buffer, and $\{x, y\}$ are the input and output samples. The parameters $\{a, b, c\}$ are generated by `combeq`. For the C case, you may use a circular buffer. Using this function, calculate and plot the impulse response $h(n)$ of all the designed filters.

12.9 The passband and stopband specifications are defined somewhat differently in FIR and IIR designs. Discuss these differences and explain why the parameter sets $\{\delta_{\text{pass}}, \delta_{\text{stop}}\}$ of Eq. (11.3.5) and $\{\varepsilon_{\text{pass}}, \varepsilon_{\text{stop}}\}$ of Eq. (12.6.4) are appropriate for FIR and IIR designs.

12.10 The parameters N and Ω_0 of an analog Butterworth filter are determined by solving the two specification equations $A(\Omega_{\text{pass}}) = A_{\text{pass}}$, $A(\Omega_{\text{stop}}) = A_{\text{stop}}$. The resulting filter order is then rounded up to the next integer value N.

Using this slightly larger N, show that if Ω_0 is found from the passband specification, that is, by solving $A(\Omega_{\text{pass}}) = A_{\text{pass}}$, then the stopband specification is more than satisfied, that is, $A(\Omega_{\text{stop}}) > A_{\text{stop}}$.

Similarly, show that if we find Ω_0 from the stopband specification $A(\Omega_{\text{stop}}) = A_{\text{stop}}$, then the passband specification is more than satisfied, that is, $A(\Omega_{\text{pass}}) < A_{\text{pass}}$.

12.11 Using the bilinear transformation and a lowpass analog Butterworth prototype filter, design a *lowpass* digital filter operating at a rate of 40 kHz and having the following specifications: $f_{\text{pass}} = 10$ kHz, $A_{\text{pass}} = 3$ dB, $f_{\text{stop}} = 15$ kHz, $A_{\text{stop}} = 35$ dB. Carry out all the design steps by hand.

Draw the cascade realization form and write the difference equations and the corresponding sample processing algorithm implementing this realization in the time domain.

12.12 Using the bilinear transformation method and a Butterworth lowpass analog prototype, design (by hand) a digital *highpass* filter operating at a rate of 10 kHz and having passband and stopband frequencies of 3 kHz and 2 kHz, respectively. The maximum passband and minimum stopband attenuations are required to be 0.5 dB and 10 dB respectively.

a. What are the actual maximum passband and minimum stopband attenuations in dB achieved by the designed filter?

b. Draw the cascade realization and write the corresponding difference equations describing the time-domain operation of the filter.

c. Give a simple closed-form expression of the magnitude response of the designed filter as a function of the variable $\cot(\omega/2)$. Sketch the magnitude response over the frequency interval $0 \le f \le 20$ kHz.

12.13 Show that the generalized bilinear transformation

$$s = \frac{1 - 2cz^{-1} + z^{-2}}{1 - z^{-2}} \tag{12.13.1}$$

maps the left-hand s-plane onto the inside of the unit circle of the z-plane, provided the real constant c is such that $|c| \le 1$.

12.14 In the design of bandpass/bandstop filters, we found that the constant c of the generalized bilinear transformation of Eq. (12.13.1) was given by an expression of the form:

$$c = \frac{\sin(\omega_1 + \omega_2)}{\sin(\omega_1) + \sin(\omega_2)}$$

Show that it satisfies the stability condition $|c| \le 1$, regardless of the values of ω_1 and ω_2 in the interval $[0, \pi]$.

12.15 Using a third-order analog lowpass Butterworth prototype filter and the bandpass bilinear transformation of Eq. (12.13.1), design a *digital bandpass* filter operating at 20 kHz and having attenuation of 0.5 dB at the frequencies 2 kHz and 8 kHz.

What is the transfer function of the designed filter? What are the upper and lower 3-dB frequencies of this filter? What is its center frequency in kHz? Sketch roughly the magnitude response over the range $0 \le f \le 30$ kHz.

12.16 Carry out the algebra in Eq. (12.8.1) to show the coefficient equations (12.8.2) for designing a digital lowpass Butterworth filter. Verify also Eq. (12.8.4).

Repeat for the highpass case, Eqs. (12.9.3)—(12.9.6). Repeat for the bandpass case, Eqs. (12.10.4)–(12.10.7).

12.17 Prove Eqs. (12.10.8) for the left and right 3-dB frequencies of a bandpass Butterworth design.

12.18 The bandpass and bandstop Butterworth designs discussed in Sections 12.10 and 12.11 match the *passband* specifications exactly and use the more conservative of the two stopbands. Instead, if we were to match the stopbands exactly and use the more conservative passband, what changes in the design procedure should we make?

12.19 Carry out the algebra of the bilinear transformation to verify the design equations of Eqs. (12.12.20) and (12.12.34) for designing digital lowpass type 1 and type 2 Chebyshev filters.

12.20 Equations (12.12.8) and (12.12.13) define the Chebyshev poles s_i and the quantities $\{\Omega_0, \Omega_i\}$. Show that they satisfy the following relationship, which is used in the second-order Chebyshev sections (12.12.14):

$$|s_i|^2 = \Omega_0^2 + \Omega_i{}^2$$

12.21 The IIR cascade filtering routines `cas.c` or `cas.m` are appropriate for cascading second-order sections and can be used in the lowpass and highpass designs. In bandpass and bandstop designs, however, we have the cascade of fourth-order sections whose coefficients are stored in the $K\times 5$ matrices A and B that are generated by the filter design functions, such as `bpcheb2.m`, where K is the number of fourth-order sections.

Write C and/or MATLAB versions of the routine `cas`, say `cas4`, that works with fourth-order sections. Its inputs must be the matrices A and B, a $K\times 5$ state matrix W whose rows are the state vectors of the cascaded fourth-order sections, and the current input sample x. Its outputs must be the current output sample y and the updated state matrix W. It must have usage:

```
     y = cas4(K, B, A, W, x);        (C version)
[y, W] = cas4(K, B, A, W, x);        (MATLAB version)
```

It must call K times a single fourth-order section routine, like the `sos` routine. Then, write C and/or MATLAB filtering routines, like `casfilt` of Problem 7.15, that can filter a vector of input samples producing a vector of output samples.

12.22 *Computer Experiment: Butterworth Digital Filter Designs.* Reproduce all the results and graphs of the lowpass, highpass, bandpass, and bandstop Butterworth design Examples 12.8.1–12.11.1. You may use the MATLAB functions `lhbutt.m` and `bpsbutt.m` to design the filters.

For each design, also do the following: Plot the phase response of the filter. Compute the filter's 5% time constant (you will need to use MATLAB's root finder `roots`). Then, using the routines `cas.c` or `cas.m`, (or `cas4.c`, `cas4.m` of Problem 12.21), compute and plot the impulse response $h(n)$ of the filter over a period lasting two time constants.

12.23 *Computer Experiment: Chebyshev Digital Filter Designs.* Reproduce all the results and graphs of the lowpass, highpass, bandpass, and bandstop Chebyshev design Examples 12.12.1–12.12.5. You may use the MATLAB functions `lhcheb1`, `lhcheb2`, `bpcheb2`, and `bscheb2` to design the filters.

For each design, also do the following: Plot the phase response of the filter. Compute the filter's 5% time constant. Then, compute and plot the impulse response $h(n)$ of the filter over a period lasting two time constants.

Since the specifications of the filters are the same as those of Problem 12.22, compare the Butterworth and Chebyshev designs in terms of their order N and their phase response.

In both problems, the frequency responses can be computed with the included MATLAB functions `cas2can` and `dtft`. For example, the frequency response of a type 2 bandstop design can be computed as follows:

```
[A, B, P] = bscheb2(fs, fpa, fpb, fsa, fsb, Apass, Astop);
a = cas2can(A);                            direct-form denominator
b = cas2can(B);                            direct-form numerator
w = (0:NF-1) * pi / NF;                    NF frequencies over [0, π]
H = dtft(b, w) ./ dtft(a, w);              compute H(ω) = N(ω)/D(ω)
```

13

Elliptic Filter Design

13.1 Introduction

Elliptic filters [302–330] also known as Cauer or Zolotarev filters, achieve the smallest filter order for the same specifications, or, the narrowest transition width for the same filter order, as compared to other filter types. On the negative side, they have the most nonlinear phase response over their passband.[†] The following table compares the basic filter types with regard to filter order and phase response:

smaller order, or narrower transition	↓	Bessel Butterworth Chebyshev Elliptic	↑	more linear phase over the passband

In this chapter, we are primarily concerned with elliptic filters. But we will also discuss briefly the design of Butterworth, Chebyshev-1, and Chebyshev-2 filters and present a unified method of designing all cases. We also discuss the design of digital IIR filters using the bilinear transformation method.

The typical "brick wall" specifications for an analog lowpass filter are shown in Fig. 13.1.1 for the case of a monotonically decreasing Butterworth filter, normalized to unity gain at DC.

The passband and stopband gains G_p, G_s and the corresponding attenuations in dB are defined in terms of the "ripple" parameters $\varepsilon_p, \varepsilon_s$ as follows:

$$G_p = \frac{1}{\sqrt{1+\varepsilon_p^2}} = 10^{-A_p/20}\,,\qquad G_s = \frac{1}{\sqrt{1+\varepsilon_s^2}} = 10^{-A_s/20} \tag{13.1.1}$$

which can be inverted to give:

$$\begin{aligned} A_p &= -20\log_{10}G_p = 10\log_{10}(1+\varepsilon_p^2) \\ A_s &= -20\log_{10}G_s = 10\log_{10}(1+\varepsilon_s^2) \end{aligned} \quad\Rightarrow\quad \begin{aligned} \varepsilon_p &= \sqrt{G_p^{-2}-1} = \sqrt{10^{A_p/10}-1} \\ \varepsilon_s &= \sqrt{G_s^{-2}-1} = \sqrt{10^{A_s/10}-1} \end{aligned} \tag{13.1.2}$$

[†] Bessel filters have, by design, the most linear phase over their passband, discussed briefly in Sec. 14.4.4.

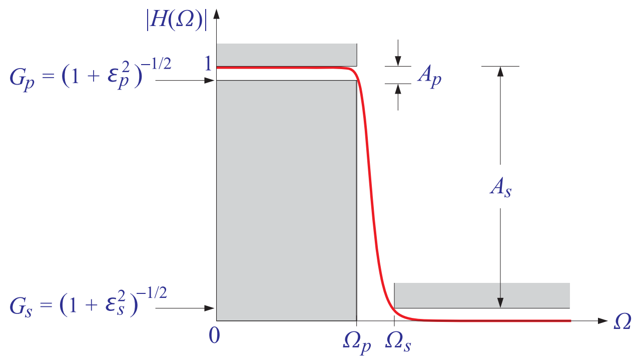

$|H(\Omega)|$
$G_p = (1+\varepsilon_p^2)^{-1/2}$
1
A_p
A_s
$G_s = (1+\varepsilon_s^2)^{-1/2}$
0
Ω_p
Ω_s
Ω

Fig. 13.1.1 Brick wall specifications for a Butterworth filter.

Associated with these specifications, we define the following design parameters k, k_1:

$$\boxed{k = \frac{\Omega_p}{\Omega_s}\,, \quad k_1 = \frac{\varepsilon_p}{\varepsilon_s}} \tag{13.1.3}$$

where k, k_1 are known as the *selectivity* and *discrimination* parameters, respectively. Both are less than unity. A narrow transition width would imply that $k \lesssim 1$, whereas a deep stopband or a flat passband would imply that $k_1 \ll 1$. Thus, for most practical desired specifications, we will have $k_1 \ll k \lesssim 1$.

The magnitude responses of the analog lowpass Butterworth, Chebyshev, and elliptic filters are given as functions of the analog frequency Ω by:[†]

$$\boxed{|H(\Omega)|^2 = \frac{1}{1+\varepsilon_p^2 F_N^2(w)}}\,, \quad w = \frac{\Omega}{\Omega_p} \tag{13.1.4}$$

where N is the filter order and $F_N(w)$ is a function of the normalized frequency w:

$$\boxed{F_N(w) = \begin{cases} w^N, & \text{Butterworth} \\ C_N(w), & \text{Chebyshev, type-1} \\ \left[k_1 C_N(k^{-1}w^{-1})\right]^{-1}, & \text{Chebyshev, type-2} \\ \mathrm{cd}(NuK_1, k_1), \quad w = \mathrm{cd}(uK, k), & \text{Elliptic} \end{cases}} \tag{13.1.5}$$

where $C_N(x)$ is the order-N Chebyshev polynomial, that is, $C_N(x) = \cos(N\cos^{-1}x)$, and $\mathrm{cd}(x,k)$ denotes the *Jacobian elliptic function* **cd** with modulus k and real quarter-period K.

[†] Ω is in units of radians per second and is related to the frequency f in Hz by $\Omega = 2\pi f$. For digital filter design, Ω is related to the physical digital frequency $\omega = 2\pi f/f_s$ via the appropriate bilinear transformation, e.g., for a lowpass design, $\Omega = \tan(\omega/2)$.

The Chebyshev-2 definition looks a little peculiar, but it is equivalent to that given previously in Sec. 12.12. Indeed, noting that, $k^{-1}w^{-1} = (\Omega_s/\Omega_p)(\Omega_p/\Omega) = \Omega_s/\Omega$, and that, $\varepsilon_s = \varepsilon_p k_1^{-1}$, we can rewrite:

$$|H(\Omega)|^2 = \frac{1}{1+\varepsilon_p^2 k_1^{-2}/C_N^2(k^{-1}w^{-1})} = \frac{1}{1+\varepsilon_s^2/C_N^2(\Omega_s/\Omega)} \tag{13.1.6}$$

The normalized frequency $w = 1$ corresponds to the *passband edge* frequency $\Omega = \Omega_p$, whereas the value $w = \Omega_s/\Omega_p = 1/k$ corresponds to the *stopband edge*, $\Omega = \Omega_s$. The requirement that the passband and stopband specifications are met at the corners $\Omega = \Omega_p$ or $w = 1$ and $\Omega = \Omega_s$ or $w = k^{-1}$ gives rise to the following design conditions:

$$\begin{aligned} |H(\Omega_p)|^2 &= \frac{1}{1+\varepsilon_p^2 F_N^2(1)} = \frac{1}{1+\varepsilon_p^2} \quad \Rightarrow \quad F_N(1) = 1 \\ |H(\Omega_s)|^2 &= \frac{1}{1+\varepsilon_p^2 F_N^2(k^{-1})} = \frac{1}{1+\varepsilon_s^2} \quad \Rightarrow \quad F_N(k^{-1}) = \frac{\varepsilon_s}{\varepsilon_p} = \frac{1}{k_1} \end{aligned} \tag{13.1.7}$$

Thus, in all four cases, the function $F_N(w)$ is normalized such that $F_N(1) = 1$ and must satisfy the following "degree equation" that relates the three design parameters N, k, k_1:[†]

$$\boxed{F_N(k^{-1}) = k_1^{-1}} \quad \text{(degree equation)} \tag{13.1.8}$$

In particular, we find that the degree equation takes the following forms in the Butterworth and both Chebyshev cases:

$$\begin{aligned} k^{-N} &= k_1^{-1} \quad \Rightarrow \quad N = \frac{\ln(k_1^{-1})}{\ln(k^{-1})} = \frac{\ln(\varepsilon_s/\varepsilon_p)}{\ln(\Omega_s/\Omega_p)} \\ C_N(k^{-1}) &= k_1^{-1} \quad \Rightarrow \quad N = \frac{\mathrm{acosh}(k_1^{-1})}{\mathrm{acosh}(k^{-1})} = \frac{\mathrm{acosh}(\varepsilon_s/\varepsilon_p)}{\mathrm{acosh}(\Omega_s/\Omega_p)} \end{aligned} \tag{13.1.9}$$

These equations may be solved for any one of the three parameters N, k, k_1 in terms of the other two. Often, in practice, one specifies Ω_p, Ω_s and $\varepsilon_p, \varepsilon_s$, which fix the values of k, k_1. Then, Eqs. (13.1.9) may be solved for N, which must be rounded up to the next integer value.

Since N is slightly increased, Eqs. (13.1.9) may be used to recompute either k in terms of N, k_1, or alternatively, k_1 in terms of N, k. Because k is an increasing function of N, and k_1 a decreasing one, it follows that the final design will have slightly improved specifications, either by making the transition width narrower (k gets nearer to 1), or by increasing the stopband or decreasing the passband attenuations (k_1 becomes smaller).

Fig. 13.1.2 shows an example designed with Butterworth, Chebyshev types-1&2, and elliptic filters. Fig. 13.1.3 shows the corresponding phase responses (their piece-wise nature arises because they are always wrapped modulo 2π to lie within $[-\pi, \pi]$.) The specifications were as follows:

$$\begin{aligned} f_p &= 4, \quad G_p = 0.95, \quad A_p = -20\log_{10} G_p = 0.4455 \text{ dB} \\ f_s &= 4.5, \quad G_s = 0.05, \quad A_s = -20\log_{10} G_s = 26.0206 \text{ dB} \end{aligned} \tag{13.1.10}$$

[†] For Chebyshev-2, it is $F_N(1) = 1$ that provides the desired relationship among N, k, k_1.

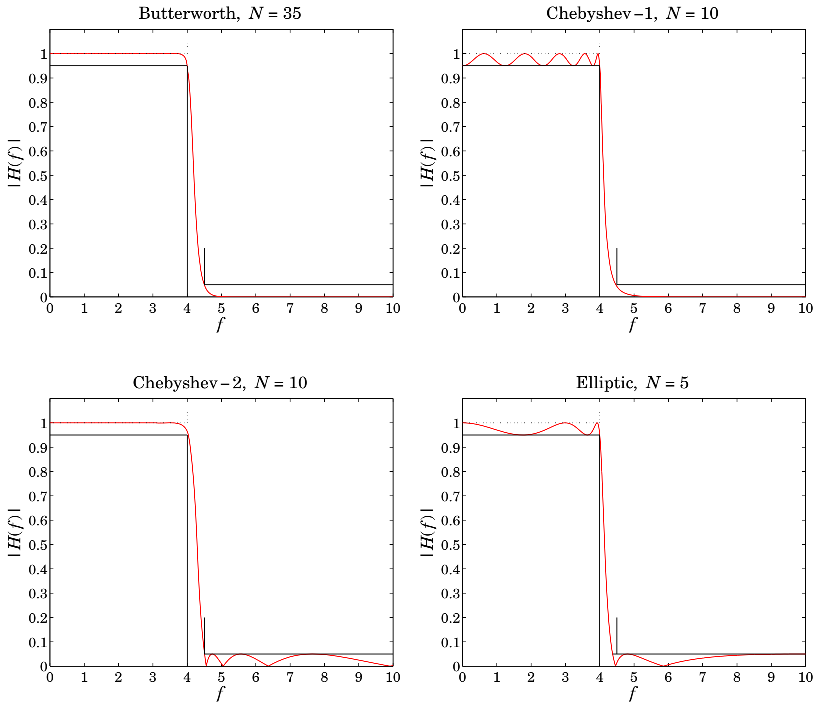


Fig. 13.1.2 Butterworth, Chebyshev, and elliptic design examples.

where the radian frequencies were computed as $\Omega_p = 2\pi f_p$, $\Omega_s = 2\pi f_s$. The design parameters k, k_1 were computed to be:

$$\begin{aligned} k &= \frac{\Omega_p}{\Omega_s} = \frac{f_p}{f_s} = 0.8889 \\ k_1 &= \frac{\varepsilon_p}{\varepsilon_s} = \frac{\sqrt{10^{A_p/10}-1}}{\sqrt{10^{A_s/10}-1}} = \frac{\sqrt{10^{0.04455}-1}}{\sqrt{10^{2.60206}-1}} = 0.0165 \end{aligned} \qquad (13.1.11)$$

We note that the elliptic design has the smallest filter order N, and the Butterworth, the largest. The difference between the Chebyshev designs is that type-1 is equiripple in the *passband*, whereas type-2 is equiripple in the *stopband*. It follows from Eq. (13.1.5) that the replacement

$$C_N(w) \longrightarrow \frac{1}{k_1 C_N(k^{-1}w^{-1})}$$

causes the type-1 case to be replaced by the type-2 case, and the equal ripples in the passband to become equal ripples in the stopband.

In the elliptic case, we want a filter that is equiripple in *both* the passband and the stopband, as shown in Fig. 13.1.2. This will be accomplished if we can find a filter function $F_N(w)$ that is equiripple in the passband and satisfies the identity:

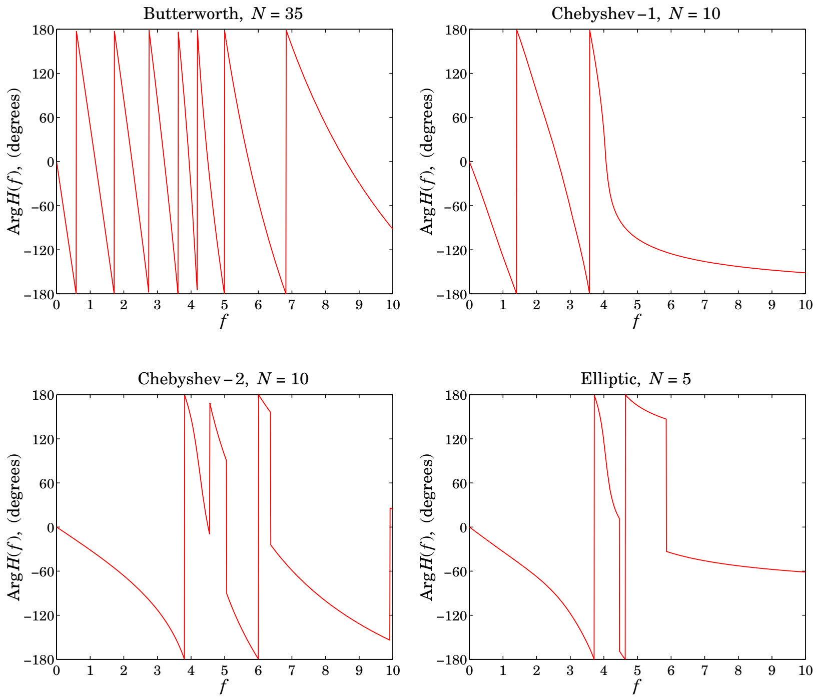


Fig. 13.1.3 Phase responses.

$$\boxed{F_N(w) = \frac{1}{k_1 F_N(k^{-1}w^{-1})}} \qquad (13.1.12)$$

which is equivalent to $\varepsilon_p F_N(\Omega/\Omega_p) = \varepsilon_s / F_N(\Omega_s/\Omega)$, so that in this case the magnitude response can be written as follows and will have ripples in both the passband and the stopband:

$$|H(\Omega)|^2 = \frac{1}{1+\varepsilon_p^2 F_N^2(\Omega/\Omega_p)} = \frac{1}{1+\varepsilon_s^2/F_N^2(\Omega_s/\Omega)} \qquad (13.1.13)$$

We note that the Butterworth filter also satisfies Eq. (13.1.12), because of the degree equation $k_1 = k^N$, but in this case $F_N(w)$ is monotonic in both the passband and the stopband.

13.2 Jacobian Elliptic Functions

Jacobian elliptic functions are a fascinating subject with many applications [313–320]. Here, we give some definitions and discuss some of the properties that are relevant in filter design [309]. The elliptic function, $w = \mathrm{sn}(z,k)$, is defined indirectly through the elliptic integral:

$$w = \sin\phi\,,\quad z = \int_0^{\phi}\frac{d\theta}{\sqrt{1-k^2\sin^2\theta}} = \int_0^{w}\frac{dt}{\sqrt{(1-t^2)(1-k^2t^2)}} \tag{13.2.1}$$

where the second integral was obtained from the first by the change of variables $t = \sin\theta$ and $w = \sin\phi$. The parameter k is called the *elliptic modulus*[†] and is assumed to be a real number in the interval, $0 \le k \le 1$. Thus, writing, $\phi = \phi(z,k)$, the function **sn** is defined as:

$$w = \mathrm{sn}(z,k) = \sin\phi(z,k) \tag{13.2.2}$$

The three related elliptic functions, **cn**, **dn**, **cd**, are defined in terms of **sn** by:

$$\begin{aligned} w &= \mathrm{cn}(z,k) = \cos\phi(z,k) = \cos\big[\arcsin\big(\mathrm{sn}(z,k)\big)\big] \\ w &= \mathrm{dn}(z,k) = \sqrt{1-k^2\,\mathrm{sn}^2(z,k)} \\ w &= \mathrm{cd}(z,k) = \frac{\mathrm{cn}(z,k)}{\mathrm{dn}(z,k)} \end{aligned} \tag{13.2.3}$$

In filter design, only the functions **sn** and **cd** are needed. In the limits $k = 0$ and $k = 1$, we obtain the trigonometric and hyperbolic functions, respectively:

$$\begin{array}{ll} \mathrm{sn}(z,0) = \sin z\,, & \mathrm{sn}(z,1) = \tanh z \\ \mathrm{cn}(z,0) = \cos z\,, & \mathrm{cn}(z,1) = \mathrm{sech}\, z \\ \mathrm{dn}(z,0) = 1\,, & \mathrm{dn}(z,1) = \mathrm{sech}\, z \\ \mathrm{cd}(z,0) = \cos z\,, & \mathrm{cd}(z,1) = 1 \end{array} \tag{13.2.4}$$

The functions **sn**, **cn**, **dn**, **cd** satisfy the following properties, where $k' = (1-k^2)^{1/2}$:

$$\begin{aligned} &\mathrm{sn}^2(z,k) + \mathrm{cn}^2(z,k) = 1 \\ &k^2\,\mathrm{sn}^2(z,k) + \mathrm{dn}^2(z,k) = 1 \\ &\mathrm{dn}^2(z,k) - k^2\,\mathrm{cn}^2(z,k) = k'^2 \\ &k'^2\,\mathrm{sn}^2(z,k) + \mathrm{cn}^2(z,k) = \mathrm{dn}^2(z,k) \\ &\mathrm{sn}^2(z,k) = \frac{1-\mathrm{cd}^2(z,k)}{1-k^2\,\mathrm{cd}^2(z,k)} \end{aligned} \tag{13.2.5}$$

The value of z at $\phi = \pi/2$ in Eq. (13.2.1) defines the so-called *complete elliptic integral* of the first kind, which is denoted by $K(k)$ or simply K:

$$\boxed{K = \int_0^{\pi/2}\frac{d\theta}{\sqrt{1-k^2\sin^2\theta}}} \quad \text{(complete elliptic integral)} \tag{13.2.6}$$

It follows from the definitions (13.2.2) and (13.2.3) that, $\mathrm{sn}(K,k) = 1$, and, $\mathrm{cd}(K,k) = 0$. Associated with an elliptic modulus k, we may define the *complementary modulus* $k' = (1-k^2)^{1/2}$ and its associated complete elliptic integral $K(k')$ denoted by $K'(k)$ or simply K':

[†] In some discussions, as well as in MATLAB, the parameter $m = k^2$ is used instead of k.

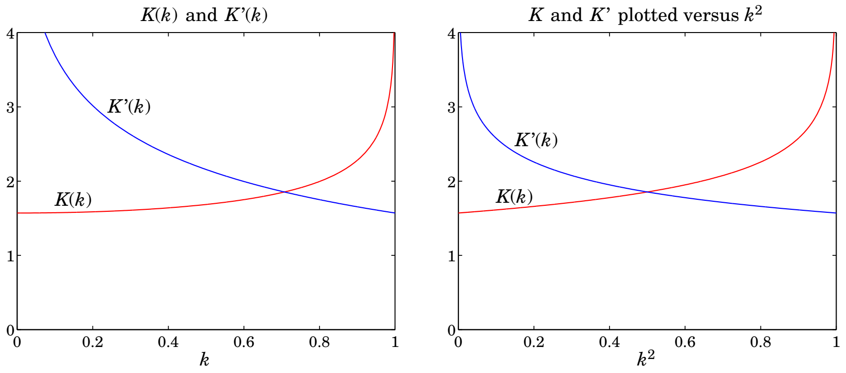


Fig. 13.2.1 Complete elliptic integrals $K(k)$ and $K'(k)$, where $K(0) = K'(1) = \pi/2$.

$$K' = \int_0^{\pi/2}\frac{d\theta}{\sqrt{1-k'^2\sin^2\theta}} = \int_0^{\pi/2}\frac{d\theta}{\sqrt{1-(1-k^2)\sin^2\theta}}\,,\qquad k' = \sqrt{1-k^2} \tag{13.2.7}$$

The quantities K, K' are referred to as *quarter periods*. At the end-point, $k = 0$, we have $K = \pi/2$, $K' = \infty$; at the other end, $k = 1$, we have $K = \infty$, $K' = \pi/2$. Fig. 13.2.1 shows a plot of K, K' versus k. The curves intersect at $k = 1/\sqrt{2}$ and are symmetric if plotted versus k^2.

The significance of the quarter periods K, K' is that **sn** and **cd** are *doubly-periodic functions* in the z-plane with a real period $4K$ and a complex period $2jK'$.

Fig. 13.2.2 shows the graphs of, $w = \mathrm{sn}(uK,k)$, and, $w = \mathrm{cd}(uK,k)$, plotted over two real periods for different values of k. The argument of the functions is $z = uK$, where u is in units of the quarter period K, so that the plotting range $-4 \le u \le 4$ is equivalent to two real periods $-4K \le z \le 4K$.

For $k \le 0.5$, $\mathrm{sn}(uK,k)$ and $\mathrm{cd}(uK,k)$ are almost identical to the trigonometric functions $\sin(u\pi/2)$ and $\cos(u\pi/2)$, that is, to the limiting case $k = 0$.

We note that $\mathrm{sn}(z,k)$ is an odd function of z, and $\mathrm{cd}(z,k)$, an even function. Moreover, by analogy with the property that a cosine and sine are shifted relative to each other by a quarter period $2\pi/4 = \pi/2$, that is, $\cos z = \sin(z+\pi/2) = \sin(\pi/2-z)$, the functions **cd** and **sn** are shifted by a quarter period K, satisfying the following identity, which is valid for all complex values of z and can be used as an alternative definition of the function **cd**:

$$\mathrm{cd}(z,k) = \mathrm{sn}(z+K,k) = \mathrm{sn}(K-z,k) \tag{13.2.8}$$

This property is evident in Fig. 13.2.2. The functions $\mathrm{dn}(uK,k)$ and $\mathrm{dn}(uK',k')$ are plotted in Fig. 13.2.3 for the values $k = 0.8$ and $k' = \sqrt{1-k^2} = 0.6$. Because $\mathrm{dn}(uK,k) = \sqrt{1-k^2\,\mathrm{sn}^2(uK,k)}$, we have the range of variation $k' \le \mathrm{dn}(uK,k) \le 1$, for real u, and similarly, $k \le \mathrm{dn}(uK',k') \le 1$.

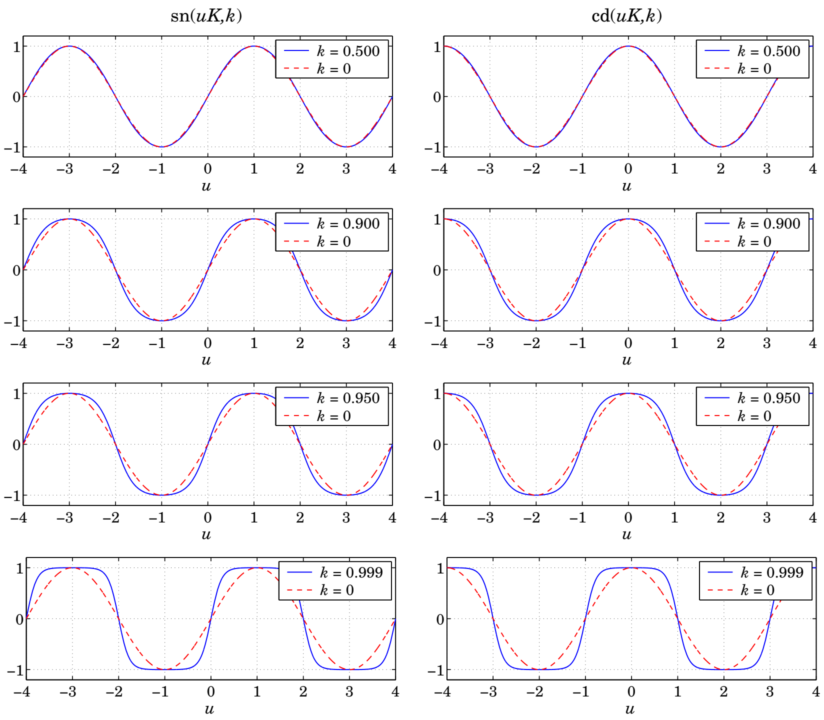


Fig. 13.2.2 Elliptic functions sn and cd.

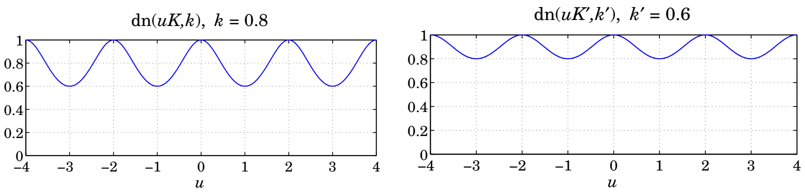


Fig. 13.2.3 The function dn with complementary moduli $k = 0.8$ and $k' = 0.6$.

Four additional properties, which will prove useful in filter design, are:

$$\operatorname{cd}\big(z + (2i-1)K, k\big) = (-1)^i \operatorname{sn}(z,k)\,, \quad \text{for any integer } i \tag{13.2.9}$$

$$\operatorname{cd}(z + 2iK, k) = (-1)^i \operatorname{cd}(z,k)\,, \quad \text{for any integer } i \tag{13.2.10}$$

$$\operatorname{cd}(z + jK', k) = \frac{1}{k\operatorname{cd}(z,k)} \tag{13.2.11}$$

$$\operatorname{cd}(jz, k) = \frac{1}{\operatorname{dn}(z,k')}\,, \quad \text{for real } z \tag{13.2.12}$$

In particular, setting $z = 0$ in (13.2.11), or, $z = K'$ in (13.2.12), we obtain:

$$\operatorname{cd}(jK', k) = \frac{1}{k} \tag{13.2.13}$$

The naming convention of the Jacobian elliptic functions may be understood with reference to the so-called *fundamental rectangle* on the complex z-plane with corners at $\{0,\, K,\, jK',\, K+jK'\}$, as shown in Fig. 13.2.4, where these corners are labeled with the letters S, C, N, D.

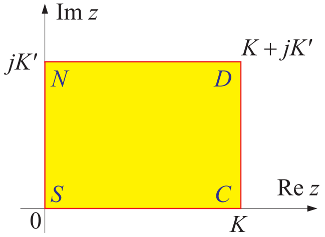


Fig. 13.2.4 The fundamental rectangle.

An elliptic function $\operatorname{pq}(z,k)$ is named such that the first letter p can be any of the four letters $\{s, c, d, n\}$, and the second letter q, any of the remaining three letters. Thus, there are $4\times 3 = 12$ Jacobian elliptic functions, namely,

sn, sd, sc, cn, cd, cs, dn, dc, ds, ns, nd, nc

Each function $\operatorname{pq}(z,k)$ has a *simple zero* at corner p and a *simple pole* at corner q of the fundamental rectangle. For example, $\operatorname{sn}(z,k)$ has a zero at the point S, $z = 0$, and a pole at the point N, $z = jK'$. Similarly, $\operatorname{cd}(z,k)$ has a zero at the point C, $z = K$, and a pole at the point D, $z = K + jK'$. Moreover, the following relationships hold:

$$\operatorname{pq}(z,k) = \frac{1}{\operatorname{qp}(z,k)}\,, \quad \operatorname{pq}(z,k) = \frac{\operatorname{pr}(z,k)}{\operatorname{qr}(z,k)} \tag{13.2.14}$$

where r is any one of the letters $\{s, c, d, n\}$ distinct from p and q, for example, as we saw in Eq. (13.2.3), $\operatorname{cd}(z,k) = \operatorname{cn}(z,k)\,/\,\operatorname{dn}(z,k)$.

The zeros and poles of the function **pq** are congruent modulo $2K$ and $2jK'$ to those at the corners p and q of the fundamental rectangle. In particular, the zeros and poles of $\operatorname{cd}(z,k)$, shown in Fig. 13.2.5, are given follows, where n, m are arbitrary integers (positive, negative, or zero):

$$\begin{aligned} &\text{zeros:} \quad z = K + 2mK + 2njK' = (2m+1)K + 2njK' \\ &\text{poles:} \quad z = K + jK' + 2mK + 2njK' = (2m+1)K + (2n+1)jK' \end{aligned} \tag{13.2.15}$$

The functions, $w = \operatorname{cd}(z,k)$, and, $w = \operatorname{sn}(z,k)$, map the z-plane conformally onto the w-plane. The smallest region of the z-plane that gets mapped onto the whole of the w-plane is called a *fundamental region*. For each function $\operatorname{pq}(z,k)$, such a region

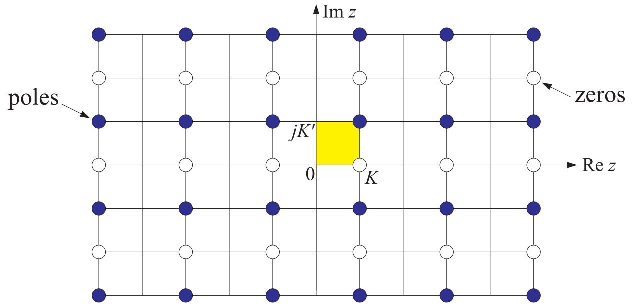


Fig. 13.2.5 Pole and zero patterns of the function $\mathrm{cd}(z,k)$.

is centered at the zero point p and surrounded by four fundamental rectangles, each rectangle being mapped onto a particular quadrant of the w-plane [309]. For example, the fundamental regions of the $\mathrm{cd}(z,k)$ and $\mathrm{sn}(z,k)$ functions are centered at the points C and S, respectively, and are defined by:

$$\begin{aligned}
&\mathrm{cd}(z,k): && 0 \le \mathrm{Re}\, z \le 2K\,, && -K' \le \mathrm{Im}\, z \le K' \\
&\mathrm{sn}(z,k): && -K \le \mathrm{Re}\, z \le K\,, && -K' \le \mathrm{Im}\, z \le K'
\end{aligned} \qquad \text{(fundamental regions)} \tag{13.2.16}$$

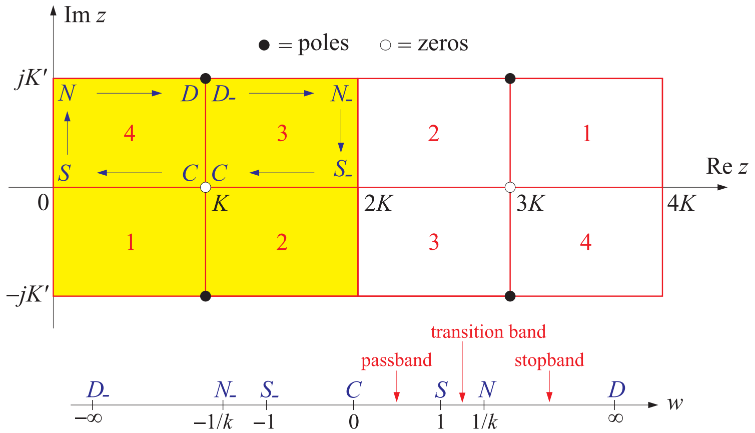


Fig. 13.2.6 Fundamental region, quadrant mappings, and period rectangle of the function $w = \mathrm{cd}(z,k)$.

These are shown in Figs. 13.2.6 and 13.2.7. The w-plane quadrants to which the z-plane quadrants map have been labeled by the quadrant numbers $\{1,2,3,4\}$. In particular, we note in Fig. 13.2.6 that the bottom two z-plane quadrants are mapped onto the first and second w-plane quadrants, that is, $z = z_1 - jz_2$ with $0 \le z_1 \le 2K$ and $0 < z_2 < K'$ gets mapped onto $w = w_1 + jw_2$ with $w_2 > 0$. Because the s-plane is related to the frequency plane by $s = jw$, it follows that the first and second w-plane quadrants will get mapped onto the left-hand s-plane, indeed, $s = j(w_1 + jw_2) = -w_2 + jw_1$. This property will be used in the construction of the analog filter's left-hand s-plane poles.

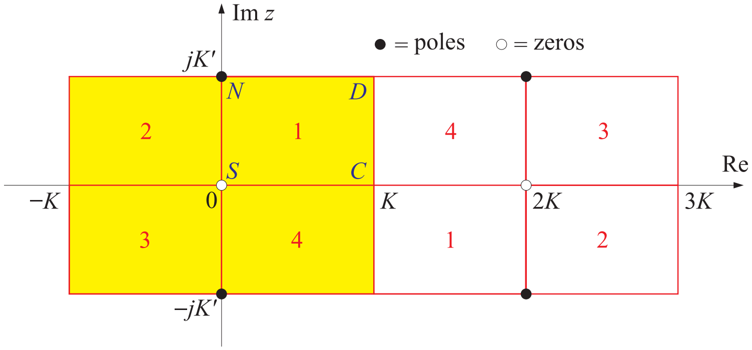


Fig. 13.2.7 Fundamental region, quadrant mappings, and period rectangle of the function $w = \mathrm{sn}(z,k)$.

Recalling that the periods of **cd** and **sn** are $4K$ and $2jK'$, we have doubled-up the fundamental regions in Figs. 13.2.6 and 13.2.7 to cover one complete *period rectangle*, that is,

$$\begin{aligned}
&\mathrm{cd}(z,k): && 0 \le \mathrm{Re}\, z \le 4K\,, && -K' \le \mathrm{Im}\, z \le K' \\
&\mathrm{sn}(z,k): && -K \le \mathrm{Re}\, z \le 3K\,, && -K' \le \mathrm{Im}\, z \le K'
\end{aligned} \qquad \text{(period rectangles)} \tag{13.2.17}$$

Of particular interest to filter design is the property that for the function, $w = \mathrm{cd}(z,k)$, the path around the fundamental rectangle $C \to S \to N \to D$ shown in Fig. 13.2.6, from the zero C to the pole D, gets mapped onto the positive real w-axis, such that the individual path segments, parametrized with the real parameter $0 \le u \le 1$, get mapped as follows:

$$\begin{array}{lllll}
\text{path } C \to S, & 0 \le u \le 1, & z = K - Ku & \Rightarrow & 0 \le w \le 1, & \text{passband} \\
\text{path } S \to N, & 0 \le u \le 1, & z = jK'u & \Rightarrow & 1 \le w \le 1/k, & \text{transition region} \\
\text{path } N \to D, & 0 \le u \le 1, & z = Ku + jK' & \Rightarrow & 1/k \le w \le \infty, & \text{stopband}
\end{array} \tag{13.2.18}$$

Because of the filter definition, Eq. (13.1.5), the above intervals of the $w = \Omega/\Omega_p$ axis will correspond to the passband, transition region, and stopband. Similarly, the continuation of the path to $D \to N_- \to S_- \to C$ covers the negative w-axis.

To verify these properties, we note that for the first segment $C \to S$, the argument $z = K - Ku$ is real and the values of $w = \mathrm{cd}(K - Ku, k) = \mathrm{sn}(Ku, k)$ will vary over the interval $0 \le w \le 1$, as seen in Fig. 13.2.2. For the segment $S \to N$, using property (13.2.12), we have $w = \mathrm{cd}(juK', k) = 1/\mathrm{dn}(uK', k')$, which increases from $w = 1$ at $u = 0$ to the value $w = \mathrm{cd}(jK', k) = 1/\mathrm{dn}(K', k') = 1/k$ at $u = 1$. Finally, for the segment $N \to D$, we use the property (13.2.11) to get:

$$w = \mathrm{cd}(Ku + jK', k) = \frac{1}{k\,\mathrm{cd}(Ku, k)}$$

with a starting value of $k\,\mathrm{cd}(0,k)=k$ in the denominator or $w=1/k$, and an ending value of $k\,\mathrm{cd}(K,k)=0$ or $w=\infty$.

In filter design, it is also required to be able to invert the functions $w=\mathrm{cd}(z,k)$ and $w=\mathrm{sn}(z,k)$, that is, to determine the value of z corresponding to a given complex-valued w. The resulting z is not unique. However, z becomes unique if it is restricted to lie within the fundamental region, that is, satisfying Eqs. (13.2.16). We will denote such an inverse by $z=\mathrm{cd}^{-1}(w,k)$ or $z=\mathrm{acd}(w,k)$. We note that within a period rectangle there are *two* values of z, the one in the fundamental region, the other in the adjacent region.

For example, if $z=\mathrm{cd}^{-1}(w,k)$ lies in the fundamental region, then $z_1=4K-z$ lies in the adjacent region and both satisfy $w=\mathrm{cd}(z,k)=\mathrm{cd}(z_1,k)$. Similarly, for the **sn** function the inverses are z and $z_1=2K-z$, with z satisfying $-K\le \mathrm{Re}\,z\le K$ and $K\le \mathrm{Re}\,z_1\le 3K$, and $w=\mathrm{sn}(z,k)=\mathrm{sn}(z_1,k)$.

The MATLAB functions **acde** and **asne** mentioned in Sect. 13.4 allow the computation of these inverse functions. Because $\mathrm{sn}(z,k)=\mathrm{cd}(K-z,k)$, the inverse of the **sn** function may be computed from the inverse of **cd**, by $z=K-\mathrm{cd}^{-1}(w,k)$.

13.3 Elliptic Rational Function and the Degree Equation

The analog filter characteristic function $F_N(w)$ was defined in the elliptic case by Eq. (13.1.5) in terms of the **cd** function [309]:

$$F_N(w)=\mathrm{cd}(NuK_1,k_1)\,,\quad w=\mathrm{cd}(uK,k) \tag{13.3.1}$$

where $w=\mathrm{cd}(uK,k)$ may be inverted to give u as a function of w, that is, $uK=\mathrm{cd}^{-1}(w,k)$. This indirect way of writing the function $F_N(w)$ is analogous to the Chebyshev -1 case, which can be thought of as the limit $k=k_1=0$ of the elliptic case:

$$C_N(w)=\cos(Nu\pi/2)\,,\quad w=\cos(u\pi/2) \tag{13.3.2}$$

where in this limit, we have, $K=K_1=\pi/2$. In order for the function $F_N(w)$ to satisfy the identity of Eq. (13.1.12), the three parameters N,k,k_1 must satisfy the following constraint, which is known as the *degree equation* for elliptic filters:

$$\boxed{N\,\frac{K'}{K}=\frac{K_1'}{K_1}}\qquad\text{(degree equation)} \tag{13.3.3}$$

where K,K_1 are the complete elliptic integrals (13.2.6) corresponding to the moduli k,k_1, and K',K_1' are the complete elliptic integrals corresponding to the complementary moduli $k'=(1-k^2)^{1/2}$ and $k_1'=(1-k_1^2)^{1/2}$. To verify this constraint, we use the definition (13.3.1) and Eq. (13.2.11) to obtain:

$$\begin{aligned}
k^{-1}w^{-1}&=\frac{1}{k\,\mathrm{cd}(uK,k)}=\mathrm{cd}(uK+jK',k)=\mathrm{cd}\left(\left(u+\frac{jK'}{K}\right)K,k\right)\\
F_N(k^{-1}w^{-1})&=\mathrm{cd}\left(N\left(u+\frac{jK'}{K}\right)K_1,k_1\right)=\mathrm{cd}\left(NuK_1+\frac{jNK'K_1}{K},k_1\right)
\end{aligned} \tag{13.3.4}$$

and using Eq. (13.2.11) again, applied with respect to the modulus k_1, we have:

$$\frac{1}{k_1F_N(w)}=\frac{1}{k_1\,\mathrm{cd}(NuK_1,k_1)}=\mathrm{cd}(NuK_1+jK_1',k_1) \tag{13.3.5}$$

Comparing Eqs. (13.3.4) and (13.3.5), we conclude that in order to satisfy the constraint, $F_N(k^{-1}w^{-1})=\big[k_1F_N(w)\big]^{-1}$, the following identity must be satisfied for all u:

$$\mathrm{cd}\left(NuK_1+\frac{jNK'K_1}{K},k_1\right)=\mathrm{cd}(NuK_1+jK_1',k_1)\quad\Rightarrow\quad\frac{NK'K_1}{K}=K_1' \tag{13.3.6}$$

from which Eq. (13.3.3) follows. We will see below that condition (13.1.8) that was obtained earlier,

$$F_N(k^{-1})=k_1^{-1} \tag{13.3.7}$$

actually provides the *solution* of Eq. (13.3.3) for the parameter k_1 in terms of N,k, or for the parameter k in terms of N,k_1. Using Eq. (13.3.3), we may also determine the values of $F_N(w)$ along the z-plane path $C\to S\to N\to D$. It follows from Eq. (13.2.18) and (13.3.1) that

$$\begin{array}{lll}
C\to S, & w=\mathrm{cd}(K-Ku,\,k), & F_N(w)=\mathrm{cd}(NK_1-NK_1u,\,k_1)\\
S\to N, & w=\mathrm{cd}(juK',k), & F_N(w)=\mathrm{cd}(juK_1',k_1)=1/\mathrm{dn}(uK_1',k_1')\\
N\to D, & w=\mathrm{cd}(Ku+jK',k), & F_N(w)=\mathrm{cd}(NK_1u+jK_1',k_1)=\big[k_1\,\mathrm{cd}(NK_1u,k_1)\big]^{-1}
\end{array} \tag{13.3.8}$$

For the path $C\to S$, $F_N(w)$ is equiripple and bounded by $|F_N(w)|\le 1$. For the path $S\to N$, we have, using the degree equation (13.3.3):

$$\begin{aligned}
&w=\mathrm{cd}\big((juK'/K)K,k\big)\quad\Rightarrow\\
&F_N(w)=\mathrm{cd}\big(jNK_1(juK'/K),\,k_1\big)=\mathrm{cd}(juK_1',k_1)=\frac{1}{\mathrm{dn}(uK_1',k_1')}
\end{aligned}$$

and therefore, $F_N(w)$ is an increasing function taking the values $1\le|F_N(w)|\le 1/k_1$. Finally, for $N\to D$, we have, using $w=\mathrm{cd}(Ku+jK',k)=\mathrm{cd}\big((u+jK'/K)K,k\big)$ and the degree equation:

$$F_N(w)=\mathrm{cd}\big(jNK_1(u+jK'/K),\,k_1\big)=\mathrm{cd}(NK_1u+jK_1',k_1)=\frac{1}{k_1\,\mathrm{cd}(NK_1u,k_1)}$$

Thus, the inverse $1/F_N(w)=k_1\,\mathrm{cd}(NK_1u,k_1)$ is equiripple and remains bounded in the interval, $\big|1/F_N(w)\big|\le k_1$. These properties cause the magnitude response (13.1.4) to be equiripple in the passband and the stopband, and be monotonically decreasing in the transition band.

Next, we construct $F_N(w)$ as a rational function of w. In the same way that Eq. (13.3.2) implies that $C_N(w)$ is a polynomial of degree N, Eq. (13.3.1) implies that $F_N(w)$ will be a rational function of w of order N.

Let us look briefly at the construction of $C_N(w)$ in terms of its zeros. Then, we will use the same technique to construct $F_N(w)$. Setting $N=2L+r$, where $r=0$ if N is even, and $r=1$ if N is odd, with L representing the number of second-order sections, we note that $C_N(w)$ is even in w if N is even, and odd if N is odd. Thus, $C_N(w)$ can be factored in the form $C_N(w)=[w]^r G(w^2)$, where $[w]^r$ means that the factor w is present if $r=1$ and absent if $r=0$, and $G(w^2)$ will be an L-th degree polynomial in w^2. To construct it, we solve the equation $C_N(w)=0$, or,

$$\cos(Nu\pi/2)=0 \quad\Rightarrow\quad Nu_i\frac{\pi}{2}=(2i-1)\frac{\pi}{2}\,,\quad \text{or,}$$

$$u_i=\frac{2i-1}{N}\,,\quad i=1,2,\dots,L \tag{13.3.9}$$

with the zeros of $C_N(w)$ constructed by

$$\zeta_i=\cos\left(\frac{u_i\pi}{2}\right),\quad i=1,2,\dots,L \tag{13.3.10}$$

resulting in the Nth degree polynomial $C_N(w)$:

$$C_N(w)=[w]^r\prod_{i=1}^{L}\frac{w^2-\zeta_i^2}{1-\zeta_i^2} \tag{13.3.11}$$

normalized such that $C_N(1)=1$. Thus, Eq. (13.3.11) is the representation of the polynomial $C_N(w)$ in terms of its N zeros.

Next, we construct the function $F_N(w)$. It follows from the definition (13.3.1) that $F_N(w)$ will be an even (odd) function of w if N is even (odd). Indeed, applying Eq. (13.2.10) with $i=1$ and $i=N$:

$$\begin{aligned}
-w &= -\,\mathrm{cd}(uK,k)=\mathrm{cd}(uK+2K,k)=\mathrm{cd}\big((u+2)K,k\big)\\
F_N(-w) &= \mathrm{cd}\big(N(u+2)K_1,k_1\big)=\mathrm{cd}(NuK_1+2NK_1,k_1)\\
&= (-1)^N\mathrm{cd}(NuK_1,k_1)=(-1)^N F_N(w)
\end{aligned}$$

The zeros of $F_N(w)$ are obtained by solving the equation, $\mathrm{cd}(NuK_1,k_1)=0$. It follows from Eq. (13.2.15) that:

$$\mathrm{cd}(NuK_1,k_1)=0 \quad\Rightarrow\quad Nu_iK_1=(2i-1)K_1 \quad \text{or,}$$

$$u_i=\frac{2i-1}{N}\,,\quad i=1,2,\dots,L \tag{13.3.12}$$

so that the u_i are the same as those in Eq. (13.3.9). Thus, the corresponding zeros of $F_N(w)$ will be at the frequencies, $w_i=\mathrm{cd}(u_iK,k)$, and we denote them by:

$$\boxed{\zeta_i=\mathrm{cd}(u_iK,k)\,,\quad i=1,2,\dots,L} \tag{13.3.13}$$

Because of the relationship $F_N(k^{-1}w^{-1})=\big[k_1F_N(w)\big]^{-1}$, the frequencies $w_i=(k\zeta_i)^{-1}$ will be the poles of $F_N(w)$. Thus, we may construct $F_N(w)$ as a rational function from its poles and zeros, and normalize it such that $F_N(1)=1$:

$$\boxed{F_N(w)=[w]^r\prod_{i=1}^{L}\left[\left(\frac{w^2-\zeta_i^2}{1-w^2k^2\zeta_i^2}\right)\left(\frac{1-k^2\zeta_i^2}{1-\zeta_i^2}\right)\right]} \tag{13.3.14}$$

Eq. (13.3.14) is known as an *elliptic rational function*, or a *Chebyshev rational function*. We note that in the limit $k=0$, Eq. (13.3.13) reduces to (13.3.10), and Eq. (13.3.14) reduces to (13.3.11).

Next, we obtain the solution of the degree equation (13.3.3). Using the condition (13.3.7) and setting $w=1/k$ and $F_N(w)=1/k_1$ in Eq. (13.3.14), we obtain the following formula for k_1 in terms of N,k:

$$k_1^{-1}=[k^{-1}]^r\prod_{i=1}^{L}\left[\left(\frac{k^{-2}-\zeta_i^2}{1-\zeta_i^2}\right)\left(\frac{1-k^2\zeta_i^2}{1-\zeta_i^2}\right)\right]=[k^{-1}]^{2L+r}\prod_{i=1}^{L}\left(\frac{1-k^2\zeta_i^2}{1-\zeta_i^2}\right)^2$$

Noting that $N=2L+r$, this can be rearranged into:

$$\boxed{k_1=k^N\prod_{i=1}^{L}\mathrm{sn}^4(u_iK,k)}\qquad\text{(degree equation)} \tag{13.3.15}$$

where we used the property $(1-\zeta_i^2)/(1-k^2\zeta_i^2)=\mathrm{sn}^2(u_iK,k)$, which follows from the last of Eqs. (13.2.5). Noting the invariance [312] of the degree equation (13.3.3) under the substitutions $k\to k_1'$ and $k_1\to k'$, we also obtain the exact solution for k in terms of N,k_1, expressed via the complementary moduli k',k_1':

$$\boxed{k'=(k_1')^N\prod_{i=1}^{L}\mathrm{sn}^4(u_iK_1',k_1')}\qquad\text{(degree equation)} \tag{13.3.16}$$

Eqs. (13.3.15) and (13.3.16)—known as the *modular equations*—were derived first by Jacobi in his original treatise on elliptic functions [313] and have been used since in the context of elliptic filter design [307,311,312].

The degree equation can also be solved approximately, and accurately, by working with the so-called *nomes* q,q_1, corresponding to the moduli k,k_1, defined by

$$q=e^{-\pi K'/K}\,,\quad q_1=e^{-\pi K_1'/K_1}$$

Exponentiating the degree equation (13.3.3), we have:

$$q_1=q^N \quad\Leftrightarrow\quad q=q_1^{1/N} \tag{13.3.17}$$

Once q has been calculated from N and q_1, the modulus k can be determined from the following series expansion [317], which converges very fast:

$$k=4\sqrt{q}\left[\frac{\displaystyle\sum_{m=0}^{\infty}q^{m(m+1)}}{\displaystyle 1+2\sum_{m=1}^{\infty}q^{m^2}}\right]^2 \tag{13.3.18}$$

For example, keeping only the terms up to $m=7$, gives a very accurate approximation.

13.4 Landen Transformations

The key tool for the evaluation of the elliptic functions $w = \mathrm{cd}(z,k)$ and $w = \mathrm{sn}(z,k)$ at any complex-valued argument z is the Landen transformation [309,318], which starts with a given elliptic modulus k and generates a sequence of *decreasing* moduli k_n via the following recursion, initialized at $k_0 = k$:

$$\boxed{k_n = \left(\frac{k_{n-1}}{1+k'_{n-1}}\right)^2, \quad n = 1,2,\dots,M} \tag{13.4.1}$$

where $k'_{n-1} = (1-k_{n-1}^2)^{1/2}$. The moduli k_n decrease rapidly to zero. The recursion is stopped at $n = M$ when k_M has become smaller than a specified tolerance level, for example, smaller than the machine epsilon.[†] For all practical values of k, such as those in the range $0 \le k \le 0.999$, the recursion may be stopped at $M = 5$, with all subsequent k_n being smaller than 10^{-15}, while for $k \le 0.99$, the subsequent k_n remain smaller than 10^{-20}. The recursion (13.4.1) may also be written in the equivalent form:

$$k_n = \frac{1-k'_{n-1}}{1+k'_{n-1}}$$

The inverse of the recursion (13.4.1) is:

$$k_{n-1} = \frac{2\sqrt{k_n}}{1+k_n}, \quad n = M, M-1, \dots, 1 \tag{13.4.2}$$

The Landen recursions (13.4.1) imply the following recursions [318] for the complete elliptic integral, $K_n = K(k_n)$, corresponding to the modulus k_n:

$$K_{n-1} = (1+k_n)K_n \tag{13.4.3}$$

The recursion (13.4.3) can be repeated to compute the elliptic integral $K = K(k)$ at the initial modulus k, that is, $K = K_0 = (1+k_1)K_1 = (1+k_1)(1+k_2)K_2$, and so on, yielding after M iterations:

$$\boxed{K = (1+k_1)(1+k_2)\cdots(1+k_M)K_M, \quad K_M = \frac{\pi}{2}} \tag{13.4.4}$$

Because k_M is almost zero, its elliptic integral will be essentially equal to $K_M = \pi/2$. The elliptic integral K' can be computed in the same way by applying the Landen recursion to k'. Floating point accuracy limits the applicability of Eq. (13.4.4) to roughly the range $0 \le k \le k_{\max}$, where $k_{\max} = (1-k_{\min}^2)^{1/2}$, with $k_{\min} = 10^{-6}$. For k in the range $k_{\max} < k \le 1-\epsilon$, where ϵ is the machine epsilon, one may use the expansion:

$$K = L + (L-1)\frac{k'^2}{2}, \quad L = -\ln\left(\frac{k'}{4}\right), \quad k' = (1-k^2)^{1/2}$$

[†] The machine epsilon for MATLAB is, $\epsilon = 2^{-52} = 2.2204\times 10^{-16}$.

The Landen transformations allow also the efficient evaluation of the elliptic functions **cd** and **sn** via the following backward recursion, known as the *Gauss transformation* [318], and written in the notation of [309]:

$$\frac{1}{\mathrm{cd}(uK_{n-1},k_{n-1})} = \frac{1}{1+k_n}\left[\frac{1}{\mathrm{cd}(uK_n,k_n)} + k_n\,\mathrm{cd}(uK_n,k_n)\right] \tag{13.4.5}$$

for $n = M, M-1, \dots, 1$. The recursion is initialized at $n = M$ where k_M is so small that the **cd** function is indistinguishable from a cosine, that is, $\mathrm{cd}(uK_M,k_M) \simeq \cos(u\pi/2)$. Thus, the computation of $w = \mathrm{cd}(uK,k)$, at any complex value of u, proceeds by calculating the quantities $w_n = \mathrm{cd}(uK_n,k_n)$, initialized at $w_M = \cos(u\pi/2)$, and ending with $w_0 = w = \mathrm{cd}(uK,k)$:

$$\boxed{w_{n-1}^{-1} = \frac{1}{1+k_n}\left[w_n^{-1} + k_n w_n\right], \quad n = M, M-1, \dots, 1} \tag{13.4.6}$$

The function, $w = \mathrm{sn}(uK,k)$, can be evaluated by the same recursion, initialized at $w_M = \sin(u\pi/2)$. The recursion (13.4.6) can also be used to calculate the *inverse* **cd** and **sn** functions by inverting it to proceed forward from $n = 1$ to $n = M$:

$$w_n^{-1} = \frac{1+k_n}{2}\left[w_{n-1}^{-1} + \sqrt{w_{n-1}^{-2} - k_{n-1}^2}\right], \quad n = 1,2,\dots,M \tag{13.4.7}$$

Starting with a given complex value $w = \mathrm{cd}(uK,k)$, and setting $w_0 = w$, the recursion will end at, $w_M = \cos(u\pi/2)$, which may be inverted to yield, $u = (2/\pi)\mathrm{acos}(w_M)$. Because u is not unique, it may be reduced to lie within its fundamental region, $0 \le \mathrm{Re}(u) \le 2$ and $0 \le |\mathrm{Im}(u)| \le K'/K$. The inverse of $w = \mathrm{sn}(uK,k)$ is obtained from the same recursion, but with, $u = (2/\pi)\mathrm{asin}(w_M)$, and reduced to lie in, $-1 \le \mathrm{Re}(u) \le 1$ and $0 \le |\mathrm{Im}(u)| \le K'/K$.

The evaluation of the cd elliptic function at a complex argument can also be carried out via the addition theorem [318]:

$$\mathrm{cd}(u+jv,k) = \frac{\mathrm{cn}(u,k)\mathrm{cn}(v,k') - j\,\mathrm{sn}(u,k)\mathrm{dn}(u,k)\mathrm{sn}(v,k')\mathrm{dn}(v,k')}{\mathrm{dn}(u,k)\mathrm{cn}(v,k')\mathrm{dn}(v,k') - jk^2\,\mathrm{sn}(u,k)\mathrm{cn}(u,k)\mathrm{sn}(v,k')} \tag{13.4.8}$$

We note also the following identity, valid for even N with the u_i defined as in Eq. (13.3.9):

$$\prod_{i=1}^{L}\mathrm{cd}(u_iK,k) = \prod_{i=1}^{L}\mathrm{sn}(u_iK,k) \tag{13.4.9}$$

where $N = 2L$. It may be used to derive the values of the function $F_N(w)$ at $w = 0$ and $w = \infty$, that is, $F_{2L}(0) = (-1)^L$ and $F_{2L}(\infty) = (-1)^L/k_1$.

All elliptic function computations described above can be carried out by the following set of MATLAB functions [329,330]:

`landen`	Landen transformation, Eq. (13.4.1)
`cde,acde`	cd elliptic function and its inverse, Eqs. (13.4.6) and (13.4.7)
`sne,asne`	sn elliptic function and its inverse, Eqs. (13.4.6) and (13.4.7)
`cne,dne`	cn and dn elliptic functions (for real arguments)
`ellipk`	complete elliptic integral $K(k)$, Eq. (13.4.4)
`ellipdeg`	exact solution of degree equation (k from N, k_1), Eq. (13.3.16)
`ellipdeg1`	exact solution of degree equation (k_1 from N, k), Eq. (13.3.15)
`ellipdeg2`	solution of degree equation using nomes, Eq. (13.3.18)
`elliprf`	elliptic rational function, Eq. (13.3.14)

13.5 Analog Elliptic Filter Design

The transfer function of an elliptic (as well as Butterworth and Chebyshev) lowpass analog filter is constructed from its zeros and poles $\{z_{ai}, p_{ai}\}$ in the second-order factored form:[†]

$$H_a(s) = H_0\left[\frac{1}{1-s/p_{a0}}\right]^r \prod_{i=1}^{L}\left[\frac{(1-s/z_{ai})\,(1-s/z_{ai}^*)}{(1-s/p_{ai})\,(1-s/p_{ai}^*)}\right] \tag{13.5.1}$$

where L is the number of analog second-order sections, related to the filter order by $N = 2L + r$. Again, the notation $[F]^r$ means that the factor F is present if $r = 1$ and absent if $r = 0$. The quantity H_0 is the gain at $\Omega = 0$ and is given as follows:

$$H_0 = \begin{cases} 1, & \text{Butterworth and Chebyshev-2} \\ G_p^{1-r}, & \text{Chebyshev-1 and Elliptic} \end{cases} \tag{13.5.2}$$

where $G_p = (1+\varepsilon_p^2)^{-1/2}$ is the passband gain. The variable s must be replaced by $s = j\Omega = j2\pi f$ to get the filter's frequency response. Multiplying the second-order factors, we may write the transfer function in the form:

$$H(s) = H_0\left[\frac{1}{1+A_{01}s}\right]^r \prod_{i=1}^{L}\left[\frac{1+B_{i1}s+B_{i2}s^2}{1+A_{i1}s+A_{i2}s^2}\right] \tag{13.5.3}$$

where the numerator and denominator coefficients are given by

$$\begin{aligned} [1,\, B_{i1},\, B_{i2}] &= \left[1,\, -2\,\mathrm{Re}\left(\frac{1}{z_{ai}}\right),\, \frac{1}{|z_{ai}|^2}\right] \\ [1,\, A_{i1},\, A_{i2}] &= \left[1,\, -2\,\mathrm{Re}\left(\frac{1}{p_{ai}}\right),\, \frac{1}{|p_{ai}|^2}\right] \\ [1,\, A_{01}] &= \left[1,\, -\frac{1}{p_{a0}}\right] \end{aligned} \tag{13.5.4}$$

Because the magnitude response corresponding to Eq. (13.5.1) is given by

$$|H(\Omega)|^2 = \frac{1}{1+\varepsilon_p^2 F_N^2(w)}, \quad w = \frac{\Omega}{\Omega_p}, \tag{13.5.5}$$

it follows that the zeros z_{ai} will arise from the poles of $F_N(w)$, and the poles p_{ai} will arise from the zeros of the denominator, that is, $1+\varepsilon_p^2 F_N^2(w) = 0$. We saw in the previous section that the poles of $F_N(w)$ occur at the normalized frequencies $w_i = (k\zeta_i)^{-1}$. Therefore, taking into account the normalization factor Ω_p, the denormalized s-plane zeros $z_{ai} = \Omega_p jw_i$ will be:

$$\boxed{z_{ai} = \Omega_p j(k\zeta_i)^{-1}, \quad i = 1, 2, \dots, L} \quad (s\text{-plane zeros}) \tag{13.5.6}$$

[†] The Butterworth and Chebyshev-1 cases do not have any zero factors.

The poles p_{ai} are found by solving the equation: $1+\varepsilon_p^2 F_N^2(w) = 0$, or,

$$F_N(w) = \pm j\frac{1}{\varepsilon_p} \tag{13.5.7}$$

The complex-frequency solutions w_i of (13.5.7) determine the denormalized poles by setting, $p_{ai} = \Omega_p jw_i$. The resulting left-hand s-plane poles p_{ai} are found to be:

$$\boxed{p_{ai} = \Omega_p j\,\mathrm{cd}\big((u_i - jv_0)K, k\big), \quad i = 1, 2, \dots, L} \quad (\text{left-hand } s\text{-plane poles}) \tag{13.5.8}$$

where the u_i are the same as in Eq. (13.3.12), and v_0 is the real-valued solution of the equation:

$$\boxed{\mathrm{sn}(jv_0NK_1, k_1) = j\frac{1}{\varepsilon_p}} \quad \Rightarrow \quad v_0 = -\frac{j}{NK_1}\,\mathrm{sn}^{-1}\left(\frac{j}{\varepsilon_p}, k_1\right) \tag{13.5.9}$$

As noted earlier in Fig. 13.2.6, the bottom two quadrants of the fundamental rectangle on the z- or u-plane get mapped onto the left-hand s-plane. If N is odd, there is an additional real-valued left-hand s-plane pole p_{a0} obtained from Eq. (13.5.8) by setting $u_i = 1$ (which corresponds to the index $i = L + 1$):

$$p_{a0} = \Omega_p j\,\mathrm{cd}\big((1 - jv_0)K, k\big) = \Omega_p j\,\mathrm{sn}(jv_0K, k) \tag{13.5.10}$$

To verify that $w_i = \mathrm{cd}\big((u_i - jv_0)K, k\big)$ is a solution of Eq. (13.5.7), we use the definition (13.3.1), property (13.2.9), and condition (13.5.9) to obtain:

$$\begin{aligned} F_N(w_i) &= \mathrm{cd}\big((u_i - jv_0)NK_1, k_1\big) \\ &= \mathrm{cd}(u_iNK_1 - jv_0NK_1, k_1) = \mathrm{cd}\big((2i-1)K_1 - jv_0NK_1, k_1\big) \\ &= (-1)^i\mathrm{sn}(-jv_0NK_1, k_1) = -(-1)^i\mathrm{sn}(jv_0NK_1, k_1) = \pm j\frac{1}{\varepsilon_p} \end{aligned}$$

13.6 Design Example

To clarify the above design steps, we give the MATLAB code for calculating the zeros, poles, and transfer function of the elliptic example of Fig. 13.1.2.

```
fp = 4; fs = 4.5; Gp = 0.95; Gs = 0.05;           % filter specifications

Wp = 2*pi*fp; Ws = 2*pi*fs;

ep = sqrt(1/Gp^2 - 1); es = sqrt(1/Gs^2 - 1);     % ripples εp = 0.3287, εs = 19.9750

k = Wp/Ws;                                        % k = 0.8889
k1 = ep/es;                                       % k1 = 0.0165

[K,Kp] = ellipk(k);                               % elliptic integrals K = 2.2353, K' = 1.6646
[K1,K1p] = ellipk(k1);                            % elliptic integrals K1 = 1.5709, K1' = 5.4937

Nexact = (K1p/K1)/(Kp/K);  N = ceil(Nexact);      % Nexact = 4.6961, N = 5
```

```
k = ellipdeg(N,k1);                               % recalculated k = 0.9143

fs_new = fp/k;                                    % new stopband fs = 4.3751

L = floor(N/2); r = mod(N,2); i = (1:L)';        % L = 2, r = 1, i = [1; 2]
u = (2*i-1)/N; zeta_i = cde(u,k);                 % ui = [0.2; 0.6], ζi = [0.9808; 0.7471]

za = Wp * j./(k*zeta_i);                          % filter zeros

v0 = -j*asne(j/ep, k1)/N;                         % v0 = 0.2331

pa = Wp * j*cde(u-j*v0, k);                       % filter poles
pa0 = Wp * j*sne(j*v0, k);

B = [ones(L,1), -2*real(1./za), abs(1./za).^2];    % numerator second-order sections
A = [ones(L,1), -2*real(1./pa), abs(1./pa).^2];    % denominator second-order sections

if r==0,                                            % prepend first-order sections
  B = [Gp, 0, 0; B];                                % DC gain is H0 = Gp, if N is even
  A = [1, 0, 0; A];
else
  B = [1, 0, 0; B];                                 % DC gain is H0 = 1, if N is odd
  A = [1, -real(1/pa0), 0; A];
end

f = linspace(0,10,2001);

for n=1:length(f),                                          % calculate frequency response
   s = j*2*pi*f(n);                                         % s = jΩ = 2πjf
   H(n) = prod((B(:,1) + B(:,2)*s + B(:,3)*s^2)./...        % cascade filter sections
               (A(:,1) + A(:,2)*s + A(:,3)*s^2));
end                                                         % alternatively, use H=fresp_a(B,A,f)

plot(f,abs(H),'r-');

xlim([0,10]); ylim([0,1.1]); grid off;
set(gca, 'xtick', 0:1:10); set(gca, 'ytick', 0:0.1:1);

title('Elliptic,  N = 5');
xlabel('f'); ylabel('|H(f)|');

line([0,fp],[1,1]); line([fp,fp],[1,1.05]);                 % draw brick-wall specs
line([0,fp],[Gp,Gp]); line([fp,fp],[Gp,0]);
line([fs_new,10],[Gs,Gs]); line([fs,fs],[4*Gs,Gs]);
```

The filter order was determined by calculating the exact value of N that satisfies the degree equation (13.3.3), that is, $N_{\text{exact}} = (K_1'/K_1)/(K'/K)$, and then, rounding it up to the next integer. With the slightly increased integer value of N, the degree equation is no longer satisfied with the given k, k_1. To satisfy it exactly, we recalculate k from N, k_1 using Eq. (13.3.16). The resulting k is slightly larger than the original one, and hence, the effective stopband $f_s = f_p/k$ will be slightly smaller, making the transition width

narrower. The calculated zeros and poles of the filter are, for $N = 5$ and $L = 2$:

$$\begin{array}{ll} z_1 = 28.0265j & p_0 = -15.1717 \\ z_2 = 36.7945j & p_1 = -1.0115 + 25.4353j \\ & p_2 = -6.2951 + 21.4113j \end{array}$$

The resulting first- and second-order numerator and denominator coefficients of the transfer function (13.5.1) are the rows of the matrices B and A, respectively:

$$B = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0.00127 \\ 1 & 0 & 0.00074 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 0.06591 & 0 \\ 1 & 0.00312 & 0.00154 \\ 1 & 0.02528 & 0.00201 \end{bmatrix} \tag{13.6.1}$$

Thus, the transfer function will be, with $s = 2\pi jf$:

$$H(s) = \frac{1}{1 + 0.06591\,s} \cdot \frac{1 + 0.00127\,s^2}{1 + 0.00312\,s + 0.00154\,s^2} \cdot \frac{1 + 0.00074\,s^2}{1 + 0.02528\,s + 0.00201\,s^2}$$

The following function `ellipap2.m` incorporates the above design steps and serves as a substitute for MATLAB's built-in function `ellipap`.

```
% ellipap2.m - analog lowpass elliptic filter design
%
% Usage: [z,p,H0,B,A] = ellipap2(N,Ap,As)
%
% N  = filter order
% Ap = passband attenuation in dB
% As = stopband attenuation in dB
%
% z  = vector of normalized filter zeros (in units of the passband frequency Ωp = 2π fp)
% p  = vector of normalized filter poles
% H0 = DC gain factor
% B  = matrix whose rows are the first- and second-order numerator coefficients
% A  = matrix whose rows are the first- and second-order denominator coefficients
%
% notes: serves as a substitute for the built-in function ELLIPAP
%        the gain factor g returned by ELLIPAP is related to the dc gain by g = abs(H0*prod(p)/prod(z))
%
%        N = 2*L+r, r = mod(N,2), L = floor(N/2) = no. second-order sections
%
%        length(p) = N, length(z) = 2*L
%        transfer function: H(s) = H0 [1/(1 - s/p0)]^r ∏_{i=1}^{L} [(1 - s/zi)(1 - s/zi*) / ((1 - s/pi)(1 - s/pi*))]
%
%        normalized s-plane variable, s = jΩ/Ωp,  Ω = 2π f,  Ωp = 2π fp,  fp = passband frequency

function [z,p,H0,B,A] = ellipap2(N,Ap,As)

if nargin==0, help ellipap2; return; end

Gp = 10^(-Ap/20);                                   % passband gain
ep = sqrt(10^(Ap/10) - 1);                          % ripple factors
es = sqrt(10^(As/10) - 1);

k1 = ep/es;
k = ellipdeg(N,k1);                                 % solve degree equation
```

```
L = floor(N/2); r = mod(N,2);                    % L is the number of second-order sections
i = (1:L)'; ui = (2*i-1)/N;
zeta_i = cde(ui,k);                              % zeros of elliptic rational function

z = j./(k*zeta_i);                               % filter zeros = poles of elliptic rational function

v0 = -j*asne(j/ep, k1)/N;                        % solution of sn(jv0NK1,k1) = j/εp

p = j*cde(ui-j*v0, k);                           % filter poles
p0 = j*sne(j*v0, k);                             % first-order pole, needed when N is odd

B = [ones(L,1), -2*real(1./z), abs(1./z).^2];         % second-order numerator sections
A = [ones(L,1), -2*real(1./p), abs(1./p).^2];         % second-order denominator sections

if r==0,                                         % prepend first-order sections
  B = [Gp, 0, 0; B];  A = [1, 0, 0; A];
else
  B = [1, 0, 0; B];  A = [1, -real(1/p0), 0; A];
end

z = cplxpair([z; conj(z)]);                      % append conjugate zeros
p = cplxpair([p; conj(p)]);                      % append conjugate poles
if r==1, p = [p; p0]; end                        % append first-order pole when N is odd

H0 = Gp^(1-r);                                   % dc gain
```

13.7 Butterworth and Chebyshev Designs

For completeness, we discuss also the design of Butterworth, Chebyshev-1, and Chebyshev-2 analog filters. For given specifications $\{\Omega_p, \Omega_s, A_p, A_s\}$, one calculates the parameters k, k_1 from Eq. (13.1.3), and solves for the filter order from Eq. (13.1.9), rounding it up to the next integer. In all cases, the filter poles are obtained by solving the equation:

$$1+\varepsilon_p^2 F_N^2(w)=0 \quad\Rightarrow\quad F_N^2(w)=-\frac{1}{\varepsilon_p^2} \tag{13.7.1}$$

As in the elliptic case, we define the quantities:

$$r=\mathrm{mod}(N,2)\,,\quad L=\frac{N-r}{2}\,,\quad u_i=\frac{2i-1}{N}\,,\quad i=1,2,\dots,L \tag{13.7.2}$$

For the Butterworth case, we obtain the conjugate poles $\{p_{ai}, p_{ai}^*\}$ of the second-order factors of Eq. (13.5.1) and the real-valued pole p_{a0} when N is odd (corresponding to the value $u_i = 1$):

$$\boxed{\begin{aligned} p_{ai} &= \Omega_p\, \varepsilon_p^{-1/N} j e^{j u_i \pi/2}\,, \quad i=1,2,\dots,L \\ p_{a0} &= \Omega_p\, \varepsilon_p^{-1/N} j e^{j\pi/2} = -\Omega_p\, \varepsilon_p^{-1/N} \end{aligned}} \quad \text{(Butterworth)} \tag{13.7.3}$$

We note that the quantity $\Omega_0 = \Omega_p\, \varepsilon_p^{-1/N}$ is the 3-dB frequency of the Butterworth filter. Indeed, since $\Omega_0^{2N} = \Omega_p^{2N}/\varepsilon_p^2$, the magnitude response can be written as

$$|H(\Omega)|^2 = \frac{1}{1+\varepsilon_p^2 w^{2N}} = \frac{1}{1+\varepsilon_p^2\left(\dfrac{\Omega}{\Omega_p}\right)^{2N}} = \frac{1}{1+\left(\dfrac{\Omega}{\Omega_0}\right)^{2N}}$$

In the Chebyshev-1 case, we have:

$$\boxed{\begin{aligned} p_{ai} &= \Omega_p j\cos\big((u_i - jv_0)\pi/2\big)\,, \quad i=1,2,\dots,L \\ p_{a0} &= \Omega_p j\cos\big((1-jv_0)\pi/2\big) = -\Omega_p \sinh(v_0\pi/2) \end{aligned}} \quad \text{(Chebyshev-1)} \tag{13.7.4}$$

where v_0 is obtained from the solution of:

$$\sinh(Nv_0\pi/2) = \frac{1}{\varepsilon_p} \quad\Rightarrow\quad v_0 = \frac{\mathrm{asinh}(1/\varepsilon_p)}{N\pi/2} \tag{13.7.5}$$

In the Chebyshev-2 case, the transfer function (13.5.1) has both poles and zeros, the latter arising from the numerator of Eq. (13.1.6), that is,

$$|H(\Omega)|^2 = \frac{1}{1+\varepsilon_p^2 k_1^{-2}/C_N^2(k^{-1}w^{-1})} = \frac{C_N^2(k^{-1}w^{-1})}{C_N^2(k^{-1}w^{-1})+\varepsilon_p^2 k_1^{-2}}$$

The resulting conjugate s-plane zeros $\{z_{ai}, z_{ai}^*\}$ and poles $\{p_{ai}, p_{ai}^*\}$, and the extra real-valued pole p_{a0} when N is odd, are essentially the inverses of those of the Chebyshev-1 case because of the correspondence $w \to k^{-1}w^{-1}$:

$$\boxed{\begin{aligned} k^{-1}z_{ai}^{-1} &= \Omega_p^{-1} j\cos(u_i\pi/2)\,, \quad i=1,2,\dots,L \\ k^{-1}p_{ai}^{-1} &= \Omega_p^{-1} j\cos\big((u_i - jv_0)\pi/2\big)\,, \quad i=1,2,\dots,L \\ k^{-1}p_{a0}^{-1} &= \Omega_p^{-1} j\cos\big((1-jv_0)\pi/2\big) = -\Omega_p^{-1}\sinh(v_0\pi/2) \end{aligned}} \quad \text{(Chebyshev-2)} \tag{13.7.6}$$

where v_0 is the solution of the equation:

$$\sinh(Nv_0\pi/2) = \varepsilon_p k_1^{-1} = \varepsilon_s \quad\Rightarrow\quad v_0 = \frac{\mathrm{asinh}(\varepsilon_s)}{N\pi/2} \tag{13.7.7}$$

Because k is used in Eq. (13.7.6), it must be recalculated by solving the second of Eqs. (13.1.9) for k in terms of k_1 and the rounded-up value of N, that is, $k = 1/\cosh\big(\mathrm{acosh}(k_1^{-1})/N\big)$.

In all cases, the poles lie in the left-hand s-plane, that is, $\mathrm{Re}(p_{ai}) < 0$. The overall transfer function is constructed by Eqs. (13.5.2)–(13.5.4), where in the Butterworth and Chebyshev-1 cases one may set $B_{i1} = B_{i2} = 0$ in the second-order numerator factors.

In all cases, the passband specification is matched exactly, while the stopband specification is exceeded because of the rounding of the exact N to the next integer.

The above design steps, as well as those for elliptic filters, have been incorporated into the MATLAB function `lpa.m`, listed below:

```
% lpa.m - lowpass analog filter design
%
% function [N,B,A] = lpa(Wp, Ws, Ap, As, type)
%
% Wp,Ws = passband and stopband frequencies in rad/sec
% Ap,As = passband and stopband attenuations in dB
% type  = 0,1,2,3 for Butterworth, Chebyshev-1, Chebyshev-2, Elliptic
%
% N   = filter order
% B,A = (L+1)x3 matrices of numerator and denominator coefficients, L=floor(N/2)

function [N,B,A] = lpa(Wp, Ws, Ap, As, type)

if nargin==0, help lpa; return; end

ep = sqrt(10^(Ap/10)-1); es = sqrt(10^(As/10)-1);

k = Wp/Ws; k1 = ep/es;                 % selectivity and discrimination parameters

switch type                            % determine order N
  case 0
     Nexact = log(1/k1) / log(1/k);
     N = ceil(Nexact);
  case 1
     Nexact = acosh(1/k1) / acosh(1/k);
     N = ceil(Nexact);
  case 2
     Nexact = acosh(1/k1) / acosh(1/k);
     N = ceil(Nexact);
     k = 1/cosh(acosh(1/k1) / N);      % recalculate k to satisfy degree equation
  case 3
     [K,Kp] = ellipk(k);
     [K1,K1p] = ellipk(k1);
     Nexact = (K1p/K1)/(Kp/K);
     N = ceil(Nexact);
     k = ellipdeg(N,k1);               % recalculate k to satisfy degree equation
end

r = mod(N,2); L = (N-r)/2; i = (1:L)'; u=(2*i-1)/N;

switch type                            % determine poles and zeros
  case 0
     pa = Wp * j * exp(j*u*pi/2) / ep^(1/N);
     pa0 = -Wp / ep^(1/N);
  case 1
     v0 = asinh(1/ep) / (N*pi/2);
     pa = Wp * j * cos((u-j*v0)*pi/2);
     pa0 = -Wp * sinh(v0*pi/2);
  case 2
     v0 = asinh(es) / (N*pi/2);
     za =  Wp ./ (j*k*cos(u*pi/2));
     pa =  Wp ./ (j*k*cos((u-j*v0)*pi/2));
     pa0 = -Wp / (k*sinh(v0*pi/2));
  case 3
     v0 = -j * asne(j/ep, k1) / N;
     za = Wp * j ./(k*cde(u,k));
     pa = Wp * j * cde(u-j*v0, k);
     pa0 = Wp * j * sne(j*v0, k);
```

```
end

B = [ones(L+1,1), zeros(L+1,2)];
A = [ones(L,1), -2*real(1./pa), abs(1./pa).^2];    % coefficient matrices

A = [[1, -r*real(1/pa0), 0]; A];

if type==2 | type==3,
   B(2:L+1,:) = [ones(L,1), -2*real(1./za), abs(1./za).^2];
end

Gp = 10^(-Ap/20);

if type==1 | type==3,                  % adjust dc gain
   B(1,1) = Gp^(1-r);
end
```

The elliptic portion of `lpa` is essentially equivalent to `ellipap2`. The function outputs the filter order N and the numerator and denominator coefficient matrices B, A, as given for example in Eq. (13.6.1), with the corresponding transfer function given by Eq. (13.5.3). The graphs of Fig. 13.1.2 can be generated by the following code fragment:

```
fp = 4; fs = 4.5;
Wp = 2*pi*fp; Ws = 2*pi*fs;
Gp = 0.95; Gs = 0.05;
Ap = -20*log10(Gp); As = -20*log10(Gs);

type = 3;    % type = 0,1,2,3 for butter, cheby-1, cheby-2, elliptic

[N,B,A] = lpa(Wp,Ws,Ap,As,type);

f = linspace(0,10,1001);
s = j*2*pi*f;             % s-domain

H = 1;
for i=1:size(B,1),
   H = H .* (B(i,1) + B(i,2)*s + B(i,3)*s.^2) ./ (A(i,1) + A(i,2)*s + A(i,3)*s.^2);
end

plot(f,abs(H),'r');

xtick(0:1:10); ylim([0,1.1]); ytick(0:0.1:1); grid off;

line([0,fp],[1,1],'LineStyle',':');
line([fp,fp],[1,1.05],'LineStyle',':');
line([0,fp],[Gp,Gp]); line([fp,fp],[Gp,0]);
line([fs,10],[Gs,Gs]); line([fs,fs],[4*Gs,Gs]);
```

13.8 Highpass, Bandpass, and Bandstop Analog Filters

The design of highpass, bandpass, or bandstop filters can be accomplished by applying an s-domain *frequency transformation* that maps a lowpass analog prototype to the desired filter. Such a transformation takes the following form, where s' is the equivalent lowpass variable:

$$s' = F(s) \tag{13.8.1}$$

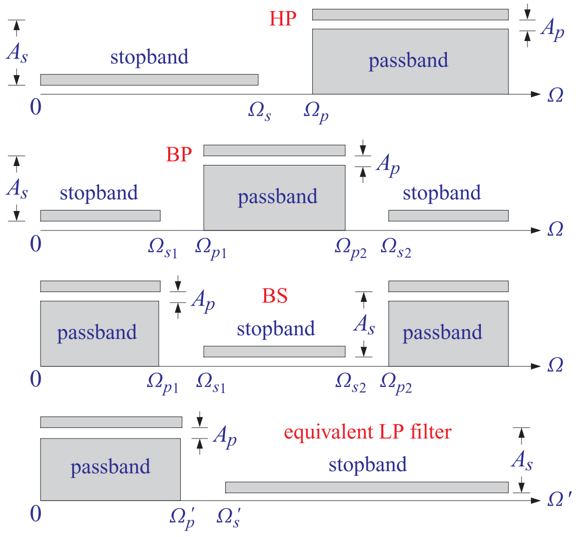


Fig. 13.8.1 Specifications of HP, BP, BS filters and of the equivalent LP filter.

The specifications of the desired highpass, bandpass, or bandstop filter are mapped by the same transformation into specifications, such as $\{\Omega'_p, \Omega'_s, A_p, A_s\}$, for the equivalent lowpass filter. Based on the transformed specifications, the equivalent lowpass filter's transfer function can be designed in the following form (using for example the function `lpa`):

$$H_{\text{LP}}(s') = H_0\left[\frac{1}{1+A_{01}s'}\right]^r \prod_{i=1}^{L}\left[\frac{1+B_{i1}s'+B_{i2}s'^2}{1+A_{i1}s'+A_{i2}s'^2}\right] \tag{13.8.2}$$

and then mapped onto the desired filter by:

$$H(s) = H_{\text{LP}}(s') = H_{\text{LP}}(F(s)) \tag{13.8.3}$$

The brick-wall specifications of the highpass, bandpass, and bandstop filters, and the corresponding specifications of the equivalent lowpass filter are shown in Fig. 13.8.1.

The mapping function $F(s)$ and the corresponding mapping specifications are given as follows in the three cases. For *highpass* designs, define:

$$s' = \frac{1}{s}\,,\quad \Omega'_p = \frac{1}{\Omega_p}\,,\quad \Omega'_s = \frac{1}{\Omega_s} \tag{13.8.4}$$

For *bandpass* designs, the bandwidth and center frequency of the passband are:

$$\Delta\Omega = \Omega_{p2} - \Omega_{p1}\,,\quad \Omega_0 = \sqrt{\Omega_{p1}\Omega_{p2}} \tag{13.8.5}$$

Then, the corresponding LP parameters are:

$$s' = s + \frac{\Omega_0^2}{s}\,,\quad \Omega'_p = \Delta\Omega\,,\quad \Omega'_s = \min\big(|\Omega'_{s1}|, |\Omega'_{s2}|\big) \tag{13.8.6}$$

where

$$\Omega'_{s1} = \Omega_{s1} - \frac{\Omega_0^2}{\Omega_{s1}}\,,\quad \Omega'_{s2} = \Omega_{s2} - \frac{\Omega_0^2}{\Omega_{s2}} \tag{13.8.7}$$

These are justified as follows. Setting $s' = j\Omega'$ and $s = j\Omega$ into Eq. (13.8.6), we obtain the corresponding mapping of the bandpass frequency Ω to the lowpass frequency Ω':

$$\Omega' = \Omega - \frac{\Omega_0^2}{\Omega}$$

and demand that the passband interval $[\Omega_{p1}, \Omega_{p2}]$ get mapped onto $[-\Omega'_p, \Omega'_p]$, that is,

$$-\Omega'_p = \Omega_{p1} - \frac{\Omega_0^2}{\Omega_{p1}}\,,\quad \Omega'_p = \Omega_{p2} - \frac{\Omega_0^2}{\Omega_{p2}}$$

These may be solved to give:

$$\Omega'_p = \Omega_{p2} - \Omega_{p1}\,,\quad \Omega_0^2 = \Omega_{p1}\Omega_{p2}$$

Once Ω_0 is fixed from the passband frequencies, it is no longer possible to map the stopband interval $[\Omega_{s1}, \Omega_{s2}]$ onto the symmetric lowpass interval $[-\Omega'_s, \Omega'_s]$. Therefore, Ω'_s is selected on the basis of the shorter of the two mapped stopband frequencies of Eq. (13.8.7).

For *bandstop* filters, the bandwidth $\Delta\Omega$ and center frequency Ω_0 are selected on the basis of the stopband interval:

$$\Delta\Omega = \Omega_{s2} - \Omega_{s1}\,,\quad \Omega_0 = \sqrt{\Omega_{s1}\Omega_{s2}} \tag{13.8.8}$$

Then, the corresponding LP parameters are:

$$s' = \frac{1}{s + \dfrac{\Omega_0^2}{s}}\,,\quad \Omega'_p = \max\big(|\Omega'_{p1}|, |\Omega'_{p2}|\big)\,,\quad \Omega'_s = \frac{1}{\Delta\Omega} \tag{13.8.9}$$

where

$$\Omega'_{p1} = \frac{1}{\Omega_{p1} - \dfrac{\Omega_0^2}{\Omega_{p1}}}\,,\quad \Omega'_{p2} = \frac{1}{\Omega_{p2} - \dfrac{\Omega_0^2}{\Omega_{p2}}} \tag{13.8.10}$$

In all three cases, once the equivalent frequencies Ω'_p, Ω'_s have been determined, the selectivity parameter can be calculated by $k = \Omega'_p/\Omega'_s$. The discrimination parameter $k_1 = \varepsilon_p/\varepsilon_s$ remains the same. Based on the values of k, k_1, the equivalent lowpass filter can be designed as a Butterworth, Chebyshev, or elliptic filter.

Fig. 13.8.2 shows some design examples. To clarify the design steps, the following code fragment implements the elliptic bandpass example:

```
fp1 = 3;
fp2 = 6;
fs1 = 2.5;
fs2 = 6.5;

Wp1 = 2*pi*fp1;
Wp2 = 2*pi*fp2;
```

```
Ws1 = 2*pi*fs1;
Ws2 = 2*pi*fs2;

Gp = 0.95;
Gs = 0.05;

Ap = -20*log10(Gp);
As = -20*log10(Gs);

Wp = Wp2-Wp1; W0 = sqrt(Wp1*Wp2);

W1 = Ws1 - W0^2/Ws1; W2 = Ws2 - W0^2/Ws2;

Ws = min(abs([W1,W2]));

type = 3;
[N,B,A] = lpa(Wp,Ws,Ap,As,type);

f = linspace(0,10,1001);
s = j*2*pi*f;
s = s + W0^2./s;                          % division by zero warning can be ignored

H = 1;
for i=1:size(B,1),
   H = H .* (B(i,1) + B(i,2)*s + B(i,3)*s.^2) ./ (A(i,1) + A(i,2)*s + A(i,3)*s.^2);
end

figure; plot(f,abs(H),’r’);

xtick(0:1:10); ylim([0,1.1]); ytick(0:0.1:1); grid off;

line([fp1,fp2],[1,1],’LineStyle’,’:’);

line([fp1,fp1],[1,1.05],’LineStyle’,’:’); line([fp2,fp2],[1,1.05],’LineStyle’,’:’);

line([fp1,fp2],[Gp,Gp]); line([fp1,fp1],[Gp,0]); line([fp2,fp2],[Gp,0]);

line([0,fs1],[Gs,Gs]); line([fs1,fs1],[4*Gs,Gs]);

line([fs2,10],[Gs,Gs]); line([fs2,fs2],[4*Gs,Gs]);
```

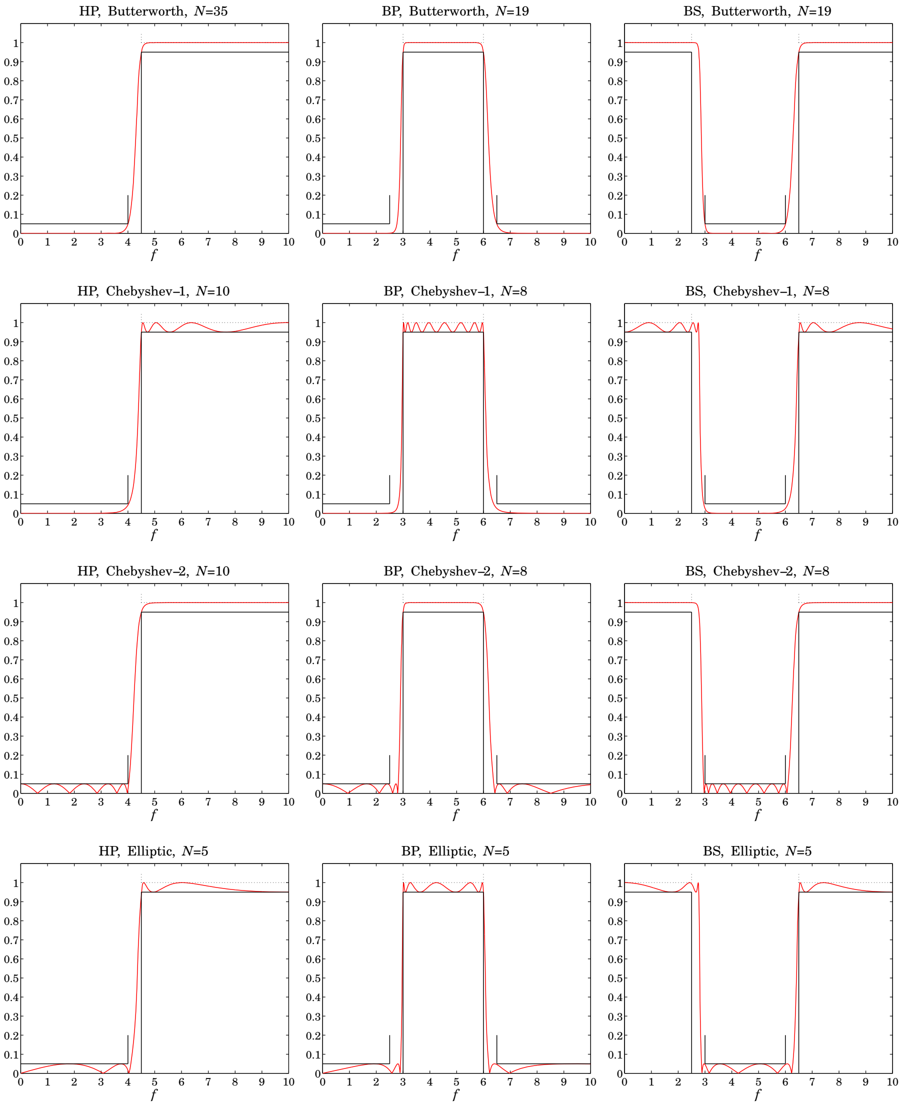


Fig. 13.8.2 Highpass, bandpass, and bandstop analog filters.

13.9 Digital Filter Design

Digital filters may be designed by the bilinear transformation method carried out by the following steps: (a) the specifications of the digital filter are transformed into equivalent specifications for a lowpass analog prototype filter, (b) the analog prototype is designed as a Butterworth, Chebyshev, or elliptic filter using the methods discussed above, and (c) the analog filter's transfer function $H_a(s)$ is transformed into the desired digital filter's transfer function $H(z)$ with an appropriate bilinear transformation, that is, a mapping between the s-plane and the z-plane of the form $s = f(z)$:

$$H(z) = H_a(s)\Big|_{s=f(z)} = H_a\big(f(z)\big) \tag{13.9.1}$$

The mappings used for lowpass, highpass, bandpass, and bandstop filters, and the corresponding frequency mappings obtained by setting $s = j\Omega$ and $z = e^{j\omega}$, where Ω is the equivalent analog frequency and $\omega = 2\pi f/f_s$, the digital frequency, are as follows:

$$\begin{aligned}
&\text{(LP)} \qquad s = \frac{1-z^{-1}}{1+z^{-1}}\,, && \Omega = \tan\left(\frac{\omega}{2}\right) \\
&\text{(HP)} \qquad s = \frac{1+z^{-1}}{1-z^{-1}}\,, && \Omega = -\cot\left(\frac{\omega}{2}\right) \\
&\text{(BP)} \qquad s = \frac{1-2c_0z^{-1}+z^{-2}}{1-z^{-2}}\,, && \Omega = \frac{c_0-\cos\omega}{\sin\omega} \\
&\text{(BS)} \qquad s = \frac{1-z^{-2}}{1-2c_0z^{-1}+z^{-2}}\,, && \Omega = -\frac{\sin\omega}{c_0-\cos\omega}
\end{aligned} \tag{13.9.2}$$

where $c_0 = \cos\omega_0$, with ω_0 corresponding to the center of the bandpass or bandstop filter.

13.10 Pole and Zero Transformations

We begin with lowpass digital filters constructed with the lowpass bilinear transformation:

$$s = \frac{1-z^{-1}}{1+z^{-1}} \tag{13.10.1}$$

Assuming that the equivalent lowpass analog filter $H_a(s)$ has already been constructed in terms of its zeros and poles, the lowpass digital filter's transfer function will be:

$$H_{\text{LP}}(z) = H_a(s)\,\Big|_{s=\frac{1-z^{-1}}{1+z^{-1}}} = H_0\left[\frac{1}{1-s/p_{a0}}\right]^r \prod_{i=1}^{L}\left[\frac{(1-s/z_{ai})(1-s/z_{ai}^*)}{(1-s/p_{ai})(1-s/p_{ai}^*)}\right]\Bigg|_{s=\frac{1-z^{-1}}{1+z^{-1}}} \tag{13.10.2}$$

After mapping the analog s-plane poles and zeros $\{p_{ai}, z_{ai}\}$ to the digital z-plane poles and zeros $\{p_i, z_i\}$ by Eq. (13.10.1), the resulting digital transfer function will have the form:

$$H_{\text{LP}}(z) = H_0\left[G_0\,\frac{1+z^{-1}}{1-p_0z^{-1}}\right]^r \prod_{i=1}^{L}\left[|G_i|^2\,\frac{(1-z_iz^{-1})(1-z_i^*z^{-1})}{(1-p_iz^{-1})(1-p_i^*z^{-1})}\right] \tag{13.10.3}$$

Noting that the inverse of Eq. (13.10.1) is $z = \dfrac{1+s}{1-s}$, the digital poles and zeros are computed by:

$$p_0 = \frac{1+p_{a0}}{1-p_{a0}}\,, \quad p_i = \frac{1+p_{ai}}{1-p_{ai}}\,, \quad z_i = \frac{1+z_{ai}}{1-z_{ai}}\,, \quad i = 1,2,\dots,L \tag{13.10.4}$$

and the gains factors, by:

$$G_0 = \frac{1-p_0}{2}\,, \quad G_i = \frac{1-p_i}{1-z_i} \tag{13.10.5}$$

Eq. (13.10.3) follows from the transformation identity of each s-plane pole/zero factor:

$$\frac{1-s/z_{ai}}{1-s/p_{ai}} = G_i\,\frac{1-z_iz^{-1}}{1-p_iz^{-1}}\,, \quad G_i = \frac{1-p_i}{1-z_i} \tag{13.10.6}$$

The zeros of the Butterworth and Chebyshev-1 cases are simply $z_i = -1$ as follows by setting $z_{ai} = \infty$ in Eq. (13.10.4). Highpass digital filters can designed by mapping the lowpass analog prototype by the highpass version of the bilinear transformation:

$$s = \frac{1+z^{-1}}{1-z^{-1}} \tag{13.10.7}$$

which amounts to making the replacement $z^{-1} \to -z^{-1}$ in the lowpass case. Replacing the lowpass poles and zeros $\{p_i, z_i\}$ by their negatives, one obtains the highpass transfer function:

$$\begin{aligned}
H_{\text{HP}}(z) &= H_0\left[\frac{1}{1-s/p_{a0}}\right]^r \prod_{i=1}^{L}\left[\frac{(1-s/z_{ai})(1-s/z_{ai}^*)}{(1-s/p_{ai})(1-s/p_{ai}^*)}\right]\Bigg|_{s=\frac{1+z^{-1}}{1-z^{-1}}} \\
&= H_0\left[G_0\,\frac{1-z^{-1}}{1-p_0z^{-1}}\right]^r \prod_{i=1}^{L}\left[|G_i|^2\,\frac{(1-z_iz^{-1})(1-z_i^*z^{-1})}{(1-p_iz^{-1})(1-p_i^*z^{-1})}\right]
\end{aligned} \tag{13.10.8}$$

where the highpass digital poles and zeros are now given by:

$$p_0 = -\frac{1+p_{a0}}{1-p_{a0}}\,, \quad p_i = -\frac{1+p_{ai}}{1-p_{ai}}\,, \quad z_i = -\frac{1+z_{ai}}{1-z_{ai}}\,, \quad i = 1,2,\dots,L \tag{13.10.9}$$

and the gains factors, by:

$$G_0 = \frac{1+p_0}{2}\,, \quad G_i = \frac{1+p_i}{1+z_i} \tag{13.10.10}$$

More generally, the lowpass analog filter can be mapped first into a lowpass digital filter using Eq. (13.10.1), which can subsequently be mapped by a z-domain frequency transformation [321,322] to a highpass, bandpass, or bandstop digital filter:

$$\boxed{\begin{matrix}\text{LP analog}\\ s\text{-plane}\\ H_a(s)\end{matrix}} \longrightarrow \boxed{\begin{matrix}\text{LP digital}\\ \hat{z}\text{-plane}\\ \hat{H}(\hat{z})\end{matrix}} \longrightarrow \boxed{\begin{matrix}\text{HP/BP/BS digital}\\ z\text{-plane}\\ H(z)\end{matrix}} \tag{13.10.11}$$

For bandpass designs, the required transformations take the two-step form:

$$s = \frac{1-\hat{z}^{-1}}{1+\hat{z}^{-1}} = \frac{1-2c_0z^{-1}+z^{-2}}{1-z^{-2}}\,, \quad \hat{z}^{-1} = \frac{z^{-1}(c_0-z^{-1})}{1-c_0z^{-1}} \tag{13.10.12}$$

where $c_0 = \cos\omega_0$ and $\omega_0 = 2\pi f_0/f_s$ is the center frequency of the bandpass filter. Setting $\omega_0 = \pi$ or $c_0 = -1$ yields the highpass case, which has $\hat{z}^{-1} = -z^{-1}$. The lowpass case corresponds to $\omega_0 = 0$ or $c_0 = 1$, which gives $\hat{z} = z$.

The mapping $s \to \hat{z}$ transforms a lowpass analog filter $H_a(s)$ into lowpass digital filter $\hat{H}(\hat{z})$, which is further transformed into a bandpass digital filter $H_{\text{BP}}(z)$ by the mapping $\hat{z} \to z$, that is, the transfer functions are related by:

$$H_{\text{BP}}(z) = \hat{H}(\hat{z})\Big|_{\hat{z}^{-1}=\frac{z^{-1}(c_0-z^{-1})}{1-c_0z^{-1}}} = H_a(s)\Big|_{s=\frac{1-\hat{z}^{-1}}{1+\hat{z}^{-1}}} \tag{13.10.13}$$

For bandstop designs, we may use:

$$s = \frac{1-\hat{z}^{-1}}{1+\hat{z}^{-1}} = \frac{1-z^{-2}}{1-2c_0z^{-1}+z^{-2}}\,, \quad \hat{z}^{-1} = -\frac{z^{-1}(c_0-z^{-1})}{1-c_0z^{-1}} \tag{13.10.14}$$

with transfer functions related by:

$$H_{\text{BS}}(z) = \hat{H}(\hat{z})\Big|_{\hat{z}^{-1}=-\frac{z^{-1}(c_0-z^{-1})}{1-c_0z^{-1}}} = H_a(s)\Big|_{s=\frac{1-\hat{z}^{-1}}{1+\hat{z}^{-1}}} \tag{13.10.15}$$

The bandpass and bandstop cases can be combined into one by defining:

$$\boxed{s = \frac{1-\hat{z}^{-1}}{1+\hat{z}^{-1}}\,, \quad \hat{z}^{-1} = q\,\frac{z^{-1}(c_0-z^{-1})}{1-c_0z^{-1}}\,, \quad q = \begin{cases} +1, & \text{BP case} \\ -1, & \text{BS case}\end{cases}} \tag{13.10.16}$$

The transfer functions will be related by:

$$H(z) = \hat{H}(\hat{z})\Big|_{\hat{z}^{-1}=\frac{qz^{-1}(c_0-z^{-1})}{1-c_0z^{-1}}} = H_a(s)\Big|_{s=\frac{1-\hat{z}^{-1}}{1+\hat{z}^{-1}}} \tag{13.10.17}$$

Setting, $s = j\Omega$, $\hat{z} = e^{j\hat{\omega}}$, and, $z = e^{j\omega}$, we find the corresponding frequency relationships:

$$\Omega = \tan\left(\frac{\hat{\omega}}{2}\right) = \begin{cases} \dfrac{c_0-\cos\omega}{\sin\omega}\,, & \text{if } q = 1 \\[2ex] -\dfrac{\sin\omega}{c_0-\cos\omega}\,, & \text{if } q = -1 \end{cases} \tag{13.10.18}$$

Because $\hat{z}$ depends quadratically on z, each lowpass analog s-plane pole p_{ai} will first get mapped into a lowpass digital $\hat{z}$-plane pole $\hat{p}_i$, which will then be mapped into *two* z-plane poles, say, p_i^+, p_i^-. The pole $\hat{p}_i$ is constructed from the analog pole p_{ai} via Eq. (13.10.16):

$$p_{ai} = \frac{1-\hat{p}_i^{-1}}{1+\hat{p}_i^{-1}} \quad\Rightarrow\quad \boxed{\hat{p}_i = \frac{1+p_{ai}}{1-p_{ai}}} \tag{13.10.19}$$

then, the pole pairs $p_i^{\pm}$ are obtained as the two solutions for p_i of the equation:

$$\hat{p}_i = q\,\frac{p_i(c_0-p_i)}{1-c_0p_i} \quad\Rightarrow\quad p_i^{\pm} = \frac{1}{2}\left[c_0(1+q\hat{p}_i) \pm \sqrt{c_0^2(1+q\hat{p}_i)^2 - 4q\hat{p}_i}\right] \tag{13.10.20}$$

Because $\hat{z}$ remains invariant under the substitution $z \to z' = (c_0 - z)/(1 - c_0 z)$, that is,

$$\hat{z} = q\,\frac{z(c_0-z)}{1-c_0z} = q\,\frac{z'(c_0-z')}{1-c_0z'}\,, \quad \text{with} \quad z' = \frac{c_0-z}{1-c_0z}\,, \quad z = \frac{c_0-z'}{1-c_0z'}\,, \tag{13.10.21}$$

it follows that the pair p_i^+, p_i^- can be constructed, alternatively, by:

$$\boxed{p_i^+ = \frac{1}{2}\left[c_0(1+q\hat{p}_i) + \sqrt{c_0^2(1+q\hat{p}_i)^2 - 4q\hat{p}_i}\right]\,, \quad p_i^- = \frac{c_0-p_i^+}{1-c_0p_i^+}} \tag{13.10.22}$$

Thus, Eqs. (13.10.19) and (13.10.22) allow the mapping of the analog poles to the final digital filter poles. The analog zeros z_{ai} are mapped in a similar way to the zeros $\hat{z}_i$ and then to z_i^+, z_i^-. Using Eqs. (13.10.16), (13.10.19), and (13.10.22), the following identity can be verified easily:

$$\frac{1-s/z_{ai}}{1-s/p_{ai}} = G_i\,\frac{1-\hat{z}_i\hat{z}^{-1}}{1-\hat{p}_i\hat{z}^{-1}} = G_i\,\frac{(1-z_i^+z^{-1})(1-z_i^-z^{-1})}{(1-p_i^+z^{-1})(1-p_i^-z^{-1})}\,, \quad G_i = \frac{1-\hat{p}_i}{1-\hat{z}_i} \tag{13.10.23}$$

It follows that the transfer function can be expressed in the equivalent factored forms:

$$\begin{aligned} H(z) &= H_0\left[\frac{1}{1-s/p_{a0}}\right]^r \prod_{i=1}^{L}\left[\frac{(1-s/z_{ai})(1-s/z_{ai}^*)}{(1-s/p_{ai})(1-s/p_{ai}^*)}\right]\Bigg|_{s=\frac{1-\hat{z}^{-1}}{1+\hat{z}^{-1}}} \\ &= H_0\left[G_0\,\frac{1+\hat{z}^{-1}}{1-\hat{p}_0\hat{z}^{-1}}\right]^r \prod_{i=1}^{L}\left[|G_i|^2\,\frac{(1-\hat{z}_i\hat{z}^{-1})(1-\hat{z}_i^*\hat{z}^{-1})}{(1-\hat{p}_i\hat{z}^{-1})(1-\hat{p}_i^*\hat{z}^{-1})}\right]\Bigg|_{\hat{z}^{-1}=\frac{qz^{-1}(c_0-z^{-1})}{1-c_0z^{-1}}} \\ &= H_0\left[G_0\,\frac{(1-z_0^+z^{-1})(1-z_0^-z^{-1})}{(1-p_0^+z^{-1})(1-p_0^-z^{-1})}\right]^r\cdot \\ &\quad \prod_{i=1}^{L}\left[|G_i|\,\frac{(1-z_i^+z^{-1})(1-z_i^{+*}z^{-1})}{(1-p_i^+z^{-1})(1-p_i^{+*}z^{-1})}\right]\cdot\prod_{i=1}^{L}\left[|G_i|\,\frac{(1-z_i^-z^{-1})(1-z_i^{-*}z^{-1})}{(1-p_i^-z^{-1})(1-p_i^{-*}z^{-1})}\right] \end{aligned} \tag{13.10.24}$$

The poles $p_0^{\pm}$ arise from the mapping of the analog pole p_{a0}:

$$\hat{p}_0 = \frac{1+p_{a0}}{1-p_{a0}} \quad\Rightarrow\quad p_0^{\pm} = \frac{1}{2}\left[c_0(1+q\hat{p}_0) \pm \sqrt{c_0^2(1+q\hat{p}_0)^2 - 4q\hat{p}_0}\right] \tag{13.10.25}$$

Because p_{a0} is real, $p_0^{\pm}$ are either both real-valued or conjugate pairs. The zeros $z_0^{\pm}$ arise from mapping the analog zero $z_{a0} = \infty$ to $\hat{z}_0 = -1$, and are given in the two cases $q = \pm 1$ by:

$$\hat{z}_0 = -1 \quad\Rightarrow\quad z_0^{\pm} = \frac{1}{2}\left[c_0(1+q\hat{z}_0) \pm \sqrt{c_0^2(1+q\hat{z}_0)^2 - 4q\hat{z}_0}\right] = \begin{cases} \pm 1\,, & \text{if } q = 1 \\ e^{\pm j\omega_0}\,, & \text{if } q = -1 \end{cases} \tag{13.10.26}$$

Thus, the corresponding numerator second-order factor becomes:

$$(1-z_0^+z^{-1})(1-z_0^-z^{-1}) = \begin{cases} 1-z^{-2}\,, & \text{if } q = 1 \\ 1-2c_0z^{-1}+z^{-2}\,, & \text{if } q = -1 \end{cases} \tag{13.10.27}$$

Such are also the other numerator factors in the Butterworth and Chebyshev-1 cases.

In summary, Eq. (13.10.24) expresses $H(z)$ as a product of second-order sections, which is usually the preferred form. By combining the last two groups of L second-order factors, we may express $H(z)$ as a cascade of L fourth-order sections:

$$H(z) = H_0\left[G_0\,\frac{(1-z_0^+z^{-1})\,(1-z_0^-z^{-1})}{(1-p_0^+z^{-1})\,(1-p_0^-z^{-1})}\right]^r\cdot$$
$$\prod_{i=1}^{L}\left[|G_i|^2\,\frac{(1-z_i^+z^{-1})\,(1-z_i^{+*}z^{-1})\,(1-z_i^-z^{-1})\,(1-z_i^{-*}z^{-1})}{(1-p_i^+z^{-1})\,(1-p_i^{+*}z^{-1})\,(1-p_i^-z^{-1})\,(1-p_i^{-*}z^{-1})}\right] \tag{13.10.28}$$

Eq. (13.10.24) includes also the LP and HP cases, which have $q = 1$ and $c_0 = \pm 1$ resulting in $\hat{z} = \pm z$. The second group of L sections reduces to unity because $p_i^- = (c_0 - p_i^+)\,/\,(1 - c_0 p_i^+) = \pm 1$ when $c_0 = \pm 1$, and similarly $z_i^- = \pm 1$, as well as, $p_0^- = z_0^- = \pm 1$.

We note finally that the mappings defined in Eq. (13.10.16) preserve the causality and stability of the filters, in the sense that they map left-hand s-plane poles to poles inside the $\hat{z}$-plane unit circle, to poles inside the z-plane unit circle. These follows from the relationships:

$$\mathrm{Re}(s) = \frac{1-|\hat{z}|^2}{|\hat{z}+1|^2}\,,\quad 1-|\hat{z}|^2 = (1-|z|^2)\,\frac{|c_0-z|^2+s_0^2}{|1-c_0z|^2}\,,\quad s_0 = \sin\omega_0 \tag{13.10.29}$$

which show that $\mathrm{Re}(s) \le 0$ is equivalent to $|\hat{z}| \le 1$ and to $|z| \le 1$.

13.11 Transformation of Frequency Specifications

The design specifications for lowpass, highpass, bandpass, and bandstop digital filters, and the specifications of the equivalent lowpass analog filter are shown in Fig. 13.11.1, where f is in Hz and f_s is the sampling rate. The frequency transformation equations (13.9.2) give rise to the following procedures for mapping the given specifications into the analog ones.

For *lowpass* filters having passband and stopband frequencies $f_{\text{pass}} < f_{\text{stop}} < f_s/2$, we calculate $\omega_{\text{pass}} = 2\pi f_{\text{pass}}/f_s$, $\omega_{\text{stop}} = 2\pi f_{\text{stop}}/f_s$, and

$$\Omega_p = \tan\left(\frac{\omega_{\text{pass}}}{2}\right),\quad \Omega_s = \tan\left(\frac{\omega_{\text{stop}}}{2}\right) \tag{13.11.1}$$

For *highpass* designs with passband and stopband frequencies $f_{\text{stop}} < f_{\text{pass}} < f_s/2$, we calculate $\omega_{\text{pass}} = 2\pi f_{\text{pass}}/f_s$, $\omega_{\text{stop}} = 2\pi f_{\text{stop}}/f_s$, and:

$$\Omega_p = \cot\left(\frac{\omega_{\text{pass}}}{2}\right),\quad \Omega_s = \cot\left(\frac{\omega_{\text{stop}}}{2}\right) \tag{13.11.2}$$

For *bandpass* designs with a passband interval $[f_{p1}, f_{p2}]$ given as a subset of a stopband interval $[f_{s1}, f_{s2}]$, we calculate the analog filter's parameters as follows, where $\omega_{p1} = 2\pi f_{p1}/f_s$, $\omega_{p2} = 2\pi f_{p2}/f_s$, $\omega_{s1} = 2\pi f_{s1}/f_s$, $\omega_{s2} = 2\pi f_{s2}/f_s$:

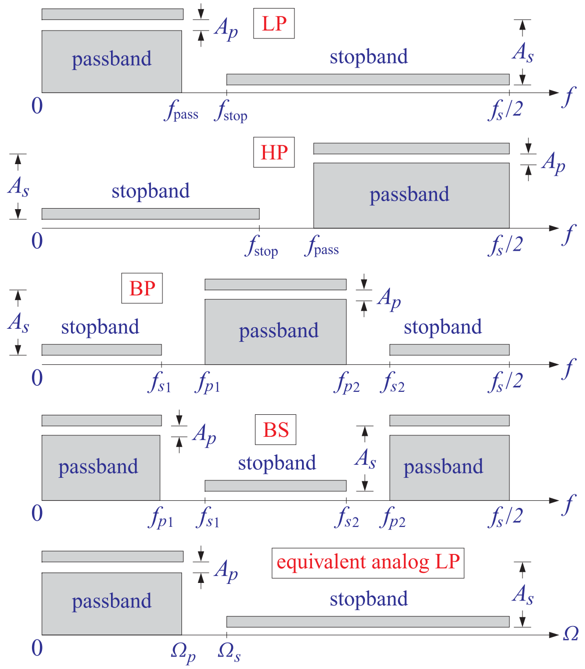


Fig. 13.11.1 Specifications of digital and equivalent analog filters (f_s is the sampling frequency).

$$\cos\omega_0 = \frac{\sin(\omega_{p1}+\omega_{p2})}{\sin\omega_{p1}+\sin\omega_{p2}}\,,\quad \Omega_{s1} = \frac{\cos\omega_0-\cos\omega_{s1}}{\sin\omega_{s1}}\,,\quad \Omega_{s2} = \frac{\cos\omega_0-\cos\omega_{s2}}{\sin\omega_{s2}}$$
$$\Omega_p = \tan\left(\frac{\omega_{p2}-\omega_{p1}}{2}\right),\quad \Omega_s = \min\left(|\Omega_{s1}|, |\Omega_{s2}|\right) \tag{13.11.3}$$

These choices match the passband specifications. An alternative choice that matches the stopband specifications is as follows:

$$\cos\omega_0 = \frac{\sin(\omega_{s1}+\omega_{s2})}{\sin\omega_{s1}+\sin\omega_{s2}}\,,\quad \Omega_{p1} = \frac{\cos\omega_0-\cos\omega_{p1}}{\sin\omega_{p1}}\,,\quad \Omega_{p2} = \frac{\cos\omega_0-\cos\omega_{p2}}{\sin\omega_{p2}}$$
$$\Omega_p = \max\left(|\Omega_{p1}|, |\Omega_{p2}|\right),\quad \Omega_s = \tan\left(\frac{\omega_{s2}-\omega_{s1}}{2}\right) \tag{13.11.4}$$

For *bandstop* designs with a stopband interval $[f_{s1}, f_{s2}]$ given as a subset of a passband interval $[f_{p1}, f_{p2}]$, we calculate the analog filter's parameters as follows, where $\omega_{p1} = 2\pi f_{p1}/f_s$, $\omega_{p2} = 2\pi f_{p2}/f_s$, $\omega_{s1} = 2\pi f_{s1}/f_s$, $\omega_{s2} = 2\pi f_{s2}/f_s$, choosing to match the passband as in Sec. 12.12:

$$\cos\omega_0 = \frac{\sin(\omega_{p1}+\omega_{p2})}{\sin\omega_{p1}+\sin\omega_{p2}}\,,\quad \Omega_{s1} = \frac{\sin\omega_{s1}}{\cos\omega_0-\cos\omega_{s1}}\,,\quad \Omega_{s2} = \frac{\sin\omega_{s2}}{\cos\omega_0-\cos\omega_{s2}}$$

$$\Omega_p = \cot\left(\frac{\omega_{p2}-\omega_{p1}}{2}\right)\,,\quad \Omega_s = \min\big(|\Omega_{s1}|,|\Omega_{s2}|\big) \tag{13.11.5}$$

Alternatively, we may match the stopband specifications:

$$\cos\omega_0 = \frac{\sin(\omega_{s1}+\omega_{s2})}{\sin\omega_{s1}+\sin\omega_{s2}}\,,\quad \Omega_{p1} = \frac{\sin\omega_{p1}}{\cos\omega_0-\cos\omega_{p1}}\,,\quad \Omega_{p2} = \frac{\sin\omega_{p2}}{\cos\omega_0-\cos\omega_{p2}}$$

$$\Omega_p = \max\big(|\Omega_{p1}|,|\Omega_{p2}|\big)\,,\quad \Omega_s = \cot\left(\frac{\omega_{s2}-\omega_{s1}}{2}\right) \tag{13.11.6}$$

13.12 MATLAB Implementation and Examples

Once the parameters Ω_p, Ω_s have been determined, the equivalent lowpass analog prototype filter can be designed, based on the specifications $\{\Omega_p, \Omega_s, A_p, A_s\}$ using for example the function `lpa`, and the analog zeros and poles may be mapped into those of the digital filter.

In order to emulate MATLAB's built-in filter design functions, the above design steps have been divided into two stages, In the first stage, the function `dford.m` uses the given specifications to determine the analog filter order N and the frequency and attenuation level that must be matched exactly:

```
% dford.m  -  digital filter order determination
%
% Usage: [N,Ad,wd] = dford(wp,ws,Ap,As,type,match);
%
% wp,ws = passband and stopband frequencies in rads/sample, (1d for LP/HP, 2d for BP/BS)
% Ap,As = passband and stopband attenuations in dB
% type  = 0,1,2,3 for Butterworth, Chebyshev-1, Chebyshev-2, Elliptic
% match = ’p’, ’s’, if omitted it defaults to ’p’ for type=0,1,3, ’s’ for type=2
%
% N  = filter order of LP analog prototype
% Ad = attenuation in dB to be matched exactly at design frequency wd
% wd = design frequency (1d or 2d) that must be matched exactly
%
% the outputs N,Ad,wd may be passed directly to DFDES to design the filter
%
% it determines the type LP/HP/BP/BS from wp,ws using the following conventions:
%
%    match=’p’ matches passband specs exactly
%    match=’s’ matches stopband specs exactly
%
%    LP: wp,ws are scalars with wp < ws
%    HP: wp,ws are scalars with wp > ws
%    BP: wp = [wp1,wp2], ws = [ws1,ws2], with ws1 < wp1 < wp2 < ws2
%    BS: wp = [wp1,wp2], ws = [ws1,ws2], with wp1 < ws1 < ws2 < wp2
```

This function serves as substitute for the built-in functions `buttord`, `cheb1ord`, `cheb2ord`, and `ellipord`. In the second stage, the function `dfdes.m` uses the calculated filter order and the matched frequency band and attenuation to calculate the analog filter poles and zeros, remap them to the digital ones, and construct the second-order or fourth-order section coefficients using Eq. (13.10.24) or (13.10.28):

```
% dfdes.m  -  digital filter design with the bilinear transformation
%
% Usage: [B,A,w0] = dfdes(N,Ad,wd,type,shape,coeffs)
%
%        [B,A,w0] = dfdes(N,Ad,wd,type,shape)          % equivalent to coeffs=’4os’
%        [B,A,w0] = dfdes(N,Ad,wd,type,shape,’4os’)    % fourth-order sections
%        [B,A,w0] = dfdes(N,Ad,wd,type,shape,’sos’)    % second-order sections
%        [B,A,w0] = dfdes(N,Ad,wd,type,shape,’hsos’)   % hat-second-order sections
%        [B,A,w0] = dfdes(N,Ad,wd,type,shape,’dir’)    % direct-form coefficients
%
% N      = filter order of LP analog prototype, digital filter order is 2*N
% Ad     = attenuation in dB at the design frequency wd, (Ad is 1d, 2d, or 3d row)
% wd     = design frequency in radians/sample, (wd is 1d for LP/HP, or, 2d for BP/BS)
% type   = 0,1,2,3 for Butterworth, Chebyshev-1, Chebyshev-2, Elliptic
% shape  = ’LP’, ’HP’, ’BP’, ’BS’, for lowpass, highpass, bandpass, bandstop designs
% coeffs = ’4os’, ’sos’, ’hsos’, ’dir’
%
% B,A = (L+1)x5 for ’4os’, (L+1)x3 for ’hsos’, (2L+1)x3 for ’sos’, (2N+1)x1 for ’dir’
% w0  = center freq (rads/sample) of the wd band for BP/BS, w0=0 or pi for LP or HP
%
% note that N = 2L+r, r=0,1, L = floor(N/2) = number of SOSs of LP analog prototype
%
% Ad,wd are entered using the following conventions:
%
% [B,A,w0] = dfdes(N,Ap,wp,0,shape,coeffs)         match Ap at wp
% [B,A,w0] = dfdes(N,Ap,wp,1,shape,coeffs)         match Ap at wp
% [B,A,w0] = dfdes(N,As,ws,2,shape,coeffs)         match As at ws
% [B,A,w0] = dfdes(N,[Ap,As],wp,3,shape,coeffs)    match Ap at wp, equiripple at Ap,As
%
% [B,A,w0] = dfdes(N,[Ap,Ab],wb,1,shape,coeffs)    match Ab at wb, equiripple at Ap
% [B,A,w0] = dfdes(N,[Ab,As],wb,2,shape,coeffs)    match Ab at wb, equiripple at As
% [B,A,w0] = dfdes(N,[Ap,Ab,As],wb,3,shape,coeffs) match Ab at wb, equiripple at Ap,As
% [B,A,w0] = dfdes(N,[Ap,As,As],ws,3,shape,coeffs) match As at ws, equiripple at Ap,As
```

This function serves as substitute for the built-in functions `butter`, `cheby1`, `cheby2`, and `ellip`. Fig. 13.12.1 depicts Butterworth, Chebyshev-1, Chebyshev-2, and elliptic digital filter designs of lowpass, highpass, bandpass, and bandstop filters designed with the same passband and stopband attenuations as in Eq. (13.1.10) and with the following frequency specifications (in kHz):

$$\begin{aligned}
&\text{LP case:}\quad f_s = 20\,,\quad f_{\text{pass}} = 4.0\,,\quad f_{\text{stop}} = 4.5\\
&\text{HP case:}\quad f_s = 20\,,\quad f_{\text{pass}} = 4.5\,,\quad f_{\text{stop}} = 4.0\\
&\text{BP case:}\quad f_s = 20\,,\quad [f_{p1}, f_{p2}] = [3.0, 6.0]\,,\quad [f_{s1}, f_{s2}] = [2.5, 6.5]\\
&\text{BS case:}\quad f_s = 20\,,\quad [f_{p1}, f_{p2}] = [2.5, 6.5]\,,\quad [f_{s1}, f_{s2}] = [3.0, 6.0]
\end{aligned}$$

For example, the elliptic designs may be generated by the code fragment:

```
Gp = 0.95; Ap = -20*log10(Gp);
Gs = 0.05; As = -20*log10(Gs);

fsamp=20;
f = linspace(0,10,1001);
w = 2*pi*f/fsamp;

% -------- LP -------------------------------------------
fp = 4; fs = 4.5; wp = 2*pi*fp/fsamp; ws = 2*pi*fs/fsamp;
[N,Ad,wd] = dford(wp,ws,Ap,As,3,'s');
[B,A] = dfdes(N,Ad,wd,3,'LP','sos');
H = fresp(B,A,w); figure; plot(f,abs(H),'r');

% -------- HP -------------------------------------------
fp = 4.5; fs = 4; wp = 2*pi*fp/fsamp; ws = 2*pi*fs/fsamp;
[N,Ad,wd] = dford(wp,ws,Ap,As,3,'s');
[B,A] = dfdes(N,Ad,wd,3,'HP','sos');
H = fresp(B,A,w); figure; plot(f,abs(H),'r');

% -------- BP -------------------------------------------
fp1=3; fp2=6; fs1=2.5; fs2=6.5;
wp = 2*pi*[fp1,fp2]/fsamp; ws = 2*pi*[fs1,fs2]/fsamp;
[N,Ad,wd] = dford(wp,ws,Ap,As,3,'s');
[B,A] = dfdes(N,Ad,wd,3,'BP','sos');
H = fresp(B,A,w); figure; plot(f,abs(H),'r');

% -------- BS -------------------------------------------
fp1=2.5; fp2=6.5; fs1=3; fs2=6;
wp = 2*pi*[fp1,fp2]/fsamp; ws = 2*pi*[fs1,fs2]/fsamp;
[N,Ad,wd] = dford(wp,ws,Ap,As,3,'s');
[B,A] = dfdes(N,Ad,wd,3,'BS','sos');
H = fresp(B,A,w); figure; plot(f,abs(H),'r');
```

where we have chosen to match the stopbands exactly and output the filter coefficients as second-order sections. The frequency response evaluation of the cascaded sections was implemented with the help of the MATLAB function `fresp.m`, borrowed from [329]. The designed transfer functions and coefficient matrices are as follows:

LP case:

$$B=\begin{bmatrix}0.3204 & 0.3204 & 0\\ 0.8591 & -0.2363 & 0.8591\\ 0.4534 & 0.1206 & 0.4534\end{bmatrix},\quad A=\begin{bmatrix}1 & -0.3593 & 0\\ 1 & -0.4436 & 0.9255\\ 1 & -0.5547 & 0.5821\end{bmatrix},\quad N=5$$

$$H_{\text{LP}}(z)=\frac{0.3204(1+z^{-1})}{1-0.3593z^{-1}}\cdot\frac{0.8591-0.2363z^{-1}+0.8591z^{-2}}{1-0.4436z^{-1}+0.9255z^{-2}}\cdot\frac{0.4534+0.1206z^{-1}+0.4534z^{-2}}{1-0.5547z^{-1}+0.5821z^{-2}}$$

HP case:

$$B=\begin{bmatrix}0.4317 & -0.4317 & 0\\ 0.8986 & -0.5866 & 0.8986\\ 0.5615 & -0.6118 & 0.5615\end{bmatrix},\quad A=\begin{bmatrix}1 & 0.1366 & 0\\ 1 & -0.4582 & 0.9257\\ 1 & -0.1727 & 0.5621\end{bmatrix},\quad N=5$$

BP case:

$$B=\begin{bmatrix}0.9500 & 0 & 0\\ 0.8161 & -1.1771 & 0.8161\\ 0.4017 & -0.7171 & 0.4017\\ 0.8161 & 0.7778 & 0.8161\\ 0.4017 & 0.6260 & 0.4017\end{bmatrix},\quad A=\begin{bmatrix}1 & 0 & 0\\ 1 & -1.2501 & 0.9253\\ 1 & -0.8124 & 0.6129\\ 1 & 0.6965 & 0.9093\\ 1 & 0.2530 & 0.5697\end{bmatrix},\quad N=4$$

BS case:

$$B=\begin{bmatrix}0.9500 & 0 & 0\\ 0.9081 & -1.0417 & 0.9081\\ 0.6221 & -0.4912 & 0.6221\\ 0.9081 & 0.5257 & 0.9081\\ 0.6221 & 0.0778 & 0.6221\end{bmatrix},\quad A=\begin{bmatrix}1 & 0 & 0\\ 1 & -1.2399 & 0.9239\\ 1 & -1.0384 & 0.5163\\ 1 & 0.7432 & 0.9090\\ 1 & 0.6453 & 0.4377\end{bmatrix},\quad N=4$$

The functions **lpa**, **dford**, and **dfdes** are available in the ISP toolbx.

13.13 Frequency-Shifted Realizations

Besides the conventional realizations based on second- or fourth-order sections, it is possible to realize the digital filter as a cascade of second-order sections in the $\hat{z}^{-1}$ variable. These realizations are also generated by the function `dfdes` and have transfer function:

$$\begin{aligned}H(z)&=H_0\left[G_0\,\frac{1+\hat{z}^{-1}}{1-\hat{p}_0\hat{z}^{-1}}\right]^r\prod_{i=1}^{L}\left[|G_i|^2\,\frac{(1-\hat{z}_i\hat{z}^{-1})\,(1-\hat{z}_i^*\hat{z}^{-1})}{(1-\hat{p}_i\hat{z}^{-1})\,(1-\hat{p}_i^*\hat{z}^{-1})}\right]\\ &\equiv H_0\left[\frac{\hat{b}_{00}+\hat{b}_{01}\hat{z}^{-1}}{1+\hat{a}_{01}\hat{z}^{-1}}\right]^r\prod_{i=1}^{L}\left[\frac{\hat{b}_{i0}+\hat{b}_{i1}\hat{z}^{-1}+\hat{b}_{i2}\hat{z}^{-2}}{1+\hat{a}_{i1}\hat{z}^{-1}+\hat{a}_{i2}\hat{z}^{-2}}\right]\end{aligned}\tag{13.13.1}$$

where $\hat{z}^{-1}$ must be replaced by

$$\hat{z}^{-1}=q\,\frac{z^{-1}(c_0-z^{-1})}{1-c_0z^{-1}}\tag{13.13.2}$$

This transformation may be represented by the block diagram shown in Fig. 13.13.1, where $c_0=\cos\omega_0$ and $s_0=\sin\omega_0$. This diagram is the so-called normalized lattice realization of Eq. (13.13.2). Other realizations of (13.13.2) are possible [323–326].

If one has a realization of Eq. (13.13.1), then each unit-delay $\hat{z}^{-1}$ may be replaced by the block diagram of Fig. 13.13.1, resulting in a realization of the final digital filter. For example, Fig. 13.13.2 shows the transposed realization of one of the second-order sections and its shifted version.

The advantage of such realizations is that they decouple the dependence on the center frequency ω_0. The hat-coefficients depend only on the desired bandwidth and attenuations, and not on ω_0. Thus, one can design a lowpass filter and shift it to any center frequency ω_0, transforming it into a bandpass (or bandstop) filter.

The MATLAB function `dfdes` calculates (with argument `coeffs` set to `'hsos'`) the hat-coefficients from the order N of the analog prototype and a prescribed frequency

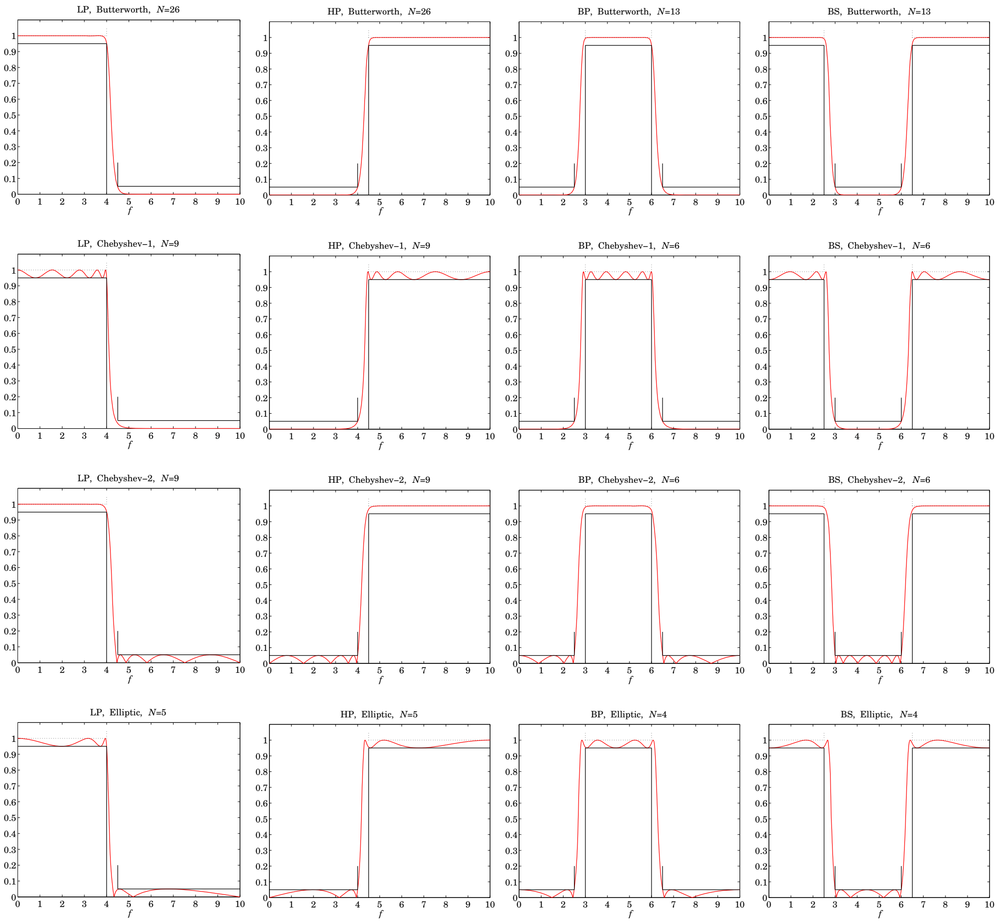


Fig. 13.12.1 Digital LP, HP, BP, and BS filters.

band $[\omega_{d1}, \omega_{d2}]$ that is matched exactly at a desired attenuation level A_d (chosen to be either the passband or the stopband.)

The bandwidth $\Delta\omega_d = \omega_{d2} - \omega_{d1}$ is used internally by `dfdes` to calculate the analog design parameter $\Omega_d = \tan(\Delta\omega_d/2)$. Then, Ω_d is mapped to the passband parameter Ω_p, which is used to design of the analog prototype filter.

To design a bandpass filter with a given bandwidth of $\Delta\omega_d$ and center frequency ω_0, one may start by designing a lowpass digital filter with cutoff frequency $\hat{\omega}_d = \Delta\omega_d$ matched at level A_d, and then shift it to ω_0. Since ω_0 and $\Delta\omega_d$ are given, the bandedge frequencies $[\omega_{d1}, \omega_{d2}]$ cannot be independently specified, but may be calculated by the following formulas [329]:

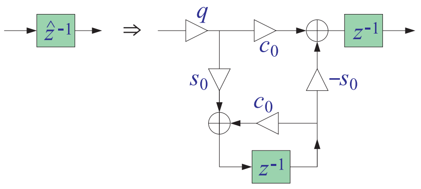


Fig. 13.13.1 LP to BP/BS frequency transformation, $q = 1$ for BP, $q = -1$ for BS.

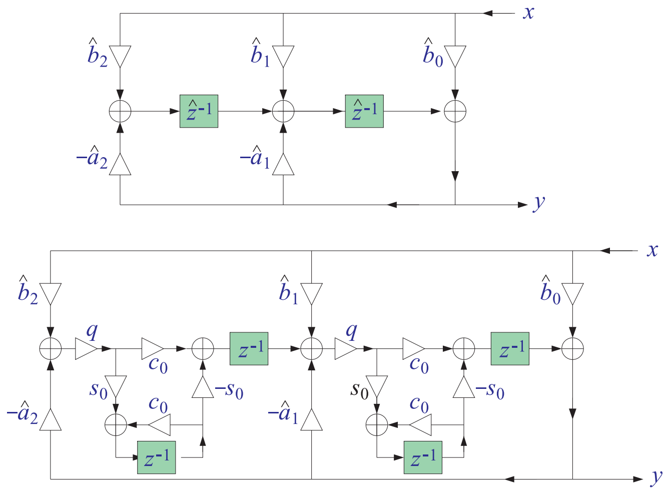


Fig. 13.13.2 Frequency-shifted transposed realization.

$$\cos\omega_{d1} = \frac{c_0 + \Omega_d\sqrt{\Omega_d^2 + s_0^2}}{1 + \Omega_d^2}, \quad \cos\omega_{d2} = \frac{c_0 - \Omega_d\sqrt{\Omega_d^2 + s_0^2}}{1 + \Omega_d^2} \tag{13.13.3}$$

where $\Omega_d = \tan(\Delta\omega_d/2)$, and $c_0 = \cos\omega_0$, $s_0 = \sin\omega_0$. These are derived by demanding that the interval $[\omega_{d1}, \omega_{d2}]$ be mapped onto the analog lowpass interval $[-\Omega_d, \Omega_d]$ through the bilinear transformation of Eq. (13.9.2), that is,

$$-\Omega_d = \frac{c_0 - \cos\omega_{d1}}{\sin\omega_{d1}}, \quad \Omega_d = \frac{c_0 - \cos\omega_{d2}}{\sin\omega_{d2}}$$

If the bandedge frequencies $[\omega_{d1}, \omega_{d2}]$ are specified, then, $\Delta\omega_d = \omega_{d2} - \omega_{d1}$, and the center frequency ω_0 is calculated by

$$\cos\omega_0 = \frac{\sin(\omega_{d1} + \omega_{d2})}{\sin\omega_{d1} + \sin\omega_{d2}} \tag{13.13.4}$$

Eqs. (13.13.3) and (13.13.4) are valid also for shifted bandstop filters, but now the lowpass filter's design frequency must be measured from Nyquist, that is, $\hat{\omega}_d = \pi - \Delta\omega_d$, because the LP stopband will be transformed to the BS stopband.

Next, we look at some design examples. Fig. 13.13.3 shows a Chebyshev-2 bandpass digital filter with sampling rate $f_s = 20$ kHz and bandwidth of $\Delta f_p = 3$ kHz measured at the passband level of $A_p = -20\log_{10}(0.95)$ dB, and shifted to the passband center frequency of $f_0 = 4$ kHz.

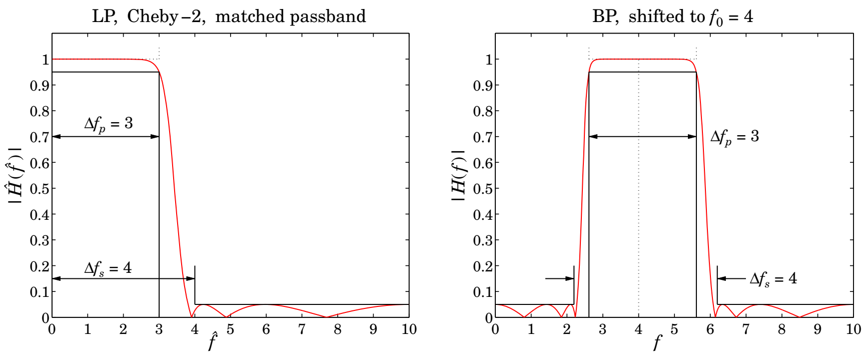


Fig. 13.13.3 LP Chebyshev-2 shifted to BP with passband centered at $f_0 = 4$.

The bandpass filter was obtained by shifting a lowpass digital filter that was designed with passband frequency $\hat{f}_{\text{pass}} = \Delta f_p = 3$ kHz at level A_p and stopband frequency $\hat{f}_{\text{stop}} = 4$ kHz at the stopband level of $A_s = -10\log_{10}(0.05)$ dB. The passband and stopband edge frequencies of the shifted filter were calculated from Eq. (13.13.3) and are shown as brick-walls on Fig. 13.13.3:

$$[f_{p1}, f_{p2}] = [2.6121\,,\ 5.6121]\ \text{kHz}\,,\quad [f_{s1}, f_{s2}] = [2.1957\,,\ 6.1957]\ \text{kHz}$$

The passband frequencies satisfy $\Delta f_p = f_{p2} - f_{p1} = 3$ kHz. The stopband edge frequencies were calculated by using the center frequency $f_0 = 4$ and the width of the lowpass filter's stopband, that is, $\Delta f_s = \hat{f}_{\text{stop}} = 4$ kHz. The resulting filter order was $N = 6$. The hat-second-order section coefficients were:

$$\hat{B} = \begin{bmatrix} 1 & 0 & 0 \\ 0.6796 & -0.4558 & 0.6796 \\ 0.4768 & -0.0352 & 0.4768 \\ 0.2919 & 0.4366 & 0.2919 \end{bmatrix}, \quad \hat{A} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & -0.8721 & 0.7755 \\ 1 & -0.4583 & 0.3767 \\ 1 & -0.0335 & 0.0539 \end{bmatrix}$$

The c_0 parameter was equal to 0.3090. The MATLAB code used to generate Fig. 13.13.3 was:

```
Gp = 0.95; Gs = 0.05; Ap = -20*log10(Gp); As = -20*log10(Gs);

fsamp = 20; fpass = 3; fstop = 4; f0 = 4;
wp = 2*pi*fpass/fsamp; ws = 2*pi*fstop/fsamp; w0 = 2*pi*f0/fsamp;
```

```
[N,Ad,wd] = dford(wp,ws,Ap,As,2,'p');          % match passband
[Bh,Ah] = dfdes(N,Ad,wd,2,'LP','hsos');        % hat-sos sections

f = linspace(0,10,1001); w = 2*pi*f/fsamp;

HLP = fresp(Bh,Ah,w);                 % frequency response of LP digital filter
HBP = fresp(Bh,Ah,w,w0);              % frequency response of shifted LP filter

figure; plot(f,abs(HLP),'r');
figure; plot(f,abs(HBP),'r');

c0 = cos(w0); s0=sin(w0);             % calculate bandedge frequencies
Wp = tan(wp/2);
wp1 = acos((c0 + Wp*sqrt(Wp^2+s0^2))/(1+Wp^2)); fp1 = wp1*fsamp/2/pi;
wp2 = acos((c0 - Wp*sqrt(Wp^2+s0^2))/(1+Wp^2)); fp2 = wp2*fsamp/2/pi;

Ws = tan(ws/2);
ws1 = acos((c0 + Ws*sqrt(Ws^2+s0^2))/(1+Ws^2)); fs1 = ws1*fsamp/2/pi;
ws2 = acos((c0 - Ws*sqrt(Ws^2+s0^2))/(1+Ws^2)); fs2 = ws2*fsamp/2/pi;
```

Fig. 13.13.4 shows another example in which the same lowpass digital filter was transformed to a bandpass filter with a passband given by $[f_{p1}, f_{p2}] = [2, 5]$ kHz, which has the same bandwidth $\Delta f_p = 5 - 2 = 3$ kHz as the previous example. In this case, because the bandedge frequencies were given, the center frequency ω_0 was calculated by Eq. (13.13.4), and the stopband edge frequencies by Eq. (13.13.3):

$$f_0 = 3.2982\ \text{kHz}\,,\quad [f_{s1}, f_{s2}] = [1.6476,\ ,\ 5.6476]\ \text{kHz}$$

By construction, the stopband width was as before, that is, $\Delta f_s = f_{s2} - f_{s1} = 4$ kHz. The hat-coefficients were the same as in the previous example, but the new value of c_0 was 0.5095.

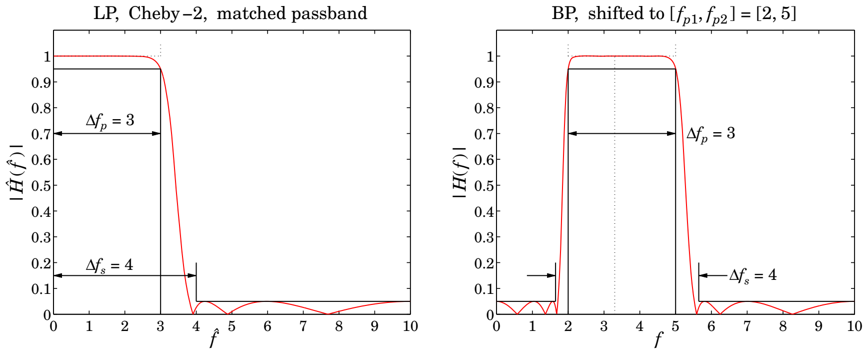


Fig. 13.13.4 LP Chebyshev-2 shifted to BP with passband at $[f_{p1}, f_{p2}] = [2, 5]$.

In the example shown in Fig. 13.13.5, the digital lowpass filter was designed with the same specifications as the previous two examples, but the stopband was matched exactly. The resulting hat-coefficients were:

$$\hat{B} = \begin{bmatrix} 1 & 0 & 0 \\ 0.6843 & -0.3796 & 0.6843 \\ 0.4830 & 0.0262 & 0.4830 \\ 0.3065 & 0.4749 & 0.3065 \end{bmatrix}, \quad \hat{A} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & -0.7805 & 0.7695 \\ 1 & -0.3760 & 0.3683 \\ 1 & 0.0340 & 0.0539 \end{bmatrix}$$

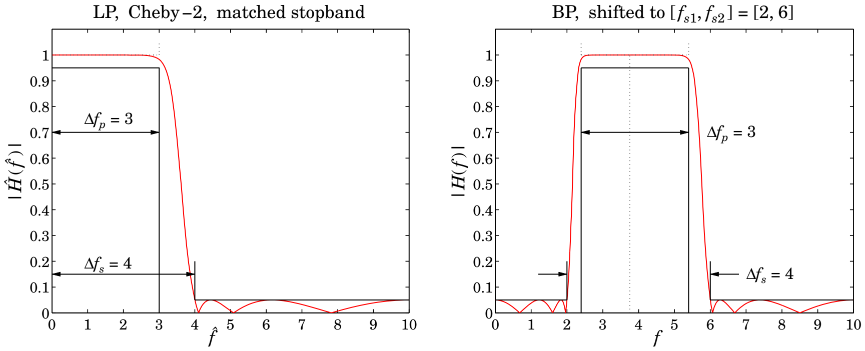


Fig. 13.13.5 LP Chebyshev-2 shifted to BP with stopband at $[f_{s1}, f_{s2}] = [2, 6]$.

The 4-kHz stopband frequency of the lowpass filter was shifted to the stopband interval $[f_{s1}, f_{s2}] = [2, 6]$ kHz, so that $\Delta f_s = 6 - 2 = 4$ kHz. The center frequency f_0 was calculated on the basis of the stopband frequencies using Eq. (13.13.4), and then, the passband frequencies were determined using (13.13.3) and satisfying $\Delta f_p = f_{p2} - f_{p1} = 3$ kHz:

$$f_0 = 3.7525 \text{ kHz}, \quad c_0 = 0.3820, \quad [f_{p1}, f_{p2}] = [2.3946, 5.3946]\text{kHz}$$

Fig. 13.13.6 shows a bandstop design with passband and stopband bandwidths of $\Delta f_p = 4$ and $\Delta f_s = 3$ kHz. The corresponding passband and stopband frequencies of the equivalent lowpass digital filter must be measured from Nyquist, that is,

$$\hat{f}_{\text{pass}} = \frac{f_s}{2} - \Delta f_p = 10 - 4 = 6 \text{ kHz}, \quad \hat{f}_{\text{stop}} = \frac{f_s}{2} - \Delta f_s = 10 - 3 = 7 \text{ kHz}$$

The lowpass digital filter was designed with the same attenuation A_p, A_s as the previous three examples with matching the passband exactly. The lowpass filter was shifted to the center frequency $f_0 = 4$ kHz, from which the passband and stopband edge frequencies of the bandstop filter were found to be:

$$[f_{p1}, f_{p2}] = [2.1957, 6.1957] \text{ kHz}, \quad [f_{s1}, f_{s2}] = [2.6121, 5.6121] \text{ kHz}$$

The c_0 parameter was 0.3090, and the hat-coefficients:

$$\hat{B} = \begin{bmatrix} 1 & 0 & 0 \\ 0.8043 & 0.9141 & 0.8043 \\ 0.6460 & 0.9598 & 0.6460 \\ 0.5565 & 1.0698 & 0.5565 \end{bmatrix}, \quad \hat{A} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0.7548 & 0.7680 \\ 1 & 0.8176 & 0.4342 \\ 1 & 0.9320 & 0.2508 \end{bmatrix}$$

The MATLAB code used to generate Fig. 13.13.6 was:

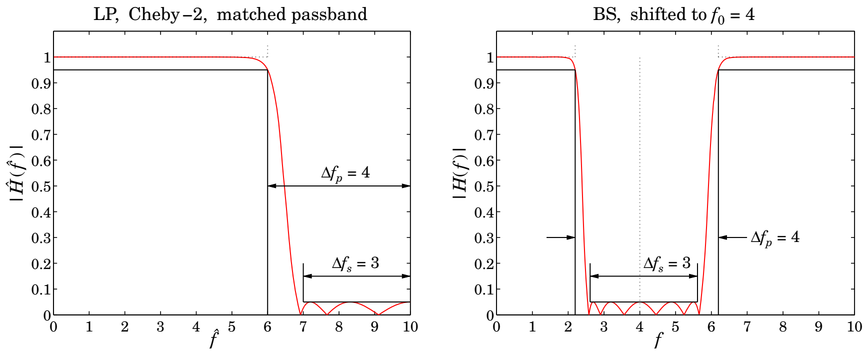


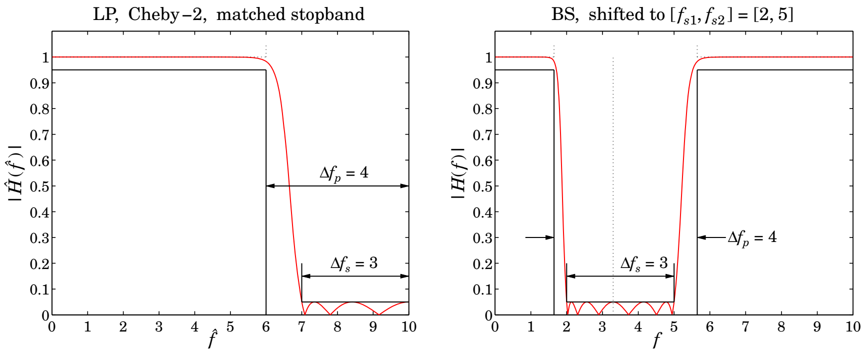
Fig. 13.13.6 LP Chebyshev-2 shifted to BS with passband centered at $f_0 = 4$.

```
Gp = 0.95; Gs = 0.05; Ap = -20*log10(Gp); As = -20*log10(Gs);

fsamp = 20; Dfp = 4; Dfs = 3; f0 = 4;
Dwp = 2*pi*Dfp/fsamp; Dws = 2*pi*Dfs/fsamp; w0 = 2*pi*f0/fsamp;

fpass = fsamp/2 - Dfp;  fstop = fsamp/2 - Dfs;
wp = 2*pi*fpass/fsamp;  ws = 2*pi*fstop/fsamp;

[N,Ad,wd] = dford(wp,ws,Ap,As,2,'p');           % match passband
[Bh,Ah] = dfdes(N,Ad,wd,2,'LP','hsos');         % output hat-sos sections

f = linspace(0,10,1001); w = 2*pi*f/fsamp;

HLP = fresp(Bh,Ah,w);                      % frequency response of LP digital filter
q = -1;                                    % shift LP to BS
HBS = fresp(Bh,Ah,w,w0,q);                 % frequency response of shifted LP filter

figure; plot(f,abs(HLP),'r');
figure; plot(f,abs(HBS),'r');

c0 = cos(w0), s0 = sin(w0);                % calculate bandedge frequencies
Wp = tan(Dwp/2);;
wp1 = acos((c0 + Wp*sqrt(Wp^2+s0^2))/(1+Wp^2)); fp1 = wp1*fsamp/2/pi;
wp2 = acos((c0 - Wp*sqrt(Wp^2+s0^2))/(1+Wp^2)); fp2 = wp2*fsamp/2/pi;

Ws = tan(Dws/2);
ws1 = acos((c0 + Ws*sqrt(Ws^2+s0^2))/(1+Ws^2)); fs1 = ws1*fsamp/2/pi;
ws2 = acos((c0 - Ws*sqrt(Ws^2+s0^2))/(1+Ws^2)); fs2 = ws2*fsamp/2/pi;
```

The shifted frequency response was computed with the help of the function `fresp` which was modified from that of [329] to handle the bandstop case with $q = -1$ as an additional input.

Fig. 13.13.7 redesigns the lowpass filter of Fig. 13.13.6 to match the stopband exactly, and then shifts it the stopband interval $[f_{s1}, f_{s2}] = [2, 5]$ kHz, from which the center frequency f_0, shift parameter c_0, and passband edge frequencies can be calculated:

$$f_0 = 3.2982 \text{ kHz}, \quad c_0 = 0.5095, \quad [f_{p1}, f_{p2}] = [1.6476, 5.6476]\text{kHz}$$


LP, Cheby−2, matched stopband

BS, shifted to $[f_{s1}, f_{s2}] = [2, 5]$

Fig. 13.13.7 LP Chebyshev-2 shifted to BS with stopband at $[f_{s1}, f_{s2}] = [2, 5]$.

14

Interpolation, Decimation, and Oversampling

14.1 Interpolation and Oversampling

Sampling rate changes are useful in many applications, such as interconnecting digital processing systems operating at different rates [347–350]. Sampling rate increase is accomplished by *interpolation*, that is, the process of inserting additional samples between the original low-rate samples. The inserted, or interpolated, samples are *calculated* by an FIR digital filter.[†] This is illustrated in Fig. 14.1.1 for the case of a 4-fold interpolator which increases the sampling rate by a factor of four, that is, $f_s{}' = 4f_s$.

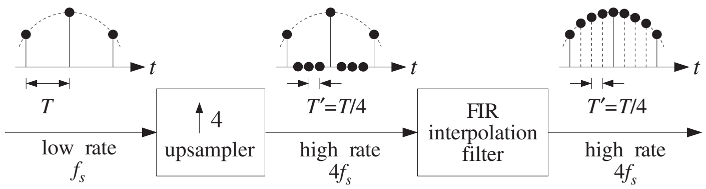


Fig. 14.1.1 Sampling rate increase with digital interpolation.

With respect to the fast time scale, the low-rate samples may be thought of as being separated by three zero samples. The 4-fold *rate expander* or *upsampler* simply inserts three zero samples for every low-rate sample. The job of the FIR filter is to replace the three zeros by the calculated interpolated values.

The interpolating filter is sometimes called an *oversampling digital filter* because it operates at the fast rate $4f_s$. However, because only one out of every four input samples is non-zero, the required filtering operations may be rearranged in such a way as to operate only on the low-rate samples, thus, effectively reducing the computational requirements of the filter—by a factor of four in this case.

This is accomplished by replacing the high-rate interpolating FIR filter by four shorter FIR subfilters, known as *polyphase* filters, operating at the *low* rate f_s. The length of each

[†] IIR filters can also be used, but are less common in practice.

subfilter is one-quarter that of the original filter. Because each low-rate input sample generates four high-rate interpolated outputs (itself and three others), each of the four low-rate subfilters is dedicated to computing only one of the four outputs. Such realization is computationally efficient and lends itself naturally to parallel multiprocessor hardware implementations in which a different DSP chip may be used to implement each subfilter.

An interesting application of interpolation is the use of oversampling digital filters in CD or DAT players, where they help to alleviate the need for high-quality analog anti-image postfilters in the playback system. Moreover, each high-rate sample can be *requantized* without loss of quality to fewer number of bits (even as low as 1 bit per sample) using appropriate *noise shaping* quantizers, thus, trading off bits for samples and simplifying the structure of the analog part of the playback system.

To understand the motivation behind this application, consider an analog signal sampled at a rate f_s, such as 44.1 kHz for digital audio. The analog signal is prefiltered by an analog lowpass *antialiasing prefilter* having cutoff frequency $f_c \leq f_s/2$ and then sampled at rate f_s and quantized. This operation is shown in Fig. 14.1.2.

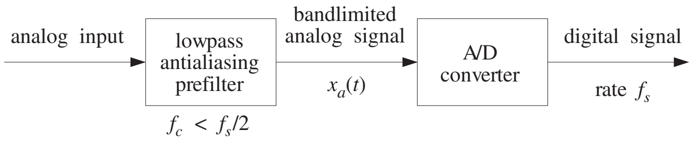


Fig. 14.1.2 Prefiltering and sampling of analog signal.

The prefilter ensures that the spectral images generated by the sampling process at integral multiples of f_s do not overlap, as required by the sampling theorem. This is shown in Fig. 14.1.3 (we ignore here the scaling factor $1/T$).

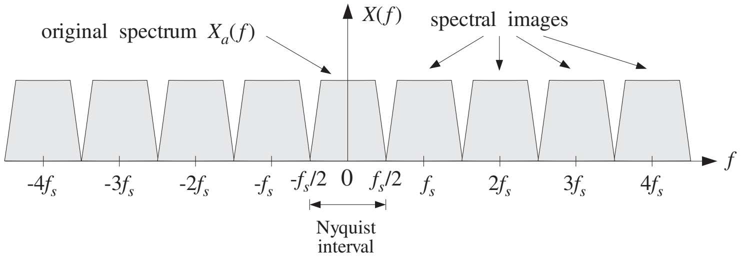


Fig. 14.1.3 Spectrum of signal sampled at low rate f_s.

After digital processing, the sampled signal is reconstructed back to analog form by a D/A *staircase* reconstructor, followed by an analog *anti-image lowpass postfilter* with effective cutoff $f_s/2$, as seen in Fig. 14.1.4.

The D/A converter, with its typical $\sin x/x$ response, removes the spectral images partially; the postfilter completes their removal. The combination of the staircase DAC and the postfilter emulates the *ideal* reconstructing analog filter. The ideal reconstructor is a lowpass filter with cutoff the Nyquist frequency $f_s/2$. It has a very sharp transition between its passband, that is, the Nyquist interval, and its stopband, as shown in Fig. 14.1.5.

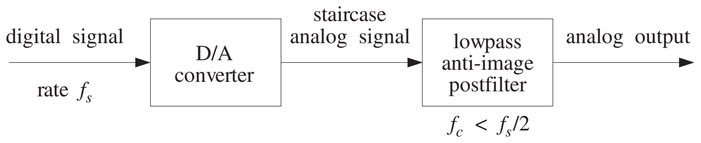


Fig. 14.1.4 Analog reconstruction of sampled signal.

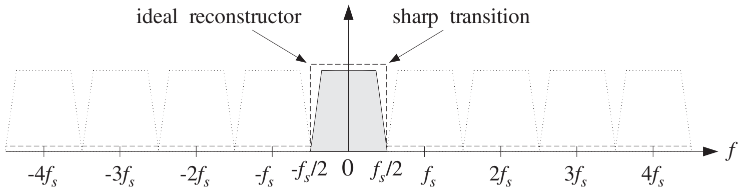


Fig. 14.1.5 Ideal reconstructor removes spectral images due to sampling.

In hi-fi applications such as digital audio, to maintain high quality in the resulting reconstructed analog signal, a very high quality analog postfilter is required, which may be expensive. One way to alleviate the need for a high quality postfilter is to increase the sampling rate. This would cause the spectral images to be more widely separated and, therefore, require a less stringent, simpler, lowpass postfilter. This is depicted in Fig. 14.1.6, for a new sampling rate that is four times higher than required, $f_s{}' = 4f_s$.

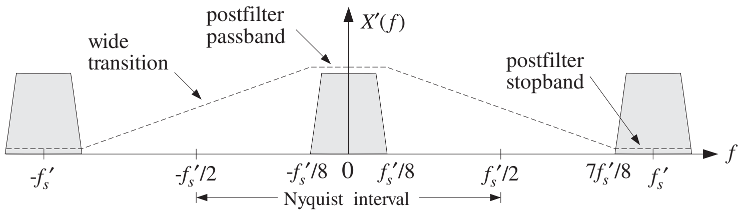


Fig. 14.1.6 Spectrum of signal resampled at high rate $4f_s$, and postfilter requirements.

The *passband* of the postfilter extends up to $f_{\text{pass}} = f_s{}'/8 = f_s/2$, but its *stopband* need only begin at $f_{\text{stop}} = f_s{}' - f_s{}'/8 = 7f_s{}'/8$. It is this wide transition region between passband and stopband that allows the use of a less stringent postfilter. For example, in oversampled digital audio applications, simple third-order Butterworth or Bessel analog postfilters are used. See Section 14.4.4.

The same conclusion can also be drawn in the time domain. Figure 14.1.7 shows the staircase output of the D/A converter for the two sampling rates f_s and $f_s{}' = 4f_s$. It is evident from this figure that the higher the sampling rate, the more closely the staircase output approximates the true signal, and the easier it is for the postfilter to smooth out the staircase levels.

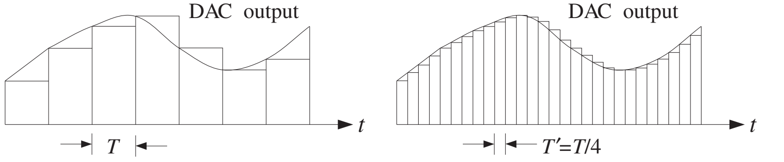


Fig. 14.1.7 Staircase DAC output is smoothed more easily in the oversampled case.

The above approach, however, is impractical because it requires the actual *resampling* of the analog signal at the higher rate $f_s{}'$. For example, in a CD player the low rate samples are already stored on the CD at the prescribed rate of 44.1 kHz and the audio signal cannot be resampled.

The philosophy of oversampling is to increase the sampling rate *digitally* using an interpolation filter which operates only on the available *low-rate* input samples. With respect to the new rate $f_s{}'$ and new Nyquist interval $[-f_s{}'/2, f_s{}'/2]$, the spectrum of the low-rate samples depicted in Fig. 14.1.3 will be as shown in Fig. 14.1.8. This is also the spectrum of the high-rate upsampled signal at the output of the rate expander in Fig. 14.1.1.

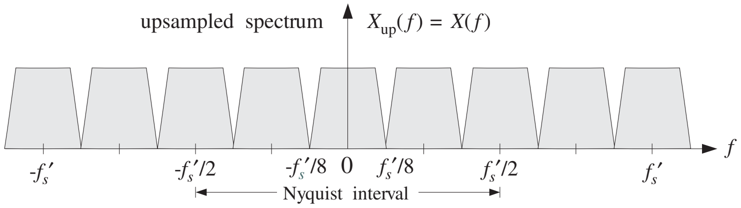


Fig. 14.1.8 Spectrum of low-rate samples with respect to the high rate $4f_s$.

A digital lowpass FIR filter with cutoff frequency $f_s{}'/8$ and operating at the high rate $f_s{}'$, would eliminate the three spectral replicas that lie between replicas at multiples of $f_s{}'$, resulting in a spectrum that is *identical* to that of a signal sampled at the high rate $f_s{}'$, like that shown in Fig. 14.1.6.

The digital filter, being periodic in f with period $f_s{}'$, cannot of course remove the spectral replicas that are centered at integral multiples of $f_s{}'$. Those are removed later by the D/A reconstructor and the anti-image analog postfilter.

The effect of such a digital filter on the spectrum of the low-rate samples is shown in Fig. 14.1.9, both with respect to the physical frequency f in Hz and the corresponding digital frequency, $\omega' = 2\pi f/f_s{}'$, in radians/sample.

In summary, a substantial part of the analog reconstruction process is accomplished by DSP methods, that is, using a digital oversampling filter to remove several adjacent spectral replicas and thereby easing the requirements of the analog postfilter. The required sharp transition characteristics of the overall reconstructor are provided by the digital filter. Thus, the high-quality *analog* postfilter is traded off for a high-quality *digital* filter operating at a higher sampling rate. The overall system is depicted in Fig. 14.1.10.

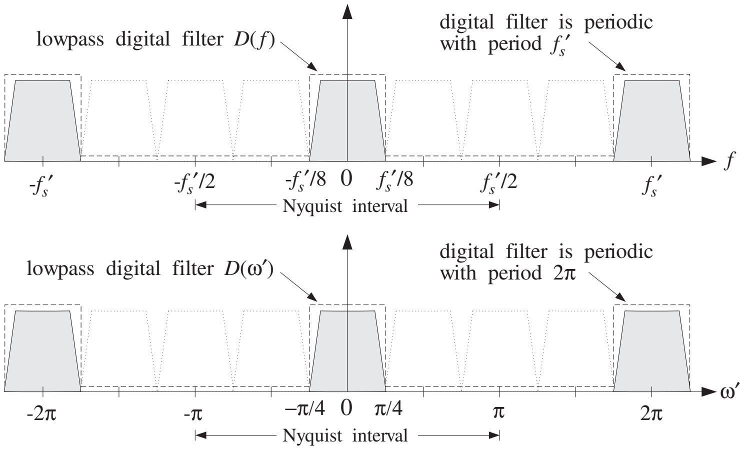

lowpass digital filter D(f)
digital filter is periodic with period f_s′
f
-f_s′
-f_s′/2
-f_s′/8
0
f_s′/8
f_s′/2
f_s′
Nyquist interval
lowpass digital filter D(ω′)
digital filter is periodic with period 2π
ω′
-2π
-π
−π/4
0
π/4
π
2π
Nyquist interval

Fig. 14.1.9 High-rate FIR interpolator removes intermediate spectral images.

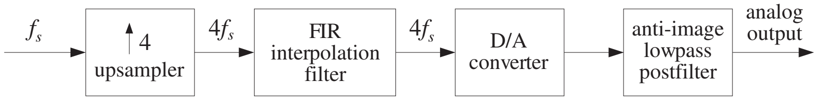


Fig. 14.1.10 4-times oversampling digital filter helps analog reconstruction.

How does an interpolation filter operate in the time domain and calculate the missing signal values between low-rate samples? To illustrate the type of operations it must carry out, consider a 4-fold interpolator and a set of six successive low-rate samples $\{A, B, C, D, E, F\}$ as shown in Fig. 14.1.11.

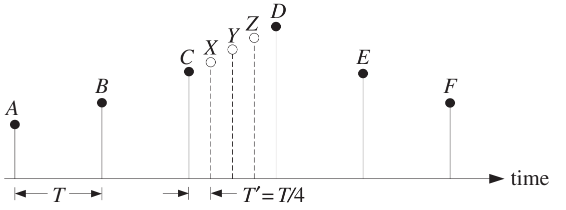


Fig. 14.1.11 Filter calculates missing samples from the surrounding low-rate samples.

The filter calculates three intermediate samples, such as $\{X, Y, Z\}$, between any two low-rate samples by forming *linear combinations* of the surrounding low-rate samples.

Depending on the type of interpolator and desired quality of the calculated values, several different ways of calculating $\{X, Y, Z\}$ are possible. For example, the simplest one is to keep the value of the previous sample C constant throughout the sampling interval and define:

$$X = Y = Z = C$$

This choice corresponds to the so-called *hold* interpolator. Another simple possibility is to interpolate *linearly* between samples $\{C, D\}$ calculating $\{X, Y, Z\}$ as follows:

$$\begin{aligned} X &= 0.75C + 0.25D \\ Y &= 0.50C + 0.50D \\ Z &= 0.25C + 0.75D \end{aligned} \tag{14.1.1}$$

Indeed, the straight line connecting C and D is parametrized as $C + (D - C)t/T$, for $0 \le t \le T$. Setting $t = T'$, $2T'$, $3T'$ with $T' = T/4$ gives the above expressions for $\{X, Y, Z\}$. For more accurate interpolation, more surrounding samples must be taken into account. For example, using four samples we have:

$$\begin{aligned} X &= -0.18B + 0.90C + 0.30D - 0.13E \\ Y &= -0.21B + 0.64C + 0.64D - 0.21E \\ Z &= -0.13B + 0.30C + 0.90D - 0.18E \end{aligned} \tag{14.1.2}$$

corresponding to a length-17 FIR approximation to the ideal interpolation filter. Similarly, a length-25 approximation to the ideal interpolator uses six surrounding low-rate samples as follows:

$$\begin{aligned} X &= 0.10A - 0.18B + 0.90C + 0.30D - 0.13E + 0.08F \\ Y &= 0.13A - 0.21B + 0.64C + 0.64D - 0.21E + 0.13F \\ Z &= 0.08A - 0.13B + 0.30C + 0.90D - 0.18E + 0.10F \end{aligned} \tag{14.1.3}$$

In general, the more the surrounding samples, the more accurate the calculated values. In typical CD players with 4-times oversampling filters, about 20–30 surrounding low-rate samples are used.

The above expressions do not quite look like the convolutional equations of linear filtering. They are special cases of the polyphase realizations of the interpolation filters and are equivalent to convolution. They will be discussed in detail in the next section, where starting with the frequency domain specifications of the filter, its impulse response and corresponding direct and polyphase realization forms are derived. See also Section 14.4.1.

14.2 Interpolation Filter Design

14.2.1 Direct Form

Consider the general case of an L-fold interpolator, which increases the sampling rate by a factor of L, that is, $f_s{}' = Lf_s$. The L-fold rate expander inserts $L - 1$ zeros between adjacent low-rate samples and the corresponding $L-1$ interpolated values are calculated by an FIR digital filter operating at the high rate Lf_s, as shown in Fig. 14.2.1.

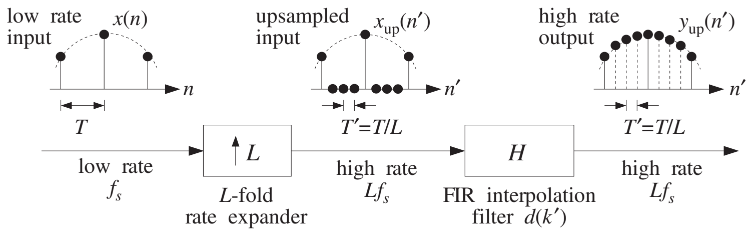


Fig. 14.2.1 L-fold digital interpolator.

Let $x(n)$ denote the low-rate samples that are input to the rate expander and let $x_{\text{up}}(n')$ be its high-rate output, consisting of the low-rate samples separated by $L - 1$ zeros. With respect to the high-rate time index n', the low-rate samples occur every L high-rate ones, that is, at integral multiples of L, $n' = nL$,

$$x_{\text{up}}(nL) = x(n) \tag{14.2.1}$$

The $L - 1$ intermediate samples between $x_{\text{up}}(nL)$ and $x_{\text{up}}(nL + L)$ are zero:

$$x_{\text{up}}(nL + i) = 0, \qquad i = 1, 2, \dots, L - 1 \tag{14.2.2}$$

This is shown in Fig. 14.2.2. More compactly, the upsampled signal $x_{\text{up}}(n')$ can be defined with respect to the high-rate time index n' by:

$$\boxed{x_{\text{up}}(n') = \begin{cases} x(n), & \text{if } n' = nL \\ 0, & \text{otherwise} \end{cases}} \tag{14.2.3}$$

Given an arbitrary value of the high-rate index n', we can always write it *uniquely* in the form $n' = nL + i$, where i is restricted to the range of values $i = 0, 1, \dots, L - 1$.

Mathematically, the integers n and i are the quotient and remainder of the division of n' by L. Intuitively, this means that n' will either fall exactly on a low-rate sample (when $i = 0$), or will fall strictly between two of them ($i \neq 0$). Using $T = LT'$, we find the absolute time in seconds corresponding to n'

$$t = n'T' = nLT' + iT' = nT + iT'$$

that is, it will be offset from a low-rate sampling time by i high-rate sampling units T'. The interpolated values must be computed at these times.

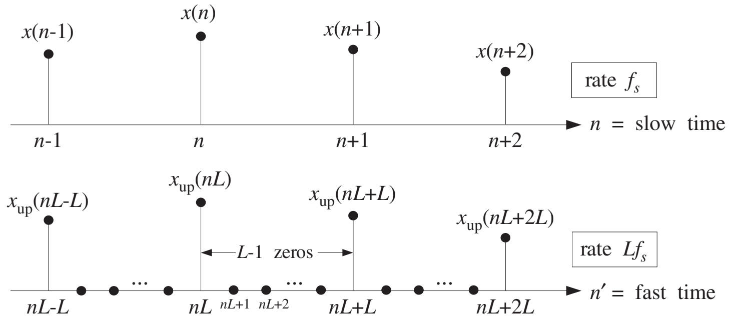


Fig. 14.2.2 Low-rate samples with respect to the slow and fast time scales.

The ideal L-fold interpolation filter is a lowpass filter, operating at the fast rate $f_s{}'$, with cutoff frequency equal to the *low-rate* Nyquist frequency $f_c = f_s/2$, or in terms of $f_s{}'$,

$$\boxed{f_c = \frac{f_s}{2} = \frac{f_s{}'}{2L}} \tag{14.2.4}$$

and expressed in units of the digital frequency, $\omega' = 2\pi f/f_s{}'$:

$$\boxed{\omega_c' = \frac{2\pi f_c}{f_s{}'} = \frac{\pi}{L}} \tag{14.2.5}$$

The frequency response of this filter is shown in Fig. 14.2.3. Its passband *gain* is taken to be L instead of unity. This is justified below. The ideal impulse response coefficients are obtained from the inverse Fourier transform:

$$d(k') = \int_{-\pi}^{\pi} D(\omega')e^{j\omega' k'}\frac{d\omega'}{2\pi} = \int_{-\pi/L}^{\pi/L} Le^{j\omega' k'}\frac{d\omega'}{2\pi} = \frac{\sin(\pi k'/L)}{\pi k'/L}$$

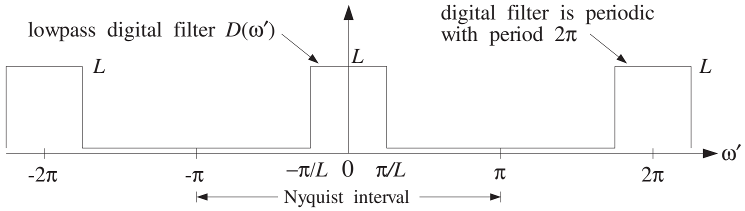


Fig. 14.2.3 Ideal lowpass digital filter operating at high rate Lf_s.

An *FIR approximation* to the ideal interpolator is obtained by truncating $d(k')$ to finite length, say $N = 2LM + 1$:

$$\boxed{d(k') = \frac{\sin(\pi k'/L)}{\pi k'/L}}, \qquad -LM \le k' \le LM \tag{14.2.6}$$

A *causal* version of the filter may be obtained by delaying it by LM samples:

$$h(n') = d(n' - LM) = \frac{\sin\big(\pi(n' - LM)/L\big)}{\pi(n' - LM)/L}, \qquad n' = 0, 1, \dots, N-1$$

And a *windowed* version is obtained by:

$$\boxed{h(n') = w(n')d(n' - LM)}, \qquad n' = 0, 1, \dots, N-1 \tag{14.2.7}$$

where $w(n')$ is an appropriate length-N window, such as a Hamming window:

$$w(n') = 0.54 - 0.46\cos\left(\frac{2\pi n'}{N-1}\right), \qquad n' = 0, 1, \dots, N-1$$

or a Kaiser window. The output of the ideal FIR interpolation filter is obtained by the convolution of the upsampled input $x_{\text{up}}(n')$ with the impulse response $d(k')$:

$$\boxed{y_{\text{up}}(n') = \sum_{k'=-LM}^{LM} d(k')x_{\text{up}}(n' - k')}, \qquad n' = 0, 1, \dots, N-1 \tag{14.2.8}$$

14.2.2 Polyphase Form

The interpolated values between the low-rate samples $x_{\text{up}}(nL)$ and $x_{\text{up}}(nL + L)$, that is, the values at the high-rate time instants $n' = nL + i$, are calculated by the filter as follows:

$$y_{\text{up}}(nL + i) = \sum_{k'=-LM}^{LM} d(k')x_{\text{up}}(nL + i - k'), \qquad i = 0, 1, \dots, L-1 \tag{14.2.9}$$

Writing uniquely $k' = kL + j$, with $0 \le j \le L - 1$, and replacing the single summation over k' by a double summation over k and j, we find

$$y_{\text{up}}(nL + i) = \sum_{k=-M}^{M-1}\sum_{j=0}^{L-1} d(kL + j)x_{\text{up}}(nL + i - kL - j)$$

To be precise, for the case $i = 0$, the summation over k should be over the range $-M \le k \le M$. But as we will see shortly, the term $k = M$ does not contribute to the sum. Defining the ith *polyphase subfilter* by[†]

$$\boxed{d_i(k) = d(kL + i)}, \qquad -M \le k \le M - 1 \tag{14.2.10}$$

for $i = 0, 1, \dots, L - 1$, we can rewrite the ith interpolated sample value as:

[†] For $i = 0$, the range of k is $-M \le k \le M$.

$$y_{\rm up}(nL+i)=\sum_{k=-M}^{M-1}\sum_{j=0}^{L-1}d_j(k)x_{\rm up}(nL-kL+i-j)$$

But the upsampled input signal is non-zero only at times that are integral multiples of L. Therefore, using Eqs. (14.2.1) and (14.2.2), we have

$$x_{\rm up}(nL-kL+i-j)=0,\qquad \text{if } i\neq j$$

This follows from the fact that $|i-j|\leq L-1$. Thus, keeping only the $j=i$ term in the above convolution sum, we obtain

$$y_{\rm up}(nL+i)=\sum_{k=-M}^{M-1}d_i(k)x_{\rm up}(nL-kL),\qquad i=0,1,\dots,L-1 \tag{14.2.11}$$

or, in terms of the low-rate samples:

$$\boxed{y_i(n)=\sum_{k=-M}^{M-1}d_i(k)x(n-k)}\,,\qquad i=0,1,\dots,L-1 \tag{14.2.12}$$

where we set $y_i(n)=y_{\rm up}(nL+i)$. Thus, the ith interpolated value, $y_{\rm up}(nL+i)$, is computed by the ith polyphase subfilter, $d_i(k)$, which has length $2M$ and is acting only on the *low-rate* input samples $x(n)$. Each interpolated value is computed as a linear combination of M low-rate samples above and M below the desired interpolation time, as shown in Fig. 14.2.4.

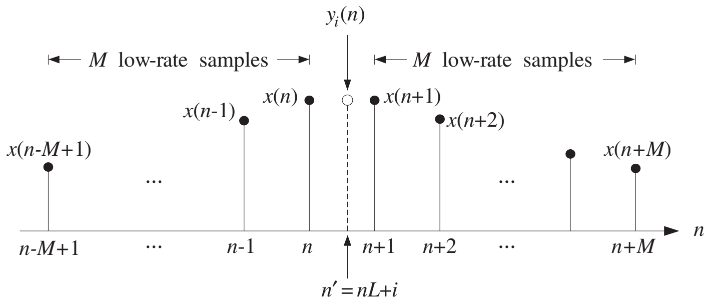


Fig. 14.2.4 Interpolator uses M low-rate samples before and after $y_{\rm up}(nL+i)$.

Using the L subfilters, interpolation is performed at a *reduced* computational cost as compared with the cost of the full, length-N, interpolation filter $d(k')$ acting on the upsampled signal $x_{\rm up}(n')$ by Eq. (14.2.9).

The computational cost of Eq. (14.2.9) is essentially $2LM$ multiplications per interpolated value, or, $2L^2M$ multiplications for computing L interpolated values. By contrast, Eq. (14.2.11) requires $2M$ multiplications per polyphase subfilter, or, $2LM$ multiplications for L interpolated values. Thus, the polyphase subfilter implementation achieves a factor of L in computational savings.

Another way to view the computational rate is in terms of the total number of *multiplications per second* required for the filtering operations, that is,

$$R=N(Lf_s)=NLf_s \qquad \text{(direct form)}$$

$$R=L(2M)f_s=Nf_s \qquad \text{(polyphase form)}$$

where in the *direct* form we have a single filter of length N operating at rate Lf_s and in the *polyphase* form we have L filters operating at f_s, each having computational rate $(2M)f_s$ multiplications per second.[†]

The polyphase implementation is depicted in Fig. 14.2.5, where during each low-rate sampling period T, the commutator reads, in sequence of $T'=T/L$ seconds, the L interpolated values at the outputs of the subfilters.

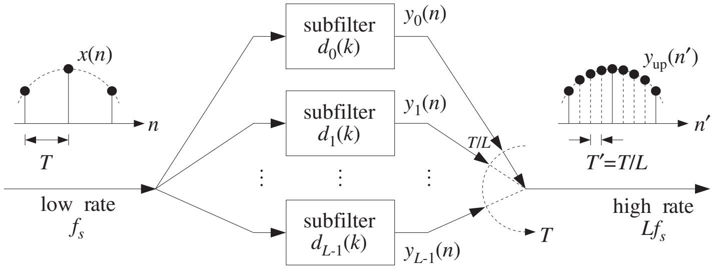


Fig. 14.2.5 Polyphase subfilter implementation of digital interpolator.

This can be seen more formally, as follows. Let ζ^{-1} denote the unit delay with respect to the high rate Lf_s and let z^{-1} denote the low-rate delay. Since L high-rate delays equal one low-rate delay, that is, $LT'=T$, we will have:

$$z=\zeta^L \quad\Leftrightarrow\quad \zeta=z^{1/L} \tag{14.2.13}$$

The ζ-transform of the high-rate filter output $y_{\rm up}(n')$ can be expressed in terms of the z-transforms of the L low-rate output signals $y_i(n)$ as follows. Writing uniquely $n'=nL+i$ with $0\leq i\leq L-1$, we have

$$\begin{aligned}Y_{\rm up}(\zeta)&=\sum_{n'=-\infty}^{\infty}y_{\rm up}(n')\zeta^{-n'}=\sum_{i=0}^{L-1}\sum_{n=-\infty}^{\infty}y_{\rm up}(nL+i)\zeta^{-nL-i}\\&=\sum_{i=0}^{L-1}\zeta^{-i}\sum_{n=-\infty}^{\infty}y_i(n)\zeta^{-Ln}\,,\qquad\text{or,}\end{aligned}$$

[†] Actually, there are $L-1$ subfilters of length $(2M)$ and one (the filter d_0) of length $(2M+1)$, giving rise to $R=(L-1)(2M)f_s+(2M+1)f_s=Nf_s$, where $N=2LM+1$.

$$\boxed{Y_{\text{up}}(\zeta)=\sum_{i=0}^{L-1}\zeta^{-i}Y_i(\zeta^L)=\sum_{i=0}^{L-1}z^{-i/L}Y_i(z)} \tag{14.2.14}$$

which shows how the L low-rate output signals are put together, in sequence of T/L high-rate delays to make up the high-rate interpolated output signal. In a similar fashion, we can derive the relationship between the ζ-transform of the high-rate filter (14.2.6) and the z-transforms of its polyphase subfilters (14.2.10), justifying the realization of Fig. 14.2.5:

$$\boxed{D(\zeta)=\sum_{i=0}^{L-1}\zeta^{-i}D_i(\zeta^L)=\sum_{i=0}^{L-1}z^{-i/L}D_i(z)} \tag{14.2.15}$$

Next, we consider the 0th polyphase subfilter, $d_0(k)$, which plays a special role. It follows from Eqs. (14.2.6) and (14.2.10) that:

$$d_0(k)=d(kL)=\frac{\sin(\pi k)}{\pi k}=\delta(k), \qquad -M\le k\le M$$

and therefore, its output will be trivially equal to its input, that is, the low-rate input sample $x(n)=x_{\text{up}}(nL)$. We have from Eq. (14.2.12):

$$y_0(n)=y_{\text{up}}(nL)=\sum_{k=-M}^{M-1}d_0(k)x(n-k)=\sum_{k=-M}^{M-1}\delta(k)x(n-k)=x(n)$$

This property is preserved even for the windowed case of Eq. (14.2.7), because all windows $w(n')$ are equal to unity at their middle. This result justifies the requirement for the passband gain of the interpolator filter in Fig. 14.2.3 to be L instead of 1. If the gain were 1, we would have $y_{\text{up}}(nL)=x(n)/L$. An alternative, frequency domain justification is given in Section 14.2.3.

The causal filter implementation of Eq. (14.2.12) requires that we either delay the output or *advance the input* by M units. We choose the latter. The polyphase subfilters in Eqs. (14.2.12) can be made causal by a delay of M low-rate samples:

$$h_i(n)=d_i(n-M)=d\big((n-M)L+i\big)=d(nL+i-LM) \tag{14.2.16}$$

for $n=0,1,\dots,2M-1$. For the windowed case, we have:

$$h_i(n)=d(nL+i-LM)w(nL+i), \qquad n=0,1,\dots,2M-1 \tag{14.2.17}$$

In terms of the causal subfilters $h_i(n)$, Eq. (14.2.12) becomes

$$y_i(n)=\sum_{k=-M}^{M-1}d_i(k)x(n-k)=\sum_{k=-M}^{M-1}h_i(k+M)x(n-k)$$

or, setting $m=k+M$ and $k=m-M$,

$$\boxed{y_i(n)=\sum_{m=0}^{P}h_i(m)x(M+n-m)}\,, \qquad i=0,1,\dots,L-1 \tag{14.2.18}$$

where $P=2M-1$ denotes the *order* of each polyphase subfilter. In other words, the interpolated samples are obtained by ordinary *causal* FIR filtering of the *time-advanced* low-rate input samples. The same result can also be obtained by z-transforms. Definition (14.2.16) reads in the z-domain:

$$H_i(z)=z^{-M}D_i(z)$$

where z^{-1} represents a low-rate delay. Similarly, Eq. (14.2.12) reads

$$Y_i(z)=D_i(z)X(z)$$

Writing $D_i(z)=z^MH_i(z)$, we obtain the z-domain equivalent of Eq. (14.2.18):

$$Y_i(z)=D_i(z)X(z)=H_i(z)\big(z^MX(z)\big)$$

The sample-by-sample processing implementation of Eq. (14.2.18) requires a *common* low-rate tapped delay line which is used in sequence by *all* the subfilters $h_i(n)$ before its contents are updated. Figure 14.4.5 shows a concrete example when $L=4$ and $M=2$. The required time-advance by M samples is implemented by initially filling the delay line with the first M low-rate samples. The internal states of the tapped delay line can be defined as

$$w_m(n)=x(M+n-m), \qquad m=0,1,\dots,P$$

Then, Eq. (14.2.18) can be written in the dot-product notation of Chapter 4:

$$y_i(n)=\text{dot}\big(P,\mathbf{h}_i,\mathbf{w}(n)\big)=\sum_{m=0}^{P}h_i(m)w_m(n), \qquad i=0,1,\dots,L-1$$

After computing the outputs of the L subfilters, the internal state $\mathbf{w}$ may be updated to the next time instant by a call to the routine `delay`, which shifts the contents:

$$w_m(n+1)=w_{m-1}(n), \qquad m=1,2,\dots,P$$

This leads to the following *sample processing algorithm* for the polyphase form: Initialize the internal state vector $\mathbf{w}(n)=[w_0(n),w_1(n),\dots,w_P(n)]$ by filling it with the first M low-rate input samples, $x(0),x(1),\dots,x(M-1)$, that is, at time $n=0$ start with

$$\mathbf{w}(0)=[0,\,x_{M-1},\,x_{M-2},\,\dots,\,x_0,\,\underbrace{0,\,0,\,\dots,\,0}_{M-1\text{ zeros}}]$$

The value $w_0(0)$ need not be initialized—it is read as the current input sample. If the low-rate samples are being read sequentially from a file or an input port, then this initialization can be implemented by the following algorithm:

```
for $m = M$ down to $m = 1$ do:
    read low-rate input sample $x$
    $w_m = x$
```
(14.2.19)

Then, proceed by reading each successive low-rate sample, $x(M+n)$, $n = 0, 1, \dots,$ and processing it by the algorithm:

```
for each low-rate input sample $x$ do:
    $w_0 = x$
    for $i = 0, 1, \dots, L-1$ compute:
        $y_i = \text{dot}(P, \mathbf{h}_i, \mathbf{w})$
    delay$(P, \mathbf{w})$
```
(14.2.20)

14.2.3 Frequency Domain Characteristics

Finally, we look in more detail at the passband and stopband characteristics of the ideal lowpass interpolation filter. Let $T = 1/f_s$ and $T' = 1/f_s{}' = T/L$ be the sampling time periods with respect to the low and high rates f_s and $f_s{}'$. With reference to Fig. 14.1.2, let $x_a(t)$ be the output of the prefilter and let $X_a(f)$ be its bandlimited spectrum. The spectrum of the sampled low-rate signal $x(n)= x_a(nT)$, shown in Fig. 14.1.3, will be related to $X_a(f)$ by the Poisson summation formula:

$$X(f)= \sum_n x(n)e^{-2\pi jfnT} = \frac{1}{T}\sum_{m=-\infty}^{\infty} X_a(f - mf_s)$$

The upsampled signal $x_{\text{up}}(n')$ at the output of the L-fold rate expander of Fig. 14.2.1 has exactly the *same spectrum* as $x(n)$, as indicated in Fig. 14.1.8. Indeed, using Eqs. (14.2.1) and (14.2.2) and $T = LT'$, we have

$$\begin{aligned} X_{\text{up}}(f) &= \sum_{n'} x_{\text{up}}(n')e^{-2\pi jfn'T'} = \sum_n x_{\text{up}}(nL)e^{-2\pi jfnLT'} \\ &= \sum_n x_{\text{up}}(nL)e^{-2\pi jfnT} = \sum_n x(n)e^{-2\pi jfnT} = X(f) \end{aligned}$$

Thus,

$$\boxed{X_{\text{up}}(f)= X(f)= \frac{1}{T}\sum_{m=-\infty}^{\infty} X_a(f - mf_s)} \tag{14.2.21}$$

The same relationship can be expressed in terms of the digital frequencies as:

$$X_{\text{up}}(\omega')= X(\omega)= X(\omega' L)$$

where

$$\omega = \frac{2\pi f}{f_s} = 2\pi fT, \qquad \omega' = \frac{2\pi f}{f_s{}'} = 2\pi fT', \qquad \omega = \omega' L$$

and

$$X_{\text{up}}(\omega')= \sum_{n'} x_{\text{up}}(n')e^{-j\omega' n'}, \qquad X(\omega)= \sum_n x(n)e^{-j\omega n}$$

Similarly, using Eq. (14.2.13), their z-transforms will be related by

$$X_{\text{up}}(\zeta)= X(z)= X(\zeta^L)$$

where the slow and fast z variables are related to the corresponding digital frequencies by

$$z = e^{j\omega} = e^{2\pi jf/f_s}, \qquad \zeta = e^{j\omega'} = e^{2\pi jf/f_s{}'} = e^{2\pi jf/Lf_s}$$

If the analog signal $x_a(t)$ had actually been *resampled* at the higher rate $f_s{}'$ giving rise to the sampled signal $x'(n')= x_a(n'T')$, then the corresponding spectrum, depicted in Fig. 14.1.6, would be:

$$X'(f)= \sum_{n'} x_a(n'T')e^{-2\pi jfn'T'} = \frac{1}{T'}\sum_{m'} X_a(f - m'f_s{}') \tag{14.2.22}$$

The difference between $x'(n')$ and $x_{\text{up}}(n')$ is that $x'(n')$ contains the correct interpolated values between low-rate samples, whereas the $x_{\text{up}}(n')$ is zero there.

In the time domain, the job of an ideal interpolation filter is to reproduce the interpolated samples correctly, that is, its output is required to be $y_{\text{up}}(n')= x'(n')$ for all n'. In the frequency domain, its job is to *reshape* the low-rate sampled spectrum $X(f)$, shown in Fig. 14.1.8, into the high-rate spectrum $X'(f)$ shown in Fig. 14.1.6. Denoting the ideal interpolation filter by $D(f)$, we have for the spectrum of the output $y_{\text{up}}(n')$:

$$Y_{\text{up}}(f)= D(f)X_{\text{up}}(f)= D(f)X(f)$$

The filter output is required to be $Y_{\text{up}}(f)= X'(f)$, thus,

$$\boxed{X'(f)= D(f)X(f)} \qquad \text{(ideal interpolation)} \tag{14.2.23}$$

for all f. This condition determines the ideal passband and stopband specifications for $D(f)$. Using Eqs. (14.2.21) and (14.2.22) and separating out the central replica of $X'(f)$ and the first L replicas of $X(f)$, we rewrite the above condition as

$$\frac{1}{T'}X_a(f)+\text{replicas} = \underbrace{\frac{1}{T}D(f)X_a(f)}_{\text{passband}} + \underbrace{\frac{1}{T}D(f)\sum_{m=1}^{L-1} X_a(f - mf_s)}_{\text{stopband}} + \text{replicas}$$

Because $X_a(f)$ is bandlimited to within $[-f_s/2, f_s/2]$, it follows that the $L-1$ intermediate replicas $\sum_{m=1}^{L-1} X_a(f - mf_s)$ will be bandlimited to within $[f_s/2, Lf_s - f_s/2]$. The filter $D(f)$ is required to remove these replicas, that is,

$$D(f)= 0, \qquad \frac{f_s}{2} \le |f| \le Lf_s - \frac{f_s}{2}$$

as shown in Fig. 14.2.3. Similarly, within the low-rate Nyquist interval $-f_s/2 \le f \le f_s/2$, the filter must satisfy:

$$\frac{1}{T'}X_a(f)=\frac{1}{T}D(f)X_a(f) \qquad \Rightarrow \qquad D(f)=\frac{T}{T'}=L$$

This justifies the choice L for the passband gain. In summary, the ideal digital interpolation filter $D(f)$ is defined as follows over the high-rate Nyquist interval $[-f_s'/2, f_s'/2]$:

$$\text{(ideal interpolator)} \qquad \boxed{D(f)=\begin{cases} L, & \text{if } |f|\le \dfrac{f_s}{2} \\[1ex] 0, & \text{if } \dfrac{f_s}{2}<|f|\le \dfrac{f_s'}{2} \end{cases}} \tag{14.2.24}$$

and is periodically extended outside that interval. It is depicted in Figs. 14.1.9 and 14.2.3. Its impulse response is given by Eq. (14.2.6).

The operation of the ideal interpolation filter, expressed by Eq. (14.2.23), can also be understood in the time domain in terms of the sampling theorem. The sampled analog signal $x(n)=x_a(nT)$ can be reconstructed to analog form by the analog reconstructor:

$$x_a(t)=\sum_n x_a(nT)h(t-nT)$$

where $h(t)$ is the ideal reconstructor for the rate f_s:

$$h(t)=\frac{\sin(\pi t/T)}{\pi t/T} \tag{14.2.25}$$

Resampling at the higher rate f_s' gives the sampled signal:

$$x'(n')=x_a(n'T')=\sum_n x_a(nT)h(n'T'-nT)=\sum_n x_a(nLT')h(n'T'-nLT')$$

Denoting

$$d(k')=h(k'T')=\frac{\sin(\pi k'T'/T)}{\pi k'T'/T}=\frac{\sin(\pi k'/L)}{\pi k'/L} \tag{14.2.26}$$

and using $x_a(nLT')=x_a(nT)=x(n)$, we obtain $h(n'T'-nLT')=d(n'-nL)$ and the filtering equation:

$$x'(n')=\sum_n d(n'-nL)x(n)=\sum_{m'} d(n'-m')x_{\text{up}}(m')$$

which is recognized as the time-domain version of Eq. (14.2.23).

In summary, the effect of the ideal interpolator in the frequency domain is shown in Fig. 14.2.6. The input spectrum consists of replicas at multiples of the *input* sampling rate f_s. The filter removes all of these replicas, *except* those that are multiples of the *output* rate Lf_s. The output spectrum consists only of replicas at multiples of Lf_s. (The scaling by the gain L is not shown.)

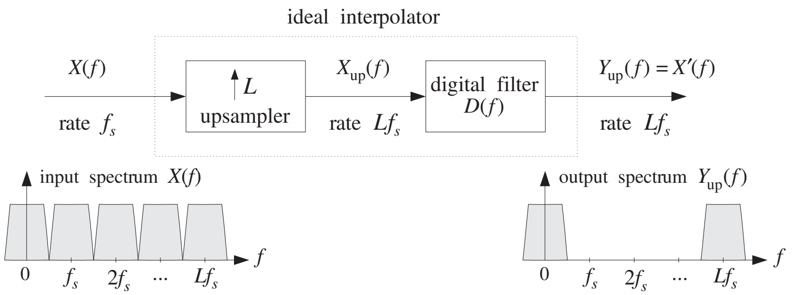


Fig. 14.2.6 Ideal interpolator in frequency domain.

14.2.4 Kaiser Window Designs

Digital interpolation filters can be designed by a variety of filter design methods, such as the Fourier series method with windowing, Parks-McClellan, or IIR designs. Here we summarize FIR designs based on the Kaiser window method.

We follow the design steps of Section 11.3, but use f_s' in place of f_s, because the interpolation filter is operating at the fast rate f_s'. For any length-N window $w(n')$, the interpolator's impulse response is computed by

$$h(n')=w(n')d(n'-LM), \qquad n'=0,1,\dots,N-1=2LM \tag{14.2.27}$$

The L length-$(2M)$ *polyphase subfilters* are defined in terms of $h(n')$ as follows. For $i=0,1,\dots,L-1$:

$$h_i(n)=h(nL+i), \qquad n=0,1,\dots,2M-1 \tag{14.2.28}$$

For a Kaiser window design, we start by specifying the desired stopband attenuation A in dB, and desired transition width Δf about the ideal cutoff frequency:

$$f_c=\frac{f_s'}{2L}=\frac{f_s}{2}$$

so that the passband and stopband frequencies are:

$$f_{\text{pass}}=f_c-\frac{1}{2}\Delta f, \quad f_{\text{stop}}=f_c+\frac{1}{2}\Delta f$$

The Kaiser window parameters are calculated by:

$$\begin{aligned}
\delta &= 10^{-A/20} \\
D &= \frac{A-7.95}{14.36} \\
\alpha &= 0.1102(A-8.7) \qquad \text{(because, typically, } A>50 \text{ dB)} \\
N-1 &\ge \frac{Df_s'}{\Delta f}=\frac{DLf_s}{\Delta f}=\frac{DL}{\Delta F}
\end{aligned} \tag{14.2.29}$$

where we used $f_s{}'$ in the formula for N and set $\Delta F = \Delta f/f_s$. Then, N must be rounded up to the smallest odd integer of the form $N = 2LM + 1$ satisfying the above inequality.

The design specifications are shown in Fig. 14.2.7. The designed length-N impulse response is given by Eq. (14.2.27), with $w(n')$ given for $n' = 0, 1, \dots, N-1$:

$$w(n') = \frac{I_0\big(\alpha\sqrt{1-(n'-LM)^2/(LM)^2}\big)}{I_0(\alpha)} = \frac{I_0\big(\alpha\sqrt{n'(2LM-n')}/LM\big)}{I_0(\alpha)}$$

The frequency response of the designed filter may be computed by:

$$H(f) = \sum_{n'=0}^{N-1} h(n')e^{-2\pi jfn'/f_s{}'} = \sum_{n'=0}^{N-1} h(n')e^{-2\pi jfn'/(Lf_s)}$$

The designed filter $h(n')$ can be implemented in its direct or polyphase forms.

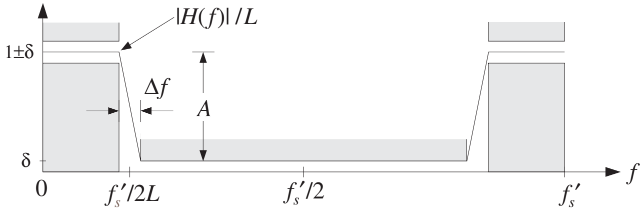


Fig. 14.2.7 Kaiser design specifications for L-fold interpolation filter.

14.2.5 *Multistage Designs*

Interpolation filters can also be implemented in a multistage form, whereby the sampling rate is gradually increased in stages until the final rate is reached. This is shown in Fig. 14.2.8. The first filter increases the sampling rate by a factor of L_0, the second by a factor of L_1, and the third by L_2, so that the overall interpolation factor is $L = L_0L_1L_2$. Such multistage realizations allow additional savings in the overall computational rate of the interpolator.

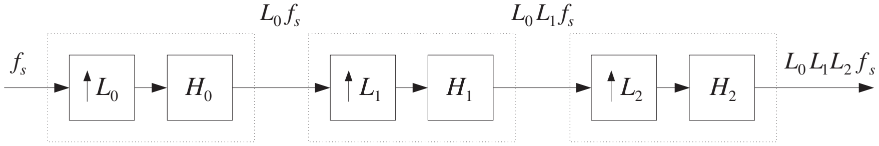


Fig. 14.2.8 Three-stage interpolation filter.

The first filter $H_0(f)$ must have the most *stringent* specifications in the sense that it has the desired transition width Δf, which is typically very narrow. The remaining stages have much *wider* transition widths and therefore smaller filter lengths.

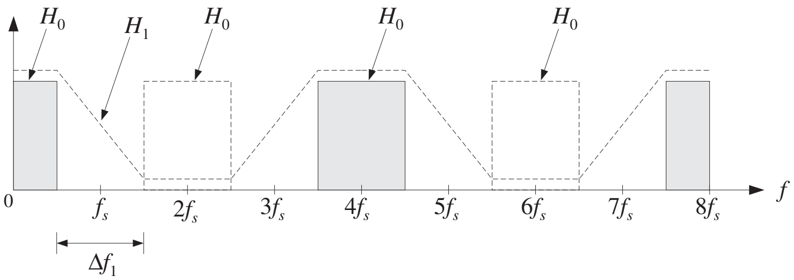


Fig. 14.2.9 Two-stage 2×2 interpolation filter.

To see this, consider the design of a 4-fold interpolator realized as the cascade of two 2-fold interpolators, $L = L_0L_1$, with $L_0 = L_1 = 2$. The desired ideal frequency characteristics of the two interpolation filters are depicted in Fig. 14.2.9.

The first interpolator H_0 is operating at the intermediate rate $f_s{}' = L_0f_s = 2f_s$ and is designed to act as an ideal lowpass filter with cutoff $f_c = f_s/2 = f_s{}'/4$. It removes all the replicas at multiples of its input rate f_s, except those that are multiples of its output rate $2f_s$.

It can be designed using a Kaiser window. For example, assuming a narrow transition width Δf about f_c, and a stopband attenuation A, we obtain from Eqs. (14.2.29):

$$N_0 - 1 = \frac{Df_s{}'}{\Delta f} = \frac{D(2f_s)}{\Delta f} = \frac{2D}{\Delta F}$$

where again $\Delta F = \Delta f/f_s$. The second interpolator H_1 is operating at the rate $2f_s{}' = 4f_s$, and must remove all replicas at multiples of its input rate $2f_s$, except those that are multiples of its output rate $4f_s$. Therefore, it has a wide transition width given by

$$\Delta f_1 = f_s{}' - f_s = 2f_s - f_s = f_s$$

Its Kaiser length will be:

$$N_1 - 1 = \frac{D(4f_s)}{\Delta f_1} = \frac{D(4f_s)}{f_s} = 4D$$

The combined effect of the two interpolation filters is to remove every three intervening replicas leaving only the replicas at multiples of $4f_s$. Because H_0 is operating at rate $2f_s$ and H_1 at rate $4f_s$, the corresponding frequency responses will be:

$$H_0(f) = \sum_{n'=0}^{N_0-1} h_0(n')e^{-2\pi jfn'/(2f_s)}, \qquad H_1(f) = \sum_{n'=0}^{N_1-1} h_1(n')e^{-2\pi jfn'/(4f_s)}$$

Assuming that both filters $h_0(n')$ and $h_1(n')$ are realized in their polyphase forms, the total computational rate of the multistage case will be, in MACs per second:

$$R_{\text{multi}} = N_0f_s + N_1(2f_s) \simeq \left(\frac{2D}{\Delta F} + 8D\right)f_s = \frac{2D}{\Delta F}(1 + 4\Delta F)f_s$$

By contrast, a single stage design would have filter length:

$$N-1=\frac{D(Lf_s)}{\Delta f}=\frac{4D}{\Delta F}$$

and polyphase computational rate:

$$R_{\text{single}}=Nf_s\simeq\frac{4D}{\Delta F}f_s$$

The relative performance of the multistage versus the single stage designs will be

$$\frac{R_{\text{multi}}}{R_{\text{single}}}=\frac{1+4\Delta F}{2}=\frac{1}{2}+2\Delta F \tag{14.2.30}$$

We note that this ratio is *independent* of the filter lengths and stopband attenuations; it depends only on the transition width. Computational savings will take place whenever:

$$\frac{1}{2}+2\Delta F<1 \qquad\Leftrightarrow\qquad \Delta F<\frac{1}{4}$$

which is usually satisfied because typical values of ΔF are of the order of 0.1. As another example, consider an 8-fold interpolator which can be realized in three different multistage ways:

$$8=2\times 2\times 2=2\times 4=4\times 2$$

The frequency characteristics of the different stages are shown in Fig. 14.2.10. The interpolator at each stage removes all replicas at multiples of *its input* rate, except those that are multiples of *its output* rate. In all three cases, the combined effect is to remove every seven intervening replicas leaving only the replicas at the multiples of $8f_s$. For the 2×2×2 case, the transition widths of the three stages are taken to be:

$$\Delta f_0=\Delta f,\quad \Delta f_1=2f_s-f_s=f_s,\quad \Delta f_2=4f_s-f_s=3f_s$$

resulting in Kaiser filter lengths:

$$N_0-1=\frac{D(2f_s)}{\Delta f_0}=\frac{2D}{\Delta F},\quad N_1-1=\frac{D(4f_s)}{\Delta f_1}=4D,\quad N_2-1=\frac{D(8f_s)}{\Delta f_2}=\frac{8D}{3}$$

and total polyphase computational rate:

$$R_{\text{multi}}=N_0f_s+N_1(2f_s)+N_2(4f_s)\simeq\left(\frac{2D}{\Delta F}+8D+\frac{32D}{3}\right)f_s$$

By contrast, the single stage design would have filter length:

$$N-1=\frac{D(8f_s)}{\Delta f}=\frac{8D}{\Delta F}$$

and polyphase computational cost:

$$R_{\text{single}}=Nf_s=\frac{8D}{\Delta F}f_s$$

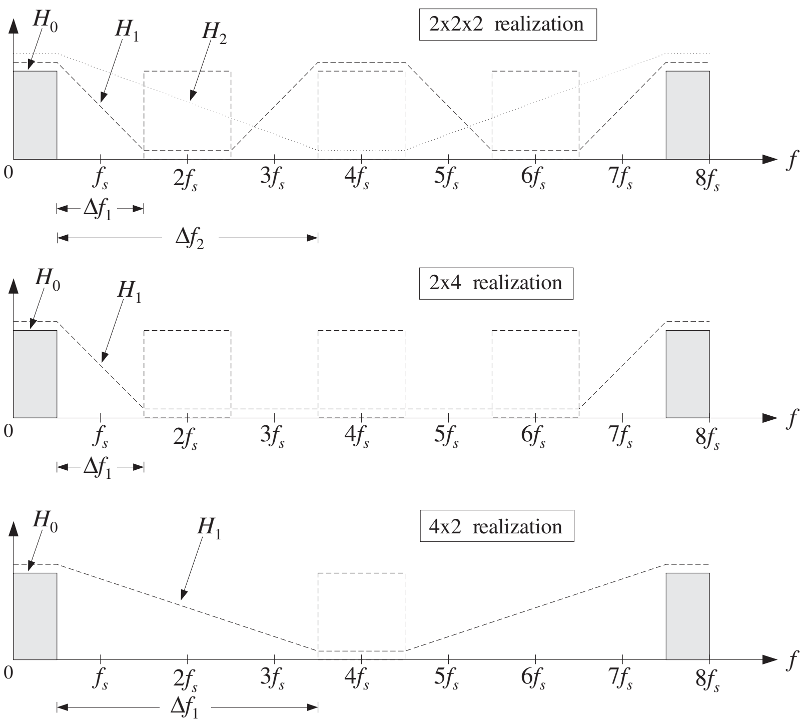


Fig. 14.2.10 Frequency characteristics of multistage 8-fold interpolators.

The relative performance of the multistage versus the single stage design is then

$$\frac{R_{\text{multi}}}{R_{\text{single}}}=\frac{1}{4}+\frac{7}{3}\Delta F$$

with savings whenever $\Delta F<9/28=0.321$. In a similar fashion, we find the multistage versus single stage polyphase computational costs of the 2×4 and 4×2 cases:

$$\frac{R_{\text{multi}}}{R_{\text{single}}}=\frac{1}{4}+2\Delta F \qquad (2\times 4\text{ case})$$

$$\frac{R_{\text{multi}}}{R_{\text{single}}}=\frac{1}{2}+\frac{4}{3}\Delta F \qquad (4\times 2\text{ case})$$

Comparing the three multistage cases, it appears that the 2×4 case is more efficient than the 2×2×2 case, which is more efficient than the 4×2 case. Indeed,

$$\frac{1}{4}+2\Delta F<\frac{1}{4}+\frac{7}{3}\Delta F<\frac{1}{2}+\frac{4}{3}\Delta F$$

the second inequality being valid for $\Delta F<1/4$. Some specific design examples will be presented later on.

The *general multistage design* procedure is as follows [347]. Assume there are K interpolation stages $H_0, H_1, \dots, H_{K-1}$ that increase the sampling rate successively by the factors $L_0, L_1, \dots, L_{K-1}$. The sampling rate at the input of the ith interpolation filter H_i will be $F_{i-1}f_s$ and at its output it will be increased by a factor L_i, that is, $F_i f_s = L_i F_{i-1} f_s$, where:

$$F_i = L_i F_{i-1} = L_0 L_1 \cdots L_i\,, \qquad i = 0, 1, \dots, K-1$$

We set $F_{-1} = 1$, so that $F_0 = L_0$. The total interpolation factor will be:

$$L = F_{K-1} = L_0 L_1 \cdots L_{K-1}$$

For a Kaiser design, we assume a given transition width Δf about the ideal cutoff frequency of the L-fold interpolator $f_c = f_s/2$ and given *stopband* attenuations in dB for each stage $A_0, A_1, \dots, A_{K-1}$. Typically, these attenuations will be the same.[†] Next, compute the Kaiser D factors, α parameters, and passband/stopband ripples δ. For $i = 0, 1, \dots, K-1$

$$\delta_i = 10^{-A_i/20}$$

$$D_i = \frac{A_i - 7.95}{14.36}$$

$$\alpha_i = 0.1102(A_i - 8.7) \qquad \text{(assuming } A_i > 50 \text{ dB)}$$

Then, compute the effective transition widths for the interpolation filters:

$$\begin{aligned} \Delta f_0 &= \Delta f \\ \Delta f_i &= F_{i-1}f_s - f_s = (F_{i-1} - 1)f_s, \quad i = 1, 2, \dots, K-1 \end{aligned} \tag{14.2.31}$$

The theoretical cutoff frequencies of these filters will be:

$$f_{ci} = \frac{F_i f_s}{2L_i} = \frac{1}{2}F_{i-1}f_s \qquad \Rightarrow \qquad \omega'_{ci} = \frac{2\pi f_{ci}}{F_i f_s} = \frac{\pi}{L_i}$$

for $i = 0, 1, \dots, K-1$. In particular, $f_{c0} = f_s/2$. For Kaiser designs, the above choices of widths imply the following passband and stopband frequencies for the filters. For $i = 0$, we have

$$f_{\text{pass},0} = f_{c0} - \frac{1}{2}\Delta f_0 = \frac{f_s}{2} - \frac{1}{2}\Delta f, \qquad f_{\text{stop},0} = f_{c0} + \frac{1}{2}\Delta f_0 = \frac{f_s}{2} + \frac{1}{2}\Delta f$$

and for $i = 1, 2, \dots, K-1$

$$f_{\text{pass},i} = f_{ci} - \frac{1}{2}\Delta f_i = \frac{f_s}{2}, \qquad f_{\text{stop},i} = f_{ci} + \frac{1}{2}\Delta f_i = F_{i-1}f_s - \frac{f_s}{2}$$

Alternatively, we can demand that all filters have exactly the same passband frequency, namely, $f_s/2 - \Delta f/2$. This changes all the stopband frequencies by shifting them down by $\Delta f/2$, that is, we can define for $i = 1, 2, \dots, K-1$

[†] If the overall multistage output is reconstructed by a DAC and analog postfilter, then the second and later stages can have less attenuation than A_0 because their suppression of the replicas will be aided by the postfilter. See Section 14.4.5.

$$f_{\text{pass},i} = \frac{f_s}{2} - \frac{1}{2}\Delta f, \qquad f_{\text{stop},i} = F_{i-1}f_s - \frac{f_s}{2} - \frac{1}{2}\Delta f$$

Note that the transition widths remain the same, but the ideal cutoff frequencies also shift down by $\Delta f/2$, that is, for $i = 1, 2, \dots, K-1$

$$f_{ci} = \frac{1}{2}(f_{\text{pass},i} + f_{\text{stop},i}) = \frac{1}{2}F_{i-1}f_s - \frac{1}{2}\Delta f \quad \Rightarrow \quad \omega'_{ci} = \frac{2\pi f_{ci}}{F_i f_s} = \frac{\pi}{L_i} - \pi\frac{\Delta F}{F_i}$$

where $\Delta F = \Delta f/f_s$. The frequency characteristics of the first filter H_0 are as shown in Fig. 14.2.7, with $L = L_0$. The specifications of the ith stage are shown in Fig. 14.2.11.

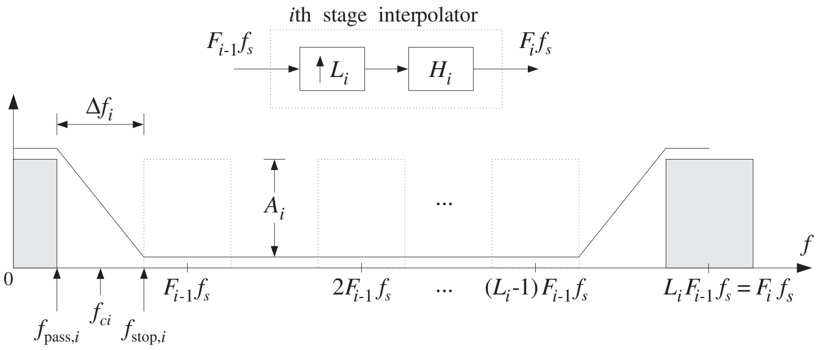


Fig. 14.2.11 H_i removes replicas at multiples of $F_{i-1}f_s$, but not at $F_i f_s$.

The filter H_i removes all replicas at multiples of its input rate $F_{i-1}f_s$ except those that are multiples of its output rate $F_i f_s$. The replicas between the first replica and that at $F_{i-1}f_s$ have already been removed by the previous stages; hence, the wide transition width Δf_i. The corresponding Kaiser filter lengths are:

$$\begin{aligned} N_0 - 1 &= \frac{D_0(F_0 f_s)}{\Delta f_0} = \frac{D_0 L_0}{\Delta F} \\ N_i - 1 &= \frac{D_i(F_i f_s)}{\Delta f_i} = \frac{D_i F_i}{F_{i-1} - 1}, \qquad i = 1, 2, \dots, K-1 \end{aligned} \tag{14.2.32}$$

where N_i must be rounded to the next smallest integer of the form:

$$N_i = 2L_i M_i + 1, \qquad i = 0, 1, \dots, K-1$$

The windowed impulse responses of the filters will be, for $i = 0, 1, \dots, K-1$

$$h_i(n') = d(L_i, n' - L_i M_i)\,w(\alpha_i, N_i, n'), \qquad n' = 0, 1, \dots, N_i - 1$$

where $d(L_i, k')$ is the ideal interpolation filter with cutoff frequency[†] $\omega'_{ci} = \pi/L_i$

[†] For the alternative case, replace ω'_{ci} by its shifted version $\omega'_{ci} = \pi/L_i - \pi\Delta F/F_i$.

$$d(L_i,k')=\frac{\sin(\omega'_{ci}k')}{\omega'_{ci}k'}=\frac{\sin(\pi k'/L_i)}{\pi k'/L_i}$$

and $w(\alpha_i,N_i,n')$ is the corresponding Kaiser window:

$$w(\alpha_i,N_i,n')=\frac{I_0\big(\alpha_i\sqrt{1-(n'-L_iM_i)^2/(L_iM_i)^2}\big)}{I_0(\alpha_i)}$$

The polyphase subfilters of each H_i are defined by

$$h_{ij}(n)=h(nL_i+j)\,,\qquad j=0,1,\dots,L_i-1,\qquad n=0,1,\dots,P_i$$

where $P_i=2M_i-1$. Finally, compute the frequency responses of the individual stages:

$$H_i(f)=\sum_{n'=0}^{N_i-1}h_i(n')e^{-2\pi jfn'/(F_if_s)}\,,\qquad i=0,1,\dots,K-1$$

and the total frequency response:

$$H_{\text{tot}}(f)=H_0(f)H_1(f)\cdots H_{K-1}(f)$$

Note that the *effective passband ripple* of the combined filter $H_{\text{tot}}(f)$ is worse than the ripples of the individual factors. This can be seen as follows. In the passband frequency range, the individual filters satisfy

$$1-\delta_i\le\left|\frac{H_i(f)}{L_i}\right|\le 1+\delta_i$$

where the DC gain L_i has been factored out. Multiplying these inequalities together, we obtain the bounds:

$$\prod_{i=0}^{K-1}(1-\delta_i)\le\left|\frac{H_0(f)H_1(f)\cdots H_{K-1}(f)}{L_0L_1\cdots L_{K-1}}\right|\le\prod_{i=0}^{K-1}(1+\delta_i)$$

For small δ_i we may use the approximation

$$\prod_{i=0}^{K-1}(1\pm\delta_i)\simeq 1\pm\sum_{i=0}^{K-1}\delta_i$$

to get the total passband ripple of the cascaded filter

$$1-\delta_{\text{tot}}\le\left|\frac{H_{\text{tot}}(f)}{L}\right|\le 1+\delta_{\text{tot}},\qquad\text{where}\quad\delta_{\text{tot}}=\sum_{i=0}^{K-1}\delta_i$$

For equal ripples $\delta_i=\delta$ and K stages, we have $\delta_{\text{tot}}=K\delta$. If so desired, this effect can be compensated by starting the design of the $H_i(f)$ using δ_i's that are smaller by a factor of K. It should be noted, however, that this may not always be necessary because the individual filters may not reach their extremum values simultaneously over the passband, and therefore the bounds $(1\pm\delta_{\text{tot}})$ may be too conservative. This is illustrated in the design examples later.

Also, in Kaiser window designs, it is the stopband attenuation A that essentially determines the passband ripples. In order to achieve reasonably high stopband attenuations, for example, in the range 70-100 dB, the corresponding passband ripples will be so small that even if they are multiplied by any K of the order of 10, they will still give rise to excellent passbands.

The total polyphase computational rate in MACs per second is obtained by adding up the computational rates of all the stages, that is,

$$R_{\text{multi}}=\sum_{i=0}^{K-1}R_i\,,\qquad\text{where}\quad R_i=N_iF_{i-1}f_s,\qquad i=0,1,\dots,K-1$$

These follow from the observation that the ith filter operates at rate F_if_s and would have computational rate $N_iF_if_s$ in its direct form. But in its polyphase form we save a factor of L_i, resulting in the rate $N_iF_if_s/L_i=N_iF_{i-1}f_s$. By comparison, the single-stage design will have Kaiser length and polyphase computational rate:

$$N-1=\frac{D(Lf_s)}{\Delta f}=\frac{DL}{\Delta F},\qquad R_{\text{single}}=Nf_s$$

where we may take $A=A_0$ for the stopband attenuation and $D=D_0$. It follows that the relative performance of multistage versus single stage designs will be:

$$\boxed{\frac{R_{\text{multi}}}{R_{\text{single}}}=\frac{N_0+\sum_{i=1}^{K-1}N_iF_{i-1}}{N}}\tag{14.2.33}$$

Assuming that all the attenuations are the same, $A_i=A$, and therefore all the D_i are the same, we may use the approximations:

$$N_0\simeq\frac{DL_0}{\Delta F}\,,\qquad N_i\simeq\frac{DF_i}{F_{i-1}-1}\,,\qquad N\simeq\frac{DL}{\Delta F}$$

to obtain the simplified expression:

$$\boxed{\frac{R_{\text{multi}}}{R_{\text{single}}}=\frac{L_0}{L}+\Delta F\sum_{i=1}^{K-1}\frac{F_iF_{i-1}}{(F_{i-1}-1)L}}\tag{14.2.34}$$

For a two-stage design with $L=L_0L_1$, $F_1=L$, and $F_0=L_0$, we have

$$\frac{R_{\text{multi}}}{R_{\text{single}}}=\frac{L_0}{L}+\frac{F_1F_0}{(F_0-1)L}\Delta F=\frac{1}{L_1}+\frac{L_0}{L_0-1}\Delta F$$

Setting $L_0=L_1=2$, or $L_0=2$, $L_1=4$, or $L_0=4$, $L_1=2$, we recover the results obtained previously for the 2×2, 2×4, and 4×2 cases. The condition that the multistage form be *more efficient* than the single-stage one is:

$$\frac{1}{L_1}+\frac{L_0}{L_0-1}\Delta F<1\qquad\Leftrightarrow\qquad\Delta F<\left(1-\frac{1}{L_0}\right)\left(1-\frac{1}{L_1}\right)$$

Given this condition, then the most efficient *ordering* of the two filters is to place first the filter with the *smaller* oversampling ratio. For example, assuming $L_0 < L_1$ then the ordering H_0H_1 is more efficient than H_1H_0 because

$$\frac{1}{L_1}+\frac{L_0}{L_0-1}\Delta F<\frac{1}{L_0}+\frac{L_1}{L_1-1}\Delta F<1$$

For a three-stage design with $F_0 = L_0$, $F_1 = L_0L_1$, and $F_2 = L_0L_1L_2 = L$, we find

$$\frac{R_{\text{multi}}}{R_{\text{single}}}=\frac{L_0}{L}+\Delta F\left[\frac{F_1F_0}{(F_0-1)L}+\frac{F_2F_1}{(F_1-1)L}\right]=\frac{1}{L_1L_2}+\Delta F\left[\frac{L_0}{L_2(L_0-1)}+\frac{L_0L_1}{L_0L_1-1}\right]$$

Setting $L_0 = L_1 = L_2 = 2$, we recover the results of the 2×2×2 case. Because the designed filter lengths N_i are slightly larger than those given by the Kaiser formulas, the correct relative computational rate should be computed using Eq. (14.2.33), whereas Eq. (14.2.34) gives only an approximation.

14.3 Linear and Hold Interpolators

We saw in Section 14.2.3 that the ideal interpolator may be thought of as the *sampled* version of the ideal *analog* reconstructor, sampled at the high rate $f_s{}'$, that is,

$$d(k')=h(k'T') \tag{14.3.1}$$

This relationship can be applied to other analog reconstructors, resulting in simpler interpolators. For any analog reconstructor $h(t)$ that reconstructs the low-rate samples by

$$y_a(t)=\sum_m h(t-mT)x(m)$$

we can obtain the interpolated samples by resampling $y_a(t)$ at the high rate $f_s{}'$:

$$y_a(n'T')=\sum_m h(n'T'-mT)x(m)$$

which can be written in the form

$$y_{\text{up}}(n')=\sum_m d(n'-mL)x(m) \tag{14.3.2}$$

where $d(k')$ is obtained from $h(t)$ via Eq. (14.3.1). The interpolation equation can be written also in terms of the *upsampled* version of $x(n)$

$$y_{\text{up}}(n')=\sum_{m'} d(n'-m')x_{\text{up}}(m')=\sum_{k'} d(k')x_{\text{up}}(n'-k') \tag{14.3.3}$$

Two of the most common interpolators are the *hold* and *linear* interpolators resulting from the sample/hold and linear analog reconstructors having impulse responses:

$$h(t)=\begin{cases}1, & \text{if } 0\le t<T\\ 0, & \text{otherwise}\end{cases}\quad\text{and}\quad h(t)=\begin{cases}1-\dfrac{|t|}{T}, & \text{if } |t|\le T\\ 0, & \text{otherwise}\end{cases}$$

They are shown in Fig. 14.3.1. Setting $t = k'T'$ in these definitions, and using the relationship $T = LT'$, we find the following discrete-time versions:

$$\text{(hold)}\qquad d(k')=\begin{cases}1, & \text{if } 0\le k'\le L-1\\ 0, & \text{otherwise}\end{cases}\;=u(k')-u(k'-L)$$

$$\text{(linear)}\qquad d(k')=\begin{cases}1-\dfrac{|k'|}{L}, & \text{if } |k'|\le L-1\\ 0, & \text{otherwise}\end{cases}$$

Note that in the linear case, the endpoints $k' = \pm L$ are not considered because $d(k')$ vanishes there. Figure 14.3.1 shows the sampled impulse responses for $L = 8$.

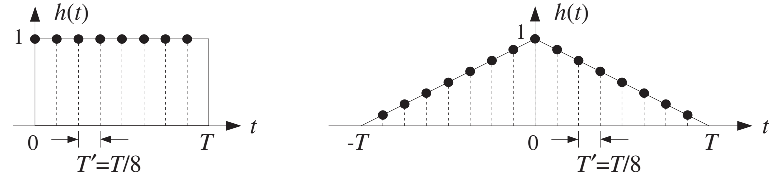


Fig. 14.3.1 Hold and linear interpolator impulse responses for $L = 8$.

The filtering operations of these interpolators are very simple. The hold interpolator holds each low-rate sample constant for L high-rate sampling times. In other words, each low-rate sample is *repeated* L times at the high rate. The linear interpolator interpolates *linearly* between a given low-rate sample and the next one.

To see this, we rewrite the filtering equation (14.3.3) in its polyphase form. As we argued for Eq. (14.2.12), we set $n' = nL + i$ and $k' = kL + j$ and use the fact that $x_{\text{up}}(nL+i-kL-j)=0$, if $i\neq j$, to get

$$y_{\text{up}}(nL+i)=\sum_k d_i(k)x(n-k) \tag{14.3.4}$$

where $d_i(k)$ are the corresponding polyphase subfilters:

$$d_i(k)=d(kL+i),\qquad i=0,1,\dots,L-1$$

The summation over k must be determined for each case. In the hold case, we have the restriction on k:

$$0\le k'\le L-1\quad\Rightarrow\quad 0\le kL+i\le L-1\quad\Rightarrow\quad 0\le k\le 1-\frac{1+i}{L}$$

Because $0 \le i \le L-1$, the only allowed value of k in that range is $k = 0$. Similarly, in the linear case, we have:

$$|k'|\le L-1\quad\Rightarrow\quad |kL+i|\le L-1\quad\Rightarrow\quad -1+\frac{1-i}{L}\le k\le 1-\frac{1+i}{L}$$

The only possible integer value in the left-hand side is $k = -1$, which is realized when $i = 1$, and the only integer value of the right-hand side is $k = 0$. Therefore, the polyphase subfilters are in the two cases:

$$
\begin{aligned}
&\text{(hold)} && d_i(k) = d_i(0)\,\delta(k)\\
&\text{(linear)} && d_i(k) = d_i(0)\,\delta(k) + d_i(-1)\,\delta(k+1)
\end{aligned}
$$

where for the hold case, we have:

$$d_i(0) = d(i) = u(i) - u(i-L) = 1 - 0 = 1$$

and for the linear case:

$$
\begin{aligned}
d_i(0) &= d(i) = 1 - \frac{|i|}{L} = 1 - \frac{i}{L}\\
d_i(-1) &= d(-L+i) = 1 - \frac{|i-L|}{L} = 1 - \frac{L-i}{L} = \frac{i}{L}
\end{aligned}
$$

Thus, the polyphase subfilters are:

$$
\begin{aligned}
&\text{(hold)} && d_i(k) = \delta(k)\\
&\text{(linear)} && d_i(k) = \Big(1 - \frac{i}{L}\Big)\delta(k) + \frac{i}{L}\delta(k+1)
\end{aligned}
\tag{14.3.5}
$$

for $i = 0, 1, \dots, L-1$. Inserting these impulse responses in Eq. (14.3.4), we find the interpolation equations in the two cases. For the hold case:

$$\boxed{y_{\text{up}}(nL+i) = x(n)}\,, \qquad i = 0, 1, \dots, L-1 \tag{14.3.6}$$

Thus, each low-rate sample is repeated L times. For the linear case we have:

$$\boxed{y_{\text{up}}(nL+i) = \Big(1 - \frac{i}{L}\Big)x(n) + \frac{i}{L}x(n+1)}\,, \qquad i = 0, 1, \dots, L-1 \tag{14.3.7}$$

They correspond to linearly weighting the two successive low-rate samples $x(n)$ and $x(n+1)$. For example, when $L = 8$ the eight interpolated samples between $x(n)$ and $x(n+1)$ are calculated by:

$$
\begin{aligned}
y_{\text{up}}(8n) &= x(n)\\
y_{\text{up}}(8n+1) &= 0.875\,x(n) + 0.125\,x(n+1)\\
y_{\text{up}}(8n+2) &= 0.750\,x(n) + 0.250\,x(n+1)\\
y_{\text{up}}(8n+3) &= 0.625\,x(n) + 0.375\,x(n+1)\\
y_{\text{up}}(8n+4) &= 0.500\,x(n) + 0.500\,x(n+1)\\
y_{\text{up}}(8n+5) &= 0.375\,x(n) + 0.625\,x(n+1)\\
y_{\text{up}}(8n+6) &= 0.250\,x(n) + 0.750\,x(n+1)\\
y_{\text{up}}(8n+7) &= 0.125\,x(n) + 0.875\,x(n+1)
\end{aligned}
$$

Figure 14.3.2 shows the interpolated signal using 8-fold hold and linear interpolators. To understand the frequency domain properties of the hold and linear interpolators and the extent to which they differ from the ideal interpolator, we compute their high-rate ζ-transforms using Eq. (14.2.15). For the hold case, taking the low-rate z-transform of $d_i(k)$ given in Eq. (14.3.5), we find

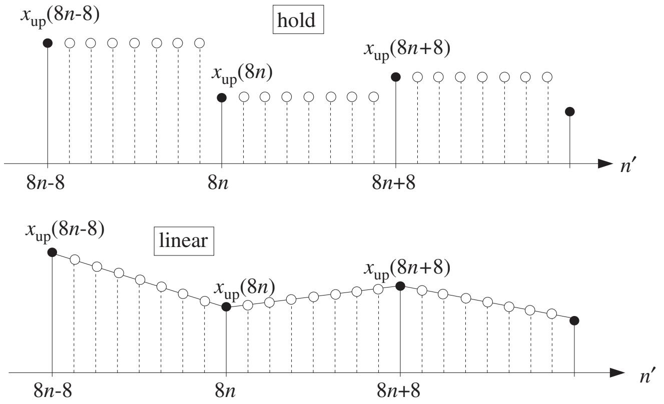


Fig. 14.3.2 8-fold hold and linear interpolators.

$$D_i(z) = 1 \qquad \Rightarrow \qquad D_i(\zeta^L) = 1$$

Then, it follows from Eq. (14.2.15)

$$D(\zeta) = \sum_{i=0}^{L-1} \zeta^{-i} D_i(\zeta^L) = \sum_{i=0}^{L-1} \zeta^{-i} = \frac{1 - \zeta^{-L}}{1 - \zeta^{-1}} \tag{14.3.8}$$

Setting $\zeta = e^{j\omega'} = e^{2\pi j f/f_s'} = e^{2\pi j f/Lf_s}$ we obtain the frequency response of the hold interpolator:

$$
\begin{aligned}
D(f) &= \frac{1 - e^{-jL\omega'}}{1 - e^{-j\omega'}} = \frac{\sin(L\omega'/2)}{\sin(\omega'/2)} e^{-j(L-1)\omega'/2}\\
&= \frac{\sin(\pi f/f_s)}{\sin(\pi f/Lf_s)} e^{-j\pi(L-1)f/Lf_s}
\end{aligned}
\tag{14.3.9}
$$

Similarly, the low-rate z-transform of $d_i(k)$ for the linear case is:

$$D_i(z) = 1 - \frac{i}{L} + \frac{i}{L}z \qquad \Rightarrow \qquad D_i(\zeta^L) = 1 + \frac{i}{L}(\zeta^L - 1)$$

From Eq. (14.2.15) we find:

$$D(\zeta) = \sum_{i=0}^{L-1} \zeta^{-i} D_i(\zeta^L) = \sum_{i=0}^{L-1} \zeta^{-i} + \frac{\zeta^L - 1}{L} \sum_{i=0}^{L-1} i\zeta^{-i}$$

and using the identity:

$$\sum_{i=0}^{L-1} i\zeta^{-i} = \frac{(1-\zeta^{-L})\zeta^{-1}}{(1-\zeta^{-1})^2} - \frac{L\zeta^{-L}}{1-\zeta^{-1}}$$

we obtain (see also Problem 5.11):

$$D(\zeta) = \frac{1}{L}\frac{(1-\zeta^{-L})(1-\zeta^{L})}{(1-\zeta^{-1})(1-\zeta)} = \frac{1}{L}\left(\frac{1-\zeta^{-L}}{1-\zeta^{-1}}\right)^2 \zeta^{L-1}$$

which leads to the frequency response:

$$D(f) = \frac{1}{L}\left|\frac{\sin(L\omega'/2)}{\sin(\omega'/2)}\right|^2 = \frac{1}{L}\left|\frac{\sin(\pi f/f_s)}{\sin(\pi f/Lf_s)}\right|^2 \tag{14.3.10}$$

Both responses (14.3.9) and (14.3.10) are periodic in f with period $f_s{}' = Lf_s$ and vanish at all multiples of f_s which are *not* multiples of Lf_s. Therefore, they partially remove the spectral replicas that are between multiples of $f_s{}'$. They are shown in Fig. 14.3.3 for the case $L = 8$, together with the ideal response.

Because of their simple structure, linear and hold interpolators are used in *multistage* implementations of interpolators, especially in the latter stages that have higher sampling rates. Some example designs are discussed in Section 14.4.5.

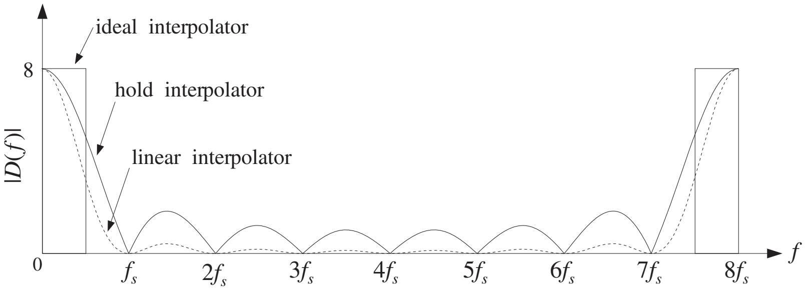


Fig. 14.3.3 Hold and linear interpolator frequency responses for $L = 8$.

14.4 Design Examples

14.4.1 4-fold Interpolators

Consider the case of a 4-fold interpolator having $L = 4$ and polyphase filter length $2M = 4$ or $M = 2$. This corresponds to a filter length $N = 2LM + 1 = 17$. The ideal impulse response will be:

$$d(k') = \frac{\sin(\pi k'/4)}{\pi k'/4}, \qquad -8 \le k' \le 8$$

or, numerically,

$$\begin{aligned}\mathbf{h} = \mathbf{d} = [&0, -0.13, -0.21, -0.18, 0, 0.30, 0.64, 0.90, 1, 0.90, 0.64,\\ &0.30, 0, -0.18, -0.21, -0.13, 0]\end{aligned} \tag{14.4.1}$$

where $\mathbf{h}$ is the causal version, with time origin shifted to the *left* of the vector, and $\mathbf{d}$ is the symmetric one with time origin at the *middle* of the vector. This truncated ideal impulse response is shown in Fig. 14.4.1. The four polyphase subfilters are defined by Eq. (14.2.10), that is, for $i = 0, 1, 2, 3$,

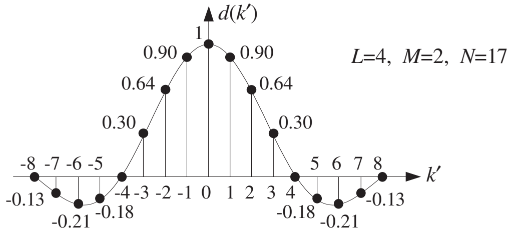


Fig. 14.4.1 Length-17 symmetric impulse response of 4-fold FIR interpolator.

$$d_i(k) = d(4k+i), \qquad -2 \le k \le 1$$

They are extracted from $\mathbf{h}$ by taking every fourth entry, starting with the ith entry:

$$\begin{aligned}\mathbf{h}_0 &= \mathbf{d}_0 = [0,\ 0,\ 1,\ 0]\\ \mathbf{h}_1 &= \mathbf{d}_1 = [-0.13,\ 0.30,\ 0.90,\ -0.18]\\ \mathbf{h}_2 &= \mathbf{d}_2 = [-0.21,\ 0.64,\ 0.64,\ -0.21]\\ \mathbf{h}_3 &= \mathbf{d}_3 = [-0.18,\ 0.90,\ 0.30,\ -0.13]\end{aligned} \tag{14.4.2}$$

The interpolated samples between $x(n) = x_{\text{up}}(4n)$ and $x(n+1) = x_{\text{up}}(4n+4)$ are calculated from Eqs. (14.2.18). All four subfilters act on the time-advanced low-rate input samples $\{x(n+2),\ x(n+1),\ x(n),\ x(n-1)\}$, or, $\{x_{\text{up}}(4n+8),\ x_{\text{up}}(4n+4),\ x_{\text{up}}(4n),\ x_{\text{up}}(4n-4)\}$. Equations (14.2.12) can be cast in a compact matrix form:

$$\begin{bmatrix} y_{\text{up}}(4n) \\ y_{\text{up}}(4n+1) \\ y_{\text{up}}(4n+2) \\ y_{\text{up}}(4n+3) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ -0.13 & 0.30 & 0.90 & -0.18 \\ -0.21 & 0.64 & 0.64 & -0.21 \\ -0.18 & 0.90 & 0.30 & -0.13 \end{bmatrix} \begin{bmatrix} x_{\text{up}}(4n+8) \\ x_{\text{up}}(4n+4) \\ x_{\text{up}}(4n) \\ x_{\text{up}}(4n-4) \end{bmatrix} \tag{14.4.3}$$

These results can be understood more intuitively using the LTI form of convolution, that is, superimposing the full length-17 symmetric impulse response $\mathbf{d}$ at the four contributing low-rate samples and summing up their contributions at the four desired time instants, that is, at $n' = 4n + i$, $i = 0, 1, 2, 3$. This is illustrated in Fig. 14.4.2.

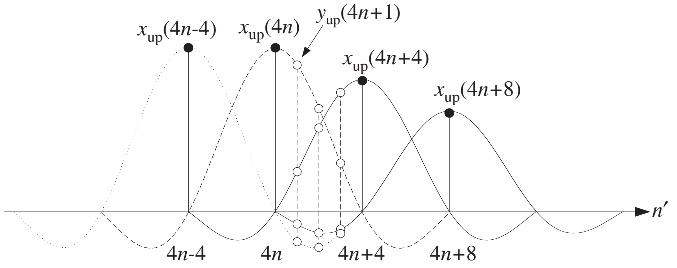


Fig. 14.4.2 LTI form of convolution by superposition of FIR impulse responses.

For example, referring to the impulse response values indicated on Fig. 14.4.1, we find that at time instant $4n+1$, the input sample $x_{\text{up}}(4n+8)$ will contribute an amount $-0.13x_{\text{up}}(4n+8)$, the sample $x_{\text{up}}(4n+4)$ will contribute an amount $0.30x_{\text{up}}(4n+4)$, the sample $x_{\text{up}}(4n)$ will contribute $0.90x_{\text{up}}(4n)$, and the sample $x_{\text{up}}(4n-4)$ an amount $-0.18x_{\text{up}}(4n-4)$. The interpolated value is built up from these four contributions:

$$y_{\text{up}}(4n+1)=-0.13x_{\text{up}}(4n+8)+0.30x_{\text{up}}(4n+4)+0.90x_{\text{up}}(4n)-0.18x_{\text{up}}(4n-4)$$

Similarly, it should be evident from Fig. 14.4.2 that $y_{\text{up}}(4n)=x_{\text{up}}(4n)$, with the contributions of the other low-rate inputs vanishing at time instant $4n$. We may also use the flip-and-slide form of convolution, in which the impulse response $d(k')$ is flipped, delayed, and positioned at the sampling instant n' to be computed. For example, at $n'=4n+1$, we have:

$$y_{\text{up}}(4n+1)=\sum_{k'}d(4n+1-k')x_{\text{up}}(k')$$

Figure 14.4.3 shows this operation. Because of symmetry, the flipped impulse response is the same as that in Fig. 14.4.1. It is then translated to $n'=4n+1$ and the above linear combination is performed.

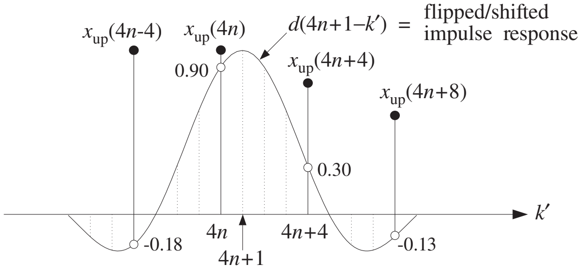


Fig. 14.4.3 Flip-and-slide form of convolution.

The only contributions come from the low-rate samples that fall within the finite extent of the impulse response. Thus, only the terms $k'=4n-4, 4n, 4n+4, 4n+8$ contribute, and each is weighted by the appropriate impulse response values that are read off from the figure, that is, $\{-0.18, 0.90, 0.30, -0.13\}$, so that again:

$$y_{\text{up}}(4n+1)=-0.18x_{\text{up}}(4n-4)+0.90x_{\text{up}}(4n)+0.30x_{\text{up}}(4n+4)-0.13x_{\text{up}}(4n+8)$$

The Hamming windowed version of the filter is obtained by multiplying the full length-17 filter response $\mathbf{h}$ by a length-17 Hamming window. The resulting impulse response becomes:

$$\begin{aligned}\mathbf{h}=\;&[0,-0.02,-0.05,-0.07,0,0.22,0.55,0.87,1,0.87,0.55,0.22,0,\\&-0.07,-0.05,-0.02,0]\end{aligned}$$

The polyphase interpolation equations become in this case:

$$\begin{bmatrix} {}_{\text{up}}(4n) \\ {}_{\text{up}}(4n+1) \\ {}_{\text{up}}(4n+2) \\ {}_{\text{up}}(4n+3) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ -0.02 & 0.22 & 0.87 & -0.07 \\ -0.05 & 0.55 & 0.55 & -0.05 \\ -0.07 & 0.87 & 0.22 & -0.02 \end{bmatrix} \begin{bmatrix} x_{\text{up}}(4n+8) \\ x_{\text{up}}(4n+4) \\ x_{\text{up}}(4n) \\ x_{\text{up}}(4n-4) \end{bmatrix}$$

The graphs in Fig. 14.4.4 compare the *magnitude responses* of the rectangularly and Hamming windowed interpolation filters. A *block diagram* realization of the polyphase form for this example is shown in Fig. 14.4.5. It is based on Eqs. (14.2.14) and (14.2.15), that is,

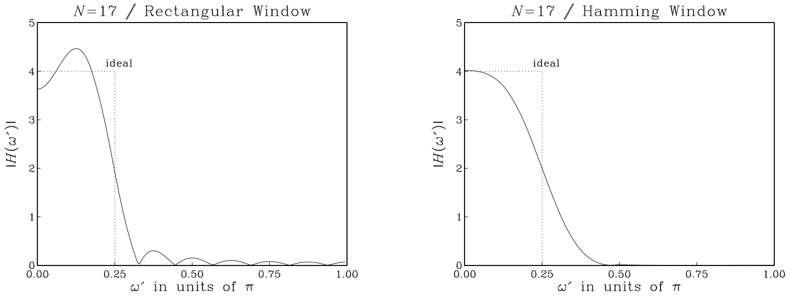


Fig. 14.4.4 Magnitude response $|H(\omega')|$ versus $\omega'=2\pi f/f_s{}'$.

$$\begin{aligned}H(\zeta)&=H_0(\zeta^4)+\zeta^{-1}H_1(\zeta^4)+\zeta^{-2}H_2(\zeta^4)+\zeta^{-3}H_3(\zeta^4)\\&=H_0(z)+z^{-1/4}H_1(z)+z^{-2/4}H_2(z)+z^{-3/4}H_3(z)\end{aligned}$$

with all the subfilters using the *same tapped delay line* holding the incoming low-rate samples. The block diagram is equivalent to the commutator model of Fig. 14.2.5. The polyphase subfilters are defined by:

$$\mathbf{h}_i=[h_{i0},\,h_{i1},\,h_{i2},\,h_{i3}],\qquad H_i(z)=h_{i0}+h_{i1}z^{-1}+h_{i2}z^{-2}+h_{i3}z^{-3}$$

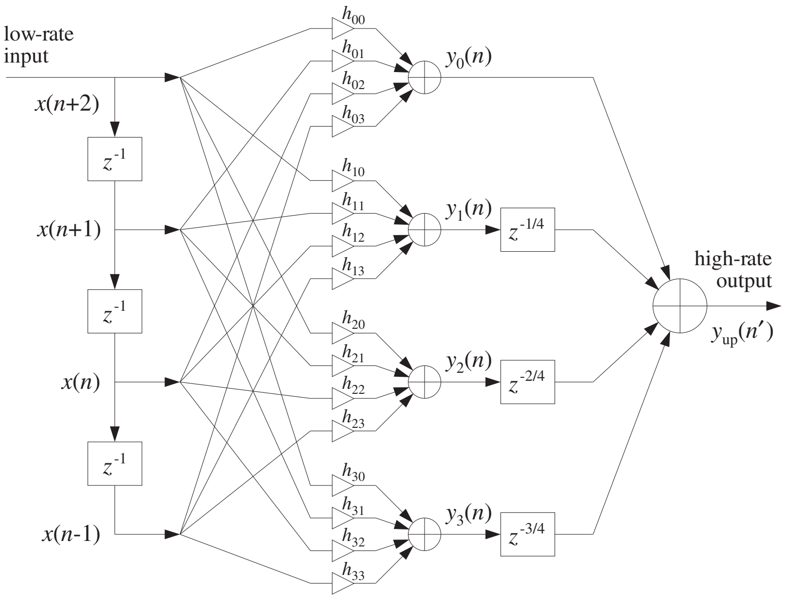


Fig. 14.4.5 Polyphase realization of 4-fold interpolator.

for $i = 0, 1, 2, 3$, where $\mathbf{h}_i$ are given by Eq. (14.4.2).

The possibility of a parallel multiprocessor implementation is evident from this diagram. The four outputs of the filters $H_i(z)$ are produced simultaneously in a parallel implementation, but they are not sent to the overall output simultaneously. During each low-rate sampling period T, the sample $y_0(n)$ is sent out first, then $T/4$ seconds later (represented by the delay $z^{-1/4}$) the second computed interpolated sample $y_1(n)$ is sent out, another $T/4$ seconds later the third sample $y_2(n)$ is sent out, and $T/4$ seconds after that, the fourth interpolated sample $y_3(n)$ is sent out.

As a concrete filtering example, consider the following low-rate input signal $x(n)$ consisting of 25 DC samples, and depicted in Fig. 14.4.6 with respect to the fast time scale:

$$x(n)= \{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1\}$$

The interpolated values between these low-rate samples are shown in Fig. 14.4.7 for the cases of the rectangularly and Hamming windowed interpolating filters. They were computed by the polyphase sample processing algorithm of Eq. (14.2.20). The input-on and input-off transients are evident.

As another example, consider the case of $L = 4$ and $M = 12$, that is, interpolation filter length $N = 2LM + 1 = 97$. This is a more realistic length for typical 4-fold oversampling digital filters used in CD players. The corresponding rectangularly and Hamming windowed magnitude responses are shown in Fig. 14.4.8. The interpolated output signals from these two filters are shown in Fig. 14.4.9 for the same low-rate input signal $x(n)$. Note the longer input-on and input-off transients.

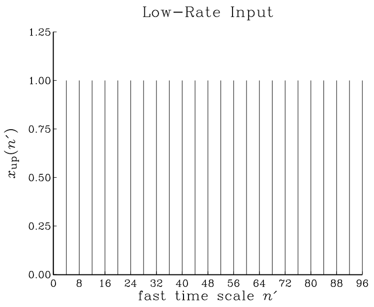


Fig. 14.4.6 Low-rate input samples $x_{up}(n')$.

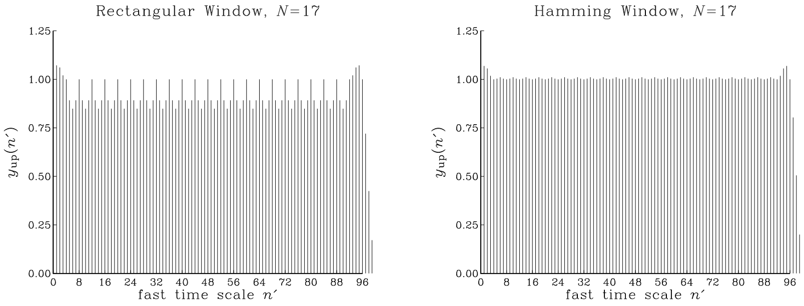


Fig. 14.4.7 High-rate interpolated output samples $y_{up}(n')$.

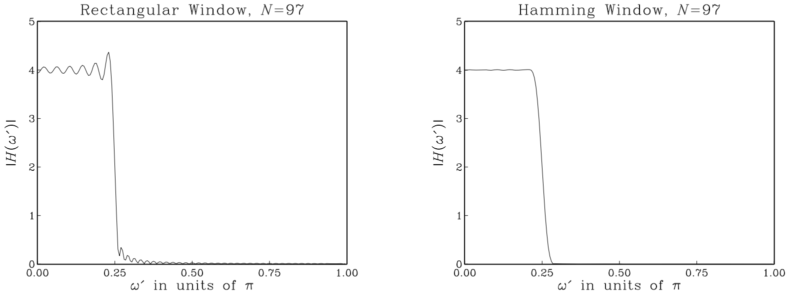


Fig. 14.4.8 Magnitude responses of length-97 interpolation filters.

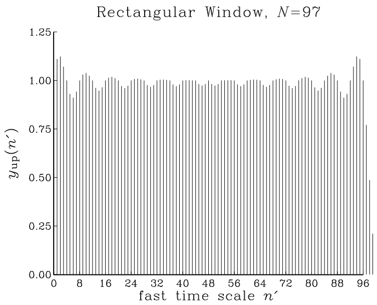


Fig. 14.4.9 High-rate interpolated outputs $y_{\text{up}}(n')$.

14.4.2 Multistage 4-fold Interpolators

Here, we follow the discussion of Section 14.2.5 and design multistage and single stage digital interpolation filters using the Kaiser window method. Such filters may be used as oversampling digital filters in CD players.[†]

We take $L = 4$ for the oversampling ratio, and assume a nominal digital audio sampling rate of $f_s = 40$ kHz, so that the fast rate will be $f_s{}' = Lf_s = 4 \times 40 = 160$ kHz. We take the transition width to be $\Delta f = 5$ kHz about the ideal cutoff frequency $f_c = f_s{}'/(2L) = f_s/2 = 20$ kHz. Therefore, the passband and stopband frequencies of the filter will be

$$f_{\text{pass}} = 20 - \frac{5}{2} = 17.5 \text{ kHz}, \quad f_{\text{stop}} = 20 + \frac{5}{2} = 22.5 \text{ kHz}$$

and the normalized transition width

$$\Delta F = \frac{\Delta f}{f_s} = \frac{5}{40} = 0.125$$

The stopband attenuation is taken to be $A = 80$ dB, which gives rise to a passband/stopband ripple $\delta = 10^{-4}$, passband attenuation $A_{\text{pass}} = 0.0017$ dB, and the following values for the Kaiser window parameters D and α:

$$D = \frac{A - 7.95}{14.36} = 5.017, \qquad \alpha = 0.1102(A - 8.7) = 7.857$$

For a 2×2 multistage design, shown in Fig. 14.4.10, we may use the general design equations (14.2.31) and (14.2.32) to find the Kaiser lengths of the two filters H_0 and H_1:

$$N_0 - 1 = \frac{DL_0}{\Delta F} = \frac{2D}{\Delta F} = 80.27, \qquad N_1 - 1 = \frac{DF_1}{F_0 - 1} = 4D = 20.07$$

which get rounded up to the values:

[†] See Ref. [356] for actual DSP chips with comparable design characteristics.

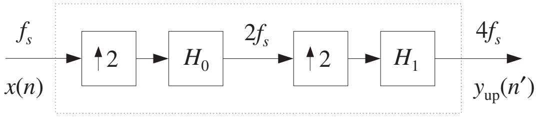


Fig. 14.4.10 2×2 = 4-fold oversampling filter.

$$N_0 = 85 = 2L_0M_0 + 1 = 4M_0 + 1 \quad \Rightarrow \quad M_0 = 21$$

$$N_1 = 25 = 2L_1M_1 + 1 = 4M_1 + 1 \quad \Rightarrow \quad M_1 = 6$$

The magnitude response of the filter $H_0(f)$ in dB and a typical low-rate sinusoidal input to be interpolated are shown in Fig. 14.4.11. The 2-fold interpolated output of $H_0(f)$ is shown in Fig. 14.4.12. It serves as the input to the next filter $H_1(f)$ whose magnitude response is also shown in Fig. 14.4.12.

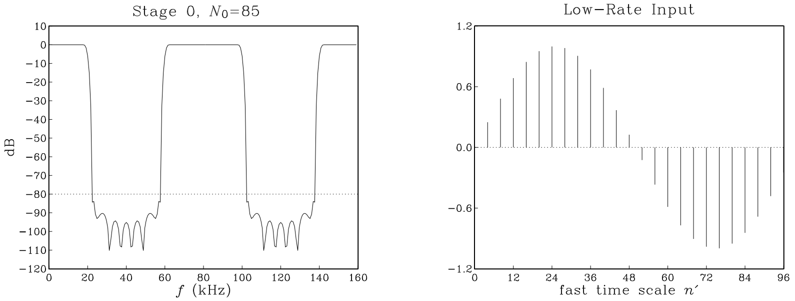


Fig. 14.4.11 Filter $H_0(f)$ and its low-rate input signal.

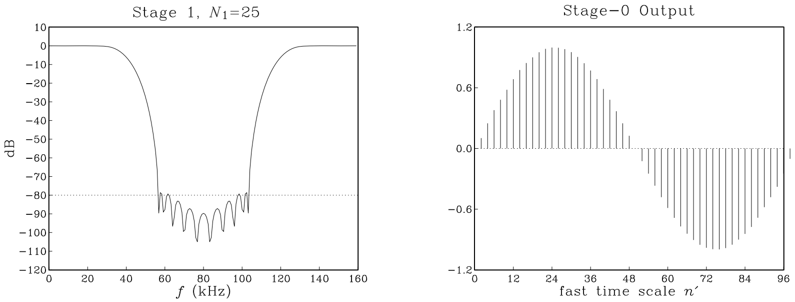


Fig. 14.4.12 Filter $H_1(f)$ and its input which is the output of $H_0(f)$.

The 2-fold interpolated output of $H_1(f)$ will be the final 4-fold interpolated output. It is shown in Fig. 14.4.13 together with the superimposed plots of the filters $H_0(f)$ and $H_1(f)$. In these figures, the frequency responses have been normalized by their DC values, that is, $H_0(f)/L_0$, $H_1(f)/L_1$.

Finally, we compare the multistage design to an equivalent single stage Kaiser design. In this case the Kaiser filter length will be

$$N - 1 = \frac{4D}{\Delta F} = 160.56$$

which is rounded up to the value

$$N = 169 = 2LM + 1 = 8M + 1 \qquad \Rightarrow \qquad M = 21$$

Its magnitude response $H(f)$ is shown in Fig. 14.4.14 together with the magnitude response of the combined multistage filter $H_{\rm tot}(f) = H_0(f)H_1(f)$. Again, we have normalized them to their DC values, namely, $H(f)/L$, and $H_{\rm tot}(f)/L$.

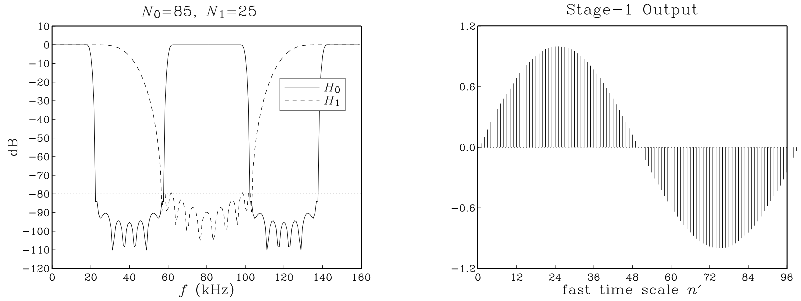


Fig. 14.4.13 Filters $H_0(f)$, $H_1(f)$ and the overall interpolated output.

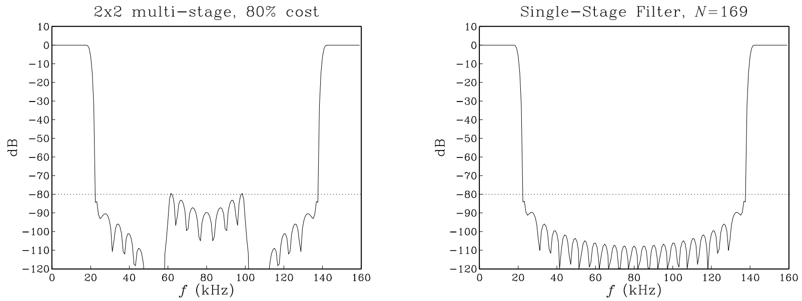


Fig. 14.4.14 Multistage filter $H_0(f)H_1(f)$ and single-stage filter $H(f)$.

The multistage realization requires only 80 percent of the computational rate of the single-stage design. Indeed, the relative computational rate of the multistage versus the single stage designs is given according to Eq. (14.2.33) by:

$$\frac{R_{\rm multi}}{R_{\rm single}} = \frac{N_0 + N_1F_0}{N} = \frac{85 + 25\cdot 2}{169} = 0.80$$

which compares well with the approximate result of Eq. (14.2.30).

$$\frac{R_{\rm multi}}{R_{\rm single}} = \frac{1}{2} + 2\Delta F = 0.5 + 2\cdot 0.125 = 0.75$$

Finally, as we saw earlier, the passband of the total filter $H_{\rm tot}(f) = H_0(f)H_1(f)$ tends to be worse than the passbands of the individual factors. Let δ be the common passband ripple, as calculated from Eq. (14.2.29). Then, the two individual filters will satisfy within their passbands:

$$1 - \delta \le \left|\frac{H_0(f)}{L_0}\right| \le 1 + \delta\,, \qquad 1 - \delta \le \left|\frac{H_1(f)}{L_1}\right| \le 1 + \delta$$

Multiplying the two inequalities, we find for the total filter

$$(1-\delta)^2 \le \left|\frac{H_0(f)H_1(f)}{L_0L_1}\right| \le (1+\delta)^2$$

or, approximately if δ is small,

$$1 - 2\delta \le \left|\frac{H_{\rm tot}(f)}{L}\right| \le 1 + 2\delta$$

Thus, the passband ripple is effectively doubled, $\delta_{\rm tot} = 2\delta$. Taking logs of both sides, we obtain the following bounds for the passband attenuations in dB:

$$-8.7\delta \le 20\log_{10}\left|\frac{H_0(f)}{L_0}\right| \le 8.7\delta, \qquad -8.7\delta \le 20\log_{10}\left|\frac{H_1(f)}{L_1}\right| \le 8.7\delta$$

$$-8.7(2\delta) \le 20\log_{10}\left|\frac{H_{\rm tot}(f)}{L}\right| \le 8.7(2\delta)$$

where we used the small-δ approximation:

$$20\log_{10}(1+\delta) \simeq 8.7\delta$$

Figure 14.4.15 shows a magnified plot of the passband region of the individual and total filters for the above designs, with the passband bounds placed on the figure. It is evident that the actual passband ripple of $H_{\rm tot}(f)$ is less than the worst-case ripple $\delta_{\rm tot} = 2\delta$.

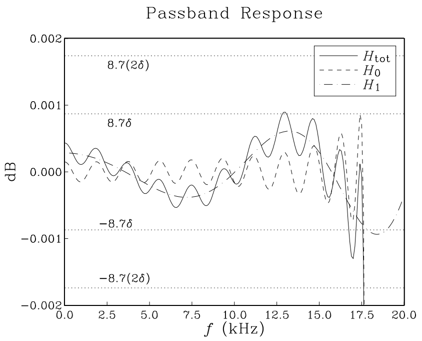


Fig. 14.4.15 Magnified passband of 2×2 interpolator.

14.4.3 DAC Equalization

In an oversampling DSP system, the interpolator output samples are reconstructed by a staircase D/A converter operating at the high rate $f_s{}' = Lf_s$. Its frequency response (normalized to unity gain at DC) is

$$H_{\rm dac}(f)=\frac{\sin(\pi f/f_s{}')}{\pi f/f_s{}'}e^{-j\pi f/f_s{}'}$$

It causes some attenuation within the Nyquist interval, with maximum of about 4 dB at the Nyquist frequency $f_s{}'/2$. For an L-fold interpolation filter which has cutoff at $f_c = f_s/2 = f_s{}'/2L$, the maximum attenuation within the filter's passband will be:

$$|H_{\rm dac}(f_c)| = \left|\frac{\sin(\pi f_c/f_s{}')}{\pi f_c/f_s{}'}\right| = \frac{\sin(\pi/2L)}{\pi/2L} \tag{14.4.4}$$

For large values of the oversampling ratio L, this attenuation is insignificant, approaching 0 dB. Thus, one of the benefits of oversampling is that the aperture effect of the DAC can be neglected.

However, for smaller values of L (for example, $L \le 8$) it may be desirable to compensate this attenuation by designing the interpolation filter to have an *inverse* shape to the $\sin x/x$ DAC response over the relevant passband range. The desired *equalized* ideal interpolation filter can then be defined by the following equation, replacing Eq. (14.2.24):

$$D(f)=\begin{cases} LD_{\rm eq}(f), & \text{if } |f|\le\dfrac{f_s}{2}\\ 0, & \text{if } \dfrac{f_s}{2}<|f|\le\dfrac{f_s{}'}{2}\end{cases} \tag{14.4.5}$$

where $D_{\rm eq}(f)$ is essentially the inverse response $1/H_{\rm dac}(f)$ with the phase removed in order to keep $D(f)$ real and even in f:

$$D_{\rm eq}(f)=\frac{\pi f/f_s{}'}{\sin(\pi f/f_s{}')},\qquad |f|\le\frac{f_s}{2} \tag{14.4.6}$$

In units of the high-rate digital frequency $\omega' = 2\pi f/f_s{}'$, Eq. (14.4.5) becomes:

$$D(\omega')=\begin{cases} L\dfrac{\omega'/2}{\sin(\omega'/2)}, & \text{if } |\omega'|\le\dfrac{\pi}{L}\\ 0, & \text{if } \dfrac{\pi}{L}<|\omega'|\le\pi\end{cases} \tag{14.4.7}$$

Such a filter can be designed by the *frequency sampling* design method of Section 11.4. If the filter order is known, say $N = 2LM + 1$, then we can compute the desired filter weights by the inverse N-point DFT:

$$\tilde d(k')=\frac{1}{N}\sum_{i=-LM}^{LM}D(\omega_i')e^{j\omega_i'k'},\qquad -LM\le k'\le LM \tag{14.4.8}$$

where ω_i' are the N DFT frequencies spanning the symmetric Nyquist interval $[-\pi,\pi]$:

$$\boxed{\omega_i'=\frac{2\pi i}{N}},\qquad -LM\le i\le LM$$

The designed causal windowed filter will be

$$h(n')=\tilde d(n'-LM)w(n'),\qquad 0\le n'\le N-1 \tag{14.4.9}$$

In the Hamming window case, we must assume a desired value for N. In the Kaiser case, we may start with a desired stopband attenuation A and transition width Δf, and then determine the filter length N and the window parameter α. Because the filter is sloping upwards in the passband, to achieve a true attenuation A in the stopband, we may have to carry out the design with a slightly larger value of A. This is illustrated in the examples below.

Note also that because $\tilde d(k')$ and $D(\omega')$ are real-valued, we may replace the right-hand side of Eq. (14.4.8) by its real part and write it in the cosine form:

$$\tilde d(k')=\frac{1}{N}\sum_{i=-LM}^{LM}D(\omega_i')\cos(\omega_i'k'),\qquad -LM\le k'\le LM$$

and because $D(\omega')$ is even in ω'

$$\tilde d(k')=\frac{1}{N}\left[D(\omega_0')+2\sum_{i=1}^{LM}D(\omega_i')\cos(\omega_i'k')\right],\qquad -LM\le k'\le LM$$

where $D(\omega_0')=D(0)=L$. This expression can be simplified even further by noting that $D(\omega_i')$ is non-zero only for

$$0<\omega_i'<\frac{\pi}{L}\quad\Rightarrow\quad 0<\frac{2\pi i}{N}<\frac{\pi}{L}\quad\Rightarrow\quad 0<i<\frac{N}{2L}=\frac{2LM+1}{2L}=M+\frac{1}{2L}$$

Thus, the summation can be restricted over $1\le i\le M$, giving

$$\tilde d(k')=\frac{L}{N}\left[1+2\sum_{i=1}^{M}\frac{\omega_i'/2}{\sin(\omega_i'/2)}\cos(\omega_i'k')\right],\qquad -LM\le k'\le LM$$

As a first example, consider a 2-times oversampling filter for a CD player. Assume a nominal audio rate of $f_s = 40$ kHz, transition width $\Delta f = 5$ kHz, and stopband attenuation $A = 80$ dB. The normalized width is $\Delta F = \Delta f/f_s = 0.125$. The Kaiser D parameter and filter length N will be

$$D = \frac{A - 7.95}{14.36} = 5.017\,, \qquad N - 1 \geq= \frac{DL}{\Delta F} = 80.3$$

which rounds up to $N = 85$. Fig. 14.4.16 shows the designed filter together with the inverse DAC response $1/|H_{\text{dac}}(f)|$ over the high-rate Nyquist interval. It also shows the passband in a magnified scale. Notice how the inverse DAC response reaches 4 dB at the Nyquist frequency of $f_s{}'/2 = 40$ kHz.

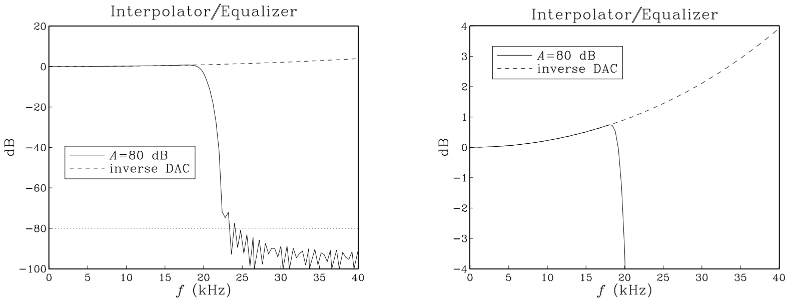


Fig. 14.4.16 2-fold interpolator/equalizer designed with $A = 80$ dB.

The actual stopband attenuation is somewhat less than the prescribed 80 dB, namely, about 72 dB at $f = f_s/2 + \Delta f/2 = 22.5$ kHz. Thus, we may wish to redesign the filter starting out with a somewhat larger attenuation. For example, assuming $A = 90$ dB, we obtain filter length $N = 93$. See [356] for a similar design. The redesigned filter is shown in Fig. 14.4.17. It achieves 80 dB attenuation at 22.5 kHz.

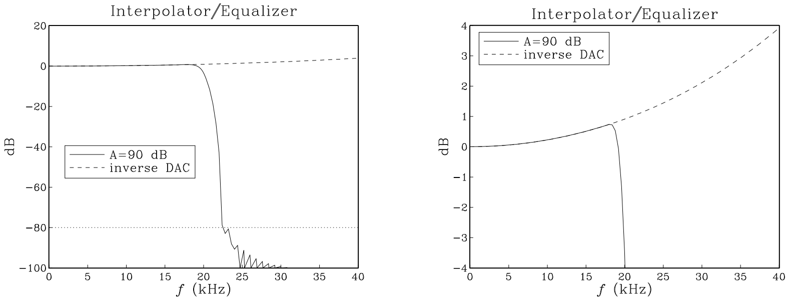


Fig. 14.4.17 2-fold interpolator/equalizer redesigned with $A = 90$ dB.

As another example, we design an equalized 4-times oversampling filter for digital audio, assuming 40 kHz audio rate, 5 kHz transition width, and a stopband attenuation of 60 dB. The Kaiser parameters are:

$$D = \frac{A - 7.95}{14.36} = 3.625\,, \qquad N - 1 \geq= \frac{DL}{\Delta F} = 115.98 \quad \Rightarrow \quad N = 121$$

The designed filter and its passband are shown in Fig. 14.4.18.

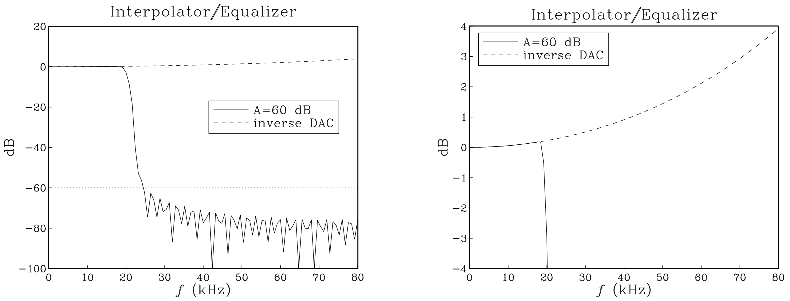


Fig. 14.4.18 4-fold interpolator/equalizer filter and its magnified passband.

14.4.4 Postfilter Design and Equalization

In addition to compensating for the attenuation of the DAC, one may wish to compensate for other effects. For example, the staircase output of the DAC will be fed into an analog anti-image postfilter which introduces its own slight attenuation within the desired passband. This attenuation can be equalized digitally by the interpolation filter. Figure 14.4.19 shows this arrangement.

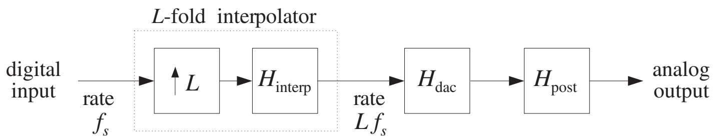


Fig. 14.4.19 Interpolation filter equalizes DAC and postfilter responses.

As we saw in Section 14.1, because of oversampling, the postfilter will have a wide transition region resulting in low filter order, such as 2 or 3. The postfilter must provide enough attenuation in its stopband to remove the spectral images of the interpolated signal at multiples of $f_s{}'$, as shown in Fig. 14.4.20. Its passband extends up to the low-rate Nyquist frequency[†] and its stopband begins at the left edge of the first spectral image, that is,

[†] If the interpolator is not ideal but has transition width Δf about $f_s/2$, we may use the more accurate expressions $f_{\text{pass}} = f_s/2 - \Delta f/2$ and $f_{\text{stop}} = Lf_s - f_s/2 - \Delta f/2$.

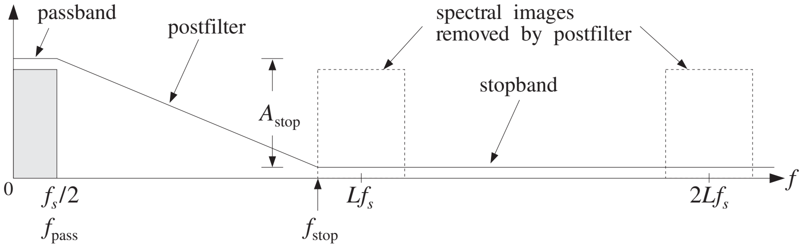


Fig. 14.4.20 Analog anti-image postfilter specifications.

$$\boxed{f_{\rm pass}=\frac{f_s}{2}\,,\qquad f_{\rm stop}=f_s{}'-f_{\rm pass}=Lf_s-\frac{f_s}{2}}$$

At $f_{\rm stop}$, the DAC already provides a certain amount of attenuation given by:

$$|H_{\rm dac}(f_{\rm stop})|=\left|\frac{\sin(\pi f_{\rm stop}/f_s{}')}{\pi f_{\rm stop}/f_s{}'}\right|=\frac{\sin(\pi-\pi/2L)}{\pi-\pi/2L}=\frac{\sin(\pi/2L)}{\pi-\pi/2L}$$

which, for large L, becomes approximately:

$$|H_{\rm dac}(f_{\rm stop})|=\frac{\sin(\pi/2L)}{\pi-\pi/2L}\simeq\frac{1}{2L}$$

or, in dB

$$A_{\rm dac}\simeq 20\log_{10}(2L) \tag{14.4.10}$$

The analog postfilter must supply an additional amount of attenuation $A_{\rm stop}$, raising the total attenuation at $f_{\rm stop}$ to a desired level, say $A_{\rm tot}$ dB:

$$A_{\rm tot}=A_{\rm dac}+A_{\rm stop}$$

For example, suppose $f_s=40$ kHz and $L=4$, and we require the total suppression of the replicas to be more than $A_{\rm tot}=60$ dB. The stopband frequency will be $f_{\rm stop}=Lf_s-f_s/2=160-20=140$ kHz. At that frequency the DAC will provide an attenuation $A_{\rm dac}=20\log_{10}(8)=18$ dB, and therefore the postfilter must provide the rest:

$$A_{\rm stop}=A_{\rm tot}-A_{\rm dac}=60-18=42\ \text{dB}$$

Suppose we use a third-order Butterworth filter with magnitude response [298–300]:

$$|H_{\rm post}(f)|^2=\frac{1}{1+\left(\dfrac{f}{f_0}\right)^6}$$

and attenuation in dB:

$$A_{\rm post}(f)=-10\log_{10}|H_{\rm post}(f)|^2=10\log_{10}\left[1+\left(\frac{f}{f_0}\right)^6\right] \tag{14.4.11}$$

where f_0 is the 3-dB normalization frequency to be determined. Then, the requirement that at $f_{\rm stop}$ the attenuation be equal to $A_{\rm stop}$ gives:

$$A_{\rm stop}=10\log_{10}\left[1+\left(\frac{f_{\rm stop}}{f_0}\right)^6\right]$$

which can be solved for f_0:

$$f_0=f_{\rm stop}\left[10^{A_{\rm stop}/10}-1\right]^{-1/6}=140\cdot\left[10^{42/10}-1\right]^{-1/6}=28\ \text{kHz}$$

The third-order Butterworth analog transfer function of the postfilter will be:

$$H_{\rm post}(s)=\frac{1}{1+2\left(\dfrac{s}{\Omega_0}\right)+2\left(\dfrac{s}{\Omega_0}\right)^2+\left(\dfrac{s}{\Omega_0}\right)^3} \tag{14.4.12}$$

where $\Omega_0=2\pi f_0$. This postfilter will adequately remove the spectral images at multiples of $f_s{}'$, but it will also cause a small amount of attenuation within the desired passband. The maximum passband attenuations caused by the postfilter and the DAC at $f_{\rm pass}=f_s/2$ can be computed from Eqs. (14.4.11) and (14.4.4):

$$A_{\rm post}(f_{\rm pass})=10\log_{10}[1+\left(\frac{f_{\rm pass}}{f_0}\right)^6]=10\log_{10}\left[1+(\frac{20}{28})^6\right]=0.54\ \text{dB}$$

$$A_{\rm dac}(f_{\rm pass})=-20\log_{10}\left[\frac{\sin(\pi/2L)}{\pi/2L}\right]=0.22\ \text{dB}$$

resulting in a total passband attenuation of $0.54+0.22=0.76$ dB. This combined attenuation of the DAC and postfilter can be equalized by the interpolator filter. Using the frequency sampling design, we replace the interpolator's defining equation (14.4.7) by the equalized version:

$$D(\omega')=\begin{cases} L\left[\dfrac{\omega'/2}{\sin(\omega'/2)}\right]\left[1+(\dfrac{\omega'}{\omega_0})^6\right]^{1/2}, & \text{if } |\omega'|\leq\dfrac{\pi}{L}\\ 0, & \text{if } \dfrac{\pi}{L}<|\omega'|\leq\pi\end{cases} \tag{14.4.13}$$

where $\omega_0=2\pi f_0/f_s{}'$. The impulse response coefficients will be calculated by:

$$\tilde{d}(k')=\frac{L}{N}\left[1+2\sum_{i=1}^{M}\left[\frac{\omega_i'/2}{\sin(\omega_i'/2)}\right]\left[1+(\frac{\omega_i'}{\omega_0})^6\right]^{1/2}\cos(\omega_i'k')\right]$$

for $-LM\leq k'\leq LM$. These coefficients must be weighted by an appropriate window, as in Eq. (14.4.9).

Figure 14.4.21 shows a Kaiser design corresponding to interpolator stopband attenuation of $A = 60$ dB and a transition width of $\Delta f = 5$ kHz. As before, the resulting filter length is $N = 121$.

For reference, the DAC response $H_{\text{dac}}(f)$, postfilter response $H_{\text{post}}(f)$, and total response $H_{\text{dac}}(f)H_{\text{post}}(f)$ are superimposed on the figure. Notice how they meet their respective specifications at $f_{\text{stop}} = 140$ kHz. The DAC response vanishes (i.e., it has infinite attenuation) at $f_s{}' = 160$ kHz and all its multiples. The figure also shows the filter's passband in a magnified scale, together with the plots of the total filter $H_{\text{dac}}(f)H_{\text{post}}(f)$ and total inverse filter $1/\big(H_{\text{dac}}(f)H_{\text{post}}(f)\big)$.

The effective overall analog reconstructor $H_{\text{interp}}(f)H_{\text{dac}}(f)H_{\text{post}}(f)$, consisting of the equalized interpolator, DAC, and postfilter, is shown in Fig. 14.4.22. The spectral images at multiples of $f_s{}' = 160$ kHz are suppressed by more than 60 dB and the 20 kHz passband is essentially flat. The figure also shows the passband in a magnified scale.

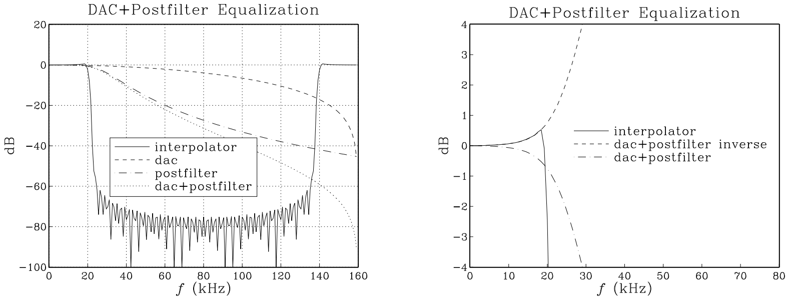


Fig. 14.4.21 4-fold interpolator with equalization of DAC and Butterworth postfilter.

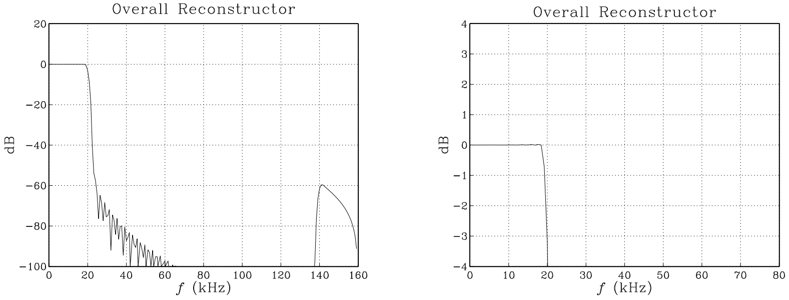


Fig. 14.4.22 Effective reconstructor has flat passband.

In digital audio applications, Bessel postfilters may also be used instead of Butterworth filters; see [353] for an example. Bessel filters provide the additional benefit that they have approximately *linear phase* over their passband. In particular, the transfer function and magnitude response of a third-order Bessel filter are given by [298–301]:

$$H_{\text{post}}(s) = \frac{15}{15 + 15\left(\dfrac{s}{\Omega_0}\right) + 6\left(\dfrac{s}{\Omega_0}\right)^2 + \left(\dfrac{s}{\Omega_0}\right)^3} \tag{14.4.14}$$

$$|H_{\text{post}}(f)|^2 = \frac{225}{225 + 45\left(\dfrac{f}{f_0}\right)^2 + 6\left(\dfrac{f}{f_0}\right)^4 + \left(\dfrac{f}{f_0}\right)^6} \tag{14.4.15}$$

where $\Omega_0 = 2\pi f_0$ and f_0 is related to the 3-dB frequency of the filter by

$$f_{\text{3dB}} = 1.75 f_0$$

The passband attenuation of this filter can be equalized digitally in a similar fashion. For equal 3-dB frequencies, Bessel filters fall off somewhat less sharply than Butterworth ones, thus, suppressing the spectral images by a lesser amount.

For example, for the previous 3-dB frequency $f_{\text{3dB}} = 28$ kHz, we find the normalization frequency $f_0 = f_{\text{3dB}}/1.75 = 16$ kHz. The corresponding postfilter attenuations at the passband and stopband frequencies, $f_{\text{pass}} = 20$ kHz and $f_{\text{stop}} = 140$ kHz, calculated from Eq. (14.4.15) are:

$$A_{\text{post}}(f_{\text{pass}}) = 1.44 \text{ dB}, \qquad A_{\text{post}}(f_{\text{stop}}) = 33 \text{ dB}$$

Thus, the DAC/postfilter combination will only achieve a total stopband attenuation of $33 + 18 = 51$ dB for the removal of the spectral images. Similarly, the total passband attenuation to be compensated by the interpolator will be $0.22 + 1.44 = 1.66$ dB.

If 51 dB suppression of the spectral images is acceptable[†] then we may redesign the interpolator so that it suppresses its stopband also by 51 dB. With $A = 51$ and $L = 4$, the redesigned interpolator will have Kaiser parameters:

$$D = \frac{A - 7.95}{14.36} = 2.998$$

$$N - 1 \geq= \frac{DL}{\Delta F} = 95.93 \quad \Rightarrow \quad N = 97$$

The redesigned equalized interpolation filter and the effective overall reconstruction filter are shown in Fig. 14.4.23. The overall reconstructor has a flat passband and suppresses all spectral images by at least 51 dB.

14.4.5 Multistage Equalization

Another application where equalization may be desirable is in *multistage* implementations of interpolators. The first stage is usually a high-quality lowpass digital filter, whereas the subsequent stages may be interpolators with *simplified* structure, such as linear or hold interpolators which do not have a flat passband. In such cases, the filter in the first stage can be designed to equalize the attenuation of all the subsequent stages *within* the passband. In addition, the analog reconstructing DAC and postfilter may also be equalized by the first stage. The overall system is shown in Fig. 14.4.24.

[†] The first Philips CD player had a similar 4-times oversampling interpolator of order $N = 97$ and stopband attenuation of $A = 50$ dB; see [353] for details.

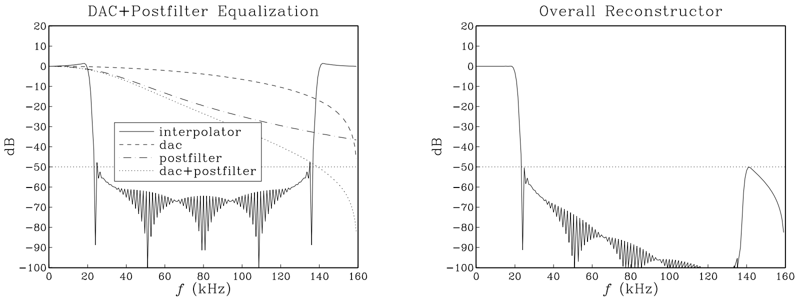


Fig. 14.4.23 4-fold interpolator with equalization of DAC and Bessel postfilter.

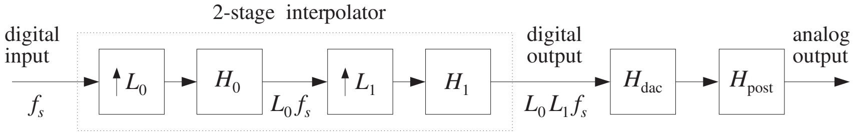


Fig. 14.4.24 First stage equalizes the passband responses of all remaining stages.

As an example, consider a 2-stage interpolator with oversampling factors $L = L_0L_1$. The first interpolation filter is a lowpass digital filter with sharp cutoff at $f_s/2$, and the second is a *linear* interpolator. The first filter H_0 increases the sampling rate from f_s to L_0f_s. The second filter H_1 increases it by a factor L_1, from L_0f_s to $L_1L_0f_s$. It has magnitude response (normalized to unity at DC):

$$|H_1(f)| = \left|\frac{\sin(\pi f/L_0f_s)}{L_1\sin(\pi f/L_1L_0f_s)}\right|^2 = \left|\frac{\sin(\omega'/2)}{L_1\sin(\omega'/2L_1)}\right|^2 \equiv D_{\text{lin}}(\omega')$$

where $\omega' = 2\pi f/(L_0f_s)$ is the digital frequency with respect to the rate L_0f_s of the filter H_0.

The job of the filter H_0 is to remove the (L_0-1) spectral replicas that lie between replicas at multiples of L_0f_s. Because it is periodic with period L_0f_s, the filter H_0 cannot remove the replicas at multiples of L_0f_s. Those are *partially* removed by the second filter H_1, which vanishes at multiples of L_0f_s that are *not* multiples of $L_1L_0f_s$. Even though these replicas vanish at their centers, their edges are only suppressed by about 33 dB. Further suppression requires the aid of a postfilter. The combined effect of the H_0 and H_1 filters is to leave only the replicas at multiples of the final sampling rate $L_1L_0f_s$. Those are also removed by the postfilter. Figures (14.4.25) and (14.4.27) illustrate these remarks for $L_0 = L_1 = 4$.

The output of H_1 is applied to a staircase DAC operating at the high rate $L_1L_0f_s$. Its frequency response (normalized to unity at DC) is:

$$|H_{\text{dac}}(f)| = \left|\frac{\sin(\pi f/L_1L_0f_s)}{\pi f/L_1L_0f_s}\right| = \left|\frac{\sin(\omega'/2L_1)}{\omega'/2L_1}\right| \equiv D_{\text{dac}}(\omega')$$

The DAC's $\sin x/x$ response vanishes at multiples of the final rate $L_1L_0f_s$. Finally, the staircase output of the DAC will be smoothed by an analog postfilter, say, a third-order Butterworth filter with 3-dB cutoff frequency f_0, having magnitude response:

$$|H_{\text{post}}(f)| = \frac{1}{\left[1+\left(\dfrac{f}{f_0}\right)^6\right]^{1/2}} = \frac{1}{\left[1+\left(\dfrac{\omega'}{\omega_0}\right)^6\right]^{1/2}} \equiv D_{\text{post}}(\omega')$$

where $\omega_0 = 2\pi f_0/L_0f_s$. The three responses $D_{\text{lin}}(\omega')$, $D_{\text{dac}}(\omega')$, and $D_{\text{post}}(\omega')$ can be equalized simultaneously by the interpolation filter H_0 by defining its passband specifications as:

$$D(\omega') = \begin{cases} \dfrac{L_0}{D_{\text{lin}}(\omega')D_{\text{dac}}(\omega')D_{\text{post}}(\omega')} & \text{if } |\omega'| \le \dfrac{\pi}{L_0} \\ 0, & \text{if } \dfrac{\pi}{L_0} < |\omega'| \le \pi \end{cases} \tag{14.4.16}$$

Assuming a filter length $N_0 = 2L_0M_0 + 1$, we obtain the coefficients of the filter H_0 by the following frequency sampling design expression:

$$\tilde{d}(k') = \frac{L_0}{N_0}\left[1 + 2\sum_{i=1}^{M_0}\frac{1}{D_{\text{lin}}(\omega_i')D_{\text{dac}}(\omega_i')D_{\text{post}}(\omega_i')}\cos(\omega_i'k')\right]$$

for $-L_0M_0 \le k' \le L_0M_0$. Note that if L_0 and L_1 are large, two simplifications may be introduced in the definition (14.4.16). First, we may omit the DAC equalization factor $D_{\text{dac}}(\omega')$ because *within* the passband $0 \le \omega' \le \pi/L_0$ it reaches a maximum attenuation of:

$$\frac{\sin(\pi/2L_1L_0)}{\pi/2L_1L_0}$$

which will be extremely small. Second, again within the passband $0 \le \omega' \le \pi/L_0$, we can approximate the linear interpolator response by its analog equivalent $(\sin x/x)^2$ response, which is *independent* of L_1:

$$D_{\text{lin}}(\omega') = \left|\frac{\sin(\omega'/2)}{L_1\sin(\omega'/2L_1)}\right|^2 \simeq \left|\frac{\sin(\omega'/2)}{\omega'/2}\right|^2$$

In deciding the specifications of the postfilter, we must consider the total attenuation of the replica at L_0f_s, which will have the *worst* attenuation. Because of the downward sloping of the postfilter, the remaining replicas will be attenuated more. At the left edge of that replica, that is, at frequency

$$f_{\text{stop}} = L_0f_s - \frac{f_s}{2} \quad\Rightarrow\quad \omega'_{\text{stop}} = \frac{2\pi f_{\text{stop}}}{L_0f_s} = \frac{\pi(2L_0-1)}{L_0}$$

the attenuations of the linear interpolator, DAC, and postfilter can be calculated by

$$\begin{aligned}
A_{\rm lin}(f_{\rm stop}) &= -20\log_{10}\left|\frac{\sin(\omega'_{\rm stop}/2)}{L_1\sin(\omega'_{\rm stop}/2L_1)}\right|^2 \\
A_{\rm dac}(f_{\rm stop}) &= -20\log_{10}\left|\frac{\sin(\omega'_{\rm stop}/2L_1)}{\omega'_{\rm stop}/2L_1}\right| \\
A_{\rm post}(f_{\rm stop}) &= 10\log_{10}\left[1+\left(\frac{f_{\rm stop}}{f_0}\right)^6\right]
\end{aligned} \tag{14.4.17}$$

From these equations, one may determine the 3-dB frequency f_0 of the postfilter in order that the total attenuation at $f_{\rm stop}$ be equal to a desired level, say $A_{\rm tot}$

$$A_{\rm tot} = A_{\rm lin}(f_{\rm stop}) + A_{\rm dac}(f_{\rm stop}) + A_{\rm post}(f_{\rm stop})$$

Similarly, one can calculate the attenuations of the linear interpolator, DAC, and postfilter at the edge of the passband, that is, at $f_{\rm pass} = f_s/2$ or $\omega'_{\rm pass} = \pi/L_0$

$$\begin{aligned}
A_{\rm lin}(f_{\rm pass}) &= -20\log_{10}\left|\frac{\sin(\omega'_{\rm pass}/2)}{L_1\sin(\omega'_{\rm pass}/2L_1)}\right|^2 \\
A_{\rm dac}(f_{\rm pass}) &= -20\log_{10}\left|\frac{\sin(\omega'_{\rm pass}/2L_1)}{\omega'_{\rm pass}/2L_1}\right| \\
A_{\rm post}(f_{\rm pass}) &= 10\log_{10}\left[1+\left(\frac{f_{\rm pass}}{f_0}\right)^6\right]
\end{aligned} \tag{14.4.18}$$

Their sum is the total passband attenuation that must be equalized by the interpolator H_0. Finally, one must verify that the last replica at $L_1L_0f_s$, which survives the combined interpolation filters H_0 and H_1, is attenuated sufficiently by the DAC/postfilter combination. At the left edge of that replica, that is, at frequency

$$f_{\rm last} = L_1L_0f_s - \frac{f_s}{2} \quad\Rightarrow\quad \omega'_{\rm last} = \frac{2\pi f_{\rm last}}{L_0f_s} = \frac{\pi(2L_1L_0-1)}{L_0}$$

the attenuations of the linear interpolator, DAC, and postfilter are

$$\begin{aligned}
A_{\rm lin}(f_{\rm last}) &= -20\log_{10}\left|\frac{\sin(\omega'_{\rm last}/2)}{L_1\sin(\omega'_{\rm last}/2L_1)}\right|^2 \\
A_{\rm dac}(f_{\rm last}) &= -20\log_{10}\left|\frac{\sin(\omega'_{\rm last}/2L_1)}{\omega'_{\rm last}/2L_1}\right| \\
A_{\rm post}(f_{\rm last}) &= 10\log_{10}\left[1+\left(\frac{f_{\rm last}}{f_0}\right)^6\right]
\end{aligned} \tag{14.4.19}$$

As a concrete design example,[†] suppose $f_s = 40$ kHz and $L_0 = L_1 = 4$, so that the total interpolation factor will be $L = 16$. For the first stage H_0, we use a Kaiser design with stopband attenuation $A = 60$ dB and transition width $\Delta f = 5$ kHz. The Kaiser parameters are:

$$D = \frac{A-7.95}{14.36} = 3.625$$

$$N_0 - 1 \geq \frac{DL_0}{\Delta F} = 115.98 \quad\Rightarrow\quad N_0 = 121$$

The linear interpolator and DAC will have frequency responses:

$$|H_1(f)| = \left|\frac{\sin(\pi f/4f_s)}{4\sin(\pi f/16f_s)}\right|^2, \qquad |H_{\rm dac}(f)| = \left|\frac{\sin(\pi f/16f_s)}{\pi f/16f_s}\right|$$

It is evident that $|H_1(f)|$ vanishes at all multiples of $4f_s$ which are not multiples of $16f_s$, whereas $|H_{\rm dac}(f)|$ vanishes at all non-zero multiples of $16f_s$.

Using Eqs. (14.4.17), we find that the requirement that at $f_{\rm stop} = L_0f_s - f_s/2 = 140$ kHz the total attenuation be more than 60 dB gives the value $f_0 = 50$ kHz for the 3-dB frequency of the postfilter.

Table 14.4.1 shows the attenuations calculated by Eqs. (14.4.17–14.4.19) at the three frequencies $f_{\rm pass} = 20$ kHz, $f_{\rm stop} = 140$ kHz, and $f_{\rm last} = 620$ kHz, or equivalently, the normalized ones $\omega'_{\rm pass} = \pi/4$, $\omega'_{\rm stop} = 7\pi/4$, and $\omega'_{\rm last} = 31\pi/4$.

	$f_{\rm pass}$	$f_{\rm stop}$	$f_{\rm last}$
$A_{\rm lin}$	0.421	32.863	0.421
$A_{\rm dac}$	0.014	0.695	29.841
$A_{\rm post}$	0.018	26.838	65.605
$A_{\rm tot}$	0.453	60.396	95.867

Table 14.4.1 Attenuations in dB.

It is evident from this table that most of the attenuation in the passband arises from the linear interpolator—one could have equalized only the linear interpolator, ignoring the DAC and postfilter, without much degradation.

At 140 kHz, the linear interpolator and postfilter contribute almost equally towards the suppression of the 160 kHz replica—the linear interpolator providing an attenuation of about 33 dB and the postfilter supplementing it with another 27 dB for a total of 60 dB. The DAC's contribution is minimal there.

At 620 kHz, the main contributors towards the suppression of the 640 kHz replica are the postfilter and the DAC providing a total of 95 dB suppression. The linear interpolator's contribution is minimal (by symmetry, its attenuation is the same at 20 kHz and 620 kHz).

Figure 14.4.25 shows the magnitude responses $H_0(f)$, $H_1(f)$, $H_{\rm dac}(f)$, $H_{\rm post}(f)$. It also shows the total interpolator response $H_0(f)H_1(f)$, which removes the replicas at multiples of L_0f_s only at the 33 dB level.

Figure 14.4.26 shows the magnified passband of the filter $H_0(f)$, together with the inverse filter $\left[H_1(f)H_{\rm dac}(f)H_{\rm post}(f)\right]^{-1}$. It also shows the magnified passband of the

[†] See [355] for a similar example having three stages $L_0 = 4$, $L_1 = 32$ linear, and $L_2 = 2$ hold interpolator, with overall $L = 256$.

total interpolator filter $H_0(f)H_1(f)$, which is essentially flat since most of the equalizing action of $H_0(f)$ goes to equalize $H_1(f)$. Figure 14.4.27 shows the effective total reconstruction filter

$$H_{\text{rec}}(f)=H_0(f)H_1(f)H_{\text{dac}}(f)H_{\text{post}}(f)$$

which has a flat passband and suppresses all spectral images by at least 60 dB.

In summary, the spectral images of the original sampled signal at multiples $mf_s = m40$ kHz, are removed in several stages: First, the interpolator H_0 removes the replicas at $m = (1,2,3)$, $(5,6,7)$, $(9,10,11)$, $(13,14,15)$, and so on. Then, the second interpolator H_1, with the help of the postfilter, removes the replicas at $m = 4,8,12$, and so on. Finally, the postfilter removes the replica $m = 16$ (and all others at multiples of 16 beyond that).

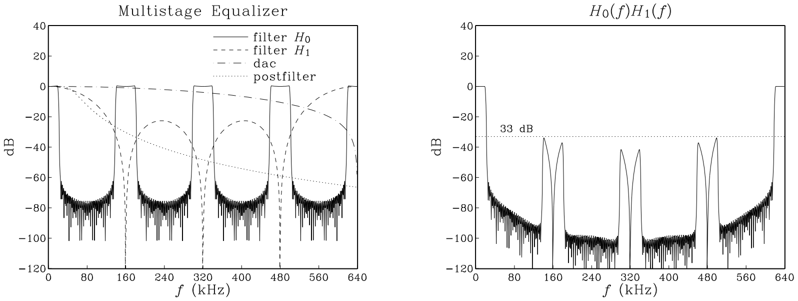


Fig. 14.4.25 16-times interpolator with DAC, postfilter, and multistage equalization.

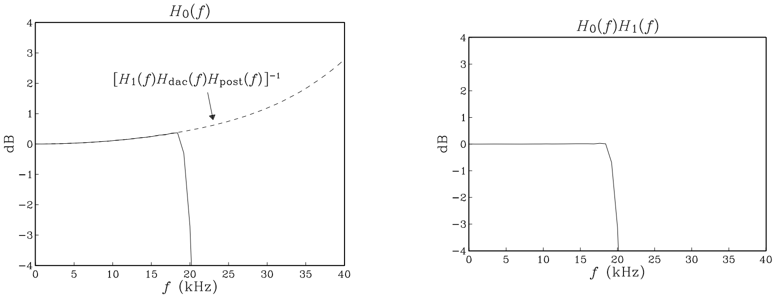


Fig. 14.4.26 Magnified passbands of $H_0(f)$ and $H_0(f)H_1(f)$.

This design example raises some additional questions: Could we have used a second-order Butterworth postfilter? A first-order one? Given a desired level, say A dB, of suppression of the images in the overall equalized reconstructor, can we predict what the lowest order of the postfilter would be? To answer these questions and to give a more accurate design technique, let us define the total filter being equalized by $H_0(f)$, namely,

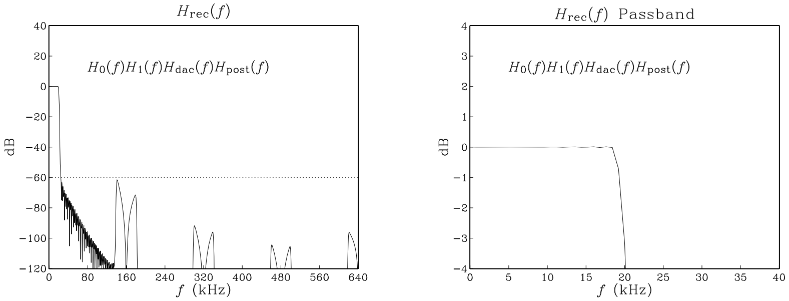


Fig. 14.4.27 Effective reconstructor has flat passband and 60 dB stopband.

$$H_{\text{eq}}(f)=H_1(f)H_{\text{dac}}(f)H_{\text{post}}(f)$$

so that the overall reconstructor is:

$$H_{\text{rec}}(f)=H_0(f)H_{\text{eq}}(f)$$

Assuming that H_0 and H_{rec} are normalized to unity at DC by dividing out a factor L_0, we write the corresponding attenuations in dB:

$$\begin{aligned}A_{\text{rec}}(f) &= -20\log_{10}|H_{\text{rec}}(f)|\\ A_0(f) &= -20\log_{10}|H_0(f)|\\ A_{\text{eq}}(f) &= -20\log_{10}|H_{\text{eq}}(f)|\end{aligned}$$

and therefore, we have:

$$A_{\text{rec}}(f)=A_0(f)+A_{\text{eq}}(f)$$

The attenuation achieved by $H_{\text{rec}}(f)$ at frequency f_{stop} should be at least A dB, that is, $A_{\text{rec}}(f_{\text{stop}})\geq A$, and therefore

$$A_0(f_{\text{stop}})+A_{\text{eq}}(f_{\text{stop}})\geq A \tag{14.4.20}$$

Because the filter $H_0(f)$ is periodic with period L_0f_s and the passband and stopband frequencies satisfy $f_{\text{stop}} = L_0f_s - f_{\text{pass}}$, it follows that

$$|H_0(f_{\text{stop}})| = |H_0(L_0f_s - f_{\text{pass}})| = |H_0(-f_{\text{pass}})| = |H_0(f_{\text{pass}})|$$

or, in dB:

$$A_0(f_{\text{stop}})=A_0(f_{\text{pass}})$$

But at frequency f_{pass}, the filter $H_0(f)$ is designed to equalize the total filter $H_{\text{eq}}(f)$, that is,

$$|H_0(f_{\text{pass}})| = \frac{1}{|H_{\text{eq}}(f_{\text{pass}})|} \qquad \Rightarrow \qquad A_0(f_{\text{stop}}) = A_0(f_{\text{pass}}) = -A_{\text{eq}}(f_{\text{pass}})$$

Therefore, we rewrite the design condition (14.4.20) as:

$$A_{\text{eq}}(f_{\text{stop}}) - A_{\text{eq}}(f_{\text{pass}}) \geq A \tag{14.4.21}$$

For the designed example above, we can subtract the two entries in the last row of Table 14.4.1 to get the value $60.396 - 0.453 = 59.943$, which is almost the desired 60 dB—it would exceed 60 dB had we chosen a slightly smaller 3-dB normalization frequency for the postfilter, for example, $f_0 = 49.8$ kHz.

Writing A_{eq} as the sum of the individual attenuations of the linear interpolator, the DAC, and the postfilter, and solving Eq. (14.4.21) for the difference of attenuations of the postfilter, we find

$$A_{\text{post}}(f_{\text{stop}}) - A_{\text{post}}(f_{\text{pass}}) \geq A - A_d \tag{14.4.22}$$

where

$$A_d = A_{\text{lin}}(f_{\text{stop}}) - A_{\text{lin}}(f_{\text{pass}}) + A_{\text{dac}}(f_{\text{stop}}) - A_{\text{dac}}(f_{\text{pass}})$$

Given the value of A, the right-hand side of Eq. (14.4.22) can be calculated using Eqs. (14.4.17) and (14.4.18). Then, Eq. (14.4.22) imposes a certain restriction on the order of the postfilter. For a Butterworth filter of order N_b, we have

$$A_{\text{post}}(f) = 10\log_{10}\left[1 + \left(\frac{f}{f_0}\right)^{2N_b}\right]$$

and therefore, the design condition (14.4.22) becomes

$$10\log_{10}\left[\frac{1 + (f_{\text{stop}}/f_0)^{2N_b}}{1 + (f_{\text{pass}}/f_0)^{2N_b}}\right] \geq A - A_d \tag{14.4.23}$$

Given N_b, Eq. (14.4.23) can be solved for the 3-dB frequency f_0. For large values of N_b, the passband term $A_{\text{post}}(f_{\text{pass}})$ can be ignored because it is much smaller than the stopband term $A_{\text{post}}(f_{\text{stop}})$, as was the case in Table 14.4.1. For small values of N_b, f_0 must also get smaller in order to provide sufficient attenuation at f_{stop}, but this also causes more attenuation within the passband, so that the difference $A_{\text{post}}(f_{\text{stop}}) - A_{\text{post}}(f_{\text{pass}})$ may never become large enough to satisfy Eq. (14.4.22).

In fact, thinking of the left-hand side of Eq. (14.4.23) as a function of f_0, one can easily verify that it is a decreasing function of f_0 and its maximum value, reached at $f_0 = 0$, is $20N_b\log_{10}(f_{\text{stop}}/f_{\text{pass}})$. Therefore, Eq. (14.4.23) will have a solution for f_0 only if N_b is large enough to satisfy

$$20N_b\log_{10}\left(\frac{f_{\text{stop}}}{f_{\text{pass}}}\right) \geq A - A_d \qquad \Rightarrow \qquad N_b \geq \frac{A - A_d}{20\log_{10}\left(\dfrac{f_{\text{stop}}}{f_{\text{pass}}}\right)}$$

Because $f_{\text{stop}}/f_{\text{pass}} = 2L_0 - 1$, we can rewrite this condition as

$$N_b \geq \frac{A - A_d}{20\log_{10}(2L_0 - 1)} \tag{14.4.24}$$

If Eq. (14.4.24) is satisfied, then the solution of Eq. (14.4.23) for f_0 is

$$f_0 = f_{\text{pass}}\left[\frac{(2L_0 - 1)^{2N_b - 1}}{10^{(A - A_d)/10} - 1} - 1\right]^{1/2N_b} \tag{14.4.25}$$

Figure 14.4.28 shows the minimum Butterworth order given in Eq. (14.4.24) as a function of the desired attenuation A, for various values of the interpolation factor L_0. One should pick, of course, the next integer above each curve. As L_0 increases, separating the $L_0 f_s$ replicas more, the allowed filter order becomes less. In the figure, $L_1 = 4$. For a given value of L_0, the dependence of the curves on L_1 is very minimal. Only the intercept A_d, not the slope, of the straight lines is slightly changed if L_1 is changed.

Inspecting this figure, we find for the above example that any filter order $N_b \geq 2$ can be used, but $N_b = 1$ cannot. Alternatively, we can calculate Eq. (14.4.24) directly. Using Table 14.4.1, we find $A_d = 33.123$ dB and therefore Eq. (14.4.24) yields $N_b \geq (60 - 33.123)/20\log_{10}(8-1) = 1.59$.

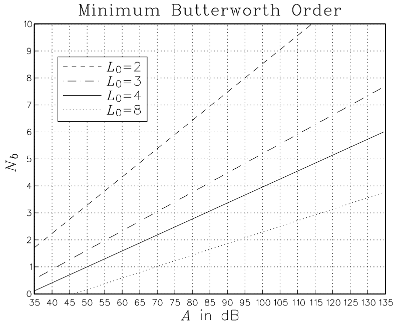


Fig. 14.4.28 Minimum Butterworth postfilter order for given stopband attenuation.

14.5 Decimation and Oversampling

Decimation by an integer factor L is the reverse of interpolation, that is, *decreasing* the sampling rate from the high rate f_s' to the lower rate $f_s = f_s'/L$.

An ideal interpolator replaces a low-rate signal $x(n)$ by the high-rate interpolated signal $x'(n')$, which would ideally correspond to the resampling of the analog signal at the higher rate. As shown in Fig. 14.2.6, the spectrum $X'(f)$ of $x'(n')$ is the spectrum of $x(n)$ with $L-1$ spectral images removed between multiples of f_s'. This ideal interpolation process can be reversed by keeping from $x'(n')$ every Lth sample and discarding the $L - 1$ samples that were interpolated between the low-rate ones.

This process of *downsampling* and its effect in the time and frequency domains is depicted in Fig. 14.5.1. Formally, the downsampled signal is defined in terms of the slow time scale as follows:

$$\boxed{x_{\text{down}}(n) = x'(n')\big|_{n'=nL} = x'(nL)} \tag{14.5.1}$$

For the ideal situation depicted in Fig. 14.5.1, the downsampled signal $x_{\text{down}}(n)$ *coincides* with the low-rate signal $x(n)$ that would have been obtained had the analog signal been resampled at the lower rate f_s, that is,

$$x(n) = x_{\text{down}}(n) = x'(nL) \tag{14.5.2}$$

The gaps in the input spectrum $X'(f)$ are necessary to guarantee this equality. Dropping the sampling rate by a factor of L, shrinks the Nyquist interval $[-f_s{}'/2, f_s{}'/2]$ by a factor of L to the new interval $[-f_s/2, f_s/2]$. Thus, if the signal had frequency components outside the new Nyquist interval, aliasing would occur and $x_{\text{down}}(n) \neq x(n)$.

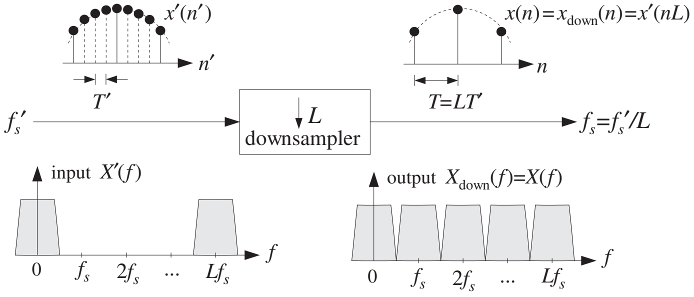


Fig. 14.5.1 Downsampler keeps one out of every L high-rate samples.

In Fig. 14.5.1, the input spectrum was already restricted to the f_s Nyquist interval, and therefore, aliasing did not occur. The rate decrease causes the spectral images of $X'(f)$ at multiples of $f_s{}'$ to be *down shifted* and become images of $X(f)$ at multiples of f_s without overlapping. The mathematical justification of this down-shifting property is derived by expressing Eq. (14.5.2) in the frequency domain. It can be shown (see Problem 14.12) that:

$$\boxed{X(f) = X_{\text{down}}(f) = \frac{1}{L}\sum_{m=0}^{L-1} X'(f - mf_s)} \tag{14.5.3}$$

Therefore, the downsampling process causes the periodic replication of the original spectrum $X'(f)$ at multiples of the low rate f_s. This operation is depicted in Fig. 14.5.2 for $L = 4$.

In general, if the high-rate signal $x'(n')$ has frequency components outside the low-rate Nyquist interval $[-f_s/2, f_s/2]$, then downsampling alone is not sufficient to perform decimation. For example, noise in the signal, such as quantization noise arising from the A/D conversion process, will have such frequency components.

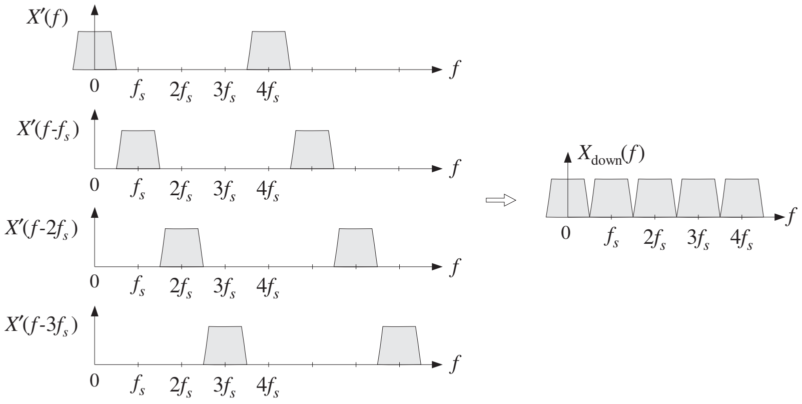


Fig. 14.5.2 Downsampled spectrum is sum of shifted high-rate spectra.

To avoid the aliasing that will arise by the spectrum replication property (14.5.3), the high-rate input $x'(n')$ must be *prefiltered* by a digital lowpass filter, called the *decimation filter*. The combined filter/downsampler system is called a *decimator* and is depicted in Fig. 14.5.3.

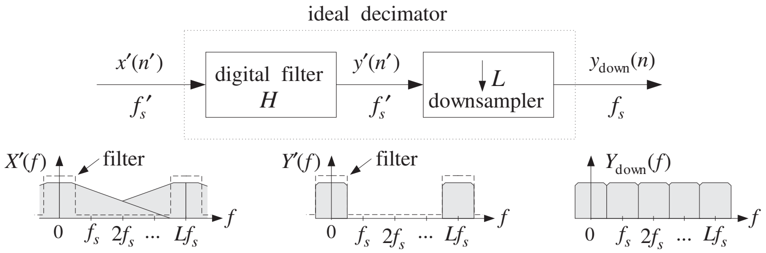


Fig. 14.5.3 Ideal digital decimator in frequency domain.

The filter operates at the high rate $f_s{}'$ and has cutoff frequency $f_c = f_s/2 = f_s{}'/2L$. It is similar to the ideal interpolation filter, except its DC gain is unity instead of L. The high-rate output of the filter is downsampled to obtain the desired low-rate decimated signal, with non-overlapping down-shifted replicas:

$$y_{\text{down}}(n) = y'(nL), \qquad Y_{\text{down}}(f) = \frac{1}{L}\sum_{m=0}^{L-1} Y'(f - mf_s) \tag{14.5.4}$$

The design of the decimation filter is identical to that of the interpolation filter. For example, a length-N FIR decimator can be obtained by windowing the (causal) ideal impulse response:

$$h(n') = w(n')d(n' - LM), \qquad \text{where} \quad d(k') = \frac{\sin(\pi k'/L)}{\pi k'}$$

where $n' = 0, 1, \dots, N-1$, and $N = 2LM + 1$. A Kaiser window $w(n')$ may be used. The downsampled output is obtained by:

$$\boxed{y_{\text{down}}(n) = y'(nL) = \sum_{m'=0}^{N-1} h(m')x'(nL - m')} \tag{14.5.5}$$

Because only every Lth output of the filter is needed, the overall computational rate is reduced by a factor of L, that is,

$$R = \frac{1}{L}Nf_s{}' = Nf_s \tag{14.5.6}$$

This is similar to the savings of the polyphase form of interpolation. A simple implementation uses a length-N tapped delay line into which the high-rate input samples are shifted at the high rate $f_s{}'$. Every L inputs, its contents are used to perform the filter's dot product output computation. A circular buffer implementation of the delay-line would, of course, avoid the time it takes to perform the shifting. Denoting by $\mathbf{w} = [w_0, w_1, \dots, w_{N-1}]$ the N-dimensional internal state vector of the filter, we may state this filtering/downsampling algorithm as follows:

```
for each high-rate input sample $x'$ do:
    $w_0 = x'$
    for every $L$th input compute:
        $y_{\text{down}} = \text{dot}(N-1, \mathbf{h}, \mathbf{w})$
    $\text{delay}(N-1, \mathbf{w})$
```
(14.5.7)

Multistage implementations of decimators are also possible [347–350]. The proper ordering of the decimation stages is the *reverse* of the interpolation case, that is, the decimator with the most *stringent* specifications is placed *last*.

Often, the earlier decimators, which also have the highest rates, are chosen to have *simplified* structures, such as simple averaging filters [351]. For example, the decimation version of the hold interpolator of Section 14.3 is obtained by dividing Eq. (14.3.8) by L to restore its DC gain to unity:

$$H(\zeta) = \frac{1}{L}\frac{1-\zeta^{-L}}{1-\zeta^{-1}} = \frac{1}{L}\big[1 + \zeta^{-1} + \zeta^{-2} + \cdots + \zeta^{-(L-1)}\big] \tag{14.5.8}$$

where ζ^{-1} is one high-rate delay. Thus, the decimator is a simple FIR *averaging* filter that averages L successive high-rate samples:

$$y_{\text{down}}(n) = \frac{x'(nL) + x'(nL-1) + x'(nL-2) + \cdots + x'(nL-L+1)}{L} \tag{14.5.9}$$

If so desired, the cruder passbands of the earlier decimators can be equalized by the last decimator, which can also equalize any imperfect passband of the analog antialiasing prefilter used prior to sampling.

Indeed, one of the main uses of decimators is to alleviate the need for high-quality analog prefilters, much as the interpolators ease the specifications of the anti-image postfilters. This idea is used in many current applications, such as the sampling systems of DAT machines, PC sound cards, speech CODECs, and various types of delta-sigma A/D converter chips.

Sampling an analog signal, such as audio, at its nominal Nyquist rate f_s would require a high-quality analog prefilter to bandlimit the input to the Nyquist frequency $f_{\max} = f_s/2$. In a sampling system that uses oversampling and decimation, the analog input is first prefiltered by a simple prefilter and then sampled at the higher rate $f_s{}' = Lf_s$. The decimation filter then reduces the bandwidth of the sampled signal to $f_s/2$. The sharp cutoffs at the Nyquist frequency $f_s/2$ are provided by the digital decimation filter instead of the prefilter.

The specifications of the prefilter are shown in Fig. 14.5.4. The decimator removes all frequencies from the range $[f_s/2, Lf_s - f_s/2]$. But because of periodicity, it cannot remove any frequencies in the range $Lf_s \pm f_s/2$. Such frequencies, if present in the analog input, must be removed by the prefilter prior to sampling; otherwise they will be aliased back into the desired Nyquist interval $[-f_s/2, f_s/2]$. Therefore, the prefilter's passband and stopband frequencies are:

$$\boxed{f_{\text{pass}} = \frac{f_s}{2}, \qquad f_{\text{stop}} = Lf_s - \frac{f_s}{2}} \tag{14.5.10}$$

The transition width of the prefilter is $\Delta f = f_{\text{stop}} - f_{\text{pass}} = (L-1)f_s$ and gets wider with the oversampling ratio L. Hence, the filter's complexity reduces with increasing L. (See Problem 14.16 for a quantitative relationship between L and filter order N.)

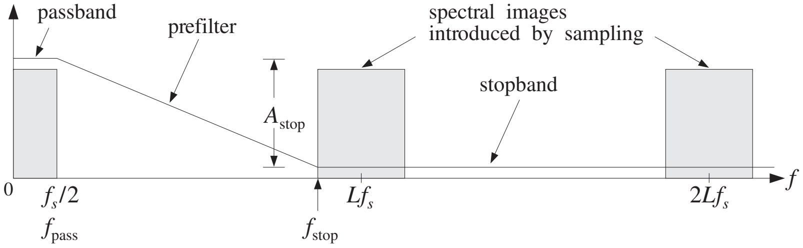


Fig. 14.5.4 Analog prefilter specifications for L-fold decimation.

In summary, oversampling in conjunction with decimation and interpolation alleviates the need for high-quality analog prefilters and postfilters by assigning the burden of achieving sharp transition characteristics to the *digital* filters. Figure 14.5.5 shows an oversampling DSP system in which sampling and reconstruction are carried out at the fast rate $f_s{}'$, and any intermediate digital processing at the low rate f_s.

A second major benefit of oversampling is that it also simplifies the structure of the A/D and D/A converters shown in the figure, so that they require fewer bits without sacrificing quality. This is accomplished by the principle of *feedback quantization*, which we discuss in Section 14.7. The changes in Fig. 14.5.5 are to replace the conventional ADC block by a delta-sigma ADC operating at fewer bits (even 1 bit), and insert between the output interpolator and the DAC a noise shaping quantizer that requantizes the output to fewer bits.

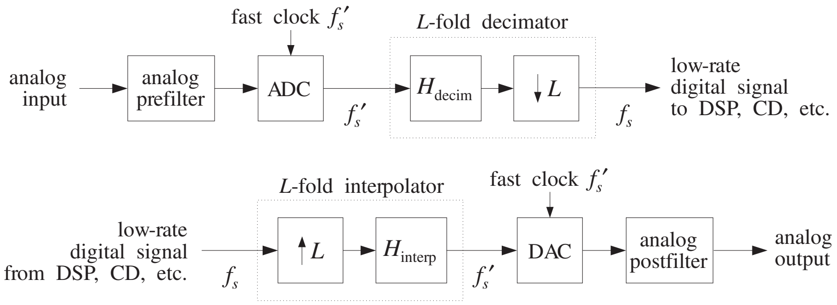


Fig. 14.5.5 Oversampling DSP system.

14.6 Sampling Rate Converters

Interpolators and decimators are examples of sampling rate converters that change the rate by *integer* factors. A more general sampling rate converter [347–350] can change the rate by an arbitrary *rational* factor, say L/M, so that the output rate will be related to the input rate by:

$$\boxed{f_s{}' = \frac{L}{M} f_s} \tag{14.6.1}$$

Such rate changes are necessary in practice for interfacing DSP systems operating at different rates. For example, to convert digital audio for broadcasting, sampled at 32 kHz, to digital audio for a DAT machine, sampled at 48 kHz, one must use a conversion factor of $48/32 = 3/2$. Similarly, to convert DAT audio to CD audio at 44.1 kHz, one must use the factor $44.1/48 = 147/160$.

The rate conversion can be accomplished by first increasing the rate by a factor of L to the high rate $f_s{}'' = Lf_s$ using an L-fold interpolator, and then decreasing the rate by a factor of M down to $f_s{}' = f_s{}''/M = Lf_s/M$ using an M-fold decimator.

Note that $f_s{}''$ is an integer multiple of both the input and output rates, and the corresponding sampling time interval $T'' = 1/f_s{}''$ is an integer fraction of both the input and output sampling times T and T':

$$f_s{}'' = Lf_s = Mf_s{}', \qquad T'' = \frac{T}{L} = \frac{T'}{M} \tag{14.6.2}$$

Because both the interpolation and decimation filters are operating at the same high rate $f_s{}''$ and both are lowpass filters, they may be combined into a single lowpass filter preceded by an upsampler and followed by a downsampler, as shown in Fig. 14.6.1.

The interpolation filter must have cutoff frequency $f_s{}''/2L = f_s/2$ and the decimation filter $f_s{}''/2M = f_s{}'/2$. Thus, the cutoff frequency of the common filter must be chosen to be the *minimum* of the two:

$$\boxed{f_c = \frac{1}{2}\min(f_s, f_s{}')} \tag{14.6.3}$$

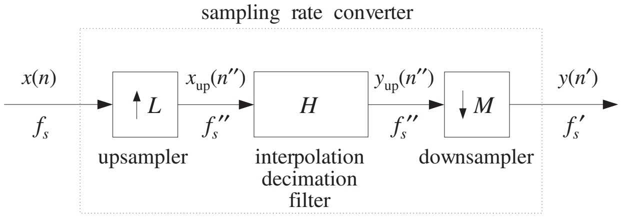


Fig. 14.6.1 Sampling rate conversion by a factor of L/M.

which can be written also in the alternative forms:

$$f_c = \min\left(1, \frac{L}{M}\right)\frac{f_s}{2} = \min\left(\frac{M}{L}, 1\right)\frac{f_s{}'}{2} = \min\left(\frac{1}{L}, \frac{1}{M}\right)\frac{f_s{}''}{2}$$

In units of the high-rate digital frequency $\omega'' = 2\pi f/f_s{}''$, we have:

$$\boxed{\omega_c'' = \frac{2\pi f_c}{f_s{}''} = \min\left(\frac{\pi}{L}, \frac{\pi}{M}\right) = \frac{\pi}{\max(L, M)}} \tag{14.6.4}$$

When $f_s{}' > f_s$, the common filter acts as an *anti-image postfilter* for the upsampler, removing the spectral replicas at multiples of f_s but not at multiples of Lf_s. When $f_s{}' < f_s$, it acts as an *antialiasing prefilter* for the downsampler, making sure that the down-shifted replicas at multiples of $f_s{}'$ do not overlap.

The design of the filter is straightforward. Assuming a filter length N of the form† $N = 2LK + 1$ and passband gain of L, we define the windowed impulse response, with respect to the high-rate time index $n'' = 0, 1, \dots, N-1$:

$$h(n'') = w(n'')d(n'' - LK), \quad \text{where} \quad d(k'') = L\,\frac{\sin(\omega_c''k'')}{\pi k''} \tag{14.6.5}$$

where $w(n'')$ is any desired length-N window. Its L polyphase subfilters of length $2K$ are defined for $i = 0, 1, \dots, L-1$:

$$h_i(n) = h(Ln + i), \qquad n = 0, 1, \dots, 2K-1 \tag{14.6.6}$$

Next, we discuss the time-domain operation and implementation of the converter. The input signal $x(n)$ is upsampled to the high rate $f_s{}''$. Then, the upsampled input $x_{\text{up}}(n'')$ is filtered, generating the interpolated output $y_{\text{up}}(n'')$, which is then downsampled by keeping one out of every M samples, that is, setting $n'' = Mn'$ to obtain the desired signal $y(n')$ resampled at rate $f_s{}'$. Thus, we have:

$$y_{\text{up}}(n'') = \sum_{m''=0}^{N-1} h(m'')x_{\text{up}}(n'' - m'') \quad \text{and} \quad y(n') = y_{\text{up}}(Mn')$$

The interpolation operation can be implemented efficiently in its polyphase realization. Setting $n'' = Ln + i$, with $i = 0, 1, \dots, L-1$, we obtain the ith sample interpolated between the input samples $x(n)$ and $x(n+1)$, from Eq. (14.2.18):

† Here, we use K instead of M to avoid confusion with the downsampling factor M.

$$y_i(n) = y_{\text{up}}(Ln+i) = \sum_{m=0}^{P} h_i(m)x(n-m) = \text{dot}\big(P, \mathbf{h}_i, \mathbf{w}(n)\big) \tag{14.6.7}$$

where we set $P = 2K - 1$ for the *order* of the polyphase subfilters (the time-advance required for causal operation is not shown here). As we saw in Eq. (14.2.20), its implementation requires a low-rate tapped delay line $\mathbf{w} = [w_0, w_1, \dots, w_P]$, which is used by all polyphase subfilters before it is updated.

Because the downsampler keeps only every Mth filter output, it is not necessary to compute all L interpolated outputs between input samples. Only those interpolated values that correspond to the output time grid need be computed. Given an output sample time $n'' = Mn'$, we can write it uniquely in the form $Mn' = Ln + i$, where $0 \le i \le L-1$. It follows that the downsampled output will be the ith interpolated value arising from the current input $x(n)$ and computed as the output of the ith polyphase subfilter $\mathbf{h}_i$:

$$y(n') = y_{\text{up}}(Mn') = y_{\text{up}}(Ln+i) = y_i(n)$$

The pattern of polyphase indices i that correspond to successive output times n' *repeats* with period L, and depends only on the relative values of L and M. Therefore, for the purpose of deriving a sample processing implementation of the converter, it proves convenient to think in terms of *blocks* of output samples of length-L. The total time duration of such an output block is LT'. Using Eq. (14.6.2), we have:

$$\boxed{T_{\text{block}} = LT' = LMT'' = MT} \tag{14.6.8}$$

Thus, within each output time block there are M input samples, LM high-rate interpolated samples, and L output samples. The M input samples get interpolated into the LM high-rate ones, from which the L output samples are selected.

The computational rate is M times smaller than the polyphase rate Nf_s required for full interpolation. Indeed, we have $2K$ MACs per polyphase filter output and L polyphase outputs in each period T_{block}, that is, $R = 2KL/T_{\text{block}} = N/T_{\text{block}} = N/MT = Nf_s/M$. Equivalently, we have *one* polyphase output in each output period T', $R = 2K/T' = 2Kf_s{}'$. Thus,

$$\boxed{R = \frac{Nf_s}{M} = \frac{Nf_s{}'}{L} = 2Kf_s{}'} \tag{14.6.9}$$

Figure 14.6.2 shows an example with $L = 5$ and $M = 3$, so that $f_s{}' = 5f_s/3$. The interpolating high rate is $f_s{}'' = 5f_s = 3f_s{}'$. The top and bottom figures show the input and output signals and their spectra. The two middle figures show the high-rate interpolated signal, viewed both with respect to the input and output time scales.

Because $f_s{}' > f_s$, the interpolation filter has cutoff $f_s/2$, and acts as an antialiasing prefilter removing the four input replicas up to $f_s{}'' = 5f_s$. The downsampling operation then downshifts the replicas at multiples of $f_s{}'$.

In the time domain, each block period $T_{\text{block}} = 15T'' = 3T = 5T'$ contains three input samples, say $\{x_0, x_1, x_2\}$, five output samples, say $\{y_0, y_1, y_2, y_3, y_4\}$, and 15 interpolated high-rate samples.

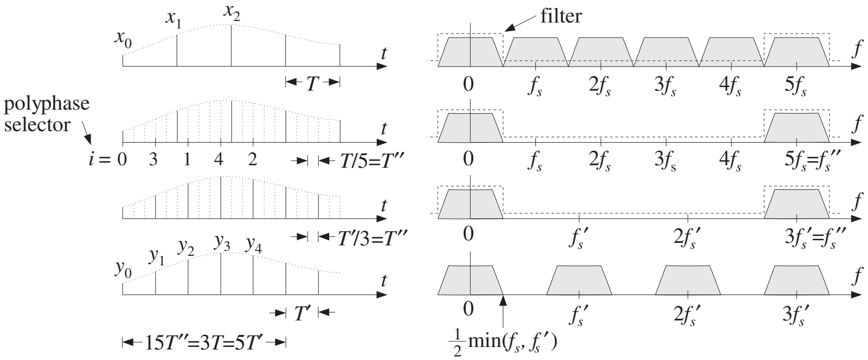


Fig. 14.6.2 Sampling rate conversion by a factor of 5/3.

As can be seen in the figure, the first input period from x_0 to x_1 contains two outputs: y_0, y_1. We have time-aligned the samples so that $y_0 = x_0$. The output y_1 is the third ($i = 3$) interpolated value, and therefore, it is obtained as the output of the polyphase filter $\mathbf{h}_3$ with current input x_0. After this operation, the input sample x_0 is no longer needed and the delay-line $\mathbf{w}$ holding the input samples may be shifted and the next input x_1 read into it.

During the next input period from x_1 to x_2, there are two more outputs: y_2, y_3. The output y_2 is the first ($i = 1$) interpolated value, and therefore, it is the output of the filter $\mathbf{h}_1$, whereas the output y_3 is the fourth ($i = 4$) interpolated value, or the output of $\mathbf{h}_4$. After this operation, the delay-line $\mathbf{w}$ may be updated and x_2 read into it.

Finally, the third input period starting at x_2 contains only one output, namely, y_4, which is the second ($i = 2$) interpolated value, or the output of $\mathbf{h}_2$ with input x_2. After this operation, the delay-line may be shifted and the same computational cycle involving the next three inputs repeated. The above steps may be summarized in the following sample processing algorithm:

```
for each input block $\{x_0, x_1, x_2\}$ do:
    $w_0 = x_0$
        $y_0 = \text{dot}(P, \mathbf{h}_0, \mathbf{w}) = x_0$
        $y_1 = \text{dot}(P, \mathbf{h}_3, \mathbf{w})$
    $\text{delay}(P, \mathbf{w})$
    $w_0 = x_1$
        $y_2 = \text{dot}(P, \mathbf{h}_1, \mathbf{w})$                    (14.6.10)
        $y_3 = \text{dot}(P, \mathbf{h}_4, \mathbf{w})$
    $\text{delay}(P, \mathbf{w})$
    $w_0 = x_2$
        $y_4 = \text{dot}(P, \mathbf{h}_2, \mathbf{w})$
    $\text{delay}(P, \mathbf{w})$
```

The outputs $\{y_0, y_1, y_2, y_3, y_4\}$ were computed by the five polyphase filters $\{\mathbf{h}_0, \mathbf{h}_3,$

$\mathbf{h}_1, \mathbf{h}_4, \mathbf{h}_2\}$ corresponding to the sequence of polyphase indices $i = \{0, 3, 1, 4, 2\}$. The input samples that were used in the computations were $\{x_0, x_0, x_1, x_1, x_2\}$, so that the corresponding index of x_n was $n = \{0, 0, 1, 1, 2\}$. When the index was repeated, the delay line was not updated.

It is easily seen from Fig. 14.6.2 that the patterns of i's and n's get repeated for every group of five outputs. These patterns can be *predetermined* as the solutions of the equations $5n + i = 3m$ for $m = 0, 1, \dots, 4$. In general, we can calculate the patterns by solving the L equations:

$$Ln_m + i_m = Mm, \qquad m = 0, 1, \dots, L-1 \tag{14.6.11}$$

with solution (where % denotes the modulo operation):

```
for  $m = 0, 1, \dots, L-1$ compute:
    $i_m = (Mm)\%L$
    $n_m = (Mm - i_m)/L$
```

(polyphase selectors) (14.6.12)

Assuming that the sequences $\{i_m, n_m\}$, $m = 0, 1, \dots, L-1$, have been *precomputed*, the general sample rate conversion algorithm that transforms each length-M input block $\{x_0, x_1, \dots, x_{M-1}\}$ into a length-L output block $\{y_0, y_1, \dots, y_{L-1}\}$, can be stated as follows:

```
for each input block $\{x_0, x_1, \dots, x_{M-1}\}$ do:
    for  $n = 0, 1, \dots, M-1$ do:
        $w_0 = x_n$
        for  $Ln/M \le m < L(n+1)/M$ do:
            $y_m = \mathrm{dot}(P, \mathbf{h}_{i_m}, \mathbf{w})$
        $\mathrm{delay}(P, \mathbf{w})$
```

(14.6.13)

The inner loop ensures that the output time index m lies between the two input times $Ln \le Mm < L(n+1)$, with respect to the T'' time scale. Because $Mm = Ln_m + i_m$, it follows that such m's will have $n_m = n$. The index i_m serves as a *polyphase filter selector.*

In the special cases of interpolation ($M = 1$), or decimation ($L = 1$), the algorithm reduces to the corresponding sample processing algorithms given in Eqs. (14.2.20) and (14.5.7). For causal processing, the initialization of the algorithm must be as in Eq. (14.2.19) (with K replacing M).

Another example is shown in Fig. 14.6.3 that has $L = 3$, $M = 5$ and decreases the sampling rate by a factor of $3/5$ so that $f_s{}' = 3f_s/5$. The interpolating high rate is now $f_s{}'' = 3f_s = 5f_s{}'$. Because $f_s{}' < f_s$, the filter's cutoff frequency must be $f_c = f_s{}'/2$, and therefore, the filter acts as an antialiasing filter for the downsampler. The filter necessarily chops off those high frequencies from the input that would otherwise be aliased by the downsampling operation, that is, the frequencies in the range $f_s{}'/2 \le f \le f_s/2$.

In the time domain, each block of five input samples $\{x_0, x_1, x_2, x_3, x_4\}$ generates a block of three output samples $\{y_0, y_1, y_2\}$. The solution of Eq. (14.6.12) gives the polyphase selector sequences, for $m = 0, 1, 2$:

$$n_m = \{0, 1, 3\}, \qquad i_m = \{0, 2, 1\}$$

which means that only the inputs $\{x_0, x_1, x_3\}$ will generate interpolated outputs, with the polyphase subfilters $\{\mathbf{h}_0, \mathbf{h}_2, \mathbf{h}_1\}$. The inputs $\{x_2, x_4\}$ will not generate outputs, but still must be shifted into the delay-line buffer. The same conclusions can also be derived by inspecting Fig. 14.6.3. The corresponding sample processing algorithm, which is a special case of Eq. (14.6.13), is:

```
for each input block $\{x_0, x_1, x_2, x_3, x_4\}$ do:
    $w_0 = x_0$
        $y_0 = \mathrm{dot}(P, \mathbf{h}_0, \mathbf{w}) = x_0$
    $\mathrm{delay}(P, \mathbf{w})$
    $w_0 = x_1$
        $y_1 = \mathrm{dot}(P, \mathbf{h}_2, \mathbf{w})$
    $\mathrm{delay}(P, \mathbf{w})$
    $w_0 = x_2$
    $\mathrm{delay}(P, \mathbf{w})$
    $w_0 = x_3$
        $y_2 = \mathrm{dot}(P, \mathbf{h}_1, \mathbf{w})$
    $\mathrm{delay}(P, \mathbf{w})$
    $w_0 = x_4$
    $\mathrm{delay}(P, \mathbf{w})$
```

(14.6.14)

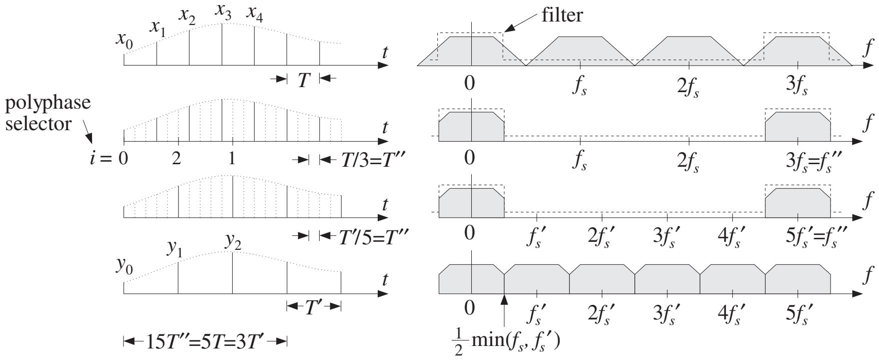


Fig. 14.6.3 Sampling rate conversion by a factor of $3/5$.

The type of converter discussed above is useful when the sampling rates remain *fixed* and synchronous. In some applications, it may be desirable to have *asynchronous* rate changes that can accommodate slowly changing input and output sampling clocks. Such converters must be able to change the rate by *arbitrary* factors, not just rational ones.

Theoretically, the sampling rate can be changed by an arbitrary factor by reconstructing the sampled signal to analog form and then resampling it at the output rate. Digitally, one can use an *extremely large* interpolation factor L to effectively obtain an analog signal and then resample it at the new rate.

The large value of L creates a very dense time grid for the interpolated signal. Every output time instant falls between two such time grid points and the output sample can be chosen to be the nearest of the two interpolated samples, or, it can be formed by linear or higher-order interpolations. References [357–362] discuss practical implementations of this idea.

For example, a recent sample rate conversion chip, designed by Analog Devices for use with digital audio [360–362], uses an interpolation ratio of $L = 2^{16} = 65536$. It can convert sampling rates from 8 to 56 kHz.

The built-in interpolation filter has length $N = 64\times 2^{16} = 2^{22} \simeq 4\times 10^6$ and is realized in its polyphase form. Thus, there are $L = 2^{16}$ polyphase filters of length $2K = 64$. The input delay-line buffer also has length 64. The computational rate of the chip is only $R = (2K)f_s{}' = 64 f_s{}'$ MAC/sec, where $f_s{}'$ is the output rate.

The chip has a polyphase filter selector (like the quantity i_m) that selects the appropriate polyphase filter to use for each output sampling time. To minimize coefficient storage for the 2^{22} filter coefficients, only 1 out of every 128 impulse response coefficients are saved in ROM; the intermediate values are computed when needed by linear interpolation. Thus, the ROM storage is $2^{22}/128 = 32768$ words.

The interpolation filter has a variable cutoff frequency $f_c = \min(f_s/2, f_s{}'/2)$. To avoid having to redesign the filter every time the cutoff changes, the filter is designed once based on a nominal input frequency f_s, such as 44.1 kHz, and then it is "time-stretched" to accommodate the variable cutoff [359,360]. To understand this, we define the scale factor $\rho = \min(1, f_s{}'/f_s)$, such that $\rho < 1$ whenever the output rate is less than the input rate. Then, we may write f_c in the form:

$$f_c = \rho\,\frac{f_s}{2} \quad\Rightarrow\quad \omega_c'' = \frac{2\pi f_c}{f_s{}''} = \rho\,\frac{\pi}{L}$$

If $\rho < 1$, the corresponding ideal impulse response is:

$$d_\rho(k'') = L\,\frac{\sin(\omega_c'' k'')}{\pi k''} = \rho\,\frac{\sin(\pi\rho k''/L)}{\pi\rho k''/L}$$

The fixed filter has response corresponding to $\rho = 1$:

$$d(k'') = \frac{\sin(\pi k''/L)}{\pi k''/L}\,, \quad -LK \le k'' \le LK$$

It follows that $d_\rho(k'')$ will be the "stretched" version of $d(k'')$:

$$d_\rho(k'') = \rho\, d(\rho k'') \tag{14.6.15}$$

The effective length of this filter must also stretch commensurately. Indeed, because the argument $\rho k''$ must lie in the designed index range of the original filter, that is, $-LK \le \rho k'' \le LK$, we must have in Eq. (14.6.15):

$$-\frac{1}{\rho}LK \le k'' \le \frac{1}{\rho}LK$$

Thus, K increases to $K_\rho = K/\rho$, and the effective length of the filter becomes $N_\rho = 2LK_\rho = 2LK/\rho = N/\rho$. The length of the tapped delay line also becomes longer, $2K_\rho = 2K/\rho$. Because the coefficients $d(k'')$ are stored in ROM only for integer values of k'', the argument of $d(\rho k'')$ must be rounded to the nearest integer. Because of the highly oversampled nature of $d(k'')$, this rounding causes only a small distortion [360].

14.7 Noise Shaping Quantizers

The main purpose of noise shaping is to reshape the spectrum of quantization noise so that most of the noise is filtered out of the relevant frequency band, such as the audio band. Noise shaping is used in four major applications:

- Oversampled *delta-sigma* A/D converters.
- Oversampled *requantizers* for D/A conversion.
- Non-oversampled dithered noise shaping for requantization.
- Non-oversampled roundoff noise shaping in digital filters.

In the oversampled cases, the main objective is to trade off bits for samples, that is, increasing the sampling rate but reducing the number of bits per sample. The resulting increase in quantization noise is compensated by a noise shaping quantizer that pushes the added noise out of the relevant frequency band in such a way as to preserve a desired level of signal quality. The reduction in the number of bits simplifies the structure of the A/D and D/A converters. See [350,351] for a review and earlier references.

In the non-oversampled cases, one objective is to minimize the accumulation of roundoff noise in digital filter structures [86–92]. Another objective is to reduce the number of bits without reducing quality. For example, in a digital audio recording and mixing system where all the digital processing is done with 20 bits, the resulting audio signal must be rounded eventually to 16 bits in order to place it on a CD. The rounding operation can cause unwanted granulation distortions. Adding a dither signal helps remove such distortions and makes the quantization noise sound like steady background white noise. However, further noise shaping can make this white noise even more inaudible by concentrating it onto spectral bands where the ear is least sensitive [75–83].

A related application in digital audio is to actually keep the bits saved from noise shaping and use them to carry extra data on a conventional CD, such as compressed images, speech, or text, and other information [84,85]. This "buried" data channel is encoded to look like pseudorandom dither which is then added (subtractively) to the CD data and subjected to noise shaping. As many as 4 bits from each 16-bit CD word may be dedicated to such hidden data without sacrificing the quality of the CD material. The resulting data rates are $4\times 44.1 = 176.4$ kbits/sec or double that for two stereo channels.

In Section 2.2, we introduced noise shaping quantizers and discussed some of their implications, such as the tradeoff between oversampling ratio and number of bits, but did not discuss how they are constructed.

Figure 14.7.1 shows a typical oversampled first-order *delta-sigma* A/D converter system.[†] The analog input is assumed to have been prefiltered by an antialiasing prefilter whose structure is simplified because of oversampling. The relevant frequency range of the input is the low-rate Nyquist interval $f_s/2$. Such converters are commonly used in oversampling DSP systems, shown in Figs. 2.2.5 and 14.5.5.

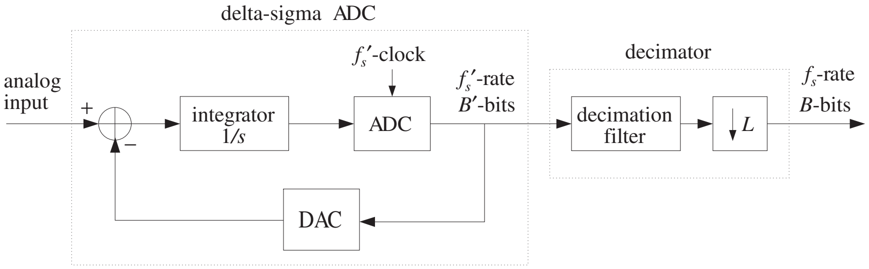


Fig. 14.7.1 Oversampled first-order delta-sigma A/D converter.

The analog part of the converter contains an ordinary A/D converter operating at the fast rate $f_s{}' = Lf_s$ and having a small number of bits, say B' bits. The most useful practical choice is $B' = 1$, that is, a two-level ADC. The output of the ADC is reconstructed back into analog form by the DAC (i.e., a two-level analog signal, if $B' = 1$) and subtracted from the input.

The difference signal (the "delta" part) is accumulated into the integrator (the "sigma" part) and provides a local average of the input. The feedback loop causes the quantization noise generated by the ADC to be *highpass* filtered, pushing its energy towards the higher frequencies (i.e., $f_s{}'/2$) and away from the signal band.

The digital part of the converter contains an L-fold decimator that reduces the sampling rate down to f_s and increases the number of bits up to a desired resolution, say B bits, where $B > B'$. In practice, the analog and digital parts reside usually on board the same chip.

The lowpass decimation filter does three jobs: (1) It removes the high-frequency quantization noise that was introduced by the feedback loop, (2) it removes any undesired frequency components beyond $f_s/2$ that were not removed by the simple analog prefilter, and (3) through its filtering operation, it increases the number of bits by linearly combining the coarsely quantized input samples with its coefficients, which are taken to have enough bits.

To see the filtering action of the feedback loop on the input and quantization noise, we consider a sampled-data equivalent model of the delta-sigma quantizer, shown in Fig. 14.7.2. The time samples, at rate $f_s{}'$, are denoted by $x'(n')$ in accordance with our notation in this chapter.

The ADC is replaced by its equivalent additive-noise model of Fig. 2.1.3 and the integrator by a discrete-time accumulator $H(\zeta)$ with transfer function:

[†] Also called a sigma-delta converter or a feedback quantizer.

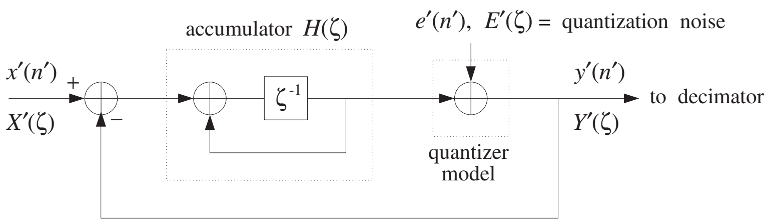


Fig. 14.7.2 Discrete-time model of first-order delta-sigma quantizer.

$$H(\zeta) = \frac{\zeta^{-1}}{1-\zeta^{-1}} \tag{14.7.1}$$

where ζ^{-1} denotes a high-rate unit delay. The numerator delay ζ^{-1} is necessary to make the feedback loop computable.

Working with ζ-transforms, we note that the input to $H(\zeta)$ is the difference signal $X'(\zeta) - Y'(\zeta)$. Its output is added to $E'(\zeta)$ to generate $Y'(\zeta)$. Thus,

$$H(\zeta)\big(X'(\zeta) - Y'(\zeta)\big) + E'(\zeta) = Y'(\zeta)$$

which may be solved for $Y'(\zeta)$ in terms of the two inputs $X'(\zeta)$ and $E'(\zeta)$:

$$Y'(\zeta) = \frac{H(\zeta)}{1+H(\zeta)}\,X'(\zeta) + \frac{1}{1+H(\zeta)}\,E'(\zeta) \tag{14.7.2}$$

It can be written in the form:

$$Y'(\zeta) = H_x(\zeta)X'(\zeta) + H_{\text{NS}}(\zeta)E'(\zeta) \tag{14.7.3}$$

where the noise shaping transfer function $H_{\text{NS}}(\zeta)$ and the transfer function for the input $H_x(\zeta)$ are defined as:

$$H_x(\zeta) = \frac{H(\zeta)}{1+H(\zeta)}\,, \qquad H_{\text{NS}}(\zeta) = \frac{1}{1+H(\zeta)} \tag{14.7.4}$$

Inserting $H(\zeta)$ from Eq. (14.7.1), we find for the first-order case:

$$\boxed{H_x(\zeta) = \zeta^{-1}, \qquad H_{\text{NS}}(\zeta) = 1 - \zeta^{-1}} \tag{14.7.5}$$

Thus, $H_{\text{NS}}(\zeta)$ is a simple highpass filter, and $H_x(\zeta)$ an allpass plain delay. The I/O equation (14.7.3) becomes:

$$Y'(\zeta) = \zeta^{-1}X'(\zeta) + (1-\zeta^{-1})E'(\zeta) \tag{14.7.6}$$

or, in the time domain:

$$\boxed{y'(n') = x'(n'-1) + \varepsilon(n')} \tag{14.7.7}$$

where we defined the filtered quantization noise:

$$\varepsilon(n') = e'(n') - e'(n'-1) \quad \Leftrightarrow \quad \mathcal{E}(\zeta) = (1-\zeta^{-1})E'(\zeta) \tag{14.7.8}$$

Thus, the quantized output $y'(n')$ is the (delayed) input plus the filtered quantization noise. Because the noise is highpass filtered, further processing of $y'(n')$ by the lowpass decimation filter will tend to average out the noise to zero and also replace the input by its locally averaged, decimated, value. A typical example of a decimator is the hold decimator of Eq. (14.5.9), which averages L successive high-rate samples.

By comparison, had we used a conventional B-bit ADC and sampled the input at the low rate f_s, the corresponding quantized output would be:

$$\boxed{y(n) = x(n) + e(n)} \tag{14.7.9}$$

where $e(n)$ is modeled as white noise over $[-f_s/2, f_s/2]$.

The “design” condition that renders the quality of the two quantizing systems equivalent and determines the tradeoff between oversampling ratio L and savings in bits, is to require that the rms quantization errors of Eqs. (14.7.7) and (14.7.9) be the *same* over the desired frequency band $[-f_s/2, f_s/2]$. As we saw in Section 2.2, the mean-square errors are obtained by integrating the power spectral densities of the noise signals over that frequency interval, yielding the condition:

$$\boxed{\sigma_e^2 = \sigma_{e'}^2 \frac{1}{f_s{}'} \int_{-f_s/2}^{f_s/2} |H_{\text{NS}}(f)|^2 \, df} \tag{14.7.10}$$

Setting $f_s{}' = Lf_s$ and $\sigma_e/\sigma_{e'} = 2^{-B}/2^{-B'} = 2^{-\Delta B}$, where $\Delta B = B - B'$, we obtain the desired relationship between L and ΔB given by Eq. (2.2.10).

Higher-order delta-sigma quantizers have highpass noise shaping transfer functions of the form:

$$H_{\text{NS}}(\zeta) = (1-\zeta^{-1})^p \tag{14.7.11}$$

where p is the order. The input/output equations for such quantizers are still of the form of Eq. (14.7.3), where $H_x(\zeta)$ is typically a multiple delay. The frequency and magnitude responses of $H_{\text{NS}}(\zeta)$ are obtained by setting $\zeta = e^{2\pi jf/f_s{}'}$:

$$H_{\text{NS}}(f) = \left(1 - e^{-2\pi jf/f_s{}'}\right)^p, \qquad |H_{\text{NS}}(f)|^2 = \left| 2\sin\left(\frac{\pi f}{f_s{}'}\right)\right|^{2p} \tag{14.7.12}$$

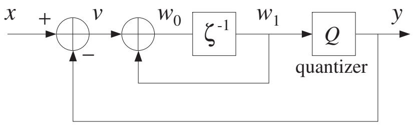
resulting in the expressions used in Eq. (2.2.8).

There exist many architectures for higher-order delta-sigma quantizers that address various circuit limitations and limit-cycle instability problems [351,364–368]. Some examples of such architectures are given in the problems.

Example 14.7.1: To illustrate the time-domain operation of a delta-sigma quantizer, consider the common 1-bit case that has a two-level ADC. Let $Q(x)$ denote the two-level quantization function defined by:

$$Q(x) = \text{sign}(x) = \begin{cases} +1, & \text{if } x \geq 0 \\ -1, & \text{if } x < 0 \end{cases} \tag{14.7.13}$$

The corresponding block diagram of the quantizer is shown below, together with the computational sample processing algorithm. The quantity w_1 is the content of the accumulator's delay:



```
for each input x do:
    $y = Q(w_1)$
    $v = x - y$
    $w_0 = w_1 + v$
    $w_1 = w_0$
```

The following table shows the computed outputs for the two constant inputs, $x = 0.4$ and $x = -0.2$, with the algorithm iterated ten times:

x	w_1	y	v	w_0
0.4	0.0	1.0	−0.6	−0.6
0.4	−0.6	−1.0	1.4	0.8
0.4	0.8	1.0	−0.6	0.2
0.4	0.2	1.0	−0.6	−0.4
0.4	−0.4	−1.0	1.4	1.0
0.4	1.0	1.0	−0.6	0.4
0.4	0.4	1.0	−0.6	−0.2
0.4	−0.2	−1.0	1.4	1.2
0.4	1.2	1.0	−0.6	0.6
0.4	0.6	1.0	−0.6	0.0

x	w_1	y	v	w_0
−0.2	0.0	1.0	−1.2	−1.2
−0.2	−1.2	−1.0	0.8	−0.4
−0.2	−0.4	−1.0	0.8	0.4
−0.2	0.4	1.0	−1.2	−0.8
−0.2	−0.8	−1.0	0.8	0.0
−0.2	0.0	1.0	−1.2	−1.2
−0.2	−1.2	−1.0	0.8	−0.4
−0.2	−0.4	−1.0	0.8	0.4
−0.2	0.4	1.0	−1.2	−0.8
−0.2	−0.8	−1.0	0.8	0.0

The average of the ten successive values of y are in the two cases, $\bar{y} = 0.4$ and $\bar{y} = -0.2$. Such averaging would take place in the decimator, for example, using a 10-fold hold decimator of the form of Eq. (14.5.9). □

Example 14.7.2: To illustrate the capability of a delta-sigma quantizer/decimator system to accurately sample an analog signal, consider the first-order quantizer of the previous example, but with a time-varying input defined with respect to the fast time scale as:

$$x'(n') = 0.5\sin(2\pi f_0 n'/f_s{}'), \qquad n' = 0, 1, \dots, N_{\text{tot}} - 1$$

We choose the values $f_0 = 8.82$ kHz, $f_s = 44.1$ kHz, $L = 10$, and $N_{\text{tot}} = 200$ samples. The fast rate is $f_s{}' = 10 \times 44.1 = 441$ kHz, and the normalized frequency $f_0/f_s{}' = 0.02$.

We want to see how the two-level quantized output $y'(n')$ of the delta-sigma quantizer is filtered by the decimation filter to effectively recover the input (and resample it at the lower rate). We compare three different decimation filters, whose frequency responses are shown in Fig. 14.7.3, with magnified passbands on the right.

The first one is an L-fold averaging decimator with transfer function given by Eq. (14.5.8). The other two are designed by the window method, and have impulse responses:

$$h(n') = w(n')\frac{\sin\big(\pi(n'-LM)/L\big)}{\pi(n'-LM)}, \qquad n' = 0, 1, \dots, N-1$$

where $N = 2LM + 1$. One has the minimum possible length, that is, $N = 2LM + 1$, with $M = 1$, giving $N = 21$, and uses a rectangular window, $w(n') = 1$. The other one is

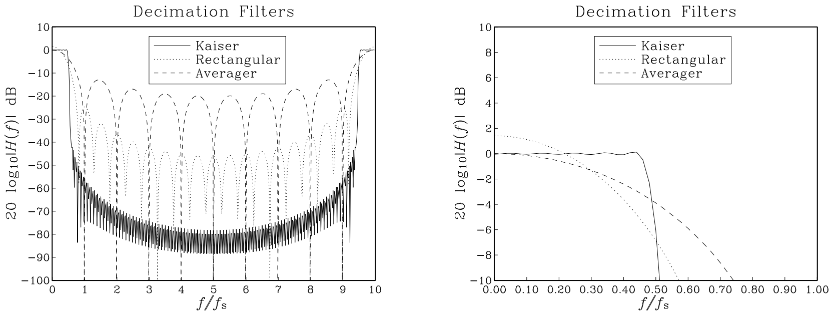


Fig. 14.7.3 Magnitude responses of decimation filters.

designed by the Kaiser method using a stopband attenuation of $A = 35$ dB and transition width $\Delta f = 4.41$ kHz, or $\Delta f/f_s = 0.1$ (about the cutoff frequency $f_c = f_s/2 = 22.05$ kHz). It has length $N = 201$, $M = 10$, and Kaiser parameters $D = 1.88$ and $\alpha = 2.78$.

The output of the quantizer $y'(n')$, which is the input to the three decimators, is shown on the left of Fig. 14.7.4; the output of the averaging decimator is on the right. The outputs of the rectangular and Kaiser decimators are shown in Fig. 14.7.5.

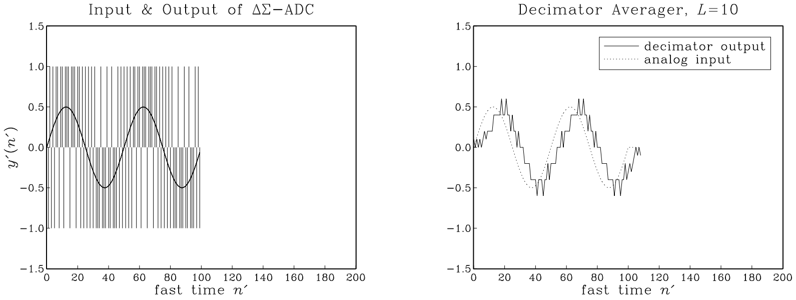


Fig. 14.7.4 Delta-sigma quantizer output and averaging decimator's output.

The averager recovers the input sinusoid only approximately and with a delay of $(L-1)/2 = 4.5$. Some of the high frequencies in $y'(n')$ get through, because they cannot be completely removed by the filter. This can be seen from the decimator's frequency response, shown in Fig. 14.7.3,

$$|H(f)| = \left|\frac{\sin(\pi f/f_s)}{L\sin(\pi f/10f_s)}\right|$$

which does not vanish everywhere between $[f_s/2, 10f_s - f_s/2]$, although it does vanish at the multiples mf_s, $m = 1, 2, \dots, 9$.

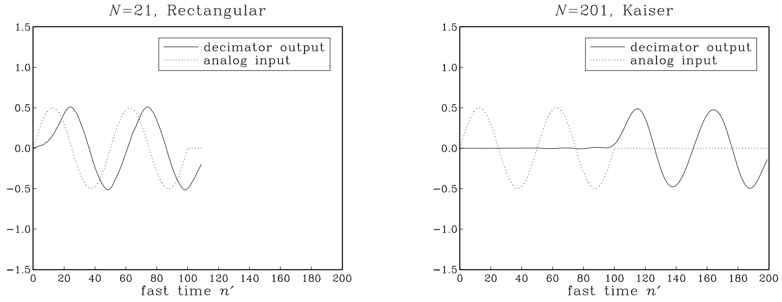


Fig. 14.7.5 Decimator filter output for rectangular and Kaiser designs.

The outputs of the window designs are faithful representations of the input sinusoid, up to the filter delay of LM samples, that is, $LM = 10$ and $LM = 100$, respectively. The Kaiser decimator gives the best output because it acts as a better lowpass filter.

What is being plotted in these graphs is the output of the decimation filter *before* it is downsampled by a factor of $L = 10$. The downsampled signal is extracted by taking every tenth output. The nine intermediate samples which are to be discarded need not be computed. However, we did compute them here for plotting purposes.

We chose simple specifications for our designs in order to get small values for the filter delays LM. In practice, stricter specifications can result in long filter lengths, for example, for a third-order noise shaper to give CD quality audio, we need $L = 64$ (see Table 2.2.1) which would require $N = DLf_s/\Delta f \simeq 4100$ for $A = 100$ dB and $\Delta f = 0.1f_s$. In such cases, a practical approach is to use *multistage* decimators. □

Next, we discuss oversampled noise shaping *requantizers* for D/A conversion. A typical requantizer system is shown in Fig. 14.7.6. The digital input is incoming at rate f_s and B-bits per sample. It is upsampled and interpolated by an L-fold interpolator, which increases the rate to $f_s{}'$. The noise shaping requantizer reduces the number of bits to $B' < B$. This output is, then, fed into an ordinary B'-bit DAC, followed by an anti-image postfilter (whose structure is greatly simplified because of oversampling).

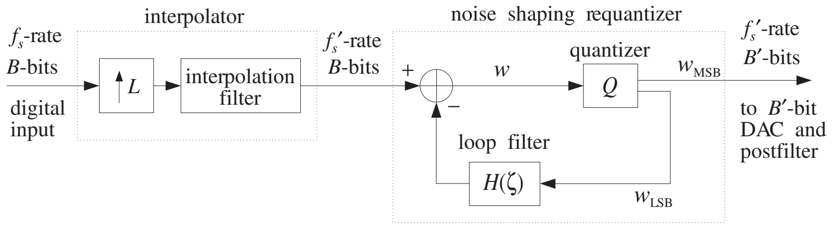


Fig. 14.7.6 Oversampled noise shaping requantizer for D/A conversion.

The quantizer Q rounds the incoming B-bit word w by keeping the B' most significant bits, say w_{MSB}, which become the output, $y = w_{MSB}$. The requantization error, that is, the $B - B'$ least significant bits of w, $w_{\text{LSB}} = w - w_{\text{MSB}}$, are fed back through a loop filter and subtracted from the input.

The feedback loop causes the quantization noise to be highpass filtered, reducing its power within the input's baseband by just the right amount to counteract the increase in noise caused by the reduction in bits.

Figure 14.7.7 shows a model of the requantizer in which the quantizer Q is replaced by its equivalent noise model and the difference of the signals around the quantizer generates the LSB signal and feeds it back.

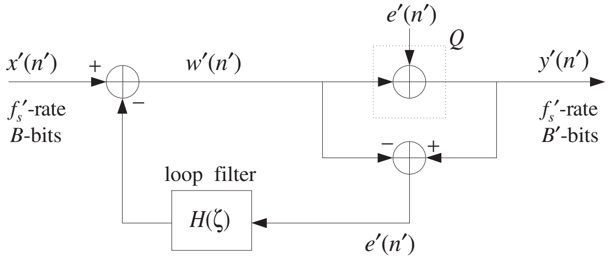


Fig. 14.7.7 Noise shaping requantizer model.

The quantized output is $y'(n') = w'(n') + e'(n')$, so that $y'(n') - w'(n') = e'(n')$. Therefore, the input to the loop filter is $e'(n')$ itself. In the ζ-domain, we have:

$$Y'(\zeta) = W'(\zeta) + E'(\zeta) \quad \text{and} \quad W'(\zeta) = X'(\zeta) - H(\zeta)E'(\zeta)$$

which gives the I/O equation:

$$Y'(\zeta) = X'(\zeta) + \big(1 - H(\zeta)\big)E'(\zeta) = X'(\zeta) + H_{\text{NS}}(\zeta)E'(\zeta) \tag{14.7.14}$$

Thus, the effective noise shaping filter is

$$H_{\text{NS}}(\zeta) = 1 - H(\zeta) \tag{14.7.15}$$

First-, second-, or higher-order filters $H_{\text{NS}}(\zeta)$ can be constructed easily by choosing the loop filter as $H(\zeta) = 1 - H_{\text{NS}}(\zeta)$, for example:

$$\begin{aligned} H(\zeta) &= \zeta^{-1} \\ H(\zeta) &= 2\zeta^{-1} - \zeta^{-2} \end{aligned} \quad \Rightarrow \quad \begin{aligned} H_{\text{NS}}(\zeta) &= (1 - \zeta^{-1}) \\ H_{\text{NS}}(\zeta) &= (1 - \zeta^{-1})^2 \end{aligned}$$

Noise shaping requantizers are based on the same principle of feedback quantization as delta-sigma A/D converters. Therefore, the tradeoff between L and ΔB remains the same. They are used routinely in the playback systems of CD players, DATs, and speech CODECs. For example, the first CD player built by Philips employed a first-order requantizer with $H(\zeta) = \zeta^{-1}$ and a 4-times oversampling interpolator [353].

14.8 Problems

14.1 Consider the 4-fold, length-17 interpolator defined in Eq. (14.4.1). Write down the low-rate transfer functions $D_i(z)$, $i = 0, 1, 2, 3$ and their causal versions $H_i(z)$, corresponding to the polyphase subfilters of Eq. (14.4.2).

Then, replace $z = \zeta^4$ and verify explicitly that the high-rate overall transfer function of the sequence **d** of Eq. (14.4.1) is given by the polyphase decomposition Eq. (14.2.15):

$$D(\zeta) = D_0(\zeta^4) + \zeta^{-1}D_1(\zeta^4) + \zeta^{-2}D_2(\zeta^4) + \zeta^{-3}D_3(\zeta^4)$$

14.2 Design a 2-fold interpolator of length $N = 9$, using a rectangular window. Show that the polyphase form of the interpolator is:

$$\begin{bmatrix} y_{\text{up}}(2n) \\ y_{\text{up}}(2n+1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ -0.21 & 0.64 & 0.64 & -0.21 \end{bmatrix} \begin{bmatrix} x_{\text{up}}(2n+4) \\ x_{\text{up}}(2n+2) \\ x_{\text{up}}(2n) \\ x_{\text{up}}(2n-2) \end{bmatrix}$$

By superimposing impulse responses, give a graphical interpretation of the above result using the LTI form of convolution, as was done in Fig. 14.4.2.

14.3 Design a 3-fold interpolator of length $N = 13$, using a rectangular and a Hamming window. Show that the polyphase form of the interpolator in the rectangular case is:

$$\begin{bmatrix} y_{\text{up}}(3n) \\ y_{\text{up}}(3n+1) \\ y_{\text{up}}(3n+2) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ -0.17 & 0.41 & 0.83 & -0.21 \\ -0.21 & 0.83 & 0.41 & -0.17 \end{bmatrix} \begin{bmatrix} x_{\text{up}}(3n+6) \\ x_{\text{up}}(3n+3) \\ x_{\text{up}}(3n) \\ x_{\text{up}}(3n-3) \end{bmatrix}$$

Determine a similar expression for the Hamming case. For the rectangular case, give a graphical interpretation of the above result using the LTI form of convolution, as in Fig. 14.4.2.

14.4 Using the LTI form of convolution, that is, superimposing impulse responses, justify the interpolation equations (14.1.3) of a length-25 rectangularly windowed ideal interpolator. Then, rewrite them in the form of Eq. (14.4.3) using the appropriate 4×6 coefficient matrix on the right.

14.5 Design a 3-fold FIR interpolation filter that uses *at most* four low-rate samples to compute the interpolated values between $x(n)$ and $x(n+1)$, that is,

$$y_{\text{up}}(3n+i) = a_i x(n+2) + b_i x(n+1) + c_i x(n) + d_i x(n-1)$$

for $i = 0, 1, 2$. Determine the values of the coefficients $\{a_i, b_i, c_i, d_i\}$, $i = 0, 1, 2, 3$, for the two cases:

a. When the filter is an *ideal* interpolator.

b. When the filter is a *linear* interpolator.

14.6 *Computer Experiment: Interpolation Filter Design.* Consider the following triangular and sinusoidal low-rate signals:

$$\begin{aligned} x(n) &= \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0\} \\ x(n) &= \sin(2\pi F_0 n), \qquad n = 0, 1, \dots, 24 \end{aligned}$$

where $F_0 = 0.04$ cycles per sample. Design a length-17 4-fold interpolation filter using a rectangular window, as in Section 14.4.1. Using the polyphase form implemented by the circular buffer version of the sample processing algorithm (14.2.20) and initialized by Eq. (14.2.19), process the above signals to get the interpolated signals $y_{\rm up}(n')$, $n' = 0, 1, \dots, 99$, and plot them versus the fast time n'.

Repeat by designing the corresponding Hamming windowed interpolation filter and filtering the two signals $x(n)$ through it.

14.7 *Computer Experiment: Multistage* $8\times$ *Interpolation Filter Design.* Design a multistage 8-times oversampling interpolation filter for digital audio applications (see [356] for a comparable design). The sampling rate, transition width, and stopband attenuation for all stages are taken to be $f_s = 40$ kHz, $\Delta f = 5$ kHz, $A = 80$ dB. There are three possible multistage designs, as shown in Fig. 14.2.10:

$$2\times 4 = 4\times 2 = 2\times 2\times 2 = 8$$

a. For each possibility, use the Kaiser method to determine the filter lengths N_0, N_1, (and N_2 for the 3-stage case). Determine also the length N of a single-stage design with the same specifications.

b. Compute the frequency responses of each stage $H_0(f)$, $H_1(f)$, (and $H_2(f)$ in the 3-stage case) and plot their magnitudes in dB and on the same graph over the range $0 \le f \le 320$ kHz. Normalize them to 0 dB at DC. Plot also the total response of the stages, that is, $H_{\rm tot}(f) = H_0(f)H_1(f)$, (or, $H_0(f)H_1(f)H_2(f)$ in the 3-stage case), and compare it with the response $H(f)$ of the single-stage design.

Note that in order to keep the overall stopband attenuation in $H_{\rm tot}(f)$ below 80 dB, you may have to increase slightly the value of A that you put into the design equations for some of the stages, for example, $A = 84$ dB.

c. Assuming all filters are realized in their polyphase form, calculate the relative computational cost $R_{\rm multi}/R_{\rm single}$ and its approximation using Eq. (14.2.34). Which of the three possibilities is the most efficient?

14.8 It is desired to design a $4\times$ oversampling digital FIR interpolation filter for a CD player. Assume the following specifications: audio sampling rate of 44.1 kHz, passband range $[0, 20]$ kHz, stopband range $[24.1, 88.2]$ kHz, and stopband attenuation of 80 dB.

Using the Kaiser window design method, determine the *filter length* and the total *computational rate* in MAC/sec for the following cases:

a. Single-stage design implemented in its polyphase form.

b. Two-stage (2×2) design implemented in its polyphase form. What are the design specifications of the two stages?

Draw a sketch of the magnitude responses of the designed filters versus frequency in the range $0 \le f \le 176.4$ kHz, and of the two individual filter responses in the two-stage design case. What are the computational savings of design (b) versus design (a)? Can a 20 MIPS DSP chip handle the computational rates?

14.9 *Computer Experiment: Bessel Postfilters.* Bessel analog filters have almost linear phase response *within* their passband. Consider the Butterworth and Bessel filters designed in Section 14.4.4, and given by Eqs. (14.4.12) and (14.4.14). Compute and on the same graph plot their phase response over the passband interval $0 \le f \le 20$ kHz. On a separate graph, plot their phase response over the range $0 \le f \le 160$ kHz. Moreover, plot their magnitude response in dB over the same range.

14.10 Consider a three-stage interpolator H_0, H_1, H_2 with oversampling factors L_0, L_1, L_2 respectively, so that the total interpolation factor is $L = L_0L_1L_2$. The filter H_0 is a very sharp lowpass filter designed by some method, such as Kaiser's. The filter H_1 is a *linear* interpolator, and H_2 a *hold* interpolator. The output of H_2 is fed into a noise shaping requantizer to reduce the number of bits and then fed at rate Lf_s into a staircase DAC, $H_{\rm dac}$, and then into a final analog postfilter $H_{\rm post}$. Such a system is used, for example, in the Philips Bitstream 1-bit DAC system for CD players [355], with $L_0 = 4$, $L_1 = 32$, $L_2 = 2$.

a. Write expressions for the magnitude responses $|H_1(f)|$, $|H_2(f)|$, $|H_{\rm dac}(f)|$, in terms of f and L_0, L_1, L_2.

b. Using part (a), show that the combined effect of the hold interpolator H_2 followed by the DAC is *equivalent* to a staircase DAC operating at the *reduced* sampling rate $L_0L_1f_s$.

Why, then, do we need the hold interpolator at all? Why not use only a two-stage interpolator and an oversampling factor of L_0L_1?

c. Consider the special case $L_0 = 4$, $L_1 = 2$, $L_2 = 2$. On the same graph, sketch roughly over the frequency range $0 \le f \le 16f_s$, the spectra at the input and output of H_0, at the output of H_1, at the output of H_2, at the output $H_{\rm dac}$, at the output of $H_{\rm post}$. What transition width did you choose for $H_{\rm post}$?

d. Sketch the *time-domain* signals at the input and output of H_2 and the output of $H_{\rm dac}$. Does that explain part (b) in the time domain?

14.11 Show that the ideal L-fold interpolation filter $D(f)$, defined in Eq. (14.2.24) over the high-rate Nyquist interval $[-f_s{}'/2, f_s{}'/2]$ and shown in Fig. 14.2.3, satisfies the replication property:

$$\frac{1}{L}\sum_{m=0}^{L-1} D(f - mf_s) = 1$$

for all f, where f_s is the low rate $f_s = f_s{}'/L$.

14.12 Consider the sampling of an analog signal $x_a(t)$ at the two sampling rates f_s and $f_s{}' = Lf_s$. The corresponding signal samples are $x(n) = x_a(nT)$ and $x'(n') = x_a(n'T')$. Because $T = LT'$, it follows that $x(n)$ will be the *downsampled* version of $x'(n')$ in the sense of Eq. (14.5.1), that is, $x(n) = x_a(nT) = x_a(nLT') = x'(nL)$. The spectra of $x(n)$ and $x'(n')$ are given by the Poisson summation formulas:

$$X(f) = \frac{1}{T}\sum_{k=-\infty}^{\infty} X_a(f - kf_s), \qquad X'(f) = \frac{1}{T'}\sum_{k'=-\infty}^{\infty} X_a(f - k'f_s{}')$$

Using the change of variables $k = k'L + m$, where $m = 0, 1, \dots, L-1$, show that the spectrum of the downsampled signal is given by the *discrete-time* version of the Poisson summation formula:

$$X(f) = \frac{1}{L}\sum_{m=0}^{L-1} X'(f - mf_s) \tag{14.8.1}$$

Why is the factor L needed? Show that the same equation can be expressed in terms of the normalized digital frequencies $\omega = 2\pi f/f_s$ and $\omega' = 2\pi f/f_s{}'$ as

$$X(\omega) = \frac{1}{L}\sum_{m=0}^{L-1} X'\big(\omega' - \frac{2\pi m}{L}\big) \tag{14.8.2}$$

14.13 The downsampled signal $x(n)$, defined in Eq. (14.5.1), can be thought of as re-sampling of $x'(n')$. More precisely, the upsampled version of the downsampled signal $x(n)$, that is, the samples $x(n)$ with $L-1$ zeros inserted between them, can be thought of as the multiplication of $x'(n')$ by a discrete-time sampling function:

$$x_{\rm up}(n') = \sum_{n=-\infty}^{\infty} x'(nL)\delta(n'-nL) = s'(n')x'(n'), \quad \text{where} \quad s'(n') = \sum_{n=-\infty}^{\infty}\delta(n'-nL)$$

First, show that $s'(n')$, being periodic in n' with period L, can be written in terms of the following discrete Fourier series, which is essentially an L-point inverse DFT:

$$s'(n') = \sum_{n=-\infty}^{\infty}\delta(n'-nL) = \frac{1}{L}\sum_{m=0}^{L-1} e^{2\pi jmn'/L} \tag{14.8.3}$$

Then, prove the downsampling property Eq. (14.8.2) using the representation Eq. (14.8.3).

14.14 Prove the downsampling equation (14.8.1) by using the property $X'(f) = D(f)X(f)$ where $D(f)$ is the ideal interpolator defined by Eq. (14.2.24), and using the results of Problem 14.11. Why can't we write $X(f) = X'(f)/D(f)$?

14.15 Consider a third-order analog Butterworth antialiasing prefilter that precedes an L-fold decimator. The passband attenuation is required to be less than 0.1 dB. Show that the *minimum* oversampling ratio L that must be used in order for the prefilter to suppress the spectral images by at least $A_{\rm stop}$ dB is given approximately by:

$$L = 0.94\cdot 10^{A_{\rm stop}/60} + 0.5$$

Make a plot of the above formula versus $A_{\rm stop}$ in the range $20 < A_{\rm stop} < 100$ dB.

14.16 Show that the order N of an analog Butterworth antialiasing prefilter to be used in conjunction with an L-fold decimator and designed with specifications $\{A_{\rm pass}, A_{\rm stop}, f_{\rm pass}, f_{\rm stop}\}$, as shown in Fig. 14.5.4, is given by:

$$N = \frac{\ln\left(\dfrac{10^{A_{\rm stop}/10}-1}{10^{A_{\rm pass}/10}-1}\right)}{2\ln(2L-1)}$$

Determine N for the values $A_{\rm pass} = 0.1$ dB, $A_{\rm stop} = 60$ dB, $L = 16$. Round N up to the next integer, say N_0. For what range of Ls does the filter order remain fixed at N_0?

14.17 Using the Kaiser window method, design a sample rate converter for up-converting CD audio at 44.1 kHz to DAT audio at 48 kHz. The required ratio is $L/M = 160/147$. Assume a transition region of $[20, 24.41]$ kHz and stopband attenuation of 95 dB.

What is the filter length N? What is the computational cost in MAC/sec assuming a polyphase realization? Can a modern DSP chip handle this cost? What are the memory requirements for such a converter?

14.18 A DAT-recorded digital audio signal is to be broadcast digitally. Using the Kaiser method, design a sampling rate converter filter for down-converting the 48 kHz DAT rate to a 32 kHz broadcast rate.

What is the filter's cutoff frequency? Assume reasonable values for the filter's transition width and stopband attenuation. What is the filter length N? What is the computational cost in MAC/sec assuming a polyphase realization? Write explicitly (i.e., in the form of Eq. (14.6.10)) the sample processing algorithm implementing the conversion algorithm.

14.19 Consider two sample rate converters for converting by the ratios 7/4 and 4/7. For each case, sketch figures similar to Figs. 14.6.2 and 14.6.3 showing the conversion stages in the time and frequency domains. For both cases, determine the polyphase filter selection indices i_m, n_m, and write explicitly (i.e., in the form of Eq. (14.6.10)) the corresponding sample processing algorithms implementing the conversion process.

14.20 Show that the time-stretching property given in Eq. (14.6.15) is preserved if the impulse response is windowed by a Hamming or Kaiser window (or, any other window).

14.21 *Computer Experiment: Sample Rate Conversion.* Write a general C or MATLAB program that implements sample rate conversion (SRC) by a factor L/M.

The SRC filter may be designed by the Kaiser method. The program must have as inputs the parameters L, M, stopband attenuation A, and normalized transition width $\Delta F = \Delta f/f_s$, where f_s is the input rate. Then, it must process an arbitrary file or array of input-rate data and output a file or array of output-rate data.

The program must initialize the $(P+1)$-dimensional state vector $\mathbf{w}$ correctly by reading in the first K input samples, as in Eq. (14.2.19). Then, it must continue processing input samples via the sample processing algorithm of Eq. (14.6.13), until the last input sample. Finally, it must calculate an additional K input-off transients to compensate for the initial delay. [*Hint:* You need to call Eq. (14.6.13) approximately K/M more times with zero input.]

As a filtering example, consider a 10 msec portion of a 100 Hz sinusoid, that is, $x(t) = \sin(2\pi t/10)$, where $0 \le t \le 10$ msec. Show that if this signal is sampled at 3 kHz, at 5 kHz, or at 15 kHz, its samples will be given respectively by:

$$\begin{aligned} x(n) &= \sin(2\pi n/30), & n &= 0,1,\dots,29 \\ x'(n') &= \sin(2\pi n'/50), & n' &= 0,1,\dots,49 \\ x''(n'') &= \sin(2\pi n''/150), & n'' &= 0,1,\dots,149 \end{aligned}$$

Design a 5/3 converter that has $A = 30$ dB and $\Delta F = 0.1$. Filter the signal $x(n)$ through the SRC filter to generate the output $y(n')$. Compare the digitally resampled signal $y(n')$ with the analog resampled signal $x'(n')$.

To compare $x(n)$ with $y(n')$, you must work with respect to the same time scale. That is, upsample the input $x(n)$ by a factor of $L = 5$ and the output $y(n')$ by a factor of $M = 3$. Then, plot the upsampled signals versus the fast time n'' and compare them.

A typical output is shown in Fig. 14.8.1. Within each 15-sample period, there are 3 input-rate samples and 5 output-rate samples. Had we not downsampled the output of the SRC filter by a factor of 3, it would be equal (for a perfect filter) to the signal $x''(n'')$.

Next, design a reverse sample rate converter to convert back by a factor of 3/5. The SRC filter has the same A and ΔF. Then, process $y(n')$ through it to see how well you may recover the original signal $x(n)$. Finally, plot the magnitude responses of the 5/3 and 3/5 filters versus frequency in the range $0 \le f \le 15$ kHz, both in absolute and decibel scales.

14.22 An alternative discrete-time model for a first-order delta-sigma quantizer is shown in Fig. 14.8.2. It uses a conventional accumulator, but puts a delay in the feedback loop to make it computable. Replace the quantizer Q by its equivalent noise model and work out the I/O relationship in the form of Eq. (14.7.3). What are the transfer functions $H_x(\zeta)$ and $H_{\rm NS}(\zeta)$? Write Eq. (14.7.3) in the n' time domain.

14.23 For the delta-sigma quantizer model shown in Fig. 14.8.2, define the action of the quantizer Q by the two-level function $Q(x)$ of Eq. (14.7.13). Using the indicated intermediate variables on the figure, write the corresponding sample processing algorithm. For the two constant

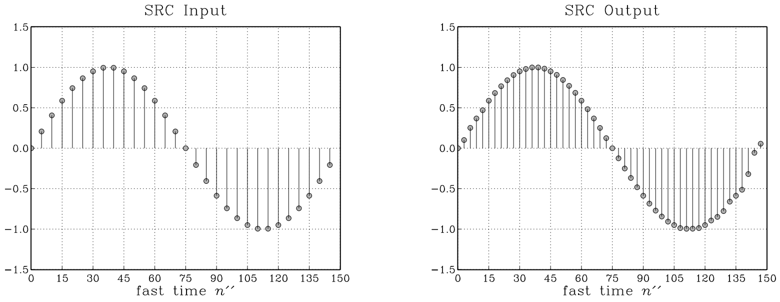


Fig. 14.8.1 Sample rate conversion by 5/3.

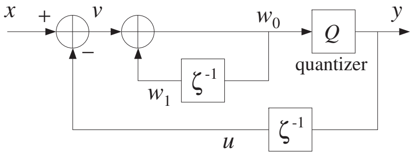


Fig. 14.8.2 Alternative model of first-order delta-sigma quantizer.

inputs, $x = 0.4$ and $x = -0.2$, iterate the algorithm ten times and make a table of the values of all the variables, as in Example 14.7.1. Compute the average of the quantized outputs y.

14.24 A discrete-time model for a second-order delta-sigma quantizer is shown in Fig. 14.8.3. Write the I/O equation in the form of Eq. (14.7.3) and determine the signal and noise transfer functions $H_x(\zeta)$ and $H_{\rm NS}(\zeta)$ in terms of the loop filters $H_1(\zeta)$ and $H_2(\zeta)$. Then, determine $H_1(\zeta)$ and $H_2(\zeta)$ such that

$$H_x(\zeta)= 1, \qquad H_{\rm NS}(\zeta)= (1-\zeta^{-1})^2$$

Redraw the full block diagram by replacing each $H_i(\zeta)$ by its realization, and write the sample processing algorithm assuming a quantizer function of the form $Q(x)$.

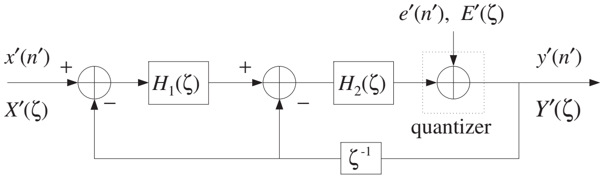


Fig. 14.8.3 Discrete-time model of second-order delta-sigma quantizer.

14.25 An alternative discrete-time model for the second-order ΔΣ quantizer is obtained by removing the delay ζ^{-1} from the feedback loop in Fig. 14.8.3. Determine $H_1(\zeta)$ and $H_2(\zeta)$ in order that the signal and noise transfer functions be:

$$H_x(\zeta)= \zeta^{-1}, \qquad H_{\rm NS}(\zeta)= (1-\zeta^{-1})^2$$

14.26 *Computer Experiment: First-Order Delta-Sigma ADC.* Write C or MATLAB programs to reproduce all the results and graphs of Example 14.7.2. Implement the decimator filtering operations in their sample-by-sample processing form using the routines `fir` or `cfir` of Chapter 4. In computing the outputs of the filters, you must also compute the input-off transients. In particular, for the window designs, you need to compute an extra LM input-off transients to compensate for the filter's delay.

Better decimators are obtained by raising the simple averaging decimator to some power. For example, a second-order L-fold "comb" decimator is defined by:

$$H(\zeta)= \left[\frac{1}{L}\,\frac{1-\zeta^{-L}}{1-\zeta^{-1}}\right]^2$$

It is similar to a linear interpolator normalized by L. For $L = 10$, determine its impulse response $\mathbf{h}$. Then, compute its output for the same quantized input as above, and compare it with the outputs of the averaging and length-21 decimators. Also, plot the magnitude response of this decimator on the same graph with the other three.

14.27 *Computer Experiment: Second-Order Delta-Sigma ADC.* Using the second-order delta-sigma quantizer and its sample processing algorithm defined in Problem 14.24 and using the same quantizer function $Q(x)$ of Eq. (14.7.13), repeat all the questions and graphs of the above computer experiment.

14.28 A second-order multistage delta-sigma quantizer architecture (known as MASH [351,363–365]) is shown in Fig. 14.8.4. It employs two identical first-order quantizers of the type of Fig. 14.8.2, with $H(\zeta)= 1/(1-\zeta^{-1})$, and $D(\zeta)= 1-\zeta^{-1}$.

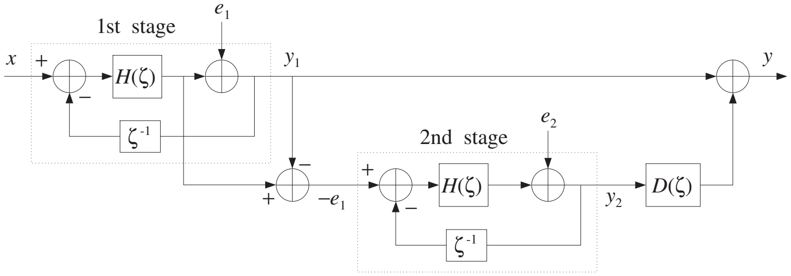


Fig. 14.8.4 MASH architecture of second-order delta-sigma quantizer.

The negative of the quantization error e_1, obtained by subtracting the two signals around the first quantizer, becomes the input to the second stage and its output is postfiltered by the differencing filter $D(\zeta)$ and added to the output of the first stage.

Using the I/O equation derived in Problem 14.22, show that the overall I/O equation of Fig. 14.8.4 involves only the second quantization error e_2, and is given by

$$Y'(\zeta)= X'(\zeta)+(1-\zeta^{-1})^2E_2'(\zeta)$$

14.29 A third-order delta-sigma MASH quantizer, can be obtained by adding a third first-order quantizer to the diagram of Fig. 14.8.4 [351,363–365]. The signal $-e_2$ can be generated from the second stage just like $-e_1$ is generated by the first stage. Then, the signal $-e_2$ is fed into the third stage, which has its own quantization noise e_3.

Draw the 3-stage block diagram and add an additional differentiator $D(\zeta)$ so that when you sum the outputs of the three stages, you get a third-order noise shaping I/O relationship, that is, combine the I/O equations of the three stages so that:

$$\begin{aligned} Y_1' &= \quad X' + DE_1' \\ Y_2' &= -E_1' + DE_2' \qquad \Rightarrow \qquad Y' = X' + D^3E_3' \\ Y_3' &= -E_2' + DE_3' \end{aligned}$$

14.30 A third-order delta-sigma MASH quantizer, can also be obtained by using a cascade combination of first- and second-order quantizers [351,363–365]. In the block diagram of Fig. 14.8.4, replace the first-order quantizer of the second stage by the second-order quantizer of Fig. 14.8.3. The differencer $D(\zeta)$ remains unchanged. Show that the overall I/O equation is now:

$$Y'(\zeta) = X'(\zeta) + (1-\zeta^{-1})^3 E_2'(\zeta)$$

14.31 Delta-sigma A/D converters are not always appropriate and must be used with caution, especially in multiplexing the sampling of several input channels or in feedback control systems. Why would you say this is so?

15

Noise Reduction and Signal Enhancement

15.1 Noise Reduction and Signal Extraction

One of the most common problems in signal processing is to extract a *desired* signal, say $s(n)$, from a noisy measured signal:

$$x(n) = s(n) + v(n) \tag{15.1.1}$$

where $v(n)$ is the *undesired* noise component.

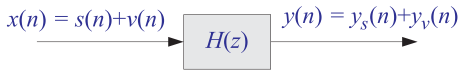
The noise signal $v(n)$ depends on the application. For example, it could be (1) a white noise signal, which is typical of the background noise picked up during the measurement process; (2) a periodic interference signal, such as the 60 Hz power-frequency pickup; (3) a low-frequency noise signal, such as radar clutter; (4) any other signal—not necessarily measurement noise—that must be separated from $s(n)$ as, for example, in separating the luminance and chrominance signal components embedded in the composite video signal in a color TV receiver.

The standard method of extracting $s(n)$ from $x(n)$ is to design an appropriate filter $H(z)$ which *removes* the noise component $v(n)$ and at the same time lets the desired signal $s(n)$ go through *unchanged*. Using linearity, we can express the output signal due to the input of Eq. (15.1.1) in the form:

$$y(n) = y_s(n) + y_v(n) \tag{15.1.2}$$

where $y_s(n)$ is the output due to $s(n)$ and $y_v(n)$ the output due to $v(n)$.

The two design conditions for the filter are that $y_v(n)$ be as small as possible and $y_s(n)$ be as similar to $s(n)$ as possible; that is, ideally we require:[†]



$$\boxed{\begin{aligned} y_s(n) &= s(n) \\ y_v(n) &= 0 \end{aligned}} \tag{15.1.3}$$

[†] An overall delay in the recovered signal is also acceptable, that is, $y_s(n) = s(n-D)$.

In general, these conditions cannot be satisfied simultaneously. To determine when they can be satisfied, we express them in the frequency domain in terms of the corresponding frequency spectra as follows: $Y_s(\omega) = S(\omega)$ and $Y_v(\omega) = 0$.

Applying the filtering equation $Y(\omega) = H(\omega)X(\omega)$ separately to the signal and noise components, we have the conditions:

$$\begin{aligned} Y_s(\omega) &= H(\omega)S(\omega) = S(\omega) \\ Y_v(\omega) &= H(\omega)V(\omega) = 0 \end{aligned} \tag{15.1.4}$$

The first requires that $H(\omega) = 1$ at all ω for which the signal spectrum is nonzero, $S(\omega) \neq 0$. The second requires that $H(\omega) = 0$ at all ω for which the noise spectrum is nonzero, $V(\omega) \neq 0$.

These two conditions can be met simultaneously *only if* the signal and noise spectra do *not* overlap, as shown in Fig. 15.1.1. In such cases, the filter $H(\omega)$ must have *passband* that coincides with the signal band, and *stopband* that coincides with the noise band. The filter removes the noise spectrum and leaves the signal spectrum unchanged.

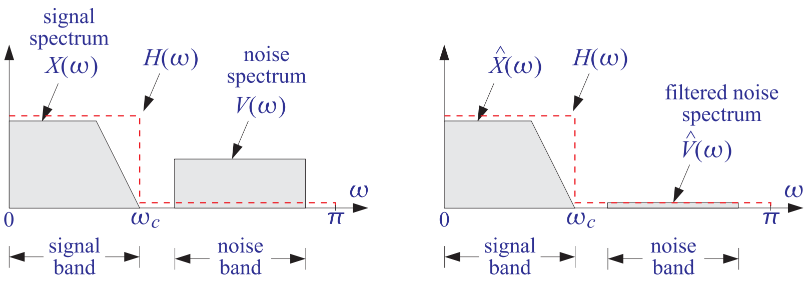


Fig. 15.1.1 Signal and noise spectra before and after filtering.

If the signal and noise spectra overlap, as is the typical case in practice, the above conditions cannot be satisfied simultaneously, because there would be values of ω such that both $S(\omega) \neq 0$ and $V(\omega) \neq 0$ and therefore the conditions (15.1.4) would require $H(\omega) = 1$ and $H(\omega) = 0$ for the same ω.

In such cases, we must compromise between the two design conditions and trade off one for the other. Depending on the application, we may decide to design the filter to remove as much noise as possible, but at the expense of distorting the desired signal. Alternatively, we may decide to leave the desired signal as undistorted as possible, but at the expense of having some noise in the output.

The latter alternative is depicted in Fig. 15.1.2 where a low-frequency signal $s(n)$ exists in the presence of a broadband noise component, such as white noise, having a flat spectrum extending over the entire[†] Nyquist interval, $-\pi \leq \omega \leq \pi$.

The filter $H(\omega)$ is chosen to be an ideal lowpass filter with passband covering the signal bandwidth, say $0 \leq \omega \leq \omega_c$. The noise energy in the filter's stopband $\omega_c \leq \omega \leq \pi$ is removed completely by the filter, thus reducing the strength (i.e., the rms value) of the noise. The spectrum of the desired signal is not affected by the filter, but neither is the portion of the noise spectrum that falls within the signal band. Thus, some noise will survive the filtering process.

[†] For discrete-time signals, the spectra are periodic in ω with period 2π.

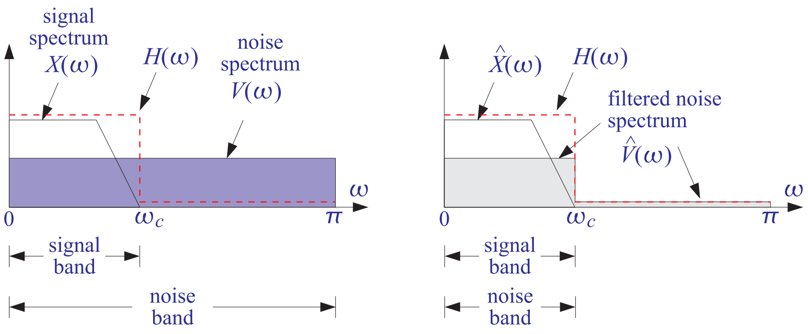


Fig. 15.1.2 Signal enhancement filter with partial noise reduction.

The amount of noise reduction achieved by this filter can be calculated using the *noise reduction ratio* (NRR) of Eq. (9.7.4) of Appendix 9.7, which is valid for white noise input signals. Denoting the input and output *mean-square* noise values by $\sigma_v^2 = E[v(n)^2]$ and $\sigma_{y_v}^2 = E[y_v(n)^2]$, we have the definition,

$$\boxed{\mathcal{R} = \frac{\sigma_{y_v}^2}{\sigma_v^2} = \int_{-\pi}^{\pi} |H(\omega)|^2 \frac{d\omega}{2\pi} = \sum_n h_n^2} \quad \text{(NRR)} \tag{15.1.5}$$

Because $H(\omega)$ is an ideal lowpass filter, the integration range reduces to the filter's passband, that is, $-\omega_c \leq \omega \leq \omega_c$. Over this range, the value of $H(\omega)$ is unity, giving:

$$\mathcal{R} = \frac{\sigma_{y_v}^2}{\sigma_v^2} = \int_{-\omega_c}^{\omega_c} 1 \cdot \frac{d\omega}{2\pi} = \frac{2\omega_c}{2\pi} = \frac{\omega_c}{\pi} \tag{15.1.6}$$

Thus, the NRR is the *proportion* of the *signal bandwidth* with respect to the Nyquist interval. The same conclusion also holds when the desired signal is a high-frequency or a mid-frequency signal. For example, if the signal spectrum extends only over the mid-frequency band $\omega_a \leq |\omega| \leq \omega_b$, then $H(\omega)$ can be designed to be unity over this band and zero otherwise. A similar calculation yields in this case:

$$\mathcal{R} = \frac{\sigma_{y_v}^2}{\sigma_v^2} = \frac{\omega_b - \omega_a}{\pi} \tag{15.1.7}$$

The noise reduction/signal enhancement capability of a filter can also be formulated in terms of the signal-to-noise ratio. The SNRs at the input and output of the filter are defined in terms of the mean-square values as:

$$SNR_{\text{in}} = \frac{E[s(n)^2]}{E[v(n)^2]}, \qquad SNR_{\text{out}} = \frac{E[y_s(n)^2]}{E[y_v(n)^2]}$$

Therefore, the relative improvement in the SNR introduced by the filter will be:

$$\frac{SNR_{\text{out}}}{SNR_{\text{in}}}=\frac{E[y_s(n)^2]}{E[y_v(n)^2]}\cdot\frac{E[v(n)^2]}{E[s(n)^2]}=\frac{1}{\mathcal{R}}\cdot\frac{E[y_s(n)^2]}{E[s(n)^2]}$$

If the desired signal is not changed by the filter, $y_s(n)\approx s(n)$, then

$$\boxed{\frac{SNR_{\text{out}}}{SNR_{\text{in}}}=\frac{1}{\mathcal{R}}} \tag{15.1.8}$$

Thus, *minimizing* the noise reduction ratio is equivalent to *maximizing* the signal-to-noise ratio at the filter's output.

The NRRs computed in Eqs. (15.1.6) or (15.1.7) give the *maximum* noise reductions achievable with *ideal* lowpass or bandpass filters that do not distort the desired signal. Such ideal filters are not realizable because they have double-sided impulse responses with infinite anticausal tails. Thus, in practice, we must use *realizable approximations* to the ideal filters. Chapters 11 and 12 discuss filter design methods that approximate the ideal responses to any desired degree.

The use of realizable noise reduction filters introduces two further design issues that must be dealt with in practice: One is the *transient response* of the filter and the other is the amount of *delay* introduced into the output.

The more closely a filter approximates the sharp transition characteristics of an ideal response, the closer to the unit circle its poles get, and the longer its transient response becomes. Stated differently, maximum noise reduction, approaching the ideal limit (15.1.6), can be achieved only at the expense of introducing long transients in the output.

The issue of the delay introduced into the output has to do with the steady-state response of the filter. We recall from Eq. (6.3.8) of Chapter 6 that after steady state has set in, different frequency components of an input signal suffer different amounts of delay, as determined by the phase delay $d(\omega)$ of the filter.

In particular, if the filter has *linear phase*, then it causes an overall delay in the output. Indeed, assuming that the filter has unity magnitude, $|H(\omega)|=1$, over its passband (i.e., the signal band) and is zero over the stopband, and assuming a constant phase delay $d(\omega)=D$, we find for the filtered version of the desired signal:

$$\begin{aligned}y_s(n)&=\int_{-\pi}^{\pi}Y_s(\omega)e^{j\omega n}\frac{d\omega}{2\pi}=\int_{-\pi}^{\pi}|H(\omega)|S(\omega)e^{j\omega(n-D)}\frac{d\omega}{2\pi}\\&=\int_{-\omega_c}^{\omega_c}S(\omega)e^{j\omega(n-D)}\frac{d\omega}{2\pi}=s(n-D)\end{aligned}$$

the last equation following from the inverse DTFT of the desired signal:

$$s(n)=\int_{-\omega_c}^{\omega_c}S(\omega)e^{j\omega n}\frac{d\omega}{2\pi}$$

Essentially all practical FIR noise reduction filters, such as the Savitzky-Golay smoothing filters discussed in Chap. 23 and the Kaiser window designs discussed in Section 11.3, have linear phase.

Next, we consider some noise reduction examples based on simple filters, calculate the corresponding noise reduction ratios, discuss the tradeoff between transient response times and noise reduction, and present some simulation examples.

15.2 IIR Exponential Smoother

It is desired to extract a constant signal $s(n)=s$ from the noisy measured signal

$$x(n)=s(n)+v(n)=s+v(n)$$

where $v(n)$ is zero-mean white Gaussian noise of variance σ_v^2. To this end, the following IIR lowpass filter is used:

$$H(z)=\frac{\alpha}{1-\lambda z^{-1}},\quad H(\omega)=\frac{\alpha}{1-\lambda e^{-j\omega}},\quad |H(\omega)|^2=\frac{\alpha^2}{1-2\lambda\cos\omega+\lambda^2}$$

where the parameter λ is restricted to the range $0<\lambda<1$. This filter is known as an "exponentially-weighted moving average" (EMA) because its impulse response is exponentially decaying,

$$h_n=\alpha\lambda^n u(n)$$

Because the desired signal $s(n)$ is constant in time, the signal band will only be the DC frequency $\omega=0$. We require, therefore, that the filter have unity response at $\omega=0$ or equivalently at $z=1$. This condition fixes the overall gain α of the filter:

$$H(z)\big|_{z=1}=\frac{\alpha}{1-\lambda}=1\qquad\Rightarrow\qquad\alpha=1-\lambda$$

The NRR of this filter can be calculated from Eq. (15.1.5) by summing the impulse response squared. Using the geometric series, we find

$$\mathcal{R}=\frac{\sigma_{y_v}^2}{\sigma_v^2}=\sum_n h_n^2=\alpha^2\sum_{n=0}^{\infty}\lambda^{2n}=\frac{\alpha^2}{1-\lambda^2}=\frac{(1-\lambda)^2}{1-\lambda^2}=\frac{1-\lambda}{1+\lambda}$$

This ratio is always less than one because λ is restricted to $0<\lambda<1$. To achieve high noise reduction, λ must be chosen near one. But, then the filter's transient time constant, given by Eq. (6.3.12), will become large:

$$n_{\text{eff}}=\frac{\ln\epsilon}{\ln\lambda}\to\infty\qquad\text{as}\qquad\lambda\to1$$

The filter's magnitude response, pole-zero pattern, and the corresponding input and output noise spectra are shown in Fig. 15.2.1. The shaded area under the $|H(\omega)|^2$ curve is the same as the NRR computed above.

The filter's 3-dB cutoff frequency ω_c can be calculated by requiring that $|H(\omega_c)|^2$ drops by 1/2, that is,

$$|H(\omega_c)|^2=\frac{\alpha^2}{1-2\lambda\cos\omega_c+\lambda^2}=\frac{1}{2}$$

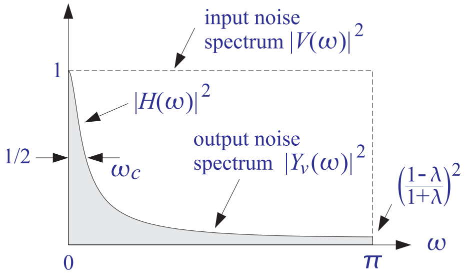


Fig. 15.2.1 Lowpass noise reduction filter of Example 15.2.

which can be solved to give $\cos\omega_c = 1-(1-\lambda)^2/2\lambda$. If a is near one, $a \lesssim 1$, we can use the approximation $\cos x \simeq 1 - x^2/2$ and solve for ω_c approximately:

$$\omega_c \simeq 1-\lambda$$

This shows that as $\lambda \to 1$, the filter becomes a narrower lowpass filter, removing more noise from the input, but at the expense of increasing the time constant.

The tradeoff between noise reduction and speed of response is illustrated in Fig. 15.2.2, where 200 samples of a simulated noisy signal $x(n)$ were filtered using the difference equation of the filter, that is, with $\alpha = 1-\lambda$

$$y(n) = \lambda y(n-1) + \alpha x(n) \tag{15.2.1}$$

and implemented with the sample processing algorithm, where $w_1(n) = y(n-1)$

```
for each input sample x do:
    $y = \lambda w_1 + \alpha x$
    $w_1 = y$
```

The value of the constant signal was $s = 5$ and the input noise variance $\sigma_v^2 = 1$. The random signal $v(n)$ was generated by successive calls to the Gaussian generator routine `gran` of Appendix A.1. The figure on the left corresponds to $\alpha = 0.90$, which has 1-percent time constant and NRR:

$$n_{\text{eff}} = \frac{\ln(0.01)}{\ln(0.90)} = 44, \qquad \mathcal{R} = \frac{1-0.90}{1+0.90} = \frac{1}{19}$$

It corresponds to an improvement of the SNR by, $10\log_{10}(1/\mathcal{R}) = 12.8$ dB. The right figure has $\lambda = 0.98$, with a longer time constant and smaller NRR:

$$n_{\text{eff}} = \frac{\ln(0.01)}{\ln(0.98)} = 228, \qquad \mathcal{R} = \frac{1-0.98}{1+0.98} = \frac{1}{99}$$

and an SNR improvement by, $10\log_{10}(1/\mathcal{R}) = 20$ dB. To understand how this filter works in the time domain and manages to reduce the noise, we rewrite the difference equation (15.2.1) in its convolutional form:

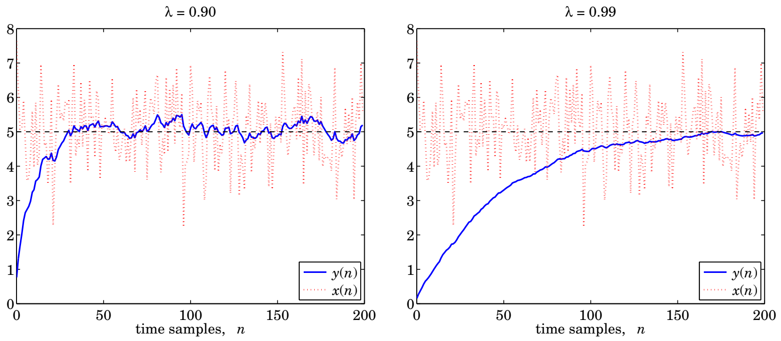


Fig. 15.2.2 Noisy input and smoothed output for Example 15.2.

$$y(n) = \alpha\sum_{m=0}^{n}\lambda^m x(n-m) = \alpha\big[x(n) + \lambda x(n-1) + \lambda^2 x(n-2) + \cdots + \lambda^n x(0)\big]$$

This sum corresponds to the accumulation or averaging of all the past samples up to the present time instant. As a result, the rapid fluctuations of the noise component $v(n)$ are averaged out. The closer λ is to 1, the more equal weighting the terms get, resulting in more effective averaging of the noise. The *exponential weighting* de-emphasizes the older samples and causes the sum to behave as though it had effectively a finite number of terms, thus, safeguarding the mean-square value of $y(n)$ from diverging. Because of the exponential weights, this filter is also called an *exponential smoother*.

The first-order EMA smoother can be applied to the smoothing of *any* low-frequency signal, not just constants. It is a standard tool in many applications requiring the smoothing of data, such as signal processing, statistics, economics, physics, and chemistry.

In general, one must make sure that the bandwidth of the desired signal $s(n)$ is *narrower* than the filter's lowpass width ω_c, so that the filter will not remove any of the higher frequencies present in $s(n)$.

Improved EMA Smoother

The NRR of Example 15.2 can be improved slightly, without affecting the speed of response, by adding a zero in the transfer function at $z = -1$ or equivalently, at $\omega = \pi$. The resulting first-order filter will be:

$$H(z) = \frac{\alpha(1+z^{-1})}{1-\lambda z^{-1}} \quad\Rightarrow\quad |H(\omega)|^2 = \frac{2\alpha^2(1+\cos\omega)}{1-2\lambda\cos\omega+\lambda^2} \tag{15.2.2}$$

where α is fixed by requiring unity gain at DC:

$$H(1) = \frac{2\alpha}{1-\lambda} = 1 \qquad\Rightarrow\qquad \alpha = \frac{1-\lambda}{2}$$

The zero at $\omega = \pi$ suppresses the high-frequency portion of the input noise spectrum even more than the filter of Example 15.2, thus, resulting in smaller NRR for the same value of λ. The impulse response of this filter can be computed using partial fractions:

$$H(z)=\frac{\alpha(1+z^{-1})}{1-\lambda z^{-1}}=A_0+\frac{A_1}{1-\lambda z^{-1}}$$

where

$$A_0=-\frac{\alpha}{\lambda},\qquad A_1=\frac{\alpha(1+\lambda)}{\lambda}$$

Therefore, the (causal) impulse response will be:

$$h_n=A_0\delta(n)+A_1\lambda^n u(n)$$

Note, in particular, $h_0 = A_0 + A_1 = \alpha$. It follows that

$$\mathcal{R}=\sum_{n=0}^{\infty}h_n^2=h_0^2+\sum_{n=1}^{\infty}h_n^2=\alpha^2+A_1^2\frac{\lambda^2}{1-\lambda^2}=\frac{1-\lambda}{2}$$

This is slightly smaller than the NRR of Example 15.2, because of the inequality:

$$\frac{1-\lambda}{2}<\frac{1-\lambda}{1+\lambda}$$

The 3-dB cutoff frequency can be calculated easily in this example. We have

$$|H(\omega_c)|^2=\frac{2\alpha^2(1+\cos\omega_c)}{1-2\lambda\cos\omega_c+\lambda^2}=\frac{1}{2}$$

which can be solved for ω_c in terms of λ:

$$l\cos\omega_c=\frac{2\lambda}{1+\lambda^2} \tag{15.2.3}$$

Conversely, we can solve for λ in terms of ω_c:

$$\lambda=\frac{1-\sin\omega_c}{\cos\omega_c}=\frac{1-\tan(\omega_c/2)}{1+\tan(\omega_c/2)} \tag{15.2.4}$$

It is easily checked that the condition $0 < \lambda < 1$ requires that $\omega_c < \pi/2$. We will encounter this example again in Chapter 12 and redesign it using the bilinear transformation. Note also that the replacement $z \to -z$ changes the filter into a highpass one. Such simple first-order lowpass or highpass filters with easily controllable widths are useful in many applications, such as the low- and high-frequency shelving filters of audio equalizers.

15.3 IIR Highpass Signal Extraction

It is desired to extract a high-frequency signal $s(n)=(-1)^n s$ from the noisy signal

$$x(n)=s(n)+v(n)=(-1)^n s+v(n)$$

where $v(n)$ is zero-mean, white Gaussian noise with variance σ_v^2. Because, the signal band is now at the Nyquist frequency $\omega = \pi$, we may use a first-order *highpass* IIR filter:

$$H(z)=\frac{\alpha}{1+\lambda z^{-1}},\quad H(\omega)=\frac{\alpha}{1+\lambda e^{-j\omega}},\quad |H(\omega)|^2=\frac{\alpha^2}{1+2\lambda\cos\omega+\lambda^2}$$

where $0 < \lambda < 1$. The gain α is fixed such that $H(\pi)=1$, or equivalently $H(z)=1$ at $z = e^{j\pi} = -1$, which gives the condition:

$$H(z)\big|_{z=-1}=\frac{\alpha}{1-\lambda}=1\qquad\Rightarrow\qquad\alpha=1-\lambda$$

The impulse response is now, $h_n = \alpha(-\lambda)^n u(n)$. The corresponding NRR can be calculated as in the previous example:

$$\mathcal{R}=\sum_n h_n^2=\alpha^2\sum_{n=0}^{\infty}(-\lambda)^{2n}=\frac{\alpha^2}{1-\lambda^2}=\frac{(1-\lambda)^2}{1-\lambda^2}=\frac{1-\lambda}{1+\lambda}$$

The noise reduction frequency characteristics of this highpass filter and its pole/zero pattern are shown in Fig. 15.3.1. Note that the pole is now at $z = -\lambda$. The 3-dB width ω_c is the same as in the previous example.

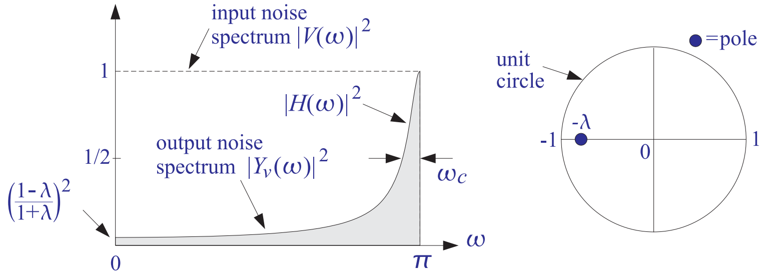


Fig. 15.3.1 Highpass noise reduction filter of Example 15.3.

Fig. 15.3.2 shows a simulation of 100 samples $x(n)$ filtered via the difference equation

$$y(n)=-\lambda y(n-1)+(1-\lambda)x(n)$$

The following values of the parameters were used: $s = 2$, $\lambda = 0.97$, $\sigma_v^2 = 1$. The corresponding one-percent time constant and NRR are in this case:

$$n_{\text{eff}}=\frac{\ln(0.01)}{\ln(0.97)}=151,\qquad\mathcal{R}=\frac{1-0.97}{1+0.97}=\frac{3}{197}=0.0101$$

which corresponds to an SNR improvement by, $10\log_{10}(1/\mathcal{R})=19.96$ dB.

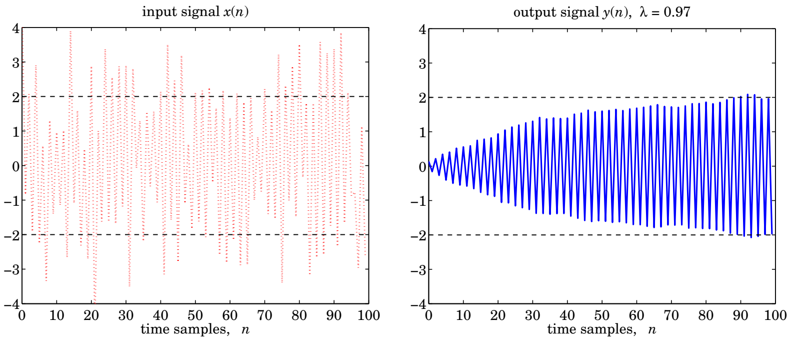


Fig. 15.3.2 Noisy input and high-frequency output for Example 15.3.

15.4 Bandpass Signal Extraction

A noisy sinusoid of frequency $f_0 = 500$ Hz is sampled at a rate of $f_s = 10$ kHz:

$$x(n)= s(n)+v(n)= \cos(\omega_0 n)+v(n)$$

where $\omega_0 = 2\pi f_0/f_s$, and $v(n)$ is a zero-mean, unit-variance, white Gaussian noise signal. The sinusoid can be extracted by a simple resonator filter of the type discussed in Section 6.4.2. The poles of the filter are placed at $z = Re^{\pm j\omega_0}$, as shown in Fig. 6.4.2.

If R is near 1, the resonator's 3-dB width given by Eq. (6.4.4), $\Delta\omega = 2(1-R)$, will be small, resulting in a very narrow bandpass filter. The narrower the filter, the more the noise will be reduced. The transfer function and impulse response of the filter were derived in Section 6.4.2:

$$H(z)= \frac{G}{1 + a_1 z^{-1} + a_2 z^{-2}}, \qquad h_n = \frac{G}{\sin\omega_0} R^n \sin(\omega_0 n + \omega_0)u(n)$$

where $a_1 = -2R\cos\omega_0$ and $a_2 = R^2$. The gain G is adjusted such that the filter's magnitude response is unity at the sinusoid's frequency, that is, $|H(\omega_0)| = 1$. In Section 6.4.2, we found

$$G = (1-R)\sqrt{1-2R\cos(2\omega_0)+R^2}$$

The NRR can be calculated in closed form:

$$\mathcal{R} = \sum_{n=0}^{\infty} h_n^2 = \frac{(1-R)(1+R^2)(1-2R\cos(2\omega_0)+R^2)}{(1+R)(1-2R^2\cos(2\omega_0)+R^4)} \qquad (15.4.1)$$

For $R = 0.99$ and $\omega_0 = 0.1\pi$, we have $\mathcal{R} = 1/99.6$, and filter parameters $a_1 = -1.8831$, $a_2 = 0.9801$, and $G = 6.1502{\times}10^{-3}$. Fig. 15.4.1 shows 300 samples of the noisy sinusoidal input $x(n)$ and the corresponding output signal $y(n)$ plotted together with desired sinusoid $s(n)$. The noise $v(n)$ was generated by the routine `gran`. The output was computed by the sample processing algorithm of the filter:

```
for each input sample x do:
    $y = -a_1 w_1 - a_2 w_2 + Gx$
    $w_2 = w_1$
    $w_1 = y$
```

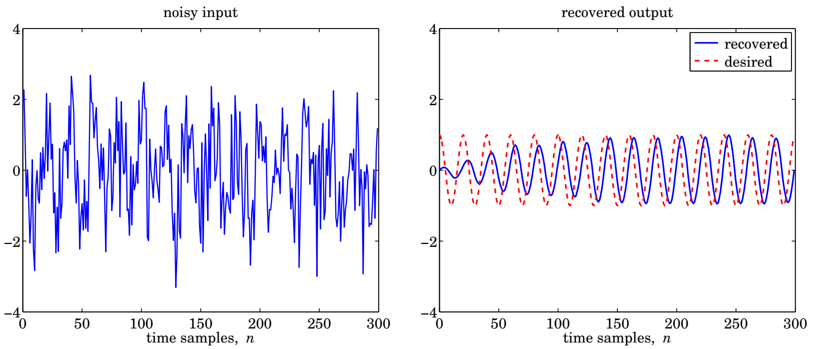


Fig. 15.4.1 Noisy sinusoidal input and extracted sinusoid.

The recovered sinusoid is slightly shifted with respect to $s(n)$ by an amount corresponding to the phase delay of the filter at $\omega = \omega_0$, that is, $n_{\rm ph}(\omega_0)= -\arg H(\omega_0)/\omega_0$. For the given numerical values, we find, $n_{\rm ph}(\omega_0)= 3.95$ samples.

15.5 FIR Averaging Filters

The problem of extracting a constant or a low-frequency signal $s(n)$ from the noisy signal $x(n)= s(n)+v(n)$ can also be approached with FIR filters. Consider, for example, the third-order filter

$$H(z)= h_0 + h_1 z^{-1} + h_2 z^{-2} + h_3 z^{-3}$$

The condition that the constant signal $s(n)$ go through the filter unchanged is the condition that the filter have unity gain at DC, which gives the constraint among the filter weights:

$$H(z)\big|_{z=1} = h_0 + h_1 + h_2 + h_3 = 1 \qquad (15.5.1)$$

The NRR of this filter will be simply:

$$\mathcal{R} = \sum_n h_n^2 = h_0^2 + h_1^2 + h_2^2 + h_3^2 \qquad (15.5.2)$$

The *best* third-order FIR filter will be the one that *minimizes* this NRR, subject to the lowpass constraint (15.5.1). To solve this minimization problem, we use the constraint to solve for one of the unknowns, say h_3:

$$h_3 = 1 - h_0 - h_1 - h_2$$

Substituting into the NRR, we find

$$\mathcal{R} = h_0^2 + h_1^2 + h_2^2 + (h_0 + h_1 + h_2 - 1)^2$$

The minimization of this expression can be carried out easily by setting the partial derivatives of $\mathcal{R}$ to zero and solving for the h's:

$$\frac{\partial}{\partial h_0}\mathcal{R} = 2h_0 + 2(h_0 + h_1 + h_2 - 1) = 2(h_0 - h_3) = 0$$

$$\frac{\partial}{\partial h_1}\mathcal{R} = 2h_1 + 2(h_0 + h_1 + h_2 - 1) = 2(h_1 - h_3) = 0$$

$$\frac{\partial}{\partial h_2}\mathcal{R} = 2h_2 + 2(h_0 + h_1 + h_2 - 1) = 2(h_2 - h_3) = 0$$

It follows that all four h's will be equal to each other, $h_0 = h_1 = h_2 = h_3$. But, because they must sum up to 1, we must have the optimum solution:

$$h_0 = h_1 = h_2 = h_3 = \frac{1}{4}$$

and the minimized NRR becomes:

$$\mathcal{R}_{\min} = \left(\frac{1}{4}\right)^2 + \left(\frac{1}{4}\right)^2 + \left(\frac{1}{4}\right)^2 + \left(\frac{1}{4}\right)^2 = 4 \cdot \left(\frac{1}{4}\right)^2 = \frac{1}{4}$$

The I/O equation for this optimum smoothing filter becomes:

$$y(n) = \frac{1}{4}\big[x(n) + x(n-1) + x(n-2) + x(n-3)\big]$$

More generally, the optimum length-N FIR filter with unity DC gain and minimum NRR is the filter with equal weights:

$$\boxed{h_n = \frac{1}{N}, \quad n = 0, 1, \dots, N-1} \tag{15.5.3}$$

and I/O equation:

$$\boxed{y(n) = \frac{1}{N}\big(x(n) + x(n-1) + x(n-2) + \cdots + x(n-N+1)\big)} \tag{15.5.4}$$

Its NRR is:

$$\boxed{\mathcal{R} = h_0^2 + h_1^2 + \cdots + h_{N-1}^2 = N \cdot \big(\frac{1}{N}\big)^2 = \frac{1}{N}} \tag{15.5.5}$$

Thus, by choosing N large enough, the NRR can be made as small as desired. Again, as the NRR decreases, the filter's time constant increases.

How does the FIR smoother compare with the IIR/EMA smoother of Example 15.2? First, we note the EMA smoother is very simple computationally, requiring only 2 MACs per output sample, whereas the FIR requires N MACs.

Second, the FIR smoother typically performs better in terms of both the NRR and the transient response, in the sense that for the same NRR value, the FIR smoother has shorter time constant, and for the same time constant, it has smaller NRR.

Given a time constant n_{eff} for an EMA smoother, the "equivalent" FIR smoother should be chosen to have the *same* length, that is,

$$N = n_{\text{eff}} = \frac{\ln \epsilon}{\ln \lambda}$$

For example, if $a = 0.90$, then $N = n_{\text{eff}} = 44$ as in Example 15.2. But then, the NRR of the FIR smoother will be $\mathcal{R} = 1/N = 1/44$, which is better than that of the IIR filter, $\mathcal{R} = 1/19$. This case is illustrated in the left graph of Fig. 15.5.1, where the FIR output was computed by Eq. (15.5.4) with $N = 44$, and implemented in MATLAB for the same noisy input of Example 15.2. The EMA output was already computed in Example 15.2.

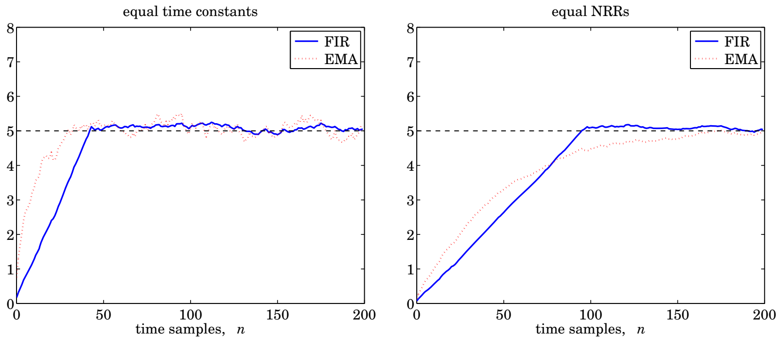


Fig. 15.5.1 Comparison of FIR and IIR smoothing filters.

Similarly, given an EMA smoother that achieves a certain NRR value, the "equivalent" FIR filter with the *same* NRR should have length N such that:

$$\mathcal{R} = \frac{1-\lambda}{1+\lambda} = \frac{1}{N} \quad \Rightarrow \quad N = \frac{1+\lambda}{1-\lambda}, \quad \Rightarrow \quad \lambda = \frac{N-1}{N+1} \tag{15.5.6}$$

For example, if $\lambda = 0.98$, then we get $N = 99$, which is much shorter than the EMA time constant, $n_{\text{eff}} = 228$ computed in Example 15.2. The right graph of Fig. 15.5.1 illustrates this case, where the FIR output was computed by Eq. (15.5.4) with $N = 99$.

An approximate relationship between the IIR time constant n_{eff} and N can be derived as follows. Using the small-x approximation $\ln\big((1+x)/(1-x)\big) \simeq 2x$, we have for large N:

$$\ln(1/\lambda) = \ln\left(\frac{1+1/N}{1-1/N}\right) \simeq \frac{2}{N}$$

It follows that,

$$n_{\text{eff}} = \frac{\ln(1/\epsilon)}{\ln(1/\lambda)} \simeq N\,\frac{1}{2}\ln\left(\frac{1}{\epsilon}\right)$$

Typically, the factor $(\ln(1/\epsilon)/2)$ is greater than one, resulting in a longer IIR time constant n_{eff} than N. For example, we have:

$$\begin{array}{lll} n_{\text{eff}} = 1.15\,N\,, & \text{if } \epsilon = 10^{-1} & \text{(10\% time constant)} \\ n_{\text{eff}} = 1.50\,N\,, & \text{if } \epsilon = 5\cdot 10^{-2} & \text{(5\% time constant)} \\ n_{\text{eff}} = 2.30\,N\,, & \text{if } \epsilon = 10^{-2} & \text{(1\% or 40 dB time constant)} \\ n_{\text{eff}} = 3.45\,N\,, & \text{if } \epsilon = 10^{-3} & \text{(0.1\% or 60 dB time constant)} \end{array}$$

Finally, we note that a further advantage of the FIR smoother is that it is a *linear phase* filter. Indeed, using the finite geometric series formula, we can write the transfer function of Eq. (15.5.4) in the form:

$$\boxed{H(z) = \frac{1}{N}\left(1 + z^{-1} + z^{-2} + \cdots + z^{-(N-1)}\right) = \frac{1}{N}\,\frac{1 - z^{-N}}{1 - z^{-1}}} \tag{15.5.7}$$

Setting, $z = e^{j\omega}$, we obtain the frequency response:

$$H(\omega) = \frac{1}{N}\,\frac{1 - e^{-jN\omega}}{1 - e^{-j\omega}} = \frac{1}{N}\,\frac{\sin(N\omega/2)}{\sin(\omega/2)}\,e^{-j\omega(N-1)/2} \tag{15.5.8}$$

which has a linear phase response. The transfer function (15.5.7) has zeros at the Nth roots of unity, except at $z = 1$, that is,

$$z_k = e^{j\omega_k}, \quad \omega_k = \frac{2\pi k}{N}, \quad k = 1, 2, \dots, N-1$$

The zeros are distributed equally around the unit circle and tend to suppress the noise spectrum along the Nyquist interval, except at $z = 1$ where there is a pole/zero cancellation and we have $H(z) = 1$.

Fig. 15.5.2 shows the magnitude and phase response of $H(\omega)$ for $N = 16$. Note that the phase response is *piece-wise linear* with slope $(N-1)/2$. It exhibits 180° jumps at $\omega = \omega_k$, where the factor $\sin(N\omega/2)/\sin(\omega/2)$ changes algebraic sign.

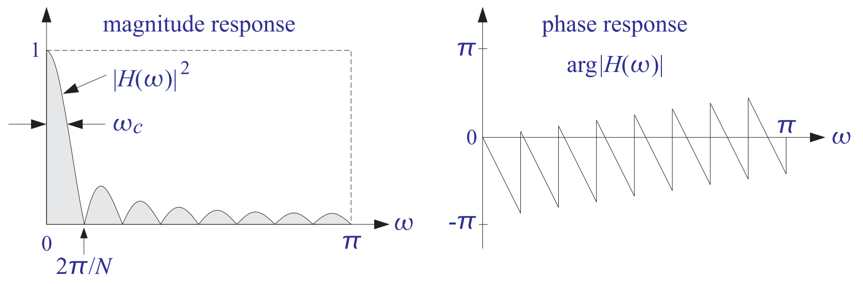


Fig. 15.5.2 Magnitude and phase responses of FIR smoother, for $N = 16$.

The cutoff frequency of the filter could be taken to be approximately *half of the base* of the mainlobe, that is,

$$\omega_c = \frac{\pi}{N}$$

This frequency corresponds to a 3.9 dB drop of the magnitude response. Indeed, setting $\omega = \omega_c = \pi/N$ we have:

$$\left|\frac{1}{N}\,\frac{\sin(N\pi/2N)}{\sin(\pi/2N)}\right|^2 \simeq \left|\frac{1}{N}\,\frac{\sin(\pi/2)}{(\pi/2N)}\right|^2 = \left|\frac{2}{\pi}\right|^2$$

where we used the approximation $\sin(\pi/2N) \simeq \pi/2N$, for large N. In decibels, we have $-10\log_{10}\left((2/\pi)^2\right) = 3.9$ dB. (Thus, ω_c is the 3.9-dB frequency.)

The 3-dB cutoff frequency is usually a more preferred measure of the passband width of a lowpass filter, defined to be the solution of the “half-power” condition,

$$|H(\omega)|^2 = \left|\frac{\sin(N\omega/2)}{N\sin(\omega/2)}\right|^2 = \frac{1}{2}$$

Setting, $\omega = 2\pi x/N$, the condition reads,

$$\left|\frac{\sin(\pi x)}{N\sin(\pi x/N)}\right|^2 = \frac{1}{2}$$

For fairly small values of N, such as $N \geq 7$, we may make the following approximation in the denominator,

$$N\sin\left(\frac{\pi x}{N}\right) \approx N\cdot\frac{\pi x}{N} = \pi x$$

so that the 3-dB condition simplifies into the equation,

$$\left|\frac{\sin(\pi x)}{\pi x}\right|^2 = \frac{1}{2} \tag{15.5.9}$$

with solution, $x \approx 0.443$, resulting in the following useful approximation for the 3-dB cutoff frequency of te FIR averager,

$$\boxed{\omega_c = 0.443\,\frac{2\pi}{N} = \frac{0.886\,\pi}{N}} \tag{15.5.10}$$

In MATLAB, the solution of Eq. (15.5.9) can be obtained by the command,

```
x = fzero(@(x) sinc(x).^2 - 0.5, 0.5);    % resulting in x = 0.44294...
```

Like its IIR/EMA counterpart of Example 15.2, the FIR averaging filter (15.5.4) can be applied to any low-frequency signal $s(n)$—not just a constant signal. The averaging of the N successive samples in Eq. (15.5.4) tends to smooth out the highly fluctuating noise component $v(n)$, while it leaves the slowly varying component $s(n)$ almost unchanged.

However, if $s(n)$ is not so slowly varying, the filter will also tend to average out these variations, especially when the averaging operation (15.5.4) reaches across many time samples when N is large. In the frequency domain, the same conclusion follows by noting that as N increases, the filter’s cutoff frequency ω_c decreases, thus removing more and more of the higher frequencies present in the desired signal.

Thus, there is a limit to the applicability of this type of smoothing filter: Its length must be chosen to be large enough to reduce the noise, but not so large as to start distorting the desired signal by smoothing it too much.

A rough quantitative criterion for the selection of the length N is as follows. If it is known that the desired signal $s(n)$ contains significant frequencies up to a maximum frequency, say $\omega_{\max}$, then we may choose N such that $\omega_c \geq \omega_{\max}$, which gives $N \leq \pi/\omega_{\max}$, and in units of Hz, $N \leq f_s/2f_{\max}$.

The FIR smoothing filter (15.5.4) will be considered in further detail in Chap. 23 and generalized to include additional linear constraints on the filter weights. Like the IIR smoother, the FIR smoother is widely used in many data analysis applications.

Recursive Realization of FIR Averager

The FIR averaging filter can also be implemented in a *recursive form* based on the summed version of the transfer function (15.5.7). For example, the direct form realization of $H(z)$ will be described by the I/O difference equation:

$$y(n)= y(n-1)+\frac{1}{N}\big(x(n)-x(n-N)\big) \qquad \text{(direct form)} \qquad (15.5.11)$$

and the *canonical* realization by the system of equations:

$$\begin{aligned} w(n) &= x(n)+w(n-1) \\ y(n) &= \frac{1}{N}\big(w(n)-w(n-N)\big) \end{aligned} \qquad \text{(canonical form)} \qquad (15.5.12)$$

These realizations are prone to roundoff accumulation errors and instabilities, and therefore, are not recommended even though they are efficient computationally.

To see the problems that may arise, consider the canonical realization. Assuming that the input $x(n)$ is a white noise signal, the equation $w(n)= w(n-1)+x(n)$ corresponds to the accumulation of $x(n)$ and, as we discuss in Appendix 9.7, this causes the mean-square value of $w(n)$ to become unstable. This is unacceptable because $w(n)$ is a required intermediate signal in this realization.

Similarly, considering the direct form realization of Eq. (15.5.11), if $y(n)$ is inadvertently initialized not to zero, but to some other constant, this constant cannot be rid of from the output because it gets canceled from the difference $y(n)-y(n-1)$. Similarly, if the operation $\big(x(n)-x(n-N)\big)/N$ is done with finite arithmetic precision, which introduces a small roundoff error, this error will get accumulated and eventually grow out of bounds.

The above recursive implementation can be stabilized using the standard procedure of pushing all the marginal poles into the unit circle. The replacement of Eq. (9.7.7) gives in this case:

$$H(z)= \frac{1}{N}\,\frac{1-\rho^N z^{-N}}{1-\rho z^{-1}}$$

where $0 < \rho \lesssim 1$. If so desired, the filter may be renormalized to unity gain at DC resulting in

$$H(z)= \frac{1-\rho}{1-\rho^N}\,\frac{1-\rho^N z^{-N}}{1-\rho z^{-1}}$$

In the limit as $\rho \to 1$, this expression converges to the original filter (15.5.7).

This stabilized version behaves comparably to the first-order smoother of Example 15.2. Indeed, if N is taken to be large for a fixed value of ρ, then we can set approximately $\rho^N \simeq 0$. In this limit, the filter reduces to the IIR smoother.

15.6 FIR Highpass Signal Extraction

In general, the substitution $z \to -z$ changes any lowpass filter into a highpass one. It corresponds to a change in the transfer function:

$$H(z)= \sum_n h_n z^{-n} \quad \longrightarrow \quad H(-z)= \sum_n h_n(-z)^{-n}= \sum_n (-1)^n h_n z^{-n}$$

and to the change in the impulse response:

$$h_n \;\longrightarrow\; (-1)^n h_n$$

We may think of Example 15.3 as being obtained from Example 15.2 by this substitution. Similarly, applying the substitution to the lowpass FIR smoother will result into a highpass FIR filter with impulse response:

$$h_n = (-1)^n\frac{1}{N}, \quad n = 0,1,\dots,N-1 \qquad (15.6.1)$$

and transfer function:

$$H(z)= \frac{1}{N}\sum_{n=0}^{N-1}(-1)^n z^{-n} = \frac{1}{N}\,\frac{1-(-1)^N z^{-N}}{1+z^{-1}}$$

The transfer function has unity gain at $\omega = \pi$, or $z = -1$; indeed,

$$H(-1)= \sum_{n=0}^{N-1}(-1)^n h_n = 1 \qquad (15.6.2)$$

The noise reduction ratio remains the same, namely, Eq. (15.5.5). In fact, one can obtain the filter (15.6.1) by minimizing the NRR of (15.5.5) subject to the highpass constraint (15.6.2).

15.7 Noise Reduction, Time Constant, Group Delay

In this section, we demonstrate with some concrete examples, the basic tradeoff between noise reduction and the time constant as well as the group delay of a filter, that is, as the filter becomes more effective in reducing noise, its time constant and group delay become longer.

For a stable and causal filter, the amount of noise reduction is quantified by the noise-reduction-ratio $\mathcal{R}$, which can be expressed in terms of the filter's impulse response, $h(n)$, or in terms of its frequency response, $H(\omega)$, or in terms of its transfer function, $H(z)$, recalling that, $H(\omega)= H(z)\big|_{z=e^{j\omega}}$, and that, $H^*(\omega)= H(-\omega)$, for a real-valued impulse response,

$$\mathcal{R}=\sum_{n=0}^{\infty}h^2(n)=\int_{-\pi}^{\pi}|H(\omega)|^2\,\frac{d\omega}{2\pi}=\oint_{u.c.}H(z)H(z^{-1})\,\frac{dz}{2\pi jz} \tag{15.7.1}$$

In addition to noise-reduction, one must also consider the time-constant of the filter that quantifies the duration of the filter transients (or, the duration of its impulse response), as well as the filter lag, or group delay, introduced by the filter.

For a stable and causal IIR filter, the filter poles must lie within the unit circle. The pole p with maximum magnitude, $|p|_{\max}$, determines the effective time constant, given as follows in time samples or in seconds,

$$n_{\rm eff}=\frac{\ln(\epsilon)}{\ln|p|_{\max}}\quad\Rightarrow\quad t_{\rm eff}=n_{\rm eff}T=\frac{1}{f_s}\,n_{\rm eff} \tag{15.7.2}$$

where, $f_s=1/T$ is the sampling rate, and ϵ is a small user-defined constant, such as, $\epsilon=10^{-2}$, or, $\epsilon=10^{-3}$. These two choices define the so-called 40-dB and 60-dB time constants, since in dB the values of ϵ are, $\epsilon_{\rm dB}=20\log_{10}(\epsilon)=-40$, and $\epsilon_{\rm dB}=-60$ dB, respectively.

The phase delay and group delay of a filter are in general functions of frequency and are defined in terms of the phase response, $\theta(\omega)$, of the filter, as follows,[†]

$$\boxed{\begin{aligned}&H(\omega)=|H(\omega)|\,e^{j\theta(\omega)}\\&n_{\rm ph}(\omega)=-\frac{\theta(\omega)}{\omega}\\&n_{\rm gr}(\omega)=-\theta'(\omega)=-\frac{d\theta(\omega)}{d\omega}\end{aligned}} \tag{15.7.3}$$

The group delay is also known as the *filter lag*. It can be shown that $n_{\rm gr}(\omega)$ is expressible directly in terms of $H(\omega)$, or in terms of $h(n)$, by,

$$n_{\rm gr}(\omega)=-\theta'(\omega)=-\,\mathrm{Re}\left[\frac{jH'(\omega)}{H(\omega)}\right]=\mathrm{Re}\left[\frac{\displaystyle\sum_{n=0}^{\infty}nh(n)e^{-j\omega n}}{\displaystyle\sum_{n=0}^{\infty}h(n)e^{-j\omega n}}\right] \tag{15.7.4}$$

The significance of the phase and group delays may be appreciated by the following result. Let, $x(n)=\cos(\omega_0 n)\cdot p(n)$, denote a sinewave modulated by slowly-varying pulse envelope $p(n)$ whose frequency spectrum, $P(\omega)$, is narrowband and highly concentrated at DC. Then, upon passing through a filter $H(\omega)$, the output signal will be given approximately by,

$$\begin{aligned}y(n)&=|H(\omega_0)|\cdot\cos\big(\omega_0 n+\theta(\omega_0)\big)\cdot p\big(n-n_{\rm gr}(\omega_0)\big)\\&=|H(\omega_0)|\cdot\cos\big(\omega_0 n-\omega_0 n_{\rm ph}(\omega_0)\big)\cdot p\big(n-n_{\rm gr}(\omega_0)\big)\\&=|H(\omega_0)|\cdot\cos\Big(\omega_0\left(n-n_{\rm ph}(\omega_0)\right)\Big)\cdot p\left(n-n_{\rm gr}(\omega_0)\right)\end{aligned} \tag{15.7.5}$$

[†] we assume that the angle $\theta(\omega)$ is reduced modulo-2π to the standard interval $[-\pi,\pi]$

that is, the envelope $p(n)$ gets delayed by the group delay at ω_0, whereas the sinusoid is modified in magnitude by the magnitude response, $|H(\omega_0)|$, and shifted in phase by the phase delay at frequency ω_0.
To derive this result, consider the complex sinusoid $e^{j\omega_0 n}$ being modulated by $p(n)$, and its DTFT,

$$x(n)=e^{j\omega_0 n}p(n)\quad\Rightarrow\quad X(\omega)=P(\omega-\omega_0)$$

where $P(\omega)$ is the DTFT of $p(n)$,

$$P(\omega)=\sum_n p(n)e^{-j\omega n}\quad\Leftrightarrow\quad p(n)=\int_{-\pi}^{\pi}P(\omega)\,e^{j\omega n}\,\frac{d\omega}{2\pi}$$

and we used the modulation theorem of Fourier transforms. It follows that the spectrum of the output signal $y(n)$ will be,

$$Y(\omega)=H(\omega)X(\omega)=H(\omega)P(\omega-\omega_0)$$

and $y(n)$ is reconstructed by the inverse DTFT,

$$y(n)=\int_{-\pi}^{\pi}Y(\omega)\,e^{j\omega n}\,\frac{d\omega}{2\pi}=\int_{-\pi}^{\pi}H(\omega)P(\omega-\omega_0)\,e^{j\omega n}\,\frac{d\omega}{2\pi} \tag{15.7.6}$$

Since $P(\omega)$ is highly concentrated at DC, it follows that $P(\omega-\omega_0)$ will be highly concentrated about ω_0. Therefore inside the integral of Eq. (15.7.6), we may expand $H(\omega)$ in the neighborhood of ω_0, that is, to the lowest order in the variable, $\omega-\omega_0$, and assume the following approximations for the magnitude and phase responses,

$$\begin{aligned}&|H(\omega)|\approx|H(\omega_0)|\\&\theta(\omega)\approx\theta(\omega_0)+(\omega-\omega_0)\theta'(\omega_0)=\theta(\omega_0)-(\omega-\omega_0)\,n_{\rm gr}(\omega_0)\\&H(\omega)=|H(\omega)|\,e^{j\theta(\omega)}\approx|H(\omega_0)|\,e^{j\theta(\omega_0)-j(\omega-\omega_0)\,n_{\rm gr}(\omega_0)}\end{aligned}$$

Thus, the output signal will be in the time domain,

$$\begin{aligned}y(n)&=\int_{-\pi}^{\pi}H(\omega)P(\omega-\omega_0)\,e^{j\omega n}\,\frac{d\omega}{2\pi}\\&\approx\int_{-\pi}^{\pi}|H(\omega_0)|\,e^{j\theta(\omega_0)-j(\omega-\omega_0)\,n_{\rm gr}(\omega_0)}P(\omega-\omega_0)\,e^{j\omega n}\,\frac{d\omega}{2\pi}\\&\approx|H(\omega_0)|\,e^{j\theta(\omega_0)}\int_{-\pi}^{\pi}e^{-j(\omega-\omega_0)\,n_{\rm gr}(\omega_0)}P(\omega-\omega_0)\,e^{j\omega n}\,\frac{d\omega}{2\pi}\\&=|H(\omega_0)|\,e^{j\theta(\omega_0)}e^{j\omega_0 n}\int_{-\pi}^{\pi}e^{-j(\omega-\omega_0)\,n_{\rm gr}(\omega_0)}P(\omega-\omega_0)\,e^{j(\omega-\omega_0)n}\,\frac{d\omega}{2\pi}\\&=|H(\omega_0)|\,e^{j\theta(\omega_0)}e^{j\omega_0 n}\int_{-\pi}^{\pi}P(\omega-\omega_0)\,e^{j(\omega-\omega_0)\left(n-n_{\rm gr}(\omega_0)\right)}\,\frac{d\omega}{2\pi}\\&=|H(\omega_0)|\,e^{j\omega_0 n+j\theta(\omega_0)}\,p\left(n-n_{\rm gr}(\omega_0)\right)\end{aligned}$$

Replacing the complex sinusoid $e^{j\omega_0 n}$ by $\cos(\omega_0 n)$ amounts to taking the real part of the above complex-valued output, resulting in Eq. (15.7.5).

A particularly important case is the filter lag at DC, $\omega = 0$, denoted by, $\bar{n} = n_{\text{gr}}(0)$, and given as follows as a special case of Eq. (15.7.4),

$$\boxed{\bar{n} = n_{gr}(0) = \frac{\displaystyle\sum_{n=0}^{\infty} n\,h(n)}{\displaystyle\sum_{n=0}^{\infty} h(n)}} \qquad \text{(filter lag)} \tag{15.7.7}$$

where we assumed that $h(n)$ is real valued. If the frequency response is normalized to unity-gain at DC, as is usually done for lowpass filters, then,

$$\bar{n} = \sum_{n=0}^{\infty} n\,h(n) \tag{15.7.8}$$

where we used the assumed fact that,

$$H(\omega)\Big|_{\omega=0} = \sum_{n=0}^{\infty} h(n) = 1$$

The DC filter lag $\bar{n}$ plays an important role in financial market indicators that are used to smooth and predict price data, as we discuss in Chap. 25. The tradeoff between lag and noise reduction is apparent here also. Too much lag is undesirable since one may miss important changes in the prices, but on the other hand, attempting to reduce the lag may be less effective in smoothing the data and reducing noise.

There are filters, such as the "predictive moving average", and the "double EMA indicator" that have zero lag, $\bar{n} = 0$, but at a price of less noise reduction. In addition, FIR filters can be designed to have optimum noise reduction and any desired value of $\bar{n}$, even negative values which correspond to time-advancing or prediction instead of delaying. We studied such filters in Chap. 24 and 25.

The significance of the DC lag may be seen by the following result. Assuming a straight-line input, $x(n) = a + bn$, with intercept a and slope b, one can verify that upon passing through a (unity-dc-gain) filter, the steady-state output will still be a straight line with the same slope, but shifted by $\bar{n}$, that is,

$$x(n) = a + bn \longrightarrow \boxed{H} \longrightarrow y(n) = a + b(n - \bar{n}) \tag{15.7.9}$$

Passing through a cascade of two identical filters $H(z)$, the lag will be doubled,

$$x(n) = a + bn \longrightarrow \boxed{H} \longrightarrow \boxed{H} \longrightarrow y(n) = a + b(n - 2\bar{n})$$

By forming the new filter,

$$\boxed{H_a(z) = 2H(z) - H^2(z)} \qquad \text{(zero-lag filter)} \tag{15.7.10}$$

the lag can be eliminated,

$$x(n) = a + bn \longrightarrow \boxed{H_a} \longrightarrow y(n) = a + bn$$

Indeed, the output of $2H(z)$ will be, $2\big[a + b(n - \bar{n})\big]$, and upon subtracting the output of $H^2(z)$, we obtain,

$$y(n) = 2\big[a + b(n - \bar{n})\big] - \big[a + b(n - 2\bar{n})\big] = a + bn$$

This method of lag reduction is known as Tukey's *twicing* operation and is the method used to construct the double EMA indicator mentioned above. The method has been generalized by Kaiser and Hamming [644].

15.8 Computer Experiment – Noise-Reduction vs. Group-Delay

To illustrate the property (15.7.5), as well as the tradeoff between noise reduction and the time constant of a filter, and between noise reduction and group delay, consider a similar example as that of Sec. 15.4, but use instead an exact design for the resonator filter as discussed in Sec. 12.3.

The input is a sinusoidal signal, $s(n) = \cos(\omega_0 n)$, in zero-mean, unit-variance, white gaussian noise, $v(n)$,

$$x(n) = s(n) + v(n) = \cos(\omega_0 n) + v(n) \tag{15.8.1}$$

The desired signal $s(n)$ can be extracted with the help of a peaking resonator filter with center frequency at ω_0. By choosing the width of the peak to be very narrow, the noise will be substantially reduced. However, as we see below, the time constant of the filter as well as the group delay increase as the width becomes narrower. The transfer function of the filter has the form,

$$H(z) = \frac{\beta}{1+\beta} \cdot \frac{1 - z^{-2}}{1 - \dfrac{2\cos\omega_0}{1+\beta}\, z^{-1} + \dfrac{1-\beta}{1+\beta}\, z^{-2}} \tag{15.8.2}$$

where β is related to the peak's 3-dB width, say, $\Delta\omega$ in rads/sample, by,

$$\beta = \tan\left(\frac{\Delta\omega}{2}\right) \tag{15.8.3}$$

The filter has the following properties. Its zeros are at DC and Nyquist, that is, at $\omega = 0$ and $\omega = \pi$. Its poles are at the locations,

$$p_{\pm} = \frac{\cos\omega_0 \pm j\sqrt{\sin^2\omega_0 - \beta^2}}{1+\beta} \tag{15.8.4}$$

The poles are complex conjugates, if $|\sin\omega_0| \geq \beta$, which is usually the practical case if the bandwidth $\Delta\omega$ is sufficiently small. Otherwise, the poles are real valued. We will assume here that the poles are complex. In this case, they have a common magnitude,

$$|p_{\pm}| = \sqrt{\frac{1-\beta}{1+\beta}} \tag{15.8.5}$$

and therefore, the ϵ-level time constant, will be,

$$n_{\text{eff}} = \frac{\ln(\epsilon)}{\ln(|p_\pm|)} = \frac{2\ln(\epsilon)}{\ln\left(\dfrac{1-\beta}{1+\beta}\right)} \tag{15.8.6}$$

The frequency response and magnitude response squared are given by,

$$\begin{aligned} H(\omega) = H(z)\Big|_{z=e^{j\omega}} &= \frac{j\beta\sin\omega}{\cos\omega - \cos\omega_0 + j\beta\sin\omega} \\ |H(\omega)|^2 &= \frac{\beta^2\sin^2\omega}{(\cos\omega-\cos\omega_0)^2 + \beta^2\sin^2\omega} \end{aligned} \tag{15.8.7}$$

Thus, the filter has unity gain at the peak, $H(\omega_0) = 1$. The group delay and its maximum value at, $\omega = \omega_0$, are as follows,

$$\begin{aligned} n_{\text{gr}}(\omega) &= \frac{\beta(1-\cos\omega\,\cos\omega_0)}{(\cos\omega-\cos\omega_0)^2+\beta^2\sin^2\omega} \\ n_{\text{gr}}(\omega_0) &= \frac{1}{\beta} \end{aligned} \tag{15.8.8}$$

The filter's left and right 3-dB frequencies, are obtained from the equations,

$$\begin{aligned} \cos\omega_L &= \frac{\cos\omega_0 + \beta\sqrt{\sin^2\omega_0+\beta^2}}{1+\beta^2} \\ \cos\omega_R &= \frac{\cos\omega_0 - \beta\sqrt{\sin^2\omega_0+\beta^2}}{1+\beta^2} \end{aligned} \tag{15.8.9}$$

and are the solutions of the 3-dB condition, $|H(\omega)|^2 = \dfrac{1}{2}$, and satisfy the constraint,

$$\omega_R - \omega_L = \Delta\omega \tag{15.8.10}$$

The filter's causal/stable impulse response $h(n)$ is given by,

$$h(n) = A_0\delta(n) + A_+ p_+^n\,u(n) + A_- p_-^n\,u(n) \tag{15.8.11}$$

where $p_\pm$ are the filter poles of Eq. (15.8.4), and,

$$A_0 = -\frac{\beta}{1-\beta}\,, \quad A_\pm = \frac{\beta}{1-\beta^2}\left[1 \pm \frac{j\beta\cos\omega_0}{\sqrt{\sin^2\omega_0-\beta^2}}\right] \tag{15.8.12}$$

The noise reduction ratio of this filter is found to be,

$$\mathcal{R} = \sum_{n=0}^{\infty} h^2(n) = \frac{\beta}{1+\beta} \tag{15.8.13}$$

To first order in $\Delta\omega$, the parameter β becomes, $\beta = \tan(\Delta\omega/2) \approx \Delta\omega/2$, implying the following approximations capturing the tradeoff between the NRR, the time-constant, and the group delay,

$$\mathcal{R} \approx \frac{\Delta\omega}{2}\,, \quad n_{\text{eff}} \approx \frac{2\ln(\epsilon^{-1})}{\Delta\omega}\,, \quad n_{\text{gr}}(\omega_0) \approx \frac{2}{\Delta\omega} \tag{15.8.14}$$

Questions

For the parameter values,

$$\omega_0 = 0.2\pi\,, \quad \Delta\omega = 0.01\pi\,, \quad [\text{rads/sample}]\,,$$

please carry out the following experiments.

(a) First, prove Eqs. (15.8.4)–(15.8.14).

(b) Calculate and plot the magnitude response squared $|H(\omega)|^2$ of the peaking filter over the frequency range, $0 \le \omega \le \pi$. *Suggestion*: For plotting purposes, you may compute $|H(\omega)|^2$ at 1001 equally-spaced frequencies in that range.

Add the left/right 3-dB frequencies, ω_L, ω_R, on the graph and connect them with a straight-line segment to indicate the 3-dB width.

(c) Calculate and plot the group delay $n_{\text{gr}}(\omega)$ over the same frequency range as in part (a). Calculate and place on the graph the maximum value of the group delay at $\omega = \omega_0$.

(d) Calculate and plot $N = 300$ samples of the impulse response $h(n)$ of the peaking filter using Eq. (15.8.11). Alternatively, calculate the impulse response numerically by sending a unit impulse $\delta(n)$ into the filter and computing $N = 300$ output samples.

Verify that the analytical and numerical calculations of $h(n)$ produce the same result (to within the double-precision of MATLAB).

(e) Load a length $N = 300$ segment of a zero-mean, unit-variance, white gaussian noise signal, $v(n)$, from the attached file, `v.mat`, via the MATLAB command,

```
load v;
```

Then, construct the time signal, $x(n)$, $n = 0, 1, \dots, N-1$, as defined in Eq. (15.8.1), and filter it through the peaking filter to obtain the output $y(n)$. You may carry out the filtering operation using the built-in MATLAB function, **filter**,

```
y = filter(num,den,x);
```

Moreover, filter separately the noise $v(n)$, obtaining its filtered version $y_v(n)$.

On four separate graphs, plot the signals, $x(n), y(n), v(n), y_v(n)$, using the *same* vertical scale in all four cases for comparison purposes. See some example graphs at the end of this section.

To the $y(n)$ graph, add the noise-free sinusoid $s(n)$ so that you can observe how well it is recovered by the filter. Are the observed transients consistent with the 40-dB time constant of the filter?

In addition, explain why the filtered noise $y_v(n)$, albeit weak, looks more like a quasi-sinusoid than noise.

(f) Using the built-in standard deviation MATLAB function, **std**, calculate the sample variances of the input and output noise signals, and calculate the estimated NRR as the ratio of the estimated variances, comparing it with the theoretical NRR given in Eq. (15.8.13),

$$\hat{\mathcal{R}} = \frac{\text{std}^2(y_v)}{\text{std}^2\,(v)}$$

(g) Next, we look at the group-delay property of Eq. (15.7.5). First, construct a slowly-varying narrowband envelope signal $p(n)$ of length $M+1$ by the Hanning window expression,

$$p(n) = \begin{cases} 0.5 - 0.5\cos\left(\dfrac{2\pi n}{M}\right), & 0 \le n \le M \\ 0\,, & \text{otherwise} \end{cases}$$

With $M = 1000$, calculate and plot the magnitude spectrum $|P(\omega)|$ of the pulse $p(n)$ over the narrow interval $|\omega| \le 0.1\pi$ using the built-in MATLAB function, **freqz**, and verify that it is narrowly concentrated about $\omega = 0$. For plotting purposes, normalize the spectrum to unity gain at DC.

Then, calculate the modulated sinusoid, with $N = 1100$,

$$x(n) = \cos(\omega_0 n)\cdot p(n)\,, \quad 0 \le n \le N-1$$

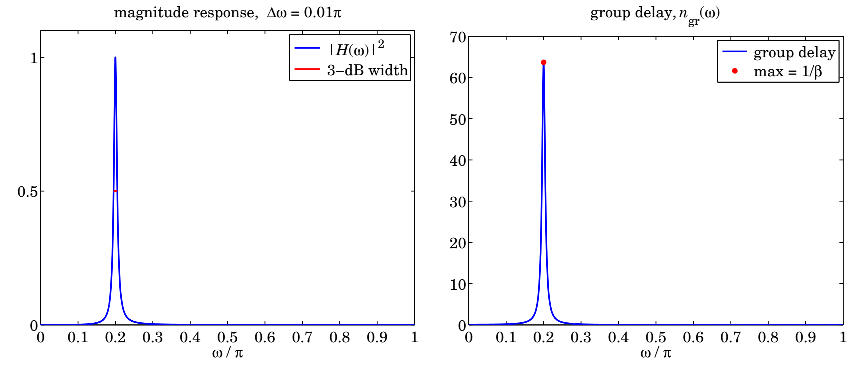
Filter $x(n)$ through the peaking filter and calculate and plot the corresponding output $y(n)$. Note that since, $H(\omega_0) = 1$, having unit magnitude and zero phase, $|H(\omega_0)| = 1$, $\text{Arg}H(\omega_0) = 0$, the expected approximate output of Eq. (15.7.5) will be of the form,

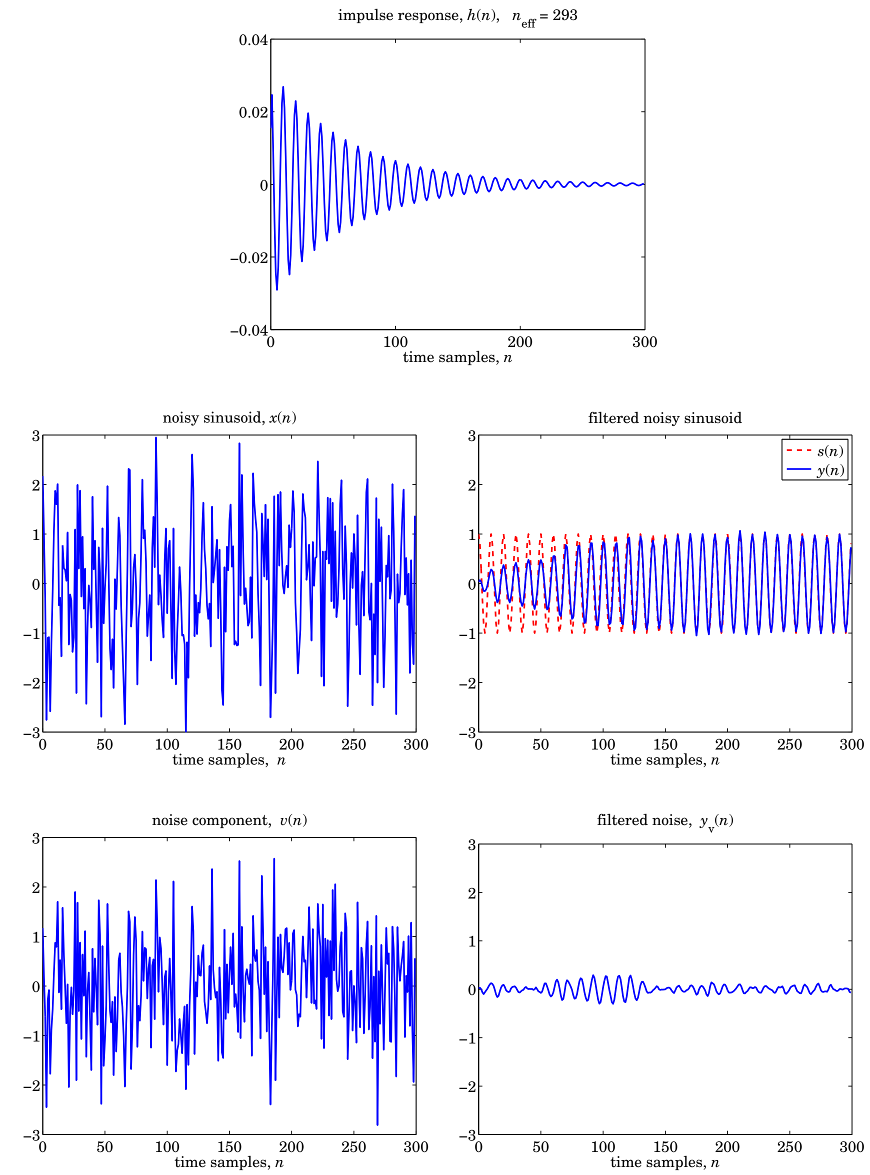
$$y(n) = \cos(\omega_0 n)\cdot p\big(n - n_{\text{gr}}(\omega_0)\big)$$

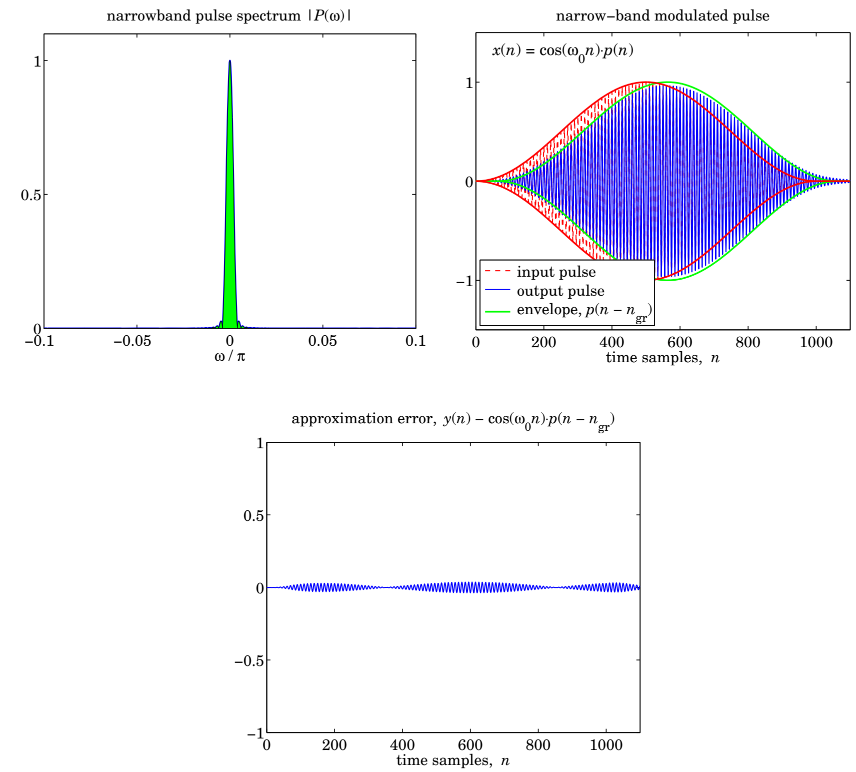
To visualize this result better, add the delayed envelope signal $p\big(n - n_{\text{gr}}(\omega_0)\big)$ to the graph of $y(n)$. Additionally, calculate and plot the error difference signal,

$$y(n) - \cos(\omega_0 n)\cdot p\big(n - n_{\text{gr}}(\omega_0)\big)\,, \quad 0 \le n \le N-1$$

Example Graphs







15.9 Computer Experiment – Time-Bandwidth Tradeoffs

In this experiment, we examine the tradeoffs between filter bandwidth and time constant and steady-state response, that is, the more narrow the bandwidth, the longer the time constant and the longer it takes to reach steady state.

Consider the following notch and peaking filters from Sec. 12.3, with center frequency ω_0 rads/sample, and 3-dB width $\Delta\omega$,

$$\begin{aligned} H_{\text{notch}}(z) &= \frac{1}{1+\beta}\cdot\frac{1-2\cos\omega_0\, z^{-1}+z^{-2}}{1-\dfrac{2\cos\omega_0}{1+\beta}z^{-1}+\dfrac{1-\beta}{1+\beta}z^{-2}} \\ H_{\text{peak}}(z) &= \frac{\beta}{1+\beta}\cdot\frac{1-z^{-2}}{1-\dfrac{2\cos\omega_0}{1+\beta}z^{-1}+\dfrac{1-\beta}{1+\beta}z^{-2}} \end{aligned} \tag{15.9.1}$$

The peaking filter is the same as in Eq. (15.8.2). The parameter β is related to the bandwidth $\Delta\omega$ as in Eq. (15.8.3), and both filters have the same poles as in Eq. (15.8.4),

but with the notch filter having zeros at $\omega = \pm\omega_0$, and the peaking filter having zeros at $\omega = 0, \pi$. Moreover, the two filters have the same time constant as in Eq. (15.8.6), the same group delay as in Eq. (15.8.8), and the same left and right 3-dB frequencies as in Eq. (15.8.9).

The two filters satisfy the following *complementarity* properties for their transfer functions, frequency responses, magnitude-square responses, impulse responses, and noise-reduction ratios, where the peaking filter's impulse response is given by Eq. (15.8.11) and its noise-reduction ratio by Eq. (15.8.13),

$$\begin{aligned} &H_{\text{notch}}(z)+H_{\text{peak}}(z)=1 \\ &H_{\text{notch}}(\omega)+H_{\text{peak}}(\omega)=1 \\ &|H_{\text{notch}}(\omega)|^2+|H_{\text{peak}}(\omega)|^2=1 \\ &h_{\text{notch}}(n)+h_{\text{peak}}(n)=\delta(n) \\ &\mathcal{R}_{\text{notch}}+\mathcal{R}_{\text{peak}}=1 \\ &\mathcal{R}_{\text{notch}}=\frac{1}{1+\beta} \\ &\mathcal{R}_{\text{peak}}=\frac{\beta}{1+\beta} \end{aligned} \tag{15.9.2}$$

Questions

(a) Prove the complementarity properties listed in Eq. (15.9.2).

(b) Consider the following analog signal of duration of 15 seconds defined as three concatenated five-second unity-amplitude sinusoidal signals of frequencies $f_1 = 2$ Hz, $f_2 = 4$ Hz, and $f_3 = 6$ Hz:

$$x_a(t)=\begin{cases}\cos(2\pi f_1 t), & 0\le t<5 \text{ sec}\\ \cos(2\pi f_2 t), & 5\le t<10 \text{ sec}\\ \cos(2\pi f_3 t), & 10\le t<15 \text{ sec}\end{cases}$$

This signal is sampled at a rate of $f_s = 200$ samples/sec, or sampling time interval $T = 1/200 = 0.005$ sec, and denote the resulting sampled times by, $t_n = nT$. In MATLAB, you could construct the vector of times as follows, with $T_{\max} = 15$ sec,

```
tn = 0:T:Tmax;
```

Calculate and plot the sampled signal $x(t_n)$ vs. t_n.

(c) Design a notch filter $H_{\text{notch}}(z)$, operating at the rate $f_s = 200$ Hz, having a notch at $f_2 = 4$ Hz, and 3-dB width of $\Delta f = 1$ Hz, so that its defining parameters are,

$$\omega_0 = \frac{2\pi f_2}{f_s} = 0.04\pi\,, \quad \Delta\omega = \frac{2\pi\Delta f}{f_s} = 0.01\pi$$

This filter is designed to knock out the middle portion of $x(t)$, after its transients die out. Calculate the 40-dB time constant of this filter in seconds, and its left/right 3-dB frequencies in Hz.

Plot the magnitude-squared response $|H_{\text{notch}}(f)|^2$ versus f in the interval $0 \le f \le 10$ Hz. Add to the graph the values corresponding the three input frequencies f_1, f_2, f_3, and also add the 3-dB frequencies connected with a straight-line segment indicating the notch width.

(d) Filter $x(t_n)$ through the filter and plot the resulting output $y(t_n)$ vs. t_n using the same vertical scales as for the previous plot of $x(t_n)$. Observe the transient and steady-state parts in the three input segments and discuss whether the calculated 40-dB time constant reasonably represents the duration of the transients.

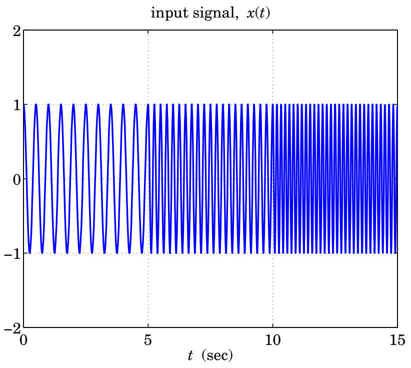
For the first and third 5-sec segment, draw horizontal lines at the expected steady-state amplitude levels as calculated from the frequency response of part (c), that is, at the values of $|H_{\text{notch}}(f_1)|$ and $|H_{\text{notch}}(f_3)|$.

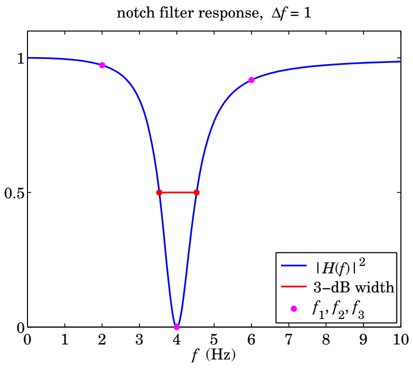
(e) Repeat parts (c,d) for the corresponding complementary peaking filter $H_{\text{peak}}(z)$ that has a peak at $f_2 = 4$ Hz and the same 3-dB width of $\Delta f = 1$ Hz. See example graphs at end.

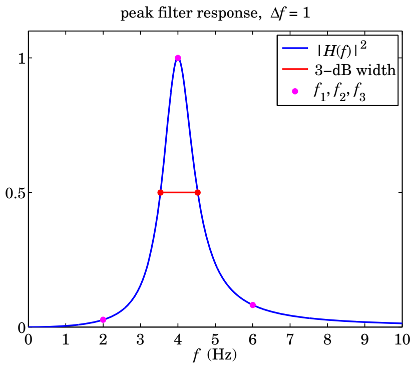
(f) Repeat parts (c,d,e) by redesigning the filters to have the shorter 3-dB width of $\Delta f = 0.3$ Hz, and comment of the tradeoff between filter width and time constant.

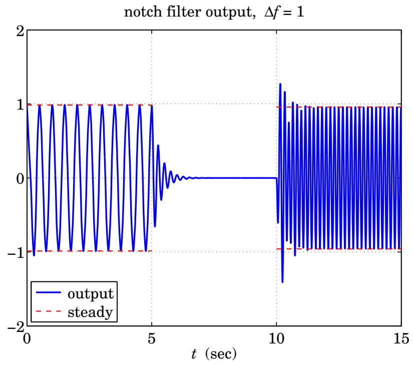
(g) Regarding the noise reduction ratios of the two filters given in Eq. (15.9.2), provide an intuitive explanation of why for narrow bandwidth $\Delta\Omega$, the NRR of the peaking filter, $\mathcal{R}_{\text{peak}}$, is very small, while that of the notch, $\mathcal{R}_{\text{notch}}$, is very large and near its maximum value of unity.

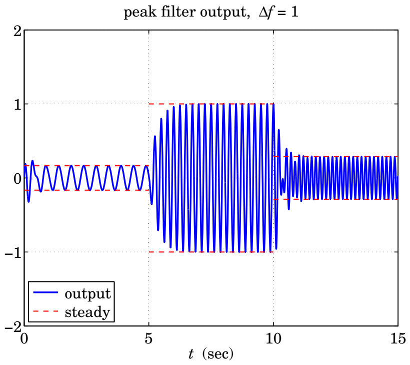
Example Graphs

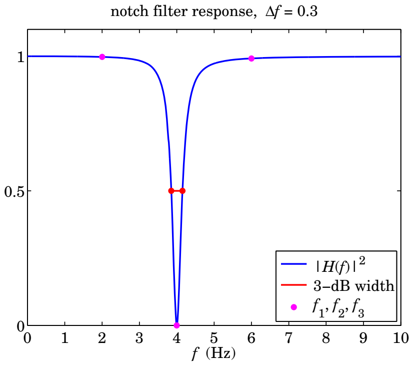


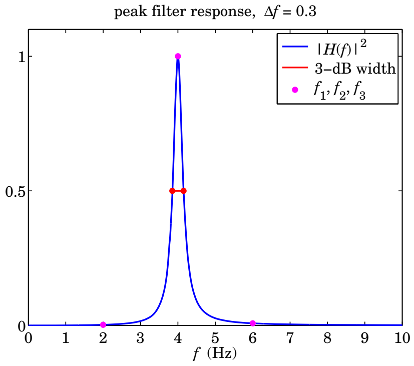


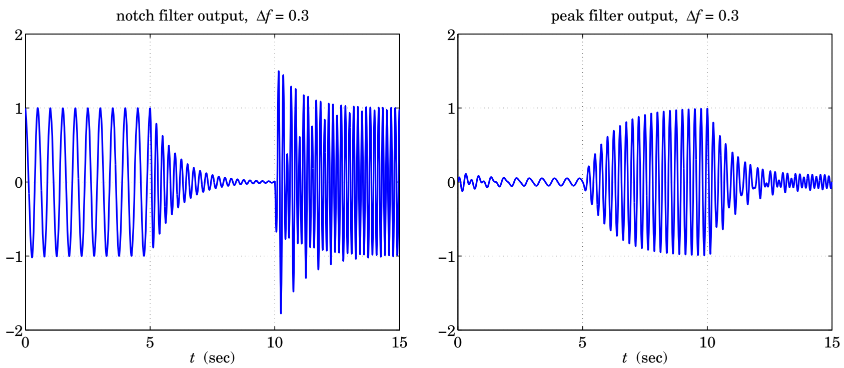












15.10 Computer Experiment – SMA, EMA, PMA, DEMA filters

Technical analysis of financial markets refers to a family of DSP methods and indicators used by stock market traders to make sense of the constantly fluctuating market data and arrive at successful "buy" or "sell" decisions. Both linear and nonlinear filtering methods are used. More details can be found in Chap. 25.

Among the linear filtering methods are smoothing filters that are used to smooth out the daily fluctuations and bring out the trends in the data. The two most commonly used filters are the FIR averaging filter, referred to in the financial context as a *simple moving average* (SMA), and the *exponentially-weighted moving average* (EMA), with both filters discussed earlier in this chapter.

The impulse responses, transfer functions and filtering equations of these filters are as follows. For the SMA case,

$$
\begin{aligned}
& h(n) = \frac{1}{N}\,, \quad 0 \le n \le N-1 \\
& H(z) = \frac{1}{N}\left(1 + z^{-1} + z^{-2} + \cdots + z^{-N+1}\right) = \frac{1}{N}\,\frac{1-z^{-N}}{1-z^{-1}} \\
& y_n = \frac{1}{N}\left(x_n + x_{n-1} + x_{n-2} + \cdots + x_{n-N+1}\right)
\end{aligned}
\tag{15.10.1}
$$

and for EMA,

$$
\begin{aligned}
& h(n) = (1-\lambda)\lambda^n\,, \quad 0 \le n < \infty \\
& H(z) = \frac{1-\lambda}{1-\lambda z^{-1}} \\
& y_n = \lambda y_{n-1} + (1-\lambda)x_n
\end{aligned}
\tag{15.10.2}
$$

Here, x_n denotes price data and the sampling rate is typically "once per day", and y_n denotes the smoothed trend over a period of days. The filter span of the SMA case is N days, while the parameter λ of the EMA case is a forgetting factor such that $0 < \lambda < 1$, which is usually specified in terms an equivalent SMA length N given by,

$$
N = \frac{1+\lambda}{1-\lambda} \quad \Leftrightarrow \quad \lambda = \frac{N-1}{N+1}
\tag{15.10.3}
$$

The typical trading rule used by traders is to "buy" when y_n is rising and y_n lies above x_n, and to "sell" when y_n is falling and y_n lies below x_n.

Unfortunately, these widely used filters have an inherent lag, which can often result in false buy/sell signals. The basic tradeoff is that longer lengths N result in longer lags, but at the same time, the filters become more effective in smoothing out and reducing noise in the data. The group delays of the SMA and EMA filters are as follows at any ω and at DC,

$$
\begin{aligned}
&\text{(SMA)} \qquad n_{\text{gr}}(\omega) = \bar{n} = \frac{N-1}{2} \\
&\text{(EMA)} \qquad n_{\text{gr}}(\omega) = \frac{\lambda(\cos\omega - \lambda)}{1+\lambda^2 - 2\lambda\cos\omega}\,, \quad \bar{n} = n_{\text{gr}}(0) = \frac{\lambda}{1-\lambda}
\end{aligned}
\tag{15.10.4}
$$

The filter lags and noise-reduction ratios will be the *same* for the two filters provided one imposes the condition of Eq. (25.2.3), as is usually done in market trading,

$$
\bar{n} = \frac{N-1}{2} = \frac{\lambda}{1-\lambda} \quad \Leftrightarrow \quad \mathcal{R} = \frac{1}{N} = \frac{1-\lambda}{1+\lambda}
\tag{15.10.5}
$$

The tradeoff is evident, with $\mathcal{R}$ decreasing and $\bar{n}$ increasing with N, or equivalently, with increasing λ towards unity. The time constants of the two filters are as follows, where in the SMA case it is simply the filter order,

$$
n_{\text{eff, SMA}} = N - 1\,, \qquad n_{\text{eff, EMA}} = \frac{\ln\epsilon}{\ln\lambda}
\tag{15.10.6}
$$

There are several methods for reducing the lag, or eliminating it altogether, as we discuss in Chap. 25. Among these are the *predictive FIR* and *double EMA* filters which are widely used.

The predictive FIR filter with zero lag is a special case of a more general length-N predictive filter designed to predict ahead by τ time steps. The impulse response of that filter is, for $n = 0, 1, \dots, N-1$,

$$
h_\tau(n) = h_a(n) + \tau h_b(n) = \frac{2(2N-1-3n)}{N(N+1)} + \tau\,\frac{6(N-1-2n)}{N(N^2-1)}
\tag{15.10.7}
$$

This filter has negative group-delay at DC equal to $-\tau$, that is, $\bar{n} = -\tau$, thus, it has a positive time advance of $+\tau$ corresponding to predicting ahead by τ units.

Actually, the time "advance" τ can be non-integer, positive, or negative. Positive τs correspond to prediction or forecasting, negative τs to delaying or lag. In fact, the SMA filter is a special case of Eq. (15.10.7) for the particular choice of $\tau = -(N-1)/2$.

The filters $h_\tau(n)$ are very flexible and useful in the trading context, and are actually the *optimal filters* that have *minimum* noise-reduction ratio, subject to the two constraints of unity DC gain and lag equal to $-\tau$, that is, for fixed N, $h_\tau(n)$ is the solution of the optimization problem:

$$\mathcal{R}_\tau = \sum_{n=0}^{N-1} h_\tau^2(n) = \min, \quad \text{subject to} \quad \begin{cases} \displaystyle\sum_{n=0}^{N-1} h_\tau(n) = 1 \\ \displaystyle\sum_{n=0}^{N-1} n h_\tau(n) = -\tau \end{cases} \tag{15.10.8}$$

The minimized value of $\mathcal{R}_\tau$ is,

$$\mathcal{R}_\tau = \frac{1}{N} + \frac{3(N-1+2\tau)^2}{N(N^2-1)} \tag{15.10.9}$$

The zero-lag predictive moving average (PMA) filter corresponds to the case, $\tau = 0$, that is, its impulse response is simply the $h_a(n)$ part of Eq. (15.10.7),

$$h_{\text{pma}}(n) = \frac{2(2N-1-3n)}{N(N+1)}, \quad n = 0, 1, \dots, N-1 \tag{15.10.10}$$

Its noise-reduction ratio is obtained by setting $\tau = 0$ in Eq. (25.3.5),

$$\mathcal{R}_{\text{pma}} = \frac{1}{N} + \frac{3(N-1)^2}{N(N^2-1)} = \frac{4N-2}{N(N+1)} \tag{15.10.11}$$

It decreases with increasing N, just not as fast as in the SMA/EMA cases.

The double EMA (DEMA) filter is another option for a zero-lag filter and is obtained by applying Tukey's twicing rule of Eq. (15.7.10) to the plain EMA filter of Eq. (15.10.2). Thus, its transfer function becomes,

$$\begin{aligned} H_{\text{ema}}(z) &= \frac{1-\lambda}{1-\lambda z^{-1}} \\ H_{\text{dema}}(z) &= 2H_{\text{ema}}(z) - H_{\text{ema}}^2(z) = \frac{1-\lambda^2-2\lambda(1-\lambda)z^{-1}}{(1-\lambda z^{-1})^2} \end{aligned} \tag{15.10.12}$$

Its DC lag is zero, $\bar{n} = 0$, while its impulse response and noise-reduction ratio are found to be,

$$\begin{aligned} h_{\text{dema}}(n) &= \left[1-\lambda^2-(1-\lambda)^2 n\right]\lambda^n u(n) \\ \mathcal{R}_{\text{dema}} &= \frac{(1-\lambda)(1+4\lambda+5\lambda^2)}{(1+\lambda)^3} \end{aligned} \tag{15.10.13}$$

Questions

(a) First, prove Eqs. (15.10.4)–(15.10.13).

(b) Construct the following length-301 piecewise linear signal,

$$s(n) = \begin{cases} 20 + 0.8n, & 0 \le n < 75 \\ 80 - 0.3(n-75), & 75 \le n < 225 \\ 35 + 0.8(n-225), & 225 \le n \le 300 \end{cases} \tag{15.10.14}$$

For $N = 24$ and λ defined through Eq. (15.10.3), filter $s(n)$ through an SMA and an EMA filter, and plot the corresponding outputs, $y_{\text{sma}}(n)$ and $y_{\text{ema}}(n)$.

Comment on whether the observed lag is consistent with Eq. (15.10.5), and whether the observed transients at the slope changes are consistent with Eq. (15.10.6) (with $\epsilon = 10^{-2}$.)

(c) Repeat part (a) using the PMA and DEMA filters with the same values of N and λ, and plot the corresponding outputs, $y_{\text{pma}}(n)$ and $y_{\text{dema}}(n)$.

Observe the elimination of lag, and comment on whether the observed transients at the slope changes are consistent with Eq. (15.10.6).

(d) The PMA filter of Eq. (15.10.7) has two parts, $h_\tau(n) = h_a(n) + \tau h_b(n)$, where $h_a(n)$ represents the zero-lag portion. The part, $h_b(n)$, is a slope-tracking filter that tracks the changing slope of the input,

$$h_b(n) = \frac{6(N-1-2n)}{N(N^2-1)}, \quad n = 0, 1, \dots, N-1, \quad \text{(PMA slope filter)} \tag{15.10.15}$$

Similarly, in the DEMA case, the corresponding slope filter is constructed by the following rule, derived in Chap. 25,

$$H_b(z) = \frac{1-\lambda}{\lambda}\left[H_{\text{ema}}(z) - H_{\text{ema}}^2(z)\right], \quad \text{(DEMA slope filter)} \tag{15.10.16}$$

The signal that represents the slope of the piecewise linear input $s(n)$ is obtained by inspection from Eq. (15.10.14),

$$s_b(n) = \begin{cases} 0.8, & 0 \le n < 75 \\ -0.3, & 75 \le n < 225 \\ 0.8, & 225 \le n \le 300 \end{cases} \tag{15.10.17}$$

Filter $s_b(n)$ through the PMA and DEMA slope filters of Eqs. (15.10.15) and (15.10.16), and plot the resulting outputs together with $s_b(n)$ on a single graph. Observe how, up to transients, both filters attempt to track the changing slope. Such slope filter outputs are also used as indicators in financial market trading.

(e) Construct a length-301 noisy version of $s(n)$ by

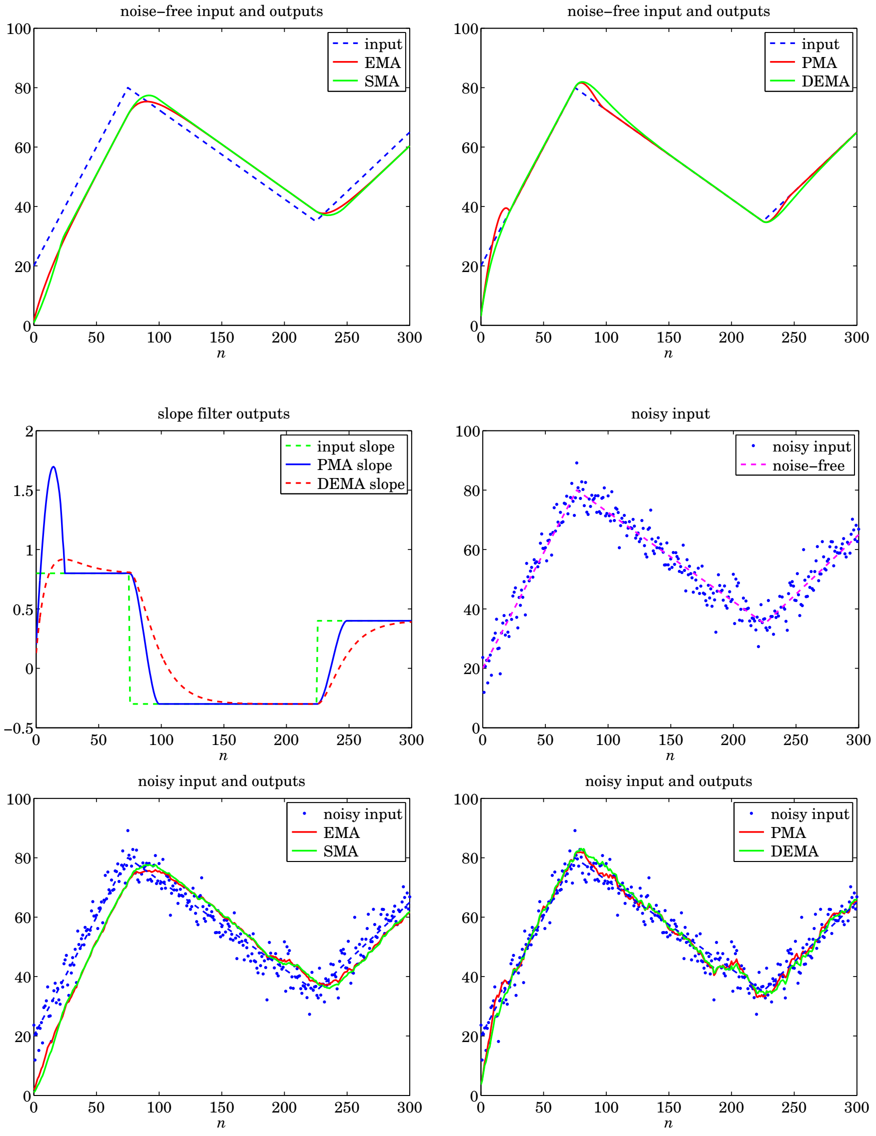
$$x(n) = s(n) + v(n) \tag{15.10.18}$$

where $v(n)$ is a length-301 segment of a zero-mean white-noise gaussian signal of variance $\sigma_v^2 = 16$. Make sure to initialize the random number generator with the same seed each time you run your program in order to always generate the same random signal. The MATLAB code would be something like,

```
% seed = ...                    % define your seed here
randn('state', seed);           % initialize generator
sigma = 4;                      % variance sigma^2 = 16
v = sigma * randn(size(s));     % noise signal, same size as s(n)
x = s + v;                      % noisy input
```

Repeat parts (a,b) using $x(n)$ as the input to the SMA, EMA, PMA, DEMA filters. Some example graphs are below.

Example Graphs


noise–free input and outputs

noise–free input and outputs

slope filter outputs

noisy input

noisy input and outputs

noisy input and outputs

15.11 Notch and Comb Filters for Periodic Signals

Two special cases of the signal enhancement/noise reduction problem arise when:

1. The noise signal $v(n)$ in Eq. (15.1.1) is *periodic*. Its spectrum is concentrated at the harmonics of a fundamental frequency. The noise reduction filter is an ideal *notch filter* with notches at these harmonics, as shown in Fig. 15.11.1.

2. The desired signal $s(n)$ is *periodic* and the noise is a wideband signal. Now, the signal enhancement filter is an ideal *comb filter* with peaks at the harmonics of the desired signal, as shown in Fig. 15.11.2.

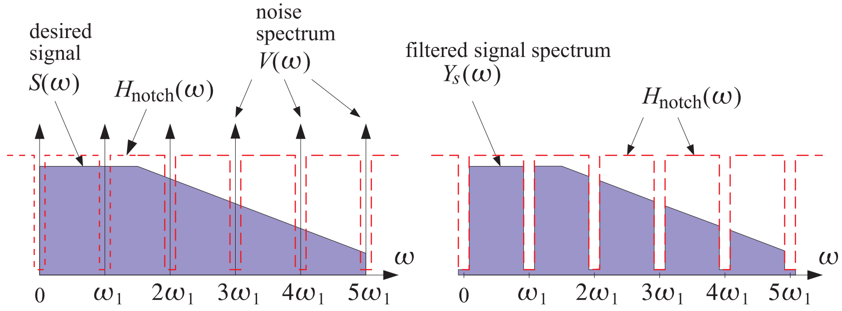


Fig. 15.11.1 Notch filter for reducing periodic interference.

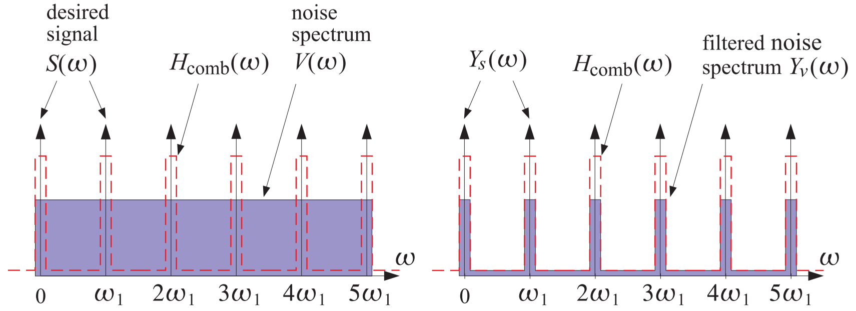


Fig. 15.11.2 Comb filter for enhancing periodic signal.

The ideal notch and comb filters of Figs. 15.11.1 and 15.11.2 are *complementary filters*, in the sense that one is zero where the other is one, so that their frequency responses add up to unity:

$$H_{\text{notch}}(\omega)+H_{\text{comb}}(\omega)=1 \tag{15.11.1}$$

A typical application of the notch case is the 60 Hz power-frequency interference picked up through insufficiently shielded instrumentation. This problem is especially acute in biomedical applications, such as measuring an electrocardiogram (ECG) by chest electrodes—a procedure which is prone to such interference. The literature on biomedical applications of DSP is extensive, with many filter design methods that are specialized for efficient microprocessor implementations [174–192].

Let f_1 be the fundamental frequency of the periodic noise (e.g., $f_1 = 60$ Hz), or, in radians per sample $\omega_1 = 2\pi f_1/f_s$. If only *one* notch at f_1 must be canceled, then a single-notch filter, such as that given in Eqs. (16.2.22) and (16.2.23) of Section 16.2.2, will be adequate.

Example 15.11.1: *Single-notch filter for ECG processing.* It is desired to design a single-notch filter to cancel the 60 Hz power-frequency pickup in an ECG recording. The ECG is sampled at a rate of 1 kHz, and we assume that the beat rate is 2 beats/sec. Thus, there are 500 samples in each beat.

The digital notch frequency will be:

$$\omega_1 = \frac{2\pi f_1}{f_s} = \frac{2\pi\, 60}{1000} = 0.12\pi \ \text{ radians/sample}$$

Assuming a Q-factor of 60 for the notch filter, we have a 3-dB width:

$$\Delta f = \frac{f_1}{Q} = 1 \ \text{Hz} \qquad \Rightarrow \qquad \Delta\omega = \frac{2\pi\Delta f}{f_s} = 0.002\pi \ \text{ radians/sample}$$

Using the design equations (16.2.22) and (16.2.23), we find the notch filter:

$$H(z) = 0.99687\,\frac{1 - 1.85955z^{-1} + z^{-2}}{1 - 1.85373z^{-1} + 0.99374z^{-2}}$$

Figure 15.11.3 shows three beats of a simulated ECG with 60 Hz noise generated by

$$x(n) = s(n) + 0.5\cos(\omega_1 n), \qquad n = 0, 1, \dots, 1500$$

The ECG signal $s(n)$ was normalized to maximum value of unity (i.e., unity QRS-peak). Thus, the noise amplitude is 50% the QRS-peak amplitude. Fig. 15.11.3 shows the noisy signal $x(n)$ and the notch filter's magnitude characteristics. The filtered signal $y(n)$ is juxtaposed next to the noise-free ECG for reference.

Except for the initial transients, the filter is very effective in removing the noise. The filter's time constant can be estimated from the filter pole radius, that is, from the last denominator coefficient $a_2 = R^2$:

$$R^2 = 0.99374 \qquad \Rightarrow \qquad R = 0.99686$$

which gives for the 1% time constant $n_{\text{eff}} = \ln(0.01)/\ln(R) = 1464$ samples. In seconds, this is $\tau = n_{\text{eff}}T = 1464/1000 = 1.464$ sec, where $T = 1/f_s = 1$ msec. □

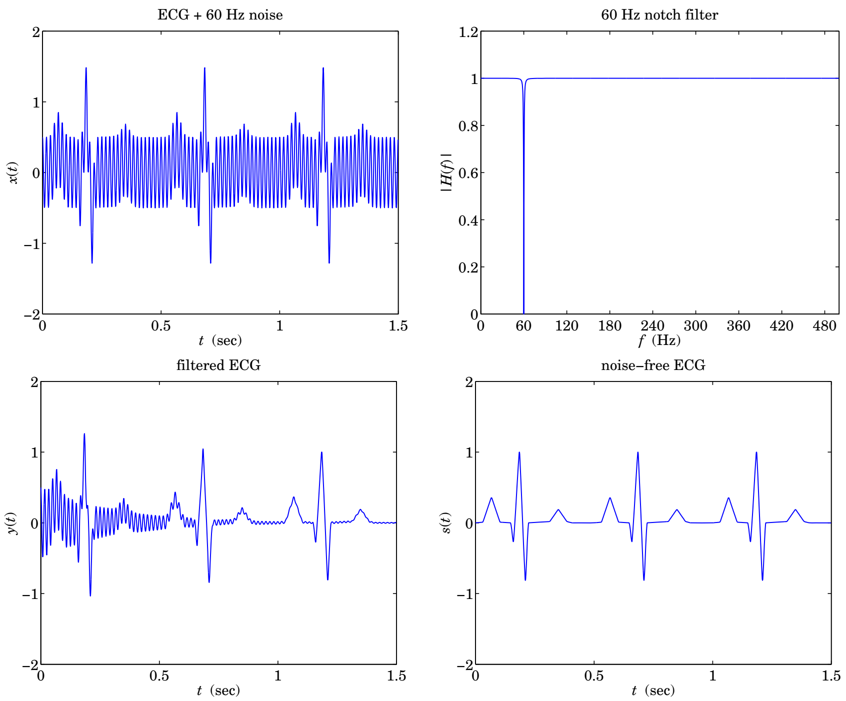


Fig. 15.11.3 Removing 60 Hz noise from ECG signal.

If *all* the harmonics of f_1 must be canceled, then it proves convenient to choose the sampling rate to be a *multiple* of the fundamental frequency, that is, $f_s = Df_1$. Then, the noise harmonics will occur at the Dth roots-of-unity frequencies:

$$f_k = kf_1 = k\frac{f_s}{D}, \quad \text{or,} \quad \omega_k = k\omega_1 = \frac{2\pi k}{D}, \qquad k = 0, 1, \dots, D-1 \tag{15.11.2}$$

In this case, we can use the general procedure discussed in Section 6.4.3. The notch polynomial having as roots the Dth roots of unity, $z_k = e^{j\omega_k} = e^{2\pi jk/D}$ is given by $N(z) = 1 - z^{-D}$. The corresponding multi-notch filter is obtained by *sharpening* the zeros, that is, putting poles *behind* the zeros by the replacement $z \to z/\rho$:

$$\boxed{H_{\text{notch}}(z) = \frac{bN(z)}{N(\rho^{-1}z)} = b\,\frac{1 - z^{-D}}{1 - az^{-D}}}\,, \qquad b = \frac{1+a}{2} \tag{15.11.3}$$

where $a = \rho^D$.

The choice $b = (1+a)/2$ ensures that $H(z)$ is normalized to unity half-way between the notches, that is, at $\omega_k = (2k+1)\pi/D$.

The value of the parameter a depends on the desired 3-dB width $\Delta\omega = 2\pi\Delta f/f_s$ of the notch dips. Using the bilinear transformation method[†] of Chapter 12, we obtain the following design equations, for a given $\Delta\omega$:

$$\boxed{\beta = \tan\Big(\frac{D\Delta\omega}{4}\Big),\quad a = \frac{1-\beta}{1+\beta},\quad b = \frac{1}{1+\beta}} \tag{15.11.4}$$

Because a must be in the interval $0 \le a < 1$, we find the restriction $0 < \beta \le 1$, which translates into the following bound on the desired 3-dB width: $D\Delta\omega/4 \le \pi/4$, or

$$\Delta\omega \le \frac{\pi}{D} \quad\Rightarrow\quad \Delta f \le \frac{f_s}{2D} \tag{15.11.5}$$

Its maximum value $\Delta\omega = \pi/D$ corresponds to $\beta = 1$ and $a = 0$.

It follows from the bilinear transformation that the magnitude response squared of the filter (15.11.3) can be written in the simple form:

$$\boxed{|H_{\text{notch}}(\omega)|^2 = \frac{\tan^2(\omega D/2)}{\tan^2(\omega D/2)+\beta^2}} \tag{15.11.6}$$

Example 15.11.2: *Multi-notch filter design.* As an example of the above design method, consider the case $D = 10$. According to Eq. (15.11.5), the 3-dB width is allowed to be in the range $0 < \Delta\omega \le \pi/D = 0.1\pi$.

Using Eqs. (15.11.4), we design the filter for the following three values of $\Delta\omega$:

$$\begin{array}{llllll}
\Delta\omega = 0.1\pi & \Rightarrow & \beta = 1 & a = 0 & b = 0.5 \\
\Delta\omega = 0.05\pi & \Rightarrow & \beta = 0.4142 & a = 0.4142 & b = 0.7071 \\
\Delta\omega = 0.0125\pi & \Rightarrow & \beta = 0.0985 & a = 0.8207 & b = 0.9103
\end{array}$$

corresponding to the three transfer functions:

$$\begin{aligned}
H_{\text{notch}}(z) &= 0.5(1-z^{-10}) \\
H_{\text{notch}}(z) &= 0.7071\,\frac{1-z^{-10}}{1-0.4142z^{-10}} \\
H_{\text{notch}}(z) &= 0.9103\,\frac{1-z^{-10}}{1-0.8207z^{-10}}
\end{aligned}$$

The magnitude squared responses, $|H_{\text{notch}}(\omega)|^2$, are plotted in Fig. 15.11.4. □

Example 15.11.3: *Multi-notch filter for ECG processing.* To illustrate the filtering operation with the notch filter of the type (15.11.3), consider the following simulated ECG signal, generated by adding a 60 Hz *square wave* to two beats of a noise-free ECG:

$$x(n) = s(n)+v(n),\qquad n = 0,1,\dots,1199$$

[†] Here, the highpass analog filter $s/(s+\beta)$ is transformed by $s = (1-z^{-D})/(1+z^{-D})$.

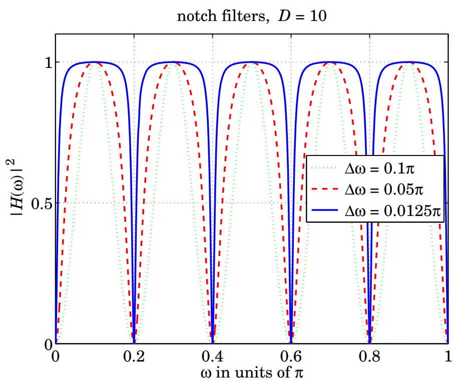


Fig. 15.11.4 Multi-notch filters of different widths ($\Delta\omega$ in units of π).

where $s(n)$ is a one-beat-per-second ECG signal sampled at a rate of 600 Hz, thus, having 600 samples per beat. The QRS-peak is normalized to unity as in Example 15.11.1. The square wave noise signal $v(n)$ has period

$$D = \frac{f_s}{f_1} = \frac{600}{60} = 10$$

and is defined as follows:

$$v(n) = 1 + 0.5w(n),\qquad w(n) = [1,1,1,1,1,-1,-1,-1,-1,-1,\dots]$$

where $w(n)$ alternates between $+1$ and -1 every 5 samples. The alternating square wave $w(n)$ has only odd harmonics, namely, f_1, $3f_1$, $5f_1$, and so on.

This particular $v(n)$ is used here only for illustrating the behavior of a multi-notch filter and does not represent any real-life noise. The nonzero mean of $v(n)$ is meant to imitate a typical *baseline shift* that can occur in ECG measurements, in addition to the 60 Hz harmonics.

The filter (15.11.3) was designed by requiring that its Q-factor be 80. This gives the 3-dB width:

$$\Delta f = \frac{f_1}{Q} = \frac{60}{80} = 0.75\ \text{Hz} \quad\Rightarrow\quad \Delta\omega = \frac{2\pi\Delta f}{f_s} = 0.0025\pi\ \text{rads/sample}$$

The design equations (15.11.4) give: $a = 0.9615$, $b = 0.9807$, with a resulting transfer function:

$$H_{\text{notch}}(z) = 0.9807\,\frac{1-z^{-10}}{1-0.9615z^{-10}}$$

Its magnitude response is similar to those of Fig. 15.11.4, but narrower. It can be implemented in its canonical form using the 11-dimensional vector of internal states $\mathbf{w} = [w_0, w_1,\dots,w_{10}]$ by the sample processing algorithm:

```
for each input sample x do:
    $w_0 = 0.9615\, w_{10} + 0.9807\, x$
    $y = w_0 - w_{10}$
    delay(10, w)
```

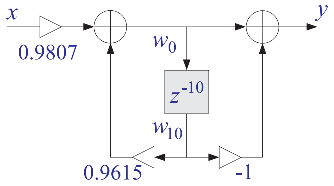


Fig. 15.11.5 shows the input $x(n)$ and the filtered output $y(n)$. To improve the visibility of the graphs, the two beats, for $0 \le t \le 1$ sec and $1 \le t \le 2$ sec, have been split into two side-by-side graphs.

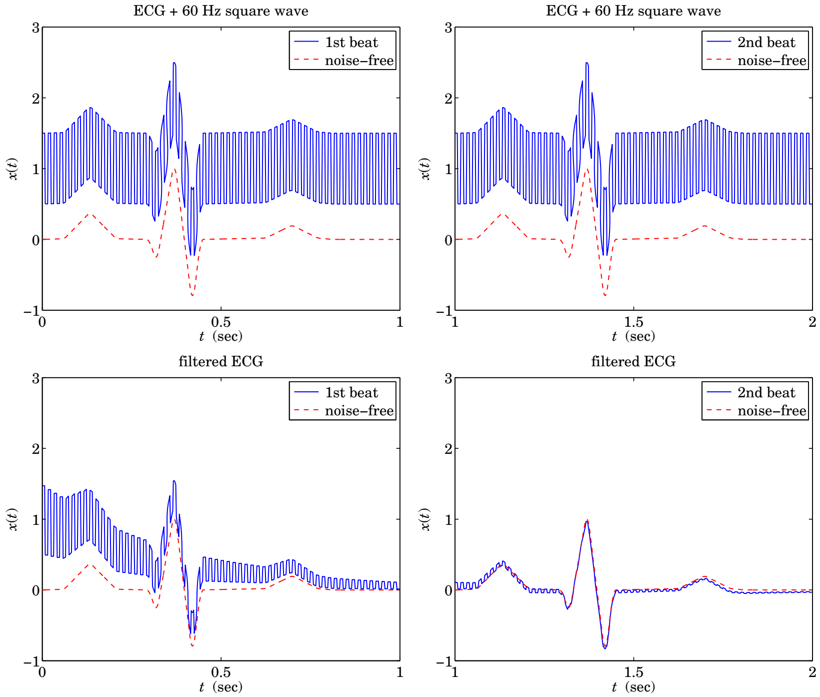


Fig. 15.11.5 Eliminating baseline shifts and 60 Hz harmonics from ECG.

Notice how the filter's zero at DC eliminates the baseline shift, while its notches at the 60 Hz harmonics eliminate the alternating square wave.

A single-notch filter at 60 Hz would not be adequate in this case, because it would not remove the DC and the higher harmonics of the noise. For example, using the method of Example 15.11.1, the single notch filter with the same $Q = 80$ and width Δf as the above multi-notch filter, is found to be:

$$H_1(z) = 0.99609\,\frac{1 - 1.61803z^{-1} + z^{-2}}{1 - 1.61170z^{-1} + 0.99218z^{-2}}$$

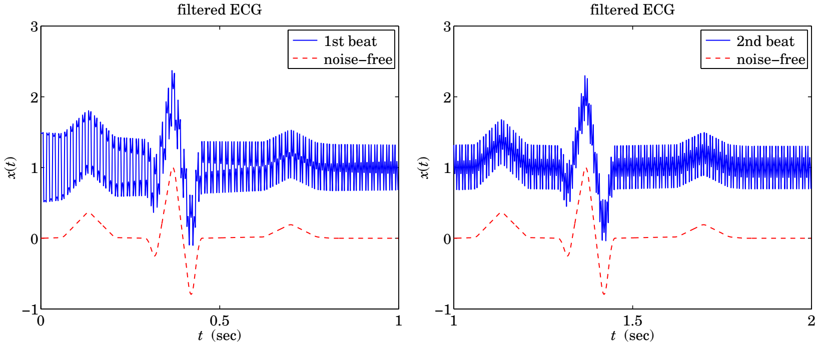


Fig. 15.11.6 Inadequate filtering by the single-notch filter $H_1(z)$.

Fig. 15.11.6 shows the filtered ECG in this case. Only the 60 Hz harmonic of the square wave noise is removed. The DC and the higher harmonics at $3f_1$, $5f_1$, and so on, are still in the output.

Actually, for this example, the highest harmonic is $5f_1$ and coincides with the Nyquist frequency $5f_1 = f_s/2 = 300$ Hz. Because D is even, all the higher odd harmonics will be aliased with one of the three odd Nyquist-interval harmonics: f_1, $3f_1$, or, $5f_1$.

We can attempt to cancel these additional harmonics by designing separate notch filters for each one. For example, using a *common* width $\Delta f = 0.75$ Hz, we design the following notch filters:

$$\begin{aligned}
H_3(z) &= 0.99609\,\frac{1 + 0.61803z^{-1} + z^{-2}}{1 + 0.61562z^{-1} + 0.99218z^{-2}} && (\text{notch at } 3f_1)\\
H_5(z) &= 0.99609\,\frac{1 + z^{-1}}{1 + 0.99218z^{-1}} && (\text{notch at } 5f_1)\\
H_0(z) &= 0.99609\,\frac{1 - z^{-1}}{1 - 0.99218z^{-1}} && (\text{notch at } 0)
\end{aligned}$$

Fig. 15.11.7 shows the output of the cascaded filter $H_1(z)H_3(z)H_5(z)$, which removes completely all the harmonics in the square wave, except DC. That can be removed by sending the output through the DC notch filter $H_0(z)$.

Note that the notch filters at DC and Nyquist frequency are first-order filters of the form:

$$H_0(z) = b\,\frac{1 - z^{-1}}{1 - az^{-1}}\,, \qquad H_5(z) = b\,\frac{1 + z^{-1}}{1 + az^{-1}}$$

These are limiting cases of the designs of Eq. (16.2.22) for $\omega_0 = 0$ and $\omega_0 = \pi$. In both cases, $b = 1/(1 + \tan(\Delta\omega/2))$ and $a = 2b - 1$. □

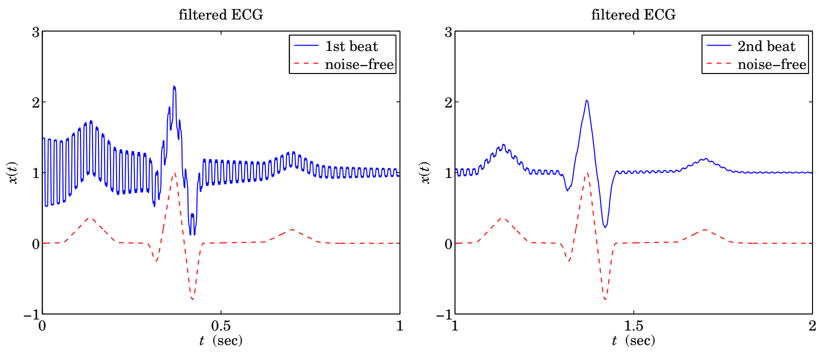


Fig. 15.11.7 Output from the cascade filter $H_1(z)H_3(z)H_5(z)$.

The multi-notch filter (15.11.3) can be obtained from the first-order filter $H(z)= b(1-z^{-1})/(1-az^{-1})$ by the substitution

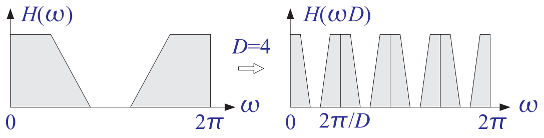
$$\boxed{z \longrightarrow z^D} \tag{15.11.7}$$

that is,

$$H(z)= b\,\frac{1-z^{-1}}{1-az^{-1}} \quad\longrightarrow\quad H(z^D)= b\,\frac{1-z^{-D}}{1-az^{-D}} \tag{15.11.8}$$

The effect of this substitution is the D-fold replication of the spectrum of the original filter. Indeed, in the frequency domain Eq. (15.11.7) gives:

$$H(\omega)\longrightarrow H(\omega D),$$



This transformation *shrinks* the original spectrum $H(\omega)$ by a factor of D and replicates it D times. Because the spectrum $H(\omega)$ has period $0\le\omega\le 2\pi$, the new spectrum $H(\omega D)$ will have period $0\le\omega D\le 2\pi$, which becomes the scaled period $0\le\omega\le 2\pi/D$ fitting exactly D times into the new Nyquist interval $0\le\omega\le 2\pi$.

The first-order filter in Eq. (15.11.8) has a single notch at $\omega=0$, which gets replicated D times and becomes a multi-notch filter.

The replicating transformation (15.11.7) can also be applied to any *narrow lowpass* filter, replicating it D times into a *comb filter*. For example, applying it to the filter of Example 15.2 we get:

$$H(z)=\frac{1-a}{1-az^{-D}} \tag{15.11.9}$$

which has a comb structure similar to that of the plain reverberator shown in Fig. 16.2.7. Similarly, the transformation (15.11.7) applied to the filter of Eq. (15.2.2), gives the following comb filter with unity-gain peaks at $\omega_k = 2k\pi/D$ and zeros at $\omega_k=(2k+1)\pi/D$:

$$\boxed{H_{\text{comb}}(z)= b\,\frac{1+z^{-D}}{1-az^{-D}}}\,,\qquad b=\frac{1-a}{2} \tag{15.11.10}$$

This filter can also be designed directly using the bilinear transformation method[†] of Chapter 12. Given a prescribed 3-dB width for the peaks, $\Delta\omega$, the filter parameters can be calculated from the design equations:

$$\boxed{\beta=\tan\Big(\frac{D\Delta\omega}{4}\Big),\quad a=\frac{1-\beta}{1+\beta},\quad b=\frac{\beta}{1+\beta}} \tag{15.11.11}$$

where, as in Eq. (15.11.4), the width is constrained to be in the interval: $0\le\Delta\omega\le\pi/D$. Like Eq. (15.11.6), the magnitude response squared of (15.11.10) can be expressed simply in the form:

$$\boxed{|H_{\text{comb}}(\omega)|^2=\frac{\beta^2}{\tan^2(\omega D/2)+\beta^2}} \tag{15.11.12}$$

The comb and notch filters of Eqs. (15.11.10) and (15.11.3) are complementary in the sense of Eq. (15.11.1); indeed, we have the identity in z:

$$H_{\text{comb}}(z)+H_{\text{notch}}(z)=\frac{1-a}{2}\,\frac{1+z^{-D}}{1-az^{-D}}+\frac{1+a}{2}\,\frac{1-z^{-D}}{1-az^{-D}}=1$$

It follows by inspecting Eqs. (15.11.6) and (15.11.12) that their magnitude responses squared also add up to one:

$$|H_{\text{comb}}(\omega)|^2+|H_{\text{notch}}(\omega)|^2=1 \tag{15.11.13}$$

This implies that both filters have the same width, as seen in the design equations (15.11.4) and (15.11.11). But, how is it possible to satisfy simultaneously Eq. (15.11.13) and $H_{\text{comb}}(\omega)+H_{\text{notch}}(\omega)=1$? This happens because their phase responses differ by 90^{o}. Indeed, it is left as an exercise to show that:

$$H_{\text{comb}}(\omega)=jH_{\text{notch}}(\omega)\tan(\omega D/2)/\beta$$

Example 15.11.4: *Comb filter design.* As a design example, consider the case $D=10$. Using Eqs. (15.11.11), we design the filter for the following three values of $\Delta\omega$:

$\Delta\omega=0.1\pi$	$\Rightarrow$	$\beta=1$	$a=0$	$b=0.5$
$\Delta\omega=0.05\pi$	$\Rightarrow$	$\beta=0.4142$	$a=0.4142$	$b=0.2929$
$\Delta\omega=0.0125\pi$	$\Rightarrow$	$\beta=0.0985$	$a=0.8207$	$b=0.0897$

corresponding to the three transfer functions:

[†] Here, the lowpass analog filter $\beta/(s+\beta)$ is transformed by $s=(1-z^{-D})/(1+z^{-D})$.

$$H_{\text{comb}}(z) = 0.5(1 + z^{-10})$$

$$H_{\text{comb}}(z) = 0.2929\,\frac{1 + z^{-10}}{1 - 0.4142z^{-10}}$$

$$H_{\text{comb}}(z) = 0.0897\,\frac{1 + z^{-10}}{1 - 0.8207z^{-10}}$$

The magnitude squared responses, $|H(\omega)|^2$, are plotted in Fig. 15.11.8. □

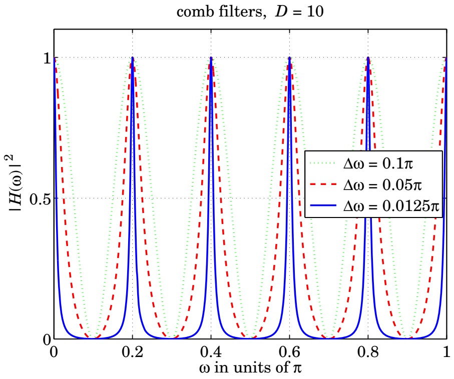


Fig. 15.11.8 Comb filters of different widths ($\Delta\omega$ in units of π).

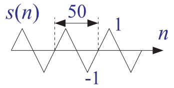
Either comb filter, (15.11.9) or (15.11.10), can be used to enhance a periodic signal buried in white noise. Their NRRs are, $NRR = (1-a)/(1+a)$ and $NRR = (1-a)/2$, respectively. This follows from the property that the substitution (15.11.7) leaves the NRR *unchanged*. Indeed, in the time domain the transformation is equivalent to inserting $D-1$ zeros between the original impulse response samples:

$$\mathbf{h} = [h_0, h_1, h_2, \dots] \longrightarrow \mathbf{h} = [h_0, 0, 0, \dots, 0, h_1, 0, 0, \dots, 0, h_2, \dots] \tag{15.11.14}$$

and, therefore, the quantity $\sum h_n^2$ remains invariant.

Example 15.11.5: *Comb filter for periodic signal enhancement.* To illustrate the noise reduction capability of the comb filter (15.11.10), consider the following signal of length 2000:

$$x(n) = s(n) + v(n), \qquad n = 0, 1, \dots, 1999,$$



where $s(n)$ is a periodic *triangular* wave of period $D = 50$, linearly alternating between ± 1 every 25 samples. Thus, there are 40 periods in $x(n)$. The noise signal $v(n)$ is a zero-mean, white Gaussian noise of rms amplitude (i.e., standard deviation) equal to 0.5, that is, 50 percent of the triangular wave.

The width of the comb filter is chosen to be $\Delta\omega = 0.0008\pi$ radians/sample, with corresponding Q-factor of:

$$Q = \frac{\omega_1}{\Delta\omega} = \frac{2\pi/D}{\Delta\omega} = 50$$

Using the design equations (15.11.11), we find $a = 0.9391$ and $b = 0.0305$, and the transfer function:

$$H_{\text{comb}}(z) = 0.0305\,\frac{1 + z^{-50}}{1 - 0.9391z^{-50}}$$

The filter's magnitude response $|H_{\text{comb}}(\omega)|^2$ and its canonical realization are shown in Fig. 15.11.9. The peaks are at $\omega_k = 2k\pi/50$, and the zeros at $\omega_k = (2k+1)\pi/50$.

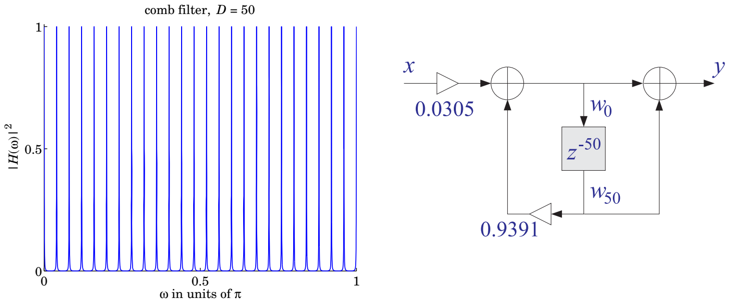


Fig. 15.11.9 Comb filter with $D = 50$ and $Q = 50$.

The canonical form uses a 51-dimensional state vector $\mathbf{w} = [w_0, w_1, \dots, w_{50}]$ to implement the delay z^{-50}. The corresponding sample processing algorithm can be formulated with a linear or a circular delay line, as follows:

```
for each input sample x do:
    $w_0 = 0.9391\,w_{50} + 0.0305\,x$
    $y = w_0 + w_{50}$
    delay(50, w)
```

```
for each input sample x do:
    $s_{50} = \text{tap}(50, \mathbf{w}, p, 50)$
    $s_0 = 0.9391\,s_{50} + 0.0305\,x$
    $y = s_0 + s_{50}$
    $*p = s_0$
    cdelay(50, w, &p)
```

where p is a circular pointer to the linear buffer $\mathbf{w}$, and s_{50}, s_0 denote the 50th and 0th components of the circular state vector pointed to by p.

Fig. 15.11.10 shows the input $x(n)$ and the filtered output $y(n)$. For plotting purposes, the signals have been split into two consecutive segments of length-1000.

The noise reduction ratio of this filter is $NRR = (1-a)/2 = 0.0305$, which corresponds to a $10\log_{10}(1/NRR) = 15.16$ dB improvement of the SNR, or equivalently, to a suppression of the rms noise value by a factor of $1/\sqrt{NRR} = 5.7$. □

The replicating transformation (15.11.7) can also be applied to the FIR averager filter of Sec. 15.5. The resulting periodic comb filter is equivalent to the method of *signal averaging* and is discussed further in Section 27.5.

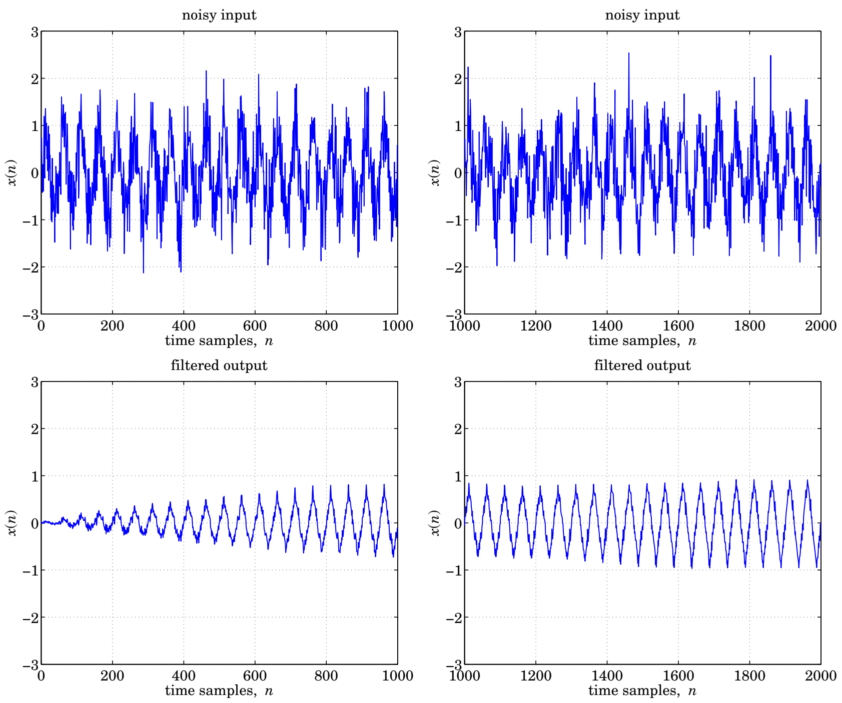


Fig. 15.11.10 Periodic signal enhancement by comb filtering.

15.12 Line and Frame Combs for Digital TV

Another application of notch and comb filters is in the case when *both* signals $s(n)$ and $v(n)$ in Eq. (15.1.1) are *periodic* and must be separated from each other.

To extract $s(n)$, one may use either a comb filter with peaks at the harmonics of $s(n)$, or a notch filter at the harmonics of $v(n)$. Similarly, to extract $v(n)$, one may use a comb at the harmonics of $v(n)$, or a notch at the harmonics of $s(n)$. For the method to work, the harmonics of $s(n)$ may not coincide with the harmonics of $v(n)$.

A major application of this idea is in color TV, digital videodisc systems, and proposed HDTV systems [193–213]. The notch/comb filters are used to separate the luminance (black & white) and chrominance (color) signals from the composite video signal, and also to reduce noise.

Consider a scanned two-dimensional still picture of horizontal and vertical dimensions a and b, as shown in Fig. 15.12.1, and assume there are N horizontal scan lines. If T_H is the time to scan one line, then the time to scan the complete picture (i.e., one frame) is equal to the scanning time for N lines, that is, (ignoring horizontal retrace and blanking times):

$$\boxed{T_F = NT_H} \tag{15.12.1}$$

The quantities T_H and T_F are called the *line and frame delays*, and their inverses are the *line and frame rates*:

$$f_H = \frac{1}{T_H}, \quad f_F = \frac{1}{T_F} \quad \Rightarrow \quad \boxed{f_H = Nf_F} \tag{15.12.2}$$

The frequencies f_H and f_F are related to the horizontal and vertical velocities of the scanning spot by

$$f_H = \frac{v_x}{a}, \quad f_F = \frac{v_y}{b} \tag{15.12.3}$$

The typical spectrum of a video signal, shown in Fig. 15.12.1, has a *macro-structure* consisting of the harmonics of the line rate f_H. About each of these, it has a *micro-structure* consisting of the harmonics of the frame rate f_F. There are N f_F-harmonics between any two f_H-harmonics. The f_H-harmonics represent the *horizontal* variations in the image, and the f_F-harmonics the *vertical* variations.

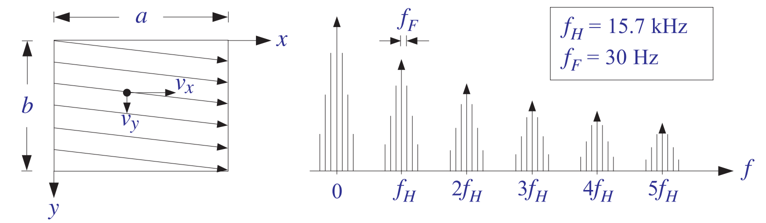


Fig. 15.12.1 Scanned image and corresponding video spectrum.

If there is motion in the image, the sharp spectral lines get smeared somewhat, but the basic macro/micro-structure is preserved. In the rest of this section, we will assume that there is no motion and that we are dealing with a still picture. At the end, we will discuss what happens when motion is introduced.

This type of spectrum can be understood qualitatively as follows: Suppose we have a still picture consisting only of a test pattern of vertical bars. Then, each scan line will be the same and the resulting signal will be periodic in time with period $T_H = 1/f_H$. Its spectrum will consist of the harmonics of f_H. Now, if there is some vertical detail in the picture, the signal will only be periodic with respect to the frame period T_F, and its spectrum will consist of the harmonics of f_F. However, because adjacent lines tend to be similar, the harmonics of f_H will still dominate the macro-structure of the spectrum.

A more mathematical explanation is as follows [200]. Let $g(x, y)$ be the brightness of the picture at position (x, y). Expanding $g(x, y)$ into a double Fourier series, we obtain:

$$g(x, y) = \sum_{k,m} c_{km} e^{2\pi jkx/a} e^{2\pi jmy/b}$$

The indices k and m correspond to the *horizontal and vertical* variations in $g(x,y)$. A scanned picture, with a uniformly moving scanning spot, is obtained by replacing $x = v_x t$ and $y = v_y t$, resulting in the *video signal*:

$$V(t)=g(v_x t, v_y t)=\sum_{k,m} c_{km}e^{2\pi jkv_x t/a}e^{2\pi jmv_y t/b}$$

Using Eq. (15.12.3), we can rewrite:

$$V(t)=\sum_{k,m} c_{km}e^{2\pi jf_{km}t} \tag{15.12.4}$$

where

$$\boxed{f_{km}=kf_H+mf_F=(kN+m)f_F=(k+\frac{m}{N})f_H} \tag{15.12.5}$$

Thus, the video signal $V(t)$ will have spectrum with sharp spectral lines at f_{km}. Because of the large difference in value between f_H and f_F, the spectrum will look as in Fig. 15.12.1, that is, exhibiting a coarse structure at the harmonics kf_H and a fine structure at the harmonics mf_F.

In the NTSC[†] TV system used in the U.S., there are $N = 525$ lines in each frame, but they are *interlaced*, with each half (i.e., a field) being presented at double the frame rate, that is, $f_{\text{field}} = 2f_F$. The field rate is approximately 60 Hz in the U.S., and the frame rate approximately 30 Hz. The exact values are [204]:

$$f_H=\frac{4.5\text{ MHz}}{286}=15.73426\text{ kHz},\qquad f_F=\frac{f_H}{525}=29.97\text{ Hz} \tag{15.12.6}$$

where, by convention, the values are derived from the *sound carrier* frequency of 4.5 MHz. The corresponding time delays are $T_H = 63.55\ \mu$sec and $T_F = 33.37$ msec.

In a color TV system, there are three scanning beams for red, green, and blue (RGB), which can be combined to yield other colors. To reduce the transmission bandwidth requirements and maintain compatibility with black and white receivers, appropriate linear combinations of the RGB colors are formed.

The black and white information (brightness and spatial details) is contained in the *luminance* signal defined by:

$$Y=0.299R+0.587G+0.114B$$

Color information (hue and saturation) can be transmitted by the *difference* signals $R-Y$ and $G-Y$. In the NTSC system, the following linear combinations—called the I and Q *chrominance* signals—are transmitted instead:

$$I=0.736(R-Y)-0.269(B-Y)$$
$$Q=0.478(R-Y)+0.413(B-Y)$$

The three RGB colors can be recovered from the three YIQ signals. The advantage of the IQ linear combinations is that they have *reduced* bandwidth requirements. The luminance bandwidth is 4.2 MHz, whereas the bandwidths of I and Q are 1.3 MHz and 0.5 MHz, respectively.

[†] National Television System Committee.

To transmit the YIQ signals efficiently, the I and Q are placed on a *color subcarrier* signal by quadrature modulation and added to the luminance component, that is, the following *composite video* signal is transmitted:

$$V(t)=Y(t)+I(t)\cos(2\pi f_{sc}t+\phi)+Q(t)\sin(2\pi f_{sc}t+\phi)$$

where $\phi = 33^{\text{o}}$.

To simplify the algebra, we work with the following complex-valued version of the above, with the understanding that we must take real parts:

$$\boxed{V(t)=Y(t)+e^{j2\pi f_{sc}t}C(t)\equiv Y(t)+Z(t)} \tag{15.12.7}$$

where $C(t)\equiv\big(I(t)-jQ(t)\big)e^{j\phi}$.

The spectra of the separate component signals $\{Y,I,Q\}$ are all similar to the basic video spectrum of Fig. 15.12.1. The subcarrier modulation *shifts* the spectra of the chrominance signals I and Q and *centers* them about the subcarrier frequency f_{sc}. Thus, the frequencies of the modulated chrominance signal $Z(t)$ will be at:

$$f_{sc}+f_{km}=f_{sc}+kf_H+mf_F \tag{15.12.8}$$

By choosing f_{sc} to be a *half-multiple* of the line frequency f_H, the chrominance peaks will fall exactly *half-way* between the luminance peaks, as shown in Fig. 15.12.2. We can take, for example,

$$\boxed{f_{sc}=(d_H+\frac{1}{2})f_H=\frac{1}{2}(2d_H+1)f_H} \tag{15.12.9}$$

Therefore, the chrominance *macro-structure* peaks are centered at half-multiples of f_H:

$$f_{sc}+f_{km}=(d_H+k+\frac{1}{2})f_H+mf_F \tag{15.12.10}$$

Moreover, because $f_H = Nf_F$ with N odd, the subcarrier frequency f_{sc} will also be equal to a half-multiple of the *frame* frequency f_F:

$$f_{sc}=(d_H+\frac{1}{2})f_H=(d_H+\frac{1}{2})Nf_F$$

Setting

$$d_F+\frac{1}{2}=(d_H+\frac{1}{2})N\qquad\Rightarrow\qquad d_F=Nd_H+\frac{N-1}{2}$$

we find,

$$\boxed{f_{sc}=(d_F+\frac{1}{2})f_F=\frac{1}{2}(2d_F+1)f_F} \tag{15.12.11}$$

It follows that the chrominance *micro-structure* peaks will be centered at half-multiples of f_F (about the kf_H macro-structure peaks), falling half-way between the micro-structure peaks of the luminance signal, as shown in Fig. 15.12.2:

$$f_{sc} + f_{km} = kf_H + (d_F + m + \frac{1}{2})f_F \qquad (15.12.12)$$

In the NTSC system, we have the choices:

$$d_H = 227, \qquad d_F = Nd_H + \frac{N-1}{2} = 119437 \qquad (15.12.13)$$

which give the subcarrier frequency:

$$f_{sc} = 227.5\,f_H = 119437.5\,f_F = 3.579545 \text{ MHz}$$

In summary, the luminance and modulated chrominance signals have spectra that are *interleaved* both at the *macro- and micro-structure* levels. This property makes them ideal candidates for comb filtering.

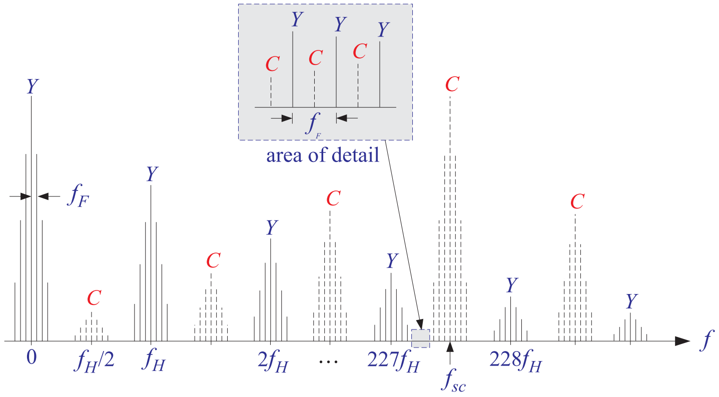


Fig. 15.12.2 Interleaved luminance and chrominance spectra.

In Fig. 15.12.2, the extent of the chrominance spectrum is somewhat exaggerated and shown to reach almost down to zero frequency. In fact, because of the small bandwidth of C, the effective extent of the chrominance spectrum about f_{sc} will be ±0.5 MHz [204], which translates to about $\pm 32 f_H$ harmonics about f_{sc}; indeed, $227.5\times 0.5/3.58 \simeq 32$.

In a conventional color TV receiver, the luminance part is extracted by lowpass filtering of the composite signal $V(t)$ with a *lowpass* filter having passband from zero to about $f_{sc} - 0.5 = 3.08$ MHz. The chrominance component is extracted by a *bandpass* filter centered at f_{sc}, with passband $f_{sc} \pm 0.5 = 3.58 \pm 0.5$ MHz. The extracted C component is then demodulated and the I and Q parts are linearly combined with Y to form the RGB signals.

These filtering operations can cause various degradations in image quality. For example, the high-frequency part of Y is filtered out, causing some loss of spatial details

in the image. Moreover, because of the finite transition widths of the lowpass and bandpass filters, some chrominance will survive the luminance lowpass filter and show up as the so-called cross-luminance or “dot-crawl” effect [205]. Similarly, some high-frequency luminance will survive the chrominance bandpass filter and will appear as the “cross-color” rainbow-type effect around sharp edges [205].

A better method of separation is to take advantage of the interlaced comb-like nature of the composite video spectrum of Fig. 15.12.2 and use digital comb filters to separate the Y and C signals. The development of large-scale digital memories that can be used to store a whole line or a whole frame [196] has made this approach possible.

A common sampling rate for digital video systems is *four* times the color subcarrier frequency, that is,

$$\boxed{f_s = 4f_{sc}} \qquad (15.12.14)$$

Using Eqs. (15.12.9) and (15.12.11), we may express the sampling rate in terms of the line frequency f_H and the frame frequency f_F:

$$\boxed{f_s = D_H f_H = D_F f_F} \qquad (15.12.15)$$

where

$$D_H = 2(2d_H + 1), \qquad D_F = ND_H = 2(2d_F + 1) \qquad (15.12.16)$$

For the NTSC system, we have from Eq. (15.12.13)

$$D_H = 910, \qquad D_F = 477750$$

and

$$f_s = 910\,f_H = 477750\,f_F = 14.31818 \text{ MHz}$$

with a corresponding sampling time interval of $T = 1/f_s = 69.84$ nsec.

Equation (15.12.15) implies that there are D_H samples along each scan line and D_F samples in each frame. In units of *radians per sample*, the subcarrier frequency f_{sc} becomes:

$$\boxed{\omega_{sc} = \frac{2\pi f_{sc}}{f_s} = \frac{\pi}{2}} \qquad (15.12.17)$$

Similarly, the frame and line frequencies are in these units:

$$\omega_F = \frac{2\pi f_F}{f_s} = \frac{2\pi}{D_F}, \qquad \omega_H = \frac{2\pi f_H}{f_s} = \frac{2\pi}{D_H} = N\omega_F$$

Using Eq. (15.12.15), the luminance video frequencies f_{km} become:

$$\omega_{km} = \frac{2\pi f_{km}}{f_s} = \frac{2\pi k}{D_H} + \frac{2\pi m}{D_F} = k\omega_H + m\omega_F \qquad (15.12.18)$$

The shifted chrominance frequencies Eq. (15.12.8) can be expressed as half-multiples of either the line or the frame digital frequencies:

$$\begin{aligned}\omega_{sc}+\omega_{km} &= (2d_H+2k+1)\frac{\pi}{D_H}+\frac{2\pi m}{D_F}\\ &= \frac{2\pi k}{D_H}+(2d_F+2m+1)\frac{\pi}{D_F}\\ &= (2kN+2d_F+2m+1)\frac{\pi}{D_F}\end{aligned} \tag{15.12.19}$$

where in the last line we replaced $D_H = D_F/N$.

The comb filters used in video systems are of the type (15.11.10), where D can be either a line delay $D = D_H$ or a frame delay $D = D_F$. Because of the high sampling rates involved, to minimize the computational cost, the filter parameters are chosen to have simple values, such as powers of two. For example, the simplest choice is the following FIR comb filter, obtained by setting $a = 0$ and $b = 1/2$ in Eq. (15.11.10):

$$H_{\text{comb}}(z) = \frac{1}{2}(1+z^{-D}) \tag{15.12.20}$$

with a complementary notch filter $H_{\text{notch}}(z) = 1 - H_{\text{comb}}(z)$:

$$H_{\text{notch}}(z) = \frac{1}{2}(1-z^{-D}) \tag{15.12.21}$$

Their magnitude responses have been plotted in Figs. 15.11.8 and 15.11.4 for $D = 10$. They have the maximum allowed 3-dB width of all the comb/notch filters of the types (15.11.10) and (15.11.3), that is, $\Delta\omega = \pi/D$.

The comb filter $H_{\text{comb}}(z)$ has (unity-gain) peaks at the multiples $2k\pi/D$ and notches at the half-multiples $(2k+1)\pi/D$. Conversely, the notch filter $H_{\text{notch}}(z)$ has peaks at the half-multiples $(2k+1)\pi/D$ and notches at $2k\pi/D$.

If D is a line delay, $D = D_H$, then the peaks of $H_{\text{comb}}(z)$ will coincide with the *macro-structure* line-frequency peaks of the luminance signal Y; and its notches will coincide with the macro-structure peaks of the modulated chrominance signal C. Thus, filtering the composite video signal V through $H_{\text{comb}}(z)$ will tend to remove C and let Y pass through, at least at the macro-structure level which is the dominant part the spectrum.

Conversely, filtering V through the notch filter $H_{\text{notch}}(z)$ will tend to remove Y and let C pass through. Thus, the two filters can be used in parallel to extract the Y and C components. A block diagram implementation of (15.12.20) and (15.12.21) is shown in Fig. 15.12.3.

The separation of Y and C is not perfect, because the line comb $H_{\text{comb}}(z)$ does not remove from the C signal its micro-structure frequencies, that is, the terms mf_F in Eq. (15.12.10). And similarly, $H_{\text{notch}}(z)$ does not remove the micro-structure frequencies of Y.

Moreover, because Eq. (15.12.20) is equivalent to the *averaging* of two successive horizontal lines, some vertical detail will be lost or averaged out, resulting in a blurrier Y signal. However, as we see below, the lost vertical detail can be restored by further filtering.

Because the luminance and chrominance spectra are interleaved at their micro-structure frame-frequency level, the delay D can also be chosen to be a frame delay, $D = D_F$. This type of comb/notch filter would do a much better job in separating the Y and C components because $H_{\text{comb}}(z)$ now has nulls at *all* the chrominance frequencies, and $H_{\text{notch}}(z)$ has nulls at all the luminance frequencies. However, such frame-delay filters can be used only if there is very little motion from frame to frame. The effect of motion is to broaden the f_F-harmonics making the separation of Y and C less than perfect.

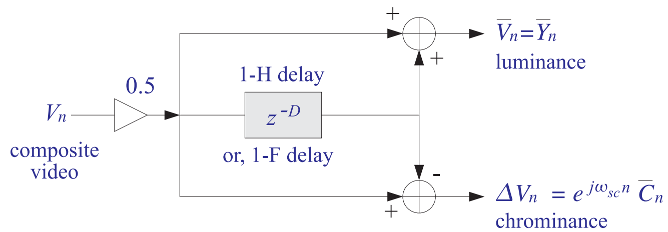

+
+
$\overline{V}_n=\overline{Y}_n$
luminance
0.5
1-H delay
V_n
z^{-D}
or, 1-F delay
composite video
-
+
$\Delta V_n = e^{j\omega_{sc}n}\,\overline{C}_n$
chrominance

Fig. 15.12.3 Line or frame comb filters.

Another simple popular type of comb filter uses two D-fold delays and is obtained by squaring Eq. (15.12.20):

$$H_{\text{comb}}(z) = \frac{1}{4}(1+z^{-D})^2 = \frac{1}{4}(1+2z^{-D}+z^{-2D}) \tag{15.12.22}$$

When $D = D_H$, it is referred to as a 2-H comb because it requires the storage of two horizontal lines. It is also known as a 1-2-1 comb because of the particular weights given to the three horizontal lines.

Its peaks and notches are the same as those of the 1-H comb (15.12.20), but here the squaring operation has the effect of making the peaks narrower and the notches flatter. The corresponding complementary notch filter is defined as:

$$H_{\text{notch}}(z) = -\frac{1}{4}(1-z^{-D})^2 = \frac{1}{4}(-1+2z^{-D}-z^{-2D}) \tag{15.12.23}$$

These definitions imply $H_{\text{comb}}(z) + H_{\text{notch}}(z) = z^{-D}$. This is required because the filters have an inherent delay of D samples. Indeed, if we advance them by D samples, we get the more symmetric definitions corresponding to truly complementary filters:

$$\begin{aligned} z^D H_{\text{comb}}(z) &= \frac{1}{2}+\frac{1}{4}(z^D+z^{-D})\\ z^D H_{\text{notch}}(z) &= \frac{1}{2}-\frac{1}{4}(z^D+z^{-D})\end{aligned} \tag{15.12.24}$$

It is instructive to also understand the above comb/notch filtering operations in the time domain. The sampled version of Eq. (15.12.7) is

$$\boxed{V_n = Y_n + Z_n = Y_n + e^{j\omega_{sc}n}C_n} \tag{15.12.25}$$

The video time index n can be mapped uniquely onto a particular pixel (i,j) on the image, as shown in Fig. 15.12.4. The *row* index i corresponds to the quotient of the division of n by D_H and the *column* index j corresponds to the remainder. That is, we can write uniquely:

$$(i,j)\longrightarrow\quad n=iD_H+j,\quad j=0,1,\dots,D_H-1 \tag{15.12.26}$$

The row index i takes on the values $i=0,1,\dots N-1$, for N lines per frame. The maximum value of n corresponding to the last pixel of the last line is obtained by setting $i=N-1$ and $j=D_H-1$, giving $n=(N-1)D_H+D_H-1=ND_H-1=D_F-1$.

Subsequent values of n will map to pixels on the next frame, and so on. Thus, two values of n separated by D_F samples correspond to the *same* pixel (i,j) on the image, as shown in Fig. 15.12.4.

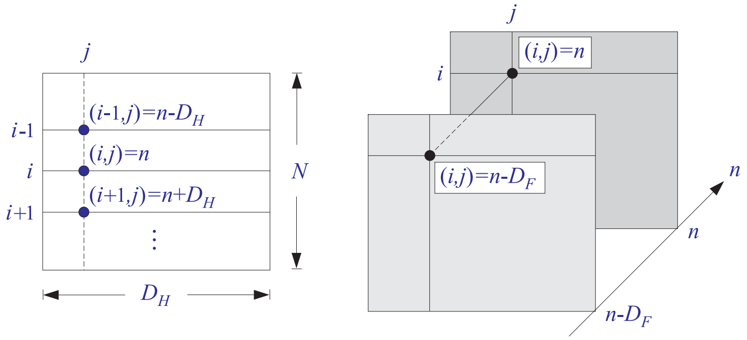


Fig. 15.12.4 Successive lines and successive frames.

Pixels on successive lines on the same column are separated by a time delay of D_H samples, as shown in Fig. 15.12.4. Indeed, we have from Eq. (15.12.26):

$$n\pm D_H=iD_H+j\pm D_H=(i\pm1)D_H+j\longrightarrow\quad(i\pm1,j)$$

With V_n as input, the output signals of the 1-H luminance and chrominance comb filters, (15.12.20) and (15.12.21), are the sum and difference signals:

$$\boxed{\begin{aligned}\overline{V}_n&=\frac{1}{2}(V_n+V_{n-D})\\ \Delta V_n&=\frac{1}{2}(V_n-V_{n-D})\end{aligned}} \tag{15.12.27}$$

If $D=D_H$, we may think of $\overline{V}_n$ as being the average of the current horizontal line with the previous one. Indeed, using the map (15.12.26), we can rewrite Eq. (15.12.27) in the equivalent form:

$$\begin{aligned}\overline{V}_{i,j}&=\frac{1}{2}(V_{i,j}+V_{i-1,j})\\ \Delta V_{i,j}&=\frac{1}{2}(V_{i,j}-V_{i-1,j})\end{aligned}$$

In a similar fashion, the outputs of the 2-H filters (15.12.24) can be expressed in terms of the video time index n:

$$\begin{aligned}\overline{V}_n&=\frac{1}{2}V_n+\frac{1}{4}(V_{n+D}+V_{n-D})\\ \Delta V_n&=\frac{1}{2}V_n-\frac{1}{4}(V_{n+D}+V_{n-D})\end{aligned}$$

or, in terms of the pixel locations, showing the weighted averaging of the current line with the lines above and below it:

$$\begin{aligned}\overline{V}_{i,j}&=\frac{1}{2}V_{i,j}+\frac{1}{4}(V_{i+1,j}+V_{i-1,j})\\ \Delta V_{i,j}&=\frac{1}{2}V_{i,j}-\frac{1}{4}(V_{i+1,j}+V_{i-1,j})\end{aligned}$$

Using Eq. (15.12.25), we have for the delayed signal V_{n-D}:

$$V_{n-D}=Y_{n-D}+Z_{n-D}=Y_{n-D}+e^{j\omega_{sc}(n-D)}C_{n-D}$$

The property that makes possible the comb filtering separation of the luminance and chrominance is that the subcarrier signal $e^{j\omega_{sc}n}$ changes sign from *line to line* and from *frame to frame*. This follows from the (intentional) choice of D_H and D_F to be even multiples of an odd integer, Eq. (15.12.16). Indeed, assuming that D is of the form $D=2(2d+1)$, we find:

$$\omega_{sc}D=\frac{\pi}{2}2(2d+1)=2\pi d+\pi$$

which corresponds to a 180° phase shift. Indeed,

$$e^{j\omega_{sc}D}=e^{2\pi jd+j\pi}=e^{j\pi}=-1$$

It follows that:

$$\boxed{V_{n-D}=Y_{n-D}-e^{j\omega_{sc}n}C_{n-D}} \tag{15.12.28}$$

The outputs of the luminance and chrominance combs can be expressed then in the form:

$$\begin{aligned}\overline{V}_n&=\frac{1}{2}(V_n+V_{n-D})=\frac{1}{2}(Y_n+Y_{n-D})+e^{j\omega_{sc}n}\frac{1}{2}(C_n-C_{n-D})\\ \Delta V_n&=\frac{1}{2}(V_n-V_{n-D})=\frac{1}{2}(Y_n-Y_{n-D})+e^{j\omega_{sc}n}\frac{1}{2}(C_n+C_{n-D})\end{aligned}$$

which can be written in terms of the corresponding sum and difference signals:

$$\boxed{\begin{aligned}\overline{V}_n&=\overline{Y}_n+e^{j\omega_{sc}n}\Delta C_n\\ \Delta V_n&=\Delta Y_n+e^{j\omega_{sc}n}\,\overline{C}_n\end{aligned}} \tag{15.12.29}$$

Consider the line-comb case first, $D=D_H$. Assuming that the chrominance signal C_n does not change much from line to line (i.e., ignoring its micro-structure frequency content), we may set $\Delta C_n \simeq 0$. Similarly, ignoring the micro-structure of Y, we may set $\Delta Y_n \simeq 0$. Then, (15.12.29) simplifies as follows:

$$\boxed{\begin{aligned} \overline{V}_n &= \overline{Y}_n \\ \Delta V_n &= e^{j\omega_{sc}n}\,\overline{C}_n \end{aligned}} \qquad (15.12.30)$$

Thus, the comb outputs are effectively the desired luminance and chrominance components. The chrominance part $e^{j\omega_{sc}n}\,\overline{C}_n$ is then sent into a subcarrier demodulator and $\overline{C}_n$ is extracted.

For the frame-comb case, $D = D_F$, the difference signals will be identically zero, $\Delta C_n = \Delta Y_n = 0$, because of the periodicity with period D_F. Thus, the frame combs are capable of separating Y and C exactly. However, they will fail when there is motion in the image which makes the video signal non-periodic. Advanced digital video systems use frame combs when there is no or very little motion, and switch to line combs when substantial motion is detected [196].

In the line-comb case, the approximation $\Delta Y_n \simeq 0$ is more severe than $\Delta C_n \simeq 0$, because the luminance signal carries most of the spatial detail information. Setting $\Delta Y_n = 0$ implies a loss of *vertical detail*, because the output of Eq. (15.12.30) gives a luminance value $\overline{Y}_n$ averaged across two lines, instead of Y_n itself.

It follows from the above that a better approximation may be obtained by setting ΔC_n to zero, but not ΔY_n. The filtering equations (15.12.29) become in this case:

$$\begin{aligned} \overline{V}_n &= \overline{Y}_n \\ \Delta V_n &= \Delta Y_n + e^{j\omega_{sc}n}\,\overline{C}_n \end{aligned} \qquad (15.12.31)$$

The averaged signal $\overline{Y}_n$ can be expressed in terms of the desired one Y_n and the missing vertical detail signal ΔY_n, as follows:

$$\overline{Y}_n = \frac{1}{2}(Y_n + Y_{n-D}) = Y_n - \frac{1}{2}(Y_n - Y_{n-D}) = Y_n - \Delta Y_n$$

Therefore, we may write Eq. (15.12.31) as:

$$\boxed{\begin{aligned} \overline{V}_n &= Y_n - \Delta Y_n \\ \Delta V_n &= \Delta Y_n + e^{j\omega_{sc}n}\,\overline{C}_n \end{aligned}} \qquad (15.12.32)$$

This result suggests a method of restoring the lost vertical detail [193,205,207]. Because the vertical detail signal ΔY_n is common in both outputs, it can be extracted from the second output ΔV_n by *lowpass filtering* and then reinserted into the first to recover Y_n. This works because most of the energy in ΔY_n is between 0–1 MHz, whereas the C-term in ΔV_n is centered around 3.58 MHz. Moreover, the term ΔY_n can be removed from ΔV_n by a *bandpass filter* centered at the subcarrier frequency.

Figure 15.12.5 shows the above method of vertical detail restoration. The delay z^{-M} compensates for the delay introduced by the filter $H_{LP}(z)$. The required lowpass filter $H_{LP}(z)$ must be chosen to remove the C-term from the ΔV_n output and be flat at low frequencies. Therefore, it is really a *bandstop filter* with a zero at the subcarrier frequency $\omega_{sc} = \pi/2$. In the z-domain, the filter must have zeros at $z = e^{\pm j\omega_{sc}} = e^{\pm j\pi/2} = \pm j$.

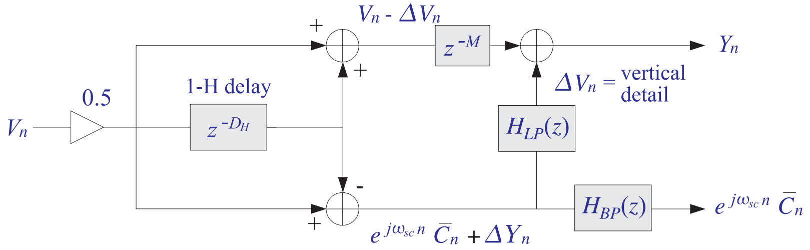

V_n
0.5
1-H delay
z^{-D_H}
+
+
-
+
V_n - ΔV_n
z^{-M}
Y_n
ΔV_n = vertical detail
$H_{LP}(z)$
$H_{BP}(z)$
$e^{j\omega_{sc}n}\,\overline{C}_n$
$e^{j\omega_{sc}n}\,\overline{C}_n + \Delta Y_n$

Fig. 15.12.5 Vertical detail reinsertion filters.

Another requirement for such a filter is that it have simple coefficients, expressible as sums of powers of two. Some examples of such filters are given below [193,205,207], normalized to unity gain at DC:

$$\begin{aligned} H_{LP}(z) &= \frac{1}{4}(1+z^{-2})^2 \\ H_{LP}(z) &= \frac{1}{16}(1+z^{-2})^2(-1+6z^{-2}-z^{-4}) \\ H_{LP}(z) &= \frac{1}{32}(1+z^{-2})^2(-3+14z^{-2}-3z^{-4}) \\ H_{LP}(z) &= \frac{1}{64}(1+z^{-2})^4(1-4z^{-2}+5z^{-4}+5z^{-8}-4z^{-10}+z^{-12}) \end{aligned} \qquad (15.12.33)$$

The factors $(1+z^{-2})$ vanish at $z = \pm j$. The corresponding bandpass filters $H_{BP}(z)$ are obtained by requiring that they be complementary to $H_{LP}(z)$, that is, they satisfy $H_{LP}(z)+H_{BP}(z)= z^{-M}$, where M is the inherent delay introduced by the filters (i.e., half the filter order). Thus, we define:

$$H_{BP}(z) = z^{-M} - H_{LP}(z) \qquad (15.12.34)$$

For the above four examples, we have $M = 2, 4, 4, 10$. Using Eqs. (15.12.33) and (15.12.34), we find:

$$\begin{aligned} H_{BP}(z) &= -\frac{1}{4}(1-z^{-2})^2 \\ H_{BP}(z) &= \frac{1}{16}(1-z^{-2})^4 \\ H_{BP}(z) &= \frac{1}{32}(1-z^{-2})^2(3-2z^{-2}+3z^{-4}) \\ H_{BP}(z) &= -\frac{1}{64}(1-z^{-2})^4(1+4z^{-2}+5z^{-4}+5z^{-8}+4z^{-10}+z^{-12}) \end{aligned} \qquad (15.12.35)$$

all having a common factor $(1-z^{-2})$, vanishing at DC and the Nyquist frequency.

Figure 15.12.6 shows the magnitude responses of the fourth filters in (15.12.33) and (15.12.35) [193], plotted over the effective video band $0 \le f \le 4.2$ MHz. Over this band, they behave as lowpass and highpass filters. The passband of the lowpass filter coincides with the significant frequency range of the vertical detail signal ΔV_n, whereas the passband of the highpass/bandpass filter coincides with the chrominance band.

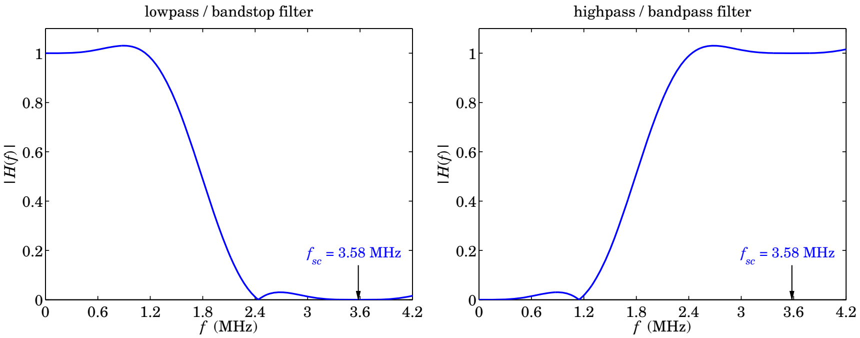


Fig. 15.12.6 Vertical detail restoration filters.

We note finally that in Fig. 15.12.5, the bandpass filter block $H_{BP}(z)$ may be eliminated and replaced by Eq. (15.12.34), which can be used to construct the required bandpass output from the lowpass output and another delay z^{-M}, thus making the implementation more efficient computationally; see, for example, Fig. 11.3.7.

15.13 Problems

15.1 A zero-mean white noise sequence $x(n)$, $n \ge 0$, of variance σ_x^2 is sent through a stable and causal filter. Using convolution, show that the variance of the output sequence $y(n)$ will be given by:

$$\sigma_y^2(n) = E[y(n)^2] = \sigma_x^2 \sum_{m=0}^{n} h(m)^2$$

so that for large n it converges to the theoretical NRR of Eq. (9.7.4).

15.2 Show that the NRR summation of Eq. (9.7.4) always converges for a stable and causal filter with rational transfer function. In particular, show that it is bounded by:

$$\mathcal{R} = \sum_{n=0}^{\infty} h_n^2 \le \frac{C}{1-|p_{\max}|^2}$$

where $p_{\max}$ is the pole of maximum magnitude and C is a constant that depends on the PF expansion coefficients. You may assume the PF expansion: $h_n = \sum_{i=1}^{M} A_i p_i^n u(n)$.

15.3 For an ideal bandpass filter with passband $\omega_a \le |\omega| \le \omega_b$, prove the theoretical NRR given by Eq. (15.1.7).

15.4 *Computer Experiment: Exponential Smoother.* Write a C program, say `smooth.c`, that implements the first-order smoother of Example 15.2 with transfer function $H(z) = (1-a)/(1-az^{-1})$, where $0 < a < 1$. The program must have usage:

```
smooth a  < x.dat  > y.dat
```

where a is a command-line argument. The input signal to be smoothed must be read from `stdin` or a file `x.dat`, and the smoothed output must be written to the `stdout` or a file `y.dat`.

15.5 *Computer Experiment: Exponential Smoother.* Using the above program `smooth.c` reproduce the graphs in Fig. 15.2.2. Generate also two similar graphs for the filter parameter values $a = 0.99$ and $a = 0.8$.

In all four cases, compute the experimental NRRs computed from the sample variances based on the $L = 200$ input and output data sequences $x(n)$, $y(n)$ with means m_x, m_y:

$$\hat{\sigma}_x^2 = \frac{1}{L}\sum_{n=0}^{L-1}(x(n)-m_x)^2, \quad \hat{\sigma}_y^2 = \frac{1}{L}\sum_{n=0}^{L-1}(y(n)-m_y)^2, \quad \widehat{\mathcal{R}} = \frac{\hat{\sigma}_y^2}{\hat{\sigma}_x^2}$$

and compare them with the theoretical values. Explain any discrepancies.

15.6 Normally, you would use a lowpass (or highpass) filter to extract a low- (or high-) frequency signal. Suppose instead you used the lowpass filter $H(z) = b/(1-az^{-1})$, where $0 < a < 1$, to extract the high-frequency signal $x(n) = s(-1)^n + v(n)$, where $v(n)$ is zero-mean white noise of variance σ_v^2.

How should you choose b so that the part $s(-1)^n$ comes out unchanged? Show that in this case the noise will be amplified. Explain this result by calculating the NRR as well as graphically by sketching the frequency spectra of the signals and filter, as in Fig. 15.2.1.

15.7 Consider the highpass FIR averager filter of Example 15.6. Using the minimization techniques outlined in Sec. 15.5, show that the optimum length-N FIR filter that minimizes the NRR subject to the highpass constraint (15.6.2) is given by Eq. (15.6.1).

15.8 Using partial fractions, derive Eq. (15.4.1) for the NRR of the bandpass resonator filter of Example 15.4.

15.9 *Computer Experiment: Bandpass Signal Extraction.* An improved version of the bandpass filter of Example 15.4, which has prescribed 3-dB width $\Delta\omega$ and center frequency ω_0, can be designed with the methods of Chapter 12. Using the design equations (12.3.21) and (12.3.22), design the following two peaking filters that have specifications:

a. Center frequency $\omega_0 = 0.1\pi$, 3-dB width $\Delta\omega = 0.05\pi$. Determine the filter's transfer function, write its sample processing algorithm, compute its NRR and its 5% time constant n_{eff}, and plot its magnitude response squared $|H(\omega)|^2$ over 400 equally spaced frequencies over $0 \le \omega < \pi$.

b. Center frequency $\omega_0 = 0.1\pi$, but with 5% time constant of $n_{\text{eff}} = 300$. Then, repeat all the questions of part (a).

c. Using the Gaussian generator `gran`, generate a noisy sinusoidal input of the form:

$$x(n) = s(n) + v(n) = \cos(\omega_0 n) + v(n), \quad n = 0, 1, \ldots, N-1$$

where $\omega_0 = 0.1\pi$, $N = 300$, and $v(n) = \text{gran}(0, 1, \&\text{iseed})$ is zero-mean, unit-variance, white Gaussian noise. Send $x(n)$ through the above two filters and compute the output $y(n)$. Plot $x(n)$ versus n. Plot the two outputs $y(n)$ together with the desired signal $s(n)$.

15.10 *Computer Experiment: Single-Notch Filter.* Consider Example 15.11.1, but with a simplified signal instead of the ECG, defined to be a double pulse which is replicated three times at a period of 0.5 sec, with a 60 Hz noise component added to it:

$$\begin{aligned} f(t) &= \big[u(t-0.15)-u(t-0.30)\big]-0.75\big[u(t-0.30)-u(t-0.45)\big] \\ s(t) &= f(t)+f(t-0.5)+f(t-1) \\ x(t) &= s(t)+0.5\cos(2\pi f_1 t) \end{aligned}$$

where t is in seconds, $u(t)$ is the unit-step function, and $f_1 = 60$ Hz. The signal $x(t)$ is sampled at a rate of 1 kHz for a period of 1.5 seconds. Let $x(n)$ denote the resulting samples. Plot $x(n)$ and the noise-free signal $s(n)$ for $0 \le n \le 1499$.

Using the design method described in Example 15.11.1, design two second-order notch filters with notch frequency at f_1, one having $Q = 6$ and the other $Q = 60$. Determine their filter coefficients and their 1% time constants. Plot their magnitude responses over $0 \le f \le f_s/2$.

Filter the sampled signal $x(n)$ through both filters, and plot the resulting output signals $y(n)$ for $0 \le n \le 1499$. Discuss the capability of the filters in removing the 60 Hz interference. Discuss also the residual ringing that is left in the output after the 60 Hz sinusoid has died out. (To study it, you may use superposition and filter $s(n)$ and the noise part separately; you may also look at the impulse responses.)

15.11 *Computer Experiment: Multi-Notch Filter.* Consider the signal $x(n)$ consisting of three periods of a pulse signal $f(n)$ plus additive noise, defined for $0 \le n < 1800$:

$$\begin{aligned} f(n) &= \big[u(n-150)-u(n-300)\big]-0.75\big[u(n-300)-u(n-450)\big] \\ s(n) &= f(n)+f(n-600)+f(n-1200) \\ x(n) &= s(n)+v(n) \end{aligned}$$

where $v(n)$ is defined as in Example 15.11.3 to be a periodic square wave of period $D = 10$. Therefore, a periodic notch filter with notches at the harmonics of $f_1 = f_s/D$ or $\omega_1 = 2\pi/D$ will remove the noise component. Using the design method of Example 15.11.3, design such a multi-notch filter having $Q = 80$.

Implement the filter using the sample processing algorithm of Example 15.11.3, and process the noisy signal $x(n)$ through it to get the output signal $y(n)$. On separate graphs, plot the signals $s(n)$, $x(n)$, and $y(n)$, for $0 \le n < 1800$. For display purposes, split each graph into three separate graphs that cover the time periods $0 \le n < 600$, $600 \le n < 1200$, and $1200 \le n < 1800$.

The noise is removed fairly well, but you will notice that the filter also distorts the desired signal $s(n)$ rather severely. To understand the origin of this distortion, filter $s(n)$ separately through the filter and plot the corresponding output. Then, design three other periodic notch filters having $Q = 200$, 400, and 800, filter $s(n)$ through them, and plot the outputs. In all cases, compute the 1% time constants of the filters and discuss the tradeoff between speed of response, noise reduction, and non-distortion of the input.

Moreover, for the two cases $Q = 80$ and $Q = 800$, plot the corresponding magnitude responses $|H(f)|$ over one Nyquist interval $0 \le f \le f_s$ assuming $f_s = 600$ Hz, so that $f_1 = f_s/D = 60$ Hz.

15.12 *Computer Experiment: ECG Processing.* Reproduce all the designs, results, and graphs of Example 15.11.1. The simulated ECG data $s(n)$ may be generated by the MATLAB routine `ecg.m` of Appendix C, as follows:

```
s  = ecg(500)';           one beat of length 500
s  = [s; s; s];           three beats
s0 = sgfilt(0, 5, s);     5-point smoother
s  = s0 / max(s0);        normalized to unity maximum
```

15.13 *Computer Experiment: ECG Processing.* Reproduce all the designs, results, and graphs of Example 15.11.3. The simulated ECG data $s(n)$ may be generated by the MATLAB routine `ecg.m` of Appendix C, as follows:

```
s  = ecg(600)';           one beat of length 600
s  = [s; s; s];           three beats
s0 = sgfilt(0, 9, s);     9-point smoother
s  = s0 / max(s0);        normalized to unity maximum
```

15.14 Show that the following periodic comb filter has NRR:

$$H(z) = \frac{1-a}{2}\,\frac{1+z^{-D}}{1-az^{-D}} \quad\Rightarrow\quad \mathcal{R} = \frac{1-a}{2}$$

Then show that if we define its Q-factor in terms of its 3-dB width $\Delta\omega$ and its first harmonic $\omega_1 = 2\pi/D$ by $Q = \omega_1/\Delta\omega$, then the parameter a can be calculated as:

$$a = \frac{1-\tan(\pi/2Q)}{1+\tan(\pi/2Q)}$$

Finally, determine and sketch its causal impulse response $h(n)$.

15.15 *Computer Experiment: Periodic Signal Enhancement.* Reproduce all the results and graphs of Example 15.11.5. Implement the comb filter using the circular-buffer version of the sample processing algorithm. (This is more appropriate because in practice the signal's period may be large.)

Repeat using Q-factors: $Q = 40$ and $Q = 30$. In all cases, compute the filter's 5% time constant and discuss the tradeoff between speed of response, signal enhancement, and noise reduction.

15.16 *Computer Experiment: TV Vertical Detail Filters.* First, verify that the vertical detail reinsertion filters given in Eqs. (15.12.33) and (15.12.35) satisfy the complementarity property of Eq. (15.12.34).

Then, plot their magnitude response $|H(f)|$ using the same scales as in Fig. 15.12.6.

Part II

Applications

16

Digital Audio Effects

16.1 Digital Waveform Generators

It is often desired to generate various types of waveforms, such as periodic square waves, sawtooth signals, sinusoids, and so on.

A filtering approach to generating such waveforms is to design a filter $H(z)$ whose *impulse response* $h(n)$ is the waveform one wishes to generate. Then, sending an impulse $\delta(n)$ as input will generate the desired waveform at the output.

In this approach, generating each sample by running the sample processing algorithm of the filter requires a certain amount of *computational overhead*. A more efficient approach is to *precompute* the samples of the waveform, store them in a table in RAM which is usually implemented as a circular buffer, and access them from the table whenever needed.

The period, or equivalently, the fundamental frequency of the generated waveform is controlled either by varying the speed of cycling around the table or by accessing a subset of the table at a fixed speed. This is the principle of the so-called *wavetable synthesis* which has been used with great success in computer music applications [110–131].

In this section, we discuss both the filtering and wavetable approaches and show how to implement them with circular buffers.

16.1.1 Sinusoidal Generators

The above filtering approach can be used to generate a (causal) sinusoidal signal of frequency f_0 and sampled at a rate f_s. Denoting the digital frequency by $\omega_0 = 2\pi f_0/f_s$, we have the z-transform pair:

$$h(n)= R^n \sin(\omega_0 n)u(n), \quad H(z)= \frac{R\sin\omega_0\, z^{-1}}{1-2R\cos\omega_0\, z^{-1}+R^2 z^{-2}} \tag{16.1.1}$$

For $0 < R < 1$, it corresponds to an exponentially decaying sinusoid of frequency ω_0. A pure sinusoid has $R = 1$. The canonical realization of this transfer function is shown in Fig. 16.1.1. The corresponding sample processing algorithm for the input $x(n)= \delta(n)$ and output $y(n)= h(n)$ is:

```
for $n = 0,1,2,\dots$ do:
    $w_0 = (2R\cos\omega_0)\, w_1 - R^2 w_2 + \delta(n)$
    $y = (R\sin\omega_0)\, w_1$
    $w_2 = w_1$
    $w_1 = w_0$
```

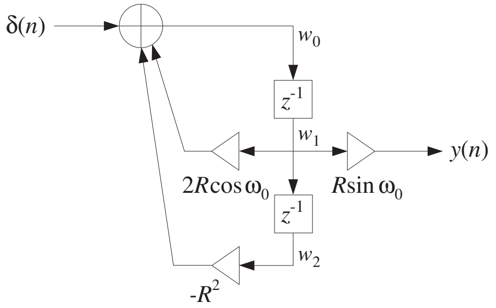


Fig. 16.1.1 Digital sinusoidal generator

In a similar fashion, we can generate an exponentially decaying cosinusoidal signal of frequency ω_0 with the following generator filter:

$$h(n)= R^n \cos(\omega_0 n)u(n), \quad H(z)= \frac{1-R\cos\omega_0\, z^{-1}}{1-2R\cos\omega_0\, z^{-1}+R^2 z^{-2}} \tag{16.1.2}$$

The canonical realization is shown in Fig. 16.1.2; its sample processing algorithm is:

```
for $n = 0,1,2,\dots$ do:
    $w_0 = (2R\cos\omega_0)\, w_1 - R^2 w_2 + \delta(n)$
    $y = w_0 - (R\cos\omega_0)\, w_1$
    $w_2 = w_1$
    $w_1 = w_0$
```

Example 16.1.1: A common application of sinusoidal generators is the all-digital touch-tone phone, known as a dual-tone multi-frequency (DTMF) transmitter/receiver [105–109]. Each key-press on the keypad generates the sum of two audible sinusoidal tones, that is, the signal

$$y(n)= \cos(\omega_L n)+\cos(\omega_H n)$$

where the two frequencies $\{\omega_L, \omega_H\}$ uniquely define the key that was pressed. Figure 16.1.3 shows the pairs of frequencies associated with each key.

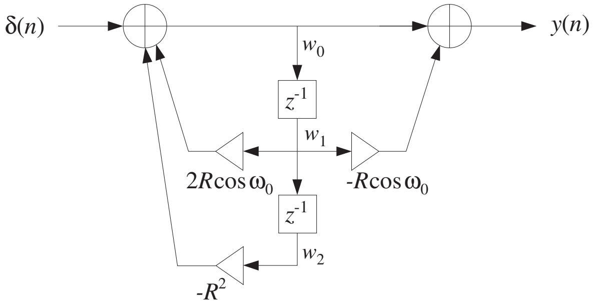
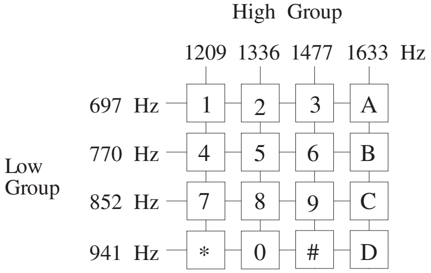


Fig. 16.1.2 Digital cosinusoidal generator

The four frequencies belonging to the low-frequency group select the four rows of the 4×4 keypad,[†] and the four high-group frequencies select the columns. A pair $\{f_L, f_H\}$ with one frequency from the low and one from the high group will select a particular key. With a typical sampling rate of $f_s = 8$ kHz, the corresponding digital frequencies are $\omega_L = 2\pi f_L/f_s$ and $\omega_H = 2\pi f_H/f_s$.



	1209	1336	1477	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

Fig. 16.1.3 DTMF keypad.

The generation of the dual tone can be implemented by using two cosinusoidal generators connected in parallel as in Fig. 16.1.4 and sending an impulse as input.

The particular values of the eight keypad frequencies have been chosen carefully so that they do not interfere with speech. At the receiving end, the dual-tone signal $y(n)$ must be processed to determine which pair of frequencies $\{f_L, f_H\}$ is present. This can be accomplished either by filtering $y(n)$ through a bank of bandpass filters tuned at the eight possible DTMF frequencies, or by computing the DFT of $y(n)$ and determining which pairs of frequency bins contain substantial energy.

Both approaches can be implemented with current DSP chips. We will discuss the DFT detection method further in Chapter 10. □

The poles of the transfer functions (16.1.1) and (16.1.2) are at the complex locations $p = Re^{j\omega_0}$ and $p^* = Re^{-j\omega_0}$. The denominator of these transfer functions factors in the form:

[†] The A,B,C,D keys appear on service keypads.

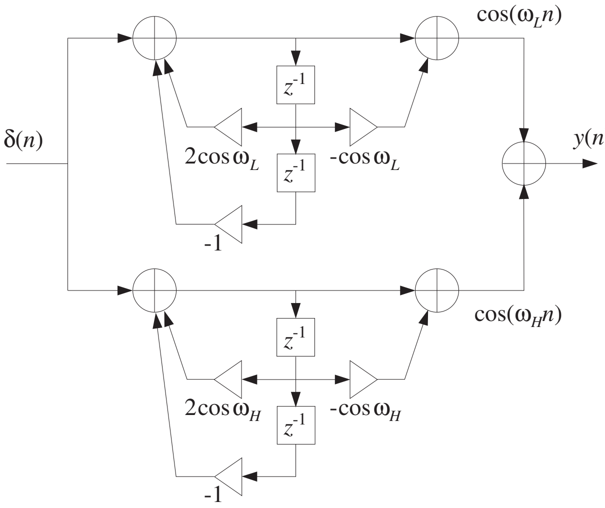


Fig. 16.1.4 DTMF tone generator.

$$1 - 2R\cos\omega_0\, z^{-1} + R^2 z^{-2} = (1 - pz^{-1})(1 - p^* z^{-1}) \tag{16.1.3}$$

Denoting by a and b the real and imaginary parts of the pole $p = a + jb$, that is, $a = R\cos\omega_0$ and $b = R\sin\omega_0$, we have $R^2 = a^2 + b^2$ and can express the common denominator as

$$1 - 2R\cos\omega_0\, z^{-1} + R^2 z^{-2} = 1 - 2az^{-1} + (a^2 + b^2)z^{-2}$$

The cosinusoidal and sinusoidal transfer functions are expressed in terms of a and b as follows:

$$\begin{aligned} H_1(z) &= \frac{1 - az^{-1}}{1 - 2az^{-1} + (a^2 + b^2)z^{-2}} \\ H_2(z) &= \frac{bz^{-1}}{1 - 2az^{-1} + (a^2 + b^2)z^{-2}} \end{aligned} \tag{16.1.4}$$

where $H_1(z)$ corresponds to Eq. (16.1.2) and $H_2(z)$ to Eq. (16.1.1).

Forming the following complex linear combination, and replacing the denominator by its factored form (16.1.3), and noting that the numerators combine to give $(1 - p^* z^{-1})$, with $p^* = a - jb$, we obtain the pole/zero cancellation:

$$\begin{aligned} H_1(z) + jH_2(z) &= \frac{1 - az^{-1} + jbz^{-1}}{1 - 2az^{-1} + (a^2 + b^2)z^{-2}} = \frac{1 - p^* z^{-1}}{(1 - pz^{-1})(1 - p^* z^{-1})} \\ &= \frac{1}{1 - pz^{-1}} \end{aligned}$$

Taking causal inverse z-transforms, we find

$$h_1(n)+jh_2(n)=p^n u(n)=R^n e^{j\omega_0 n}u(n)$$

Writing $e^{j\omega_0 n}=\cos(\omega_0 n)+j\sin(\omega_0 n)$ and extracting real and imaginary parts, gives the impulse responses:

$$h_1(n)=R^n\cos(\omega_0 n)u(n),\qquad h_2(n)=R^n\sin(\omega_0 n)u(n)$$

which agree with Eqs. (16.1.2) and (16.1.1), respectively.

The filter coefficients in Figs. 16.1.1 and 16.1.2 involve both the real and imaginary parts a, b, as well as the magnitude squared $R^2=a^2+b^2$ of the poles. In a hardware implementation, these coefficients must be quantized to a finite number of bits. One potential drawback is that to be quantized accurately, the coefficient a^2+b^2 will need twice as many bits as the individual coefficients a and b.

An alternative realization [1] that combines both the sinusoidal and cosinusoidal generators is the so-called *coupled form* and is depicted in Fig. 16.1.5. Because only a and b, not their squares, appear as filter coefficients, this form will not suffer from the above quantization drawback.

When $R=1$, it is impossible in general to find quantized coefficients a,b that satisfy $a^2+b^2=1$ exactly. In that case, one settles for R slightly less than one. There exist filter structures with improved quantization properties when the poles are near the unit circle [89–92].

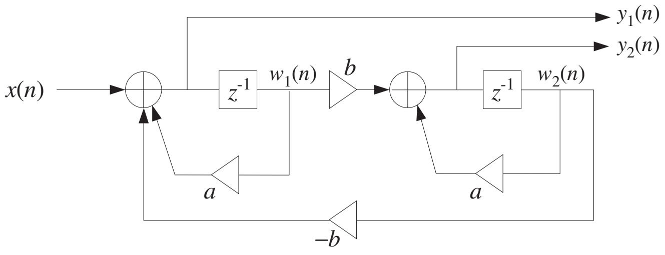


Fig. 16.1.5 Coupled form sine/cosine generator.

Noting that $w_1(n)=y_1(n-1)$ and $w_2(n)=y_2(n-1)$, the difference equations describing this form are in the time and z domains:

$$\begin{aligned} y_1(n) &= ay_1(n-1)-by_2(n-1)+x(n)\\ y_2(n) &= ay_2(n-1)+by_1(n-1) \end{aligned}$$

$$\begin{aligned} Y_1(z) &= az^{-1}Y_1(z)-bz^{-1}Y_2(z)+X(z)\\ Y_2(z) &= az^{-1}Y_2(z)+bz^{-1}Y_1(z) \end{aligned}$$

Solving for the transfer functions $H_1(z)=Y_1(z)/X(z)$ and $H_2(z)=Y_2(z)/X(z)$, we obtain Eq. (16.1.4). The sample processing algorithm that simultaneously generates the two outputs y_1 and y_2 will be:

```
for each input sample x do:
    $y_1 = aw_1 - bw_2 + x$
    $y_2 = aw_2 + bw_1$
    $w_1 = y_1$
    $w_2 = y_2$
```

16.1.2 Periodic Waveform Generators

A periodic analog signal, such as a sinusoid, does not necessarily remain periodic when sampled at a given rate f_s. For example, the samples of $x(t)=\cos(2\pi ft)$, obtained by setting $t=nT$, are:

$$x(n)=\cos(2\pi fnT)=\cos(\omega n)$$

where $\omega=2\pi fT=2\pi f/f_s$.

In order for $x(n)$ to be periodic in the time index n with some period, say of D samples, it is necessary that one *whole* period of the sinusoid fit within the D samples, that is, at $n=D$, the sinusoid must cycle by one whole period. This requires that $x(D)=x(0)$, or,

$$\cos(\omega D)=1$$

which requires that the frequency ω be such that[†]

$$\omega D=2\pi\quad\Rightarrow\quad\boxed{\omega=\frac{2\pi}{D}}\tag{16.1.5}$$

Writing $\omega=2\pi f/f_s$ and solving for f, we find the condition that a sampled sinusoid of frequency f is periodic if:

$$\boxed{f=\frac{f_s}{D}}\tag{16.1.6}$$

or, equivalently, if the sampling rate is an *integral multiple* of the frequency:

$$\boxed{f_s=Df}\tag{16.1.7}$$

These results generalize to the case of an arbitrary analog periodic signal, not just a sinusoid. Indeed, if a signal $x(t)$ has period T_D and is sampled at a rate $f_s=1/T$, then the periodicity condition $x(t+T_D)=x(t)$ implies $x(nT+T_D)=x(nT)$. In order for the sampled signal to be periodic in n, the time $nT+T_D$ must be one of the sampling times, that is,

$$nT+T_D=(n+D)T$$

[†] One could also have $\omega=2\pi c/D$, where c is an integer, but this would correspond to fitting more than one sinusoidal cycles in the D samples, that is, c cycles.

which requires that the period T_D be an integral multiple of the sampling period T:

$$\boxed{T_D = DT} \tag{16.1.8}$$

Because the *fundamental frequency* of such a periodic signal is $f = 1/T_D$, equation (16.1.8) is equivalent to Eq. (16.1.6) or (16.1.7).

In this section, we consider the generation of such discrete-time periodic signals. Because of the periodicity, it is enough to specify the signal over one period only. Denoting the time samples over one period by b_i, $i = 0, 1, \dots, D-1$, we have the periodic sequence:

$$\mathbf{h} = [b_0, b_1, \dots, b_{D-1}, b_0, b_1, \dots, b_{D-1}, b_0, b_1, \dots, b_{D-1}, \dots\,] \tag{16.1.9}$$

Figure 16.1.6 depicts such a sequence for $D = 4$. The filtering approach to generating such a periodic sequence is to think of it as the impulse response of a filter and then excite the filter with an impulsive input. The following filter has Eq. (16.1.9) as its causal impulse response:

$$H(z) = \frac{b_0 + b_1z^{-1} + b_2z^{-2} + \cdots + b_{D-1}z^{-(D-1)}}{1 - z^{-D}} \tag{16.1.10}$$

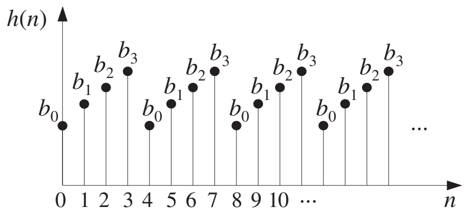


Fig. 16.1.6 Discrete-time periodic signal of period $D = 4$.

As a concrete example, consider the case $D = 4$ with transfer function:

$$H(z) = \frac{b_0 + b_1z^{-1} + b_2z^{-2} + b_3z^{-3}}{1 - z^{-4}} \tag{16.1.11}$$

Its causal impulse response can be obtained by expanding the denominator using the infinite geometric series:

$$\begin{aligned} H(z) &= (b_0 + b_1z^{-1} + b_2z^{-2} + b_3z^{-3})(1 + z^{-4} + z^{-8} + \cdots\,) \\ &= (b_0 + b_1z^{-1} + b_2z^{-2} + b_3z^{-3})\cdot 1 \\ &\quad + (b_0 + b_1z^{-1} + b_2z^{-2} + b_3z^{-3})\cdot z^{-4} \\ &\quad + (b_0 + b_1z^{-1} + b_2z^{-2} + b_3z^{-3})\cdot z^{-8} + \cdots \end{aligned}$$

Picking out the coefficients of the powers of z^{-1} gives the causal periodic impulse response sequence:

$$\mathbf{h} = [b_0, b_1, b_2, b_3, b_0, b_1, b_2, b_3, b_0, b_1, b_2, b_3, \dots\,] \tag{16.1.12}$$

Writing

$$(1 - z^{-D})H(z) = b_0 + b_1z^{-1} + \cdots + b_{D-1}z^{-(D-1)}$$

and transforming it to the time domain gives the difference equation for $h(n)$:

$$h(n) = h(n-D) + b_0\delta(n) + b_1\delta(n-1) + \cdots + b_{D-1}\delta(n-D+1)$$

For example, with $D = 4$:

$$h(n) = h(n-4) + b_0\delta(n) + b_1\delta(n-1) + b_2\delta(n-2) + b_3\delta(n-3) \tag{16.1.13}$$

which generates Eq. (16.1.12). Indeed, iterating Eq. (16.1.13) with causal initial conditions gives:

$$\begin{aligned} &h(0) = b_0, \quad h(1) = b_1, \quad h(2) = b_2, \quad h(3) = b_3 \\ &h(n) = h(n-4), \quad \text{for } n \geq 4 \end{aligned}$$

The transfer function (16.1.11) can be realized in its direct or canonical forms. It is instructive to look at the time-domain operation of these two realizations. The direct form is depicted in Fig. 16.1.7. Note that there are $D = 4$ feedback delays, but only $D - 1 = 3$ feed-forward ones. The corresponding sample processing algorithm will be as follows:

```
for $n = 0, 1, 2, \dots$ do:
    $v_0 = \delta(n)$
    $y = w_0 = w_4 + b_0v_0 + b_1v_1 + b_2v_2 + b_3v_3$
    delay$(3, \mathbf{v})$
    delay$(4, \mathbf{w})$
```

The following table shows the contents of the delay registers at successive sampling instants. The v and w delays are initialized to zero. Note that the v_0 column is the impulsive input $\delta(n)$. Similarly, the v_1 column represents the delayed version of v_0, that is, $\delta(n-1)$, and v_2, v_3 represent $\delta(n-2)$, $\delta(n-3)$.

n	v_0	v_1	v_2	v_3	w_0	w_1	w_2	w_3	w_4	$y = w_0$
0	1	0	0	0	b_0	0	0	0	0	b_0
1	0	1	0	0	b_1	b_0	0	0	0	b_1
2	0	0	1	0	b_2	b_1	b_0	0	0	b_2
3	0	0	0	1	b_3	b_2	b_1	b_0	0	b_3
4	0	0	0	0	b_0	b_3	b_2	b_1	b_0	b_0
5	0	0	0	0	b_1	b_0	b_3	b_2	b_1	b_1
6	0	0	0	0	b_2	b_1	b_0	b_3	b_2	b_2
7	0	0	0	0	b_3	b_2	b_1	b_0	b_3	b_3
8	0	0	0	0	b_0	b_3	b_2	b_1	b_0	b_0

(16.1.14)

During the first four sampling instants, $n = 0, 1, 2, 3$, the initial impulse travels through the v-delays and eventually these delays empty out. The only purpose of the first four iterations of the algorithm is to *load* the w-delay registers with the values b_i of the signal. Indeed as can be seen from the table, the contents of the w-registers at time $n = 4$ are the b_i values loaded in *reverse order*:

$$[w_1, w_2, w_3, w_4] = [b_3, b_2, b_1, b_0]$$

and, in general, at time $n = D$ the w-delays will contain the values:

$$w_i = b_{D-i}, \qquad i = 1, 2, \dots, D \tag{16.1.15}$$

For $n \geq 4$, the input part of the block diagram no longer plays a part because the v-delays are empty, whereas the contents of the w-delays recirculate according to $w_0 = w_4$, or $w_0 = w_D$, in general. Thus, an alternative way to formulate the sample processing algorithm for the generation of a periodic waveform is to break the algorithm into two parts: an initialization part

```
for $n = 0, 1, \dots, D-1$ do:
    $w_0 = b_n$
    delay$(D, \mathbf{w})$
```

(16.1.16)

and a steady state part

```
repeat forever:
    $w_0 = w_D$
    delay$(D, \mathbf{w})$
```

(16.1.17)

where $y = w_0$ is the corresponding output, as seen in Fig. 16.1.7.

Equation (16.1.16) effectively loads the w-registers with the b-values in *reverse order*. Then, Eq. (16.1.17) repeatedly recirculates the delay line, producing the periodic output.

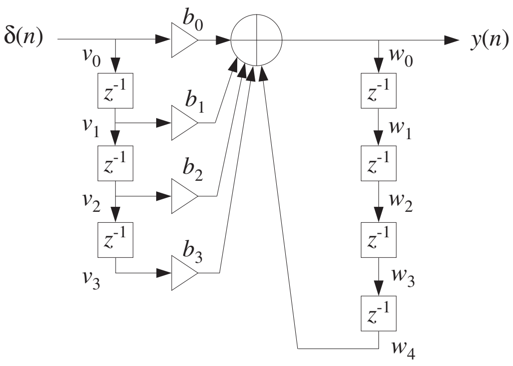


Fig. 16.1.7 Periodic generator in direct form.

The canonical realization of the transfer function (16.1.11) is shown in Fig. 16.1.8. Its operation is now by the following sample processing algorithm:

```
for $n = 0, 1, 2, \dots$ do:
    $w_0 = w_D + \delta(n)$
    $y = b_0 w_0 + b_1 w_1 + \cdots + b_{D-1} w_{D-1}$
    delay$(D, \mathbf{w})$
```

The contents of the w-register at successive sampling instants are shown below:

n	w_0	w_1	w_2	w_3	w_4	$y = b_0w_0 + b_1w_1 + b_2w_2 + b_3w_3$
0	1	0	0	0	0	b_0
1	0	1	0	0	0	b_1
2	0	0	1	0	0	b_2
3	0	0	0	1	0	b_3
4	1	0	0	0	1	b_0
5	0	1	0	0	0	b_1
6	0	0	1	0	0	b_2
7	0	0	0	1	0	b_3

The initial impulse gets trapped into the recirculating w-delay line, each time passing through only one of the b_i filter multipliers as it gets shifted from register to register.

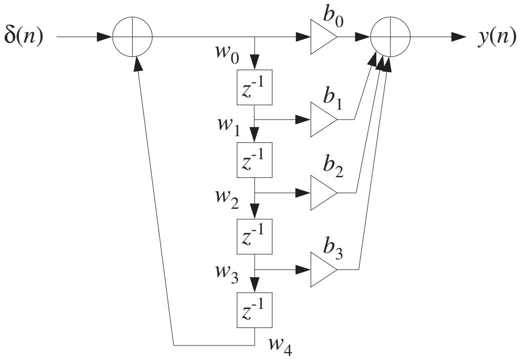


Fig. 16.1.8 Periodic generator in canonical form.

An intuitive way of understanding the operation of the canonical form is to separate out the common set of w-delays and think of the transfer function as the cascade of the two filters:

$$H(z) = \frac{1}{1 - z^{-4}} \cdot N(z) \tag{16.1.18}$$

where $N(z) = b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}$. Fig. 16.1.9 shows this interpretation. The impulse response of the first factor is a train of pulses separated by the desired period $D = 4$ with $D - 1$ zeros in between, that is, the sequence:

$$[1,0,0,0,1,0,0,0,1,0,0,0,\dots\,]$$

Every time one of these impulses hits the FIR filter $N(z)$, it generates the impulse response $\mathbf{b}=[b_0,b_1,b_2,b_3]$, translated in time to match the time of that input impulse, as required by time invariance. Because the duration of $\mathbf{b}$ is only D samples, there is no overlap of the generated impulse responses, that is, each impulse response ends just before the next one begins.

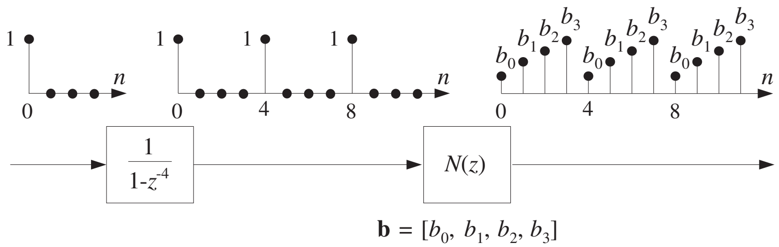


Fig. 16.1.9 Periodic pulse train causes periodic output.

Both the direct and canonical realizations can be implemented using circular buffers. For example, the direct-form sample processing algorithm described by Eqs.(16.1.16) and (16.1.17) can be written with the help of the circular version of the delay routine, `cdelay`, as follows:

```
for  $n = 0, 1, \dots, D-1$  do:
    $*p = b_n$
    $\text{cdelay}(D, \mathbf{w}, \&p)$
```
(16.1.19)

and

```
repeat forever:
    $*p = \text{tap}(D, \mathbf{w}, p, D)$
    $\text{cdelay}(D, \mathbf{w}, \&p)$
```
(16.1.20)

As discussed in Chapter 4, the circular pointer must be initialized by $p=w$. Eq. (16.1.19) loads the circular buffer with the D waveform samples, and then Eq. (16.1.20) reproduces them periodically. Alternatively, we may use the routines `cdelay2` and `tap2` that employ the offset index q such that $p=w+q$. Noting that $*p=p[0]=w[q]$, we have the generation algorithm:

```
for  $n = 0, 1, \dots, D-1$  do:
    $w[q] = b_n$
    $\text{cdelay2}(D, \&q)$
```
(16.1.21)

and

```
repeat forever:
    $w[q] = \text{tap2}(D, \mathbf{w}, q, D)$
    $\text{cdelay2}(D, \&q)$
```
(16.1.22)

where q must be initialized by $q=0$.

These circular versions provide more efficient implementations than the direct form because at each time instant only the current w-register pointed to by p is updated—being loaded with the value of the *last* state, that is, the Dth state. By contrast, in the linear delay-line implementation, the entire delay line must be shifted at each time instant.

To appreciate how the circular delay line is updated, the table below shows the contents of the vector $\mathbf{w}$ for the case $D=4$, at successive time instants (grouped every $D+1=5$ samples):

n	q	w_0	w_1	w_2	w_3	w_4	y
0	0	$\uparrow b_0$	0	0	0	0	b_0
1	4	b_0	0	0	0	$\uparrow b_1$	b_1
2	3	b_0	0	0	$\uparrow b_2$	b_1	b_2
3	2	b_0	0	$\uparrow b_3$	b_2	b_1	b_3
4	1	b_0	$\uparrow b_0$	b_3	b_2	b_1	b_0
5	0	$\uparrow b_1$	b_0	b_3	b_2	b_1	b_1
6	4	b_1	b_0	b_3	b_2	$\uparrow b_2$	b_2
7	3	b_1	b_0	b_3	$\uparrow b_3$	b_2	b_3
8	2	b_1	b_0	$\uparrow b_0$	b_3	b_2	b_0
9	1	b_1	$\uparrow b_1$	b_0	b_3	b_2	b_1
10	0	$\uparrow b_2$	b_1	b_0	b_3	b_2	b_2
11	4	b_2	b_1	b_0	b_3	$\uparrow b_3$	b_3
12	3	b_2	b_1	b_0	$\uparrow b_0$	b_3	b_0
13	2	b_2	b_1	$\uparrow b_1$	b_0	b_3	b_1
14	1	b_2	$\uparrow b_2$	b_1	b_0	b_3	b_2
15	0	$\uparrow b_3$	b_2	b_1	b_0	b_3	b_3
16	4	b_3	b_2	b_1	b_0	$\uparrow b_0$	b_0
17	3	b_3	b_2	b_1	$\uparrow b_1$	b_0	b_1
18	2	b_3	b_2	$\uparrow b_2$	b_1	b_0	b_2
19	1	b_3	$\uparrow b_3$	b_2	b_1	b_0	b_3
20	0	$\uparrow b_0$	b_3	b_2	b_1	b_0	b_0
21	4	b_0	b_3	b_2	b_1	$\uparrow b_1$	b_1
22	3	b_0	b_3	b_2	$\uparrow b_2$	b_1	b_2
23	2	b_0	b_3	$\uparrow b_3$	b_2	b_1	b_3
24	1	b_0	$\uparrow b_0$	b_3	b_2	b_1	b_0

The up-arrow symbol ↑ indicates the w-register pointed to by the current value of the output pointer p. The pointer p cycles around every $D+1=5$ samples even though the output is periodic every $D=4$ samples. Equivalently, the current w-register can be determined by the corresponding value of the offset index q, that is, the register $w[q]$.

In the direct form version of Eq. (16.1.14), the linear delay line recirculates once every $D=4$ samples, so that at $n=8$ the state vector $\mathbf{w}$ is the same as at time $n=4$. By contrast, the circular delay line recirculates much more slowly, that is, every $D(D+1)=20$ samples, so that the buffer $\mathbf{w}$ has the same contents at times $n=4$ and $n=24$. In both cases, the first $D=4$ samples correspond to the initialization phases of Eqs. (16.1.16) and (16.1.19).

The following program segment illustrates the initialization and usage of the circular-buffer generation algorithms. It is assumed that the D-dimensional array of values $b[i]$, $i=0,1,\dots,D-1$, has already been defined:

```
double *b, *w, *p;

b = (double *) calloc(D, sizeof(double));       definition of b[n] is not shown
w = (double *) calloc(D+1, sizeof(double));     (D+1)-dimensional
p = w;                                          initialize circular pointer

for (n=0; n<D; n++) {                           initialization part
       *p = b[n];                               fill buffer with b[n]'s
       printf("%lf\n", *p);                     current output
       cdelay(D, w, &p);                        update circular delay line
       }

for (n=D; n<Ntot; n++) {                        steady state part
       *p = tap(D, w, p, D);                    first state = last state
       printf("%lf\n", *p);                     current output
       cdelay(D, w, &p);                        update circular delay line
       }
```

For comparison, we also list the linear delay-line version of Eqs. (16.1.16) and (16.1.17):

```
for (n=0; n<D; n++) {                           initialization part
       w[0] = b[n];                             fill buffer with b[n]'s
       printf("%lf\n", w[0]);                   current output
       delay(D, w);                             update linear delay line
       }

for (n=D; n<Ntot; n++) {                        steady state part
       w[0] = w[D];                             first state = last state
       printf("%lf\n", w[0]);                   current output
       delay(D, w);                             update linear delay line
       }
```

The spectra of *double-sided* periodic signals consist of sharp spectral lines at the *harmonics*, which are integral multiples of the fundamental frequency.

One-sided, or causal, periodic sequences such as the sequence (16.1.9), have *comb-like* spectra, as shown in Fig. 16.1.10, with *dominant* peaks at the harmonics.

The spectrum of (16.1.9) can be obtained by setting $z=e^{j\omega}=e^{2\pi jf/f_s}$ in the generating transfer function (16.1.10). Using the trigonometric identity

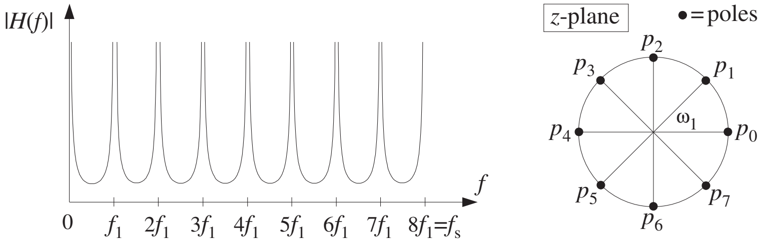


Fig. 16.1.10 Comb-like frequency spectrum, for $D=8$.

$$1-z^{-D}=1-e^{-j\omega D}=e^{-j\omega D/2}\left(e^{j\omega D/2}-e^{-j\omega D/2}\right)=2je^{j\omega D/2}\sin\left(\frac{\omega D}{2}\right)$$

we have

$$|H(\omega)|=\frac{|N(\omega)|}{|1-e^{-j\omega D}|}=\frac{|N(\omega)|}{2\left|\sin\left(\frac{\omega D}{2}\right)\right|}$$

or, replacing $\omega=2\pi f/f_s$ in terms of the physical frequency f:

$$|H(f)|=\frac{|N(f)|}{2\left|\sin\left(\frac{\pi fD}{f_s}\right)\right|}\tag{16.1.23}$$

The peaks in the spectrum are due to the zeros of the denominator which vanishes at the harmonics, that is,

$$\sin\left(\frac{\pi fD}{f_s}\right)=0\quad\Rightarrow\quad\frac{\pi fD}{f_s}=\pi m$$

with m an integer. Solving for f:

$$f_m=m\frac{f_s}{D}=mf_1\tag{16.1.24}$$

where we denoted the fundamental frequency by $f_1=f_s/D$.

Because the spectrum $H(f)$ is periodic in f with period f_s, we may restrict the index m to the D values $m=0,1,\dots,D-1$, which keep f within the Nyquist interval $[0,f_s)$. These harmonics correspond to the *poles* of $H(z)$. Indeed, solving for the zeros of the denominator, we have

$$1-z^{-D}=0\quad\Rightarrow\quad z^D=1$$

with the D solutions:

$$z=p_m=e^{j\omega_m},\qquad\omega_m=\frac{2\pi f_m}{f_s}=\frac{2\pi m}{D},\quad m=0,1,\dots,D-1$$

Note that they are the Dth roots of unity on the unit circle.

16.1.3 Wavetable Generators

The linear and circular buffer implementations of the direct form generator both use $(D+1)$-dimensional buffers $\mathbf{w} = [w_0, w_1, , \dots, w_D]$, whereas the periodic waveform has only D samples in one period: $\mathbf{b} = [b_0, b_1, \dots, b_{D-1}]$. As we saw, this causes the buffer contents to recirculate periodically.

A simpler approach is to use a buffer of length D, that is, $\mathbf{w} = [w_0, w_1, , \dots, w_{D-1}]$ referred to as a *wavetable* and store in it a copy of one period of the desired waveform. The periodic waveform is then generated by repeatedly cycling over the wavetable with the aid of a circular pointer p, which always points at the current *output* sample.

Figure 16.1.11 shows such a wavetable for the case $D = 4$. Also shown are the positions of the circular pointer at successive sampling instants n, and the corresponding values of the offset index q, such that $*p = w[q]$.

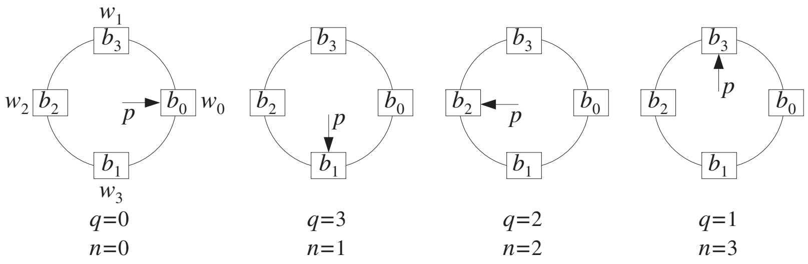


Fig. 16.1.11 Circular pointer cycles over wavetable.

As in the previous section, the waveform samples are stored in *circular reverse order*. The reverse loading of the table with the D waveform samples can be done with the following loop, initialized at $p = w$:

```
for  $i = 0, 1, \dots, D-1$  do:
    $*p = b_i$
    cdelay$(D-1, \mathbf{w}, \&p)$
```
(16.1.25)

or, in terms of the offset index q, initialized at $q = 0$:

```
for  $i = 0, 1, \dots, D-1$  do:
    $w[q] = b_i$
    cdelay2$(D-1, \&q)$
```
(16.1.26)

Note that the only difference with Eqs. (16.1.19) and (16.1.21) is the dimension of the buffer $\mathbf{w}$, which requires the `cdelay` routine to have argument $D-1$ instead of D. Upon exit from these initialization loops, the pointer p has wrapped around once and points again at the *beginning* of the buffer, $p = w$ or $q = 0$. Alternatively, the initialization of the wavetable can be done with:

```
for  $i = 0, 1, \dots, D-1$  do:
    $w[i] = b[(D-i)\%D]$
```
(16.1.27)

where the modulo operation is felt only at $i = 0$, giving in this case $w[0] = b[D\%D] = b[0]$. After the wavetable is loaded with the waveform samples, the circular pointer can be made to cycle over the wavetable by successive calls to `cdelay`:

```
repeat forever:
    output  $y = *p$
    cdelay$(D-1, \mathbf{w}, \&p)$
```
(16.1.28)

Each call to `cdelay` circularly *decrements* the pointer p to point to the *next* entry in the wavetable. In terms of the offset index q, we have similarly:

```
repeat forever:
    output  $y = w[q]$
    cdelay2$(D-1, \&q)$
```
(16.1.29)

Because the waveform was loaded in reverse order, decrementing the pointer will generate the waveform in forward order, as shown in Fig. 16.1.11.

Traditionally in the computer music literature, the wavetable is loaded in *forward* order, that is, $w[i] = b[i]$, $i = 0, 1, \dots, D-1$ and the circular pointer is *incremented* circularly [110,111]. In signal processing language, this corresponds to time advance instead of time delay. We will see how to implement time advances with the help of the generalized circular delay routine `gdelay2` discussed below.

The following program segment illustrates the initialization (16.1.25) and the steady-state operation (16.1.28):

```
double *b, *w, *p;

b = (double *) calloc(D, sizeof(double));       definition of b[i] is not shown
w = (double *) calloc(D, sizeof(double));       Note, w is D-dimensional
p = w;                                          initialize circular pointer

for (i=0; i<D; i++) {                           initialization:
       *p = b[i];                               fill buffer with b[i]'s
       cdelay(D-1, w, &p);                      decrement pointer
       }

for (n=0; n<Ntot; n++) {                        steady state operation:
       printf("%lf\n", *p);                     current output
       cdelay(D-1, w, &p);                      decrement pointer
       }
```

Often, it is desired to generate a *delayed* version of the periodic waveform. Instead of loading a delayed period into a new wavetable, we can use the same wavetable, but start cycling over it at a shifted position. For a delay of m time units such that $0 \le m \le D-1$, the starting pointer p and corresponding offset index q should be:

$$p = w + m, \qquad q = m \tag{16.1.30}$$

To understand this, denote by $b(n)$ the original periodic sequence of period D, and let $y(n) = b(n-m)$ be its delayed version by m units. The starting sample will be $y(0) = b(-m)$, but because of the periodicity, we have $y(0) = b(-m) = b(D-m)$, which by the reverse loading of the w-buffer is equal to $w[m]$ according to Eq. (16.1.27). Thus, the starting sample will be $y(0) = w[m]$, corresponding to offset $q = m$. A *time advance* by m units can be implemented by starting at $q = -m$, which wraps to the positive value $q = D - m$.

For example, referring to Fig. 16.1.11, starting at position m and cycling clockwise generates the successively delayed periodic sequences:

$$\begin{aligned}
m = 0: &\quad [b_0, b_1, b_2, b_3, b_0, b_1, b_2, b_3, b_0, b_1, b_2, b_3, \dots] \\
m = 1: &\quad [b_3, b_0, b_1, b_2, b_3, b_0, b_1, b_2, b_3, b_0, b_1, b_2, \dots] \\
m = 2: &\quad [b_2, b_3, b_0, b_1, b_2, b_3, b_0, b_1, b_2, b_3, b_0, b_1, \dots] \\
m = 3: &\quad [b_1, b_2, b_3, b_0, b_1, b_2, b_3, b_0, b_1, b_2, b_3, b_0, \dots]
\end{aligned}$$

Wavetable synthesis of periodic waveforms lies at the heart of many computer music applications and programs, such as *Music V* and its descendants [110–131]. Generating a single periodic wave is not musically very interesting. However, the generated waveform can be subjected to further operations to create more complex and interesting sounds, such as:

- Varying its amplitude or envelope to imitate the attack and decay of various instruments, or for imposing tremolo-type effects.
- Varying or modulating its frequency to imitate various effects such as vibrato, glissando, or portamento. This also leads to the popular family of FM synthesizers.
- Sending it through linear or nonlinear, time-invariant or time-varying filtering operations, generically known as *waveshaping* operations, which modify it further. They can be used to create models of various instruments, such as plucked strings or drums, or to superimpose various audio effects, such as reverb, stereo imaging, flanging, and chorusing [124–130].
- Adding together the outputs of several wavetables with different amplitudes and frequencies to imitate additive Fourier synthesis of various sounds.

The possibilities are endless. They have been and are actively being explored by the computer music community.

Here, we can only present some simple examples that illustrate the usage of wavetables. We begin by discussing how the *frequency* of the generated waveform may be changed.

Given a wavetable of length D, the period of the generated waveform is given by Eq. (16.1.8), $T_D = DT$, and its fundamental frequency by

$$f = \frac{f_s}{D} \tag{16.1.31}$$

The frequency f can be changed in two ways: by varying the sampling rate f_s or changing the effective length D of the basic period. Changing the sampling rate is not practical, especially when one is dealing with several wavetables of different frequencies, although digital music synthesizers have been built based on this principle. Thus, varying D is of more practical interest and is the preferred approach in most music synthesis programs. Replacing D by a smaller length $d \le D$ will increase the fundamental frequency to:

$$\boxed{f = \frac{f_s}{d}} \tag{16.1.32}$$

and will decrease the period to $T_d = dT$.

For example, if $d = D/2$, the effective frequency is doubled $f = f_s/(D/2) = 2f_s/D$. Replacing D by $d = D/2$ is equivalent to cycling the pointer p over a *subset* of the circular buffer **w** consisting of *every other* point in the buffer. Fig. 16.1.12 shows the positions of the pointer p at successive time instants n for the case of $D = 8$ and $d = D/2 = 4$. Skipping every other point can be accomplished by performing *two* calls to `cdelay2` at each time, that is, replacing Eq. (16.1.29) by:

```
repeat forever:
    output  $y = w[q]$
    cdelay2$(D-1, \&q)$
    cdelay2$(D-1, \&q)$
```

(16.1.33)

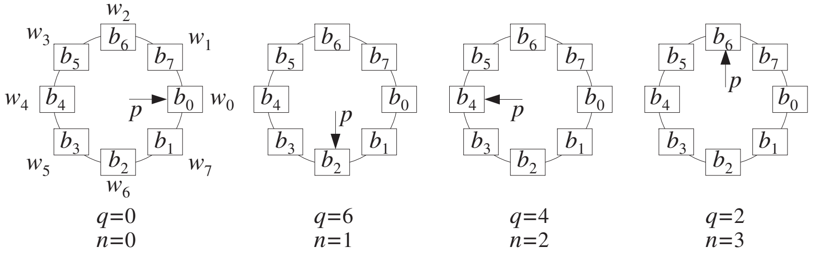


Fig. 16.1.12 Circular pointer is decremented by 2, generating a subsequence of period 4.

The two calls to `cdelay2` effectively decrement the offset index q by *two*, that is, $q = q - 2$. The generated q values will be $q = 0$, $q = 0 - 2$, which wraps modulo-8 to $q = 6$, $q = 6 - 2 = 4$, $q = 4 - 2 = 2$, and so on. Thus, the generated subsequence will be the periodic repetition of the contents of the registers $[w_0, w_6, w_4, w_2]$, or, as shown in Fig. 16.1.12:

$$[b_0, b_2, b_4, b_6, b_0, b_2, b_4, b_6, b_0, b_2, b_4, b_6, b_0, b_2, b_4, b_6, \dots]$$

which repeats with period $d = 4$. It should be compared to the full wavetable sequence, which repeats every $D = 8$ samples:

$$[b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7, \dots]$$

Similarly, cycling every four wavetable samples will generate the periodic subsequence of period $d = D/4 = 8/4 = 2$ and frequency $f = 4f_s/D$:

$$[b_0, b_4, b_0, b_4, b_0, b_4, b_0, b_4, b_0, b_4, b_0, b_4, b_0, b_4, b_0, b_4, \dots]$$

In Fig. 16.1.12, it corresponds to decrementing the pointer p or offset q by 4, that is, $q = q - 4$, and can be implemented by inserting 4 calls to `cdelay2` at each iteration:

```
repeat forever:
    output  $y = w[q]$
    cdelay2($D-1$, &$q$)
    cdelay2($D-1$, &$q$)
    cdelay2($D-1$, &$q$)
    cdelay2($D-1$, &$q$)                    (16.1.34)
```

Rather than making multiple calls to `cdelay2`, we define a *generalized* version of this routine, `gdelay2.c`, which allows for an arbitrary *real-valued shift* of the pointer index q:

```
/* gdelay2.c - generalized circular delay with real-valued shift */

void gdelay2(D, c, q)
int D;
double c, *q;                           c=shift, q=offset index
{
       *q -= c;                         decrement by c

       if (*q < 0)
              *q += D+1;

       if (*q > D)
              *q -= D+1;
}
```

There are two basic differences with `cdelay2`. First, the offset index q is allowed to take on real values as opposed to integer values. Second, each call decrements q by the real-valued shift c, that is, $q = q - c$. The reason for allowing real values will become clear shortly. Note that `cdelay2` is a special case of `gdelay2` with $c = 1.0$. In terms of this routine, Eqs. (16.1.33) or (16.1.34) will read as:

```
repeat forever:
    output  $y = w[q]$
    gdelay2($D-1, c$, &$q$)                 (16.1.35)
```

with $c = 2$ or $c = 4$, respectively. The successive calls to `gdelay2` in Eq. (16.1.35) update the offset q according to the iteration:

$$q_{n+1} = (q_n - c)\%D \tag{16.1.36}$$

where mod-D is used because the dimension of the circular buffer is D. Generally, we have the following relationship between the shift c and the sub-period d:

$$\boxed{c = \frac{D}{d}} \tag{16.1.37}$$

which gives the *number of times* the sub-period d fits into the full period D, or,

$$\boxed{d = \frac{D}{c}} \tag{16.1.38}$$

which expresses d as a *fraction* of the full period D, or,

$$\boxed{D = cd} \tag{16.1.39}$$

Combining Eqs. (16.1.32) and (16.1.38), gives for the frequency of the generated subsequence:

$$\boxed{f = \frac{f_s}{d} = c\frac{f_s}{D}} \tag{16.1.40}$$

or, in terms of the digital frequency in radians/sample:

$$\omega = \frac{2\pi f}{f_s} = \frac{2\pi}{d} = \frac{2\pi c}{D} \tag{16.1.41}$$

Equivalently, given a desired frequency f and table length D, we obtain the required value of the shift:

$$\boxed{c = D\frac{f}{f_s} = DF} \tag{16.1.42}$$

where $F = f/f_s$ is the digital frequency in units of *cycles per sample*. It follows from Eq. (16.1.42) that c is the *number of cycles* of the subsequence that are contained in the D samples of the full wavetable.

So far, our discussion assumed that both the sub-length d and the shift c were integers. Because of the constraint (16.1.39), such restriction would not allow too many choices for c or d, and consequently for f. Therefore, c, d, and q are allowed to take on real values in the definition of `gdelay2`.

To keep f within the *symmetric* Nyquist interval $|f| \le f_s/2$, requires that c satisfy the condition: $|cf_s/D| \le f_s/2$, or,

$$|c| \le \frac{D}{2} \quad \Rightarrow \quad -\frac{D}{2} \le c \le \frac{D}{2} \tag{16.1.43}$$

Negative values of c correspond to negative frequencies f. This is useful for introducing 180° phase shifts in waveforms. Any value of c in the range $D/2 < c \le D$ is wrapped modulo-D to the value $c - D$, which lies in the negative part of the Nyquist interval (16.1.43). The wrapping $c \to c - D$ is equivalent to the frequency wrapping $f \to f - f_s$.

As we mentioned earlier, in the computer music literature, the circular wavetable is loaded with the waveform in forward order. Cycling over the wavetable at frequency $f = cf_s/D$ is accomplished by *incrementing* the offset index q according to the iteration:

$$q_{n+1} = (q_n + c)\%D \tag{16.1.44}$$

Such "forward" versions can be implemented easily by the routine `gdelay2` by calling it in Eq. (16.1.35) with c replaced by $-c$. The wrap-around tests in `gdelay2` always force q to lie in the range $0 \le q < D$. If q is not an integer in that range, then it cannot be an array index that defines the output buffer sample $y = w[q]$. However, it can be approximated by an integer, for example, by truncating down, or truncating up, or rounding to the nearest integer:

$$\begin{aligned} i &= \lfloor q \rfloor && \text{(truncating down)} \\ j &= \lfloor q + 1 \rfloor\%D && \text{(truncating up)} \\ k &= \lfloor q + 0.5 \rfloor\%D && \text{(rounding)} \end{aligned} \tag{16.1.45}$$

The modulo-D operation is necessary to keep the index within the circular buffer range $\{0, 1, \dots, D-1\}$. The returned output will be in these cases:

$$\begin{aligned} y &= w[i] && \text{(truncating down)} \\ y &= w[j] && \text{(truncating up)} \\ y &= w[k] && \text{(rounding)} \end{aligned} \tag{16.1.46}$$

For example, in the first case, Eq. (16.1.35) will be replaced by:

```
repeat forever:
    $i = \lfloor q \rfloor$
    output  $y = w[i]$
    gdelay2$(D-1, c, \&q)$
```

(16.1.47)

and similarly in the other cases.

Because q lies (circularly) between the integers i and j, a more accurate output can be obtained by *linearly interpolating* between the wavetable values $w[i]$ and $w[j]$, that is, returning the output:

$$y = w[i] + (q - i)\big(w[j] - w[i]\big) \tag{16.1.48}$$

The geometric meaning of Eq. (16.1.48) is depicted in Fig. 16.1.13, where y lies on the straight line connecting $w[i]$ and $w[j]$. Note that $j = i + 1$, except when q falls in the last integer subdivision of the $[0, D)$ interval, that is, when $D - 1 \le q < D$. In that case, $i = D - 1$ and $j = D\%D = 0$, and we must interpolate between the values $w[D-1]$ and $w[0]$, as shown in Fig. 16.1.13. Equation (16.1.48) correctly accounts for this case with j computed by Eq. (16.1.45), or equivalently by $j = (i+1)\%D$.

The interpolation method produces a more accurate output than the other methods, but at the expense of increased computation. The rounding method is somewhat more accurate than either of the truncation methods. The differences between the methods become unimportant as the length D of the wavetable increases. In computer music applications typical values of D are 512–32768. The nature of the approximation error for the truncation method and the other methods has been studied in [118–120].

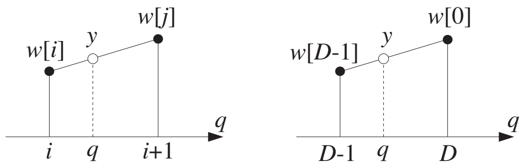


Fig. 16.1.13 Linear interpolation between two successive buffer samples.

The generation algorithm (16.1.47) starts producing the periodic sequence at the beginning of the w buffer, that is, with $q = 0$. If a delayed version of the subsequence is needed, we may shift the initial value of the offset q to a new starting position as in Eq. (16.1.30). However, because q is measured in multiples of the shift c, we must replace Eq. (16.1.30) by

$$q = mc = mDF \tag{16.1.49}$$

Because `gdelay2` decrements q, we can obtain Eq. (16.1.49) by starting with $q = 0$ and calling `gdelay2` once with the *opposite argument*:

$$\text{gdelay2}(D-1, -mc, \&q) \tag{16.1.50}$$

This expression implements both time delays ($m > 0$) and time advances ($m < 0$). Because mc must be in the interval $|mc| < D/2$, it follows that the allowed values of m are in the interval $|m| < d/2$. After this call, the generation algorithm Eq. (16.1.47) may be started with the desired value of the shift c.

Example 16.1.2: The eight waveform samples:

$$\mathbf{b} = [b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7]$$

are stored in (circular) reverse order in the 8-dimensional circular wavetable:

$$\mathbf{w} = [w_0, w_1, w_2, w_3, w_4, w_5, w_6, w_7]$$

It is desired to generate a periodic subsequence of period $d = 3$. Determine this subsequence when the output is obtained by the four methods of: (a) truncating down, (b) truncating up, (c) rounding, and (d) linear interpolation.

Solution: Here, $D = 8$ so that the shift is $c = D/d = 8/3$, which is not an integer. There are $d = 3$ possible values of the offset index q obtained by iterating Eq. (16.1.36):

$$q_0 = 0$$

$$q_1 = q_0 - c = -\frac{8}{3} \equiv 8 - \frac{8}{3} = \frac{16}{3} = 5\frac{1}{3}$$

$$q_2 = q_1 - c = \frac{16}{3} - \frac{8}{3} = \frac{8}{3} = 2\frac{2}{3}$$

The next q will be $q_3 = q_2 - c = (8/3) - (8/3) = 0$, and the above three values will be repeated. Fig. 16.1.14 shows the relative locations of the three q's with respect to the circular buffer indices.

The three q's can be obtained quickly by dividing the circular buffer D into d equal parts and counting clockwise. The direction of the three q-arrows in Fig. 16.1.14 are at relative angles ω as given by Eq. (16.1.41); here, $\omega = 2\pi/3$.

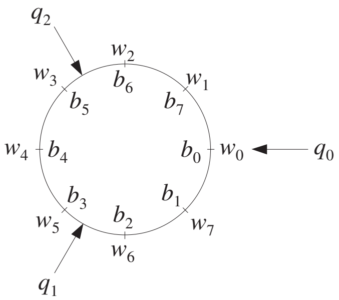


Fig. 16.1.14 Successive positions of q when $d = 3$.

It is seen that q_1 points between the buffer samples $w[5]$ and $w[6]$. If we truncate down, then, we will output the content of $w[5] = b_3$, and if we truncate up, $w[6] = b_2$. Because, q_1 points nearer to $w[5]$ than to $w[6]$, we will output $w[5] = b_3$, if we round to the nearest buffer location. If we interpolate linearly between $w[5]$ and $w[6]$, we will output the value:

$$y = w[5] + (q_1 - i_1)\,(w[6] - w[5]) = b_3 + \frac{1}{3}(b_2 - b_3) = \frac{1}{3}b_2 + \frac{2}{3}b_3$$

where $i_1 = \lfloor q_1 \rfloor = 5$, and $q_1 - i_1 = 1/3$. Similarly, the next offset index q_2 points between $w[2]$ and $w[3]$. If we truncate down, we will output $w[2] = b_6$, and if we truncate up, $w[3] = b_5$. If we round, we will output $w[3] = b_5$ because q_2 points closer to $w[3]$ than to $w[2]$. And, if we interpolate, we will output the value:

$$y = w[2] + (q_2 - i_2)\,(w[3] - w[2]) = b_6 + \frac{2}{3}(b_5 - b_6) = \frac{2}{3}b_5 + \frac{1}{3}b_6$$

where $i_2 = \lfloor q_2 \rfloor = 2$, and $q_2 - i_2 = 2/3$.

To summarize, at successive time instants $n = 0, 1, 2, \dots$, the offset q cycles repeatedly over the three values $\{q_0, q_1, q_2\}$. The output associated with each q depends on the chosen approximation method. For the four methods, we will generate the following period-3 sequences:

$$\begin{array}{ll} [b_0, b_3, b_6, b_0, b_3, b_6, \dots\,] & \text{(truncate down)} \\ [b_0, b_2, b_5, b_0, b_2, b_5, \dots\,] & \text{(truncate up)} \\ [b_0, b_3, b_5, b_0, b_3, b_5, \dots\,] & \text{(round)} \end{array}$$

and if we interpolate:

$$[b_0, \frac{1}{3}b_2 + \frac{2}{3}b_3, \frac{2}{3}b_5 + \frac{1}{3}b_6, b_0, \frac{1}{3}b_2 + \frac{2}{3}b_3, \frac{2}{3}b_5 + \frac{1}{3}b_6, \dots\,]$$

Had we used the computer music convention of forward loading the circular buffer and incrementing q according to Eq. (16.1.44), we would find that the down and up truncated sequences *reverse roles*, that is, the down-truncated sequence would be our up-truncated one. □

Example 16.1.3: Repeat Example 16.1.2 when the subsequence has period $d = 3$, but with an initial delay of $m = 1/2$ samples.

Solution: The desired delay by m samples (in units of c) can be implemented by an initial call to `gdelay2`, with an effective negative shift of $-mc$ as in Eq. (16.1.50). This initializes the offset index by shifting it from $q_0 = 0$ to

$$q_0 = mc = \frac{1}{2} \cdot \frac{8}{3} = \frac{4}{3}$$

The other two q's are obtained as in Example 16.1.2:

$$q_1 = q_0 - c = \frac{4}{3} - \frac{8}{3} = -\frac{4}{3} \equiv 8 - \frac{4}{3} = \frac{20}{3} = 6\frac{2}{3}$$

$$q_2 = q_1 - c = \frac{20}{3} - \frac{8}{3} = 4$$

The three q's are depicted in Fig. 16.1.15. The relative angle between the q-arrows is still $\omega = 2\pi c/D$, but the initial arrow for q_0 is displaced by an angle $\omega_0 = 2\pi m/d = 2\pi(mc)/D$ with respect to the horizontal axis. The original q's are shown by the dashed arrows. Note that the delay by $m = 1/2$ sample in units of c, rotates all the q's by half the original angle of $2\pi/d = 2\pi/3$, that is, by $\pi/3$.

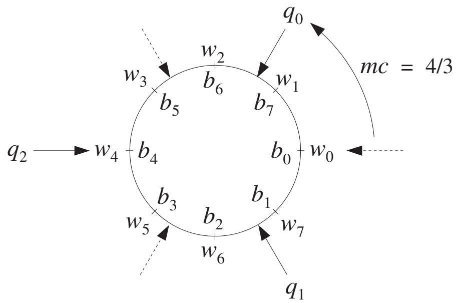


Fig. 16.1.15 Successive positions of q with $d = 3$ and delay $m = 1/2$.

Down-truncation gives the following integer values for the q's and corresponding buffer entries:

$$[q_0, q_1, q_2] = [1, 6, 4]$$

$$\big[w[1], w[6], w[4]\big] = [b_7, b_2, b_4]$$

Up-truncation gives:

$$[q_0, q_1, q_2] = [2, 7, 4]$$

$$\big[w[2], w[7], w[4]\big] = [b_6, b_1, b_4]$$

For the rounding case, we have

$$[q_0, q_1, q_2] = [1, 7, 4]$$

$$\big[w[1], w[7], w[4]\big] = [b_7, b_1, b_4]$$

For the linear interpolation case, we have the outputs:

$$\begin{aligned}
1 < q_0 < 2 \quad &\Rightarrow \quad y_0 = w[1] + (q_0 - 1)\,(w[2] - w[1]) = \frac{1}{3}b_6 + \frac{2}{3}b_7 \\
6 < q_1 < 7 \quad &\Rightarrow \quad y_1 = w[6] + (q_1 - 6)\,(w[7] - w[6]) = \frac{2}{3}b_1 + \frac{1}{3}b_2 \\
q_2 = 4 \quad &\Rightarrow \quad y_2 = w[4] = b_4
\end{aligned}$$

Thus, depending on the output method, the following period-3 delayed subsequences will be generated:

$[b_7, b_2, b_4, b_7, b_2, b_4, \dots]$	(truncate down)
$[b_6, b_1, b_4, b_6, b_1, b_4, \dots]$	(truncate up)
$[b_7, b_1, b_4, b_7, b_1, b_4, \dots]$	(round)
$[y_0, y_1, y_2, y_0, y_1, y_2, \dots]$	(interpolate)

Thus, non-integer delays can be implemented easily. □

The purpose of these examples was to show the mechanisms of producing subsequences of different periods from a fixed wavetable of a given length D.

The generation algorithm of the truncation method given in Eq. (16.1.47), as well as the algorithms of the rounding and interpolation methods, can be programmed easily with the help of the routine `gdelay2`. To this end, we rewrite Eq. (16.1.47) in the following way:

```
repeat forever:
    $i = \lfloor q \rfloor$
    output $y = Aw[i]$
    gdelay2$(D-1, DF, \&q)$
```

(16.1.51)

where we introduced an *amplitude* scale factor A and expressed the shift $c = DF$ in terms of the digital frequency $F = f/f_s$.

With A and F as inputs to the algorithm, we can control the amplitude and frequency of the generated waveform. The following routine `wavgen.c` is an implementation of Eq. (16.1.51):

```
/* wavgen.c - wavetable generator (truncation method) */

void gdelay2();

double wavgen(D, w, A, F, q)            usage: y = wavgen(D, w, A, F, &q);
int D;                                  D = wavetable length
double *w, A, F, *q;                    A = amplitude, F = frequency, q = offset index
{
       double y;
       int i;

       i = (int) (*q);                            truncate down

       y = A * w[i];

       gdelay2(D-1, D*F, q);                      shift c = DF

       return y;
}
```

The following routines `wavgenr` and `wavgeni` are implementations of the *rounding* and *linear interpolation* generator methods of Eqs. (16.1.46) and (16.1.48), with added amplitude and frequency control:

```
/* wavgenr.c - wavetable generator (rounding method) */

void gdelay2();

double wavgenr(D, w, A, F, q)           usage: y = wavgenr(D, w, A, F, &q);
int D;                                  D = wavetable length
double *w, A, F, *q;                    A = amplitude, F = frequency, q = offset index
{
       double y;
       int k;

       k = (int) (*q + 0.5);                    round

       y = A * w[k];

       gdelay2(D-1, D*F, q);                    shift c = DF

       return y;
}
```

```
/* wavgeni.c - wavetable generator (interpolation method) */

void gdelay2();

double wavgeni(D, w, A, F, q)          usage: y = wavgeni(D, w, A, F, &q);
int D;                                 D = wavetable length
double *w, A, F, *q;                   A = amplitude, F = frequency, q = offset index
{
       double y;
       int i, j;

       i = (int) *q;                          interpolate between w[i], w[j]
```

```
        j = (i + 1) % D;

        y = A * (w[i] + (*q - i) * (w[j] - w[i]));

        gdelay2(D-1, D*F, q);                   shift c = DF

        return y;
}
```

In computer music, such routines are known as *wavetable oscillators* [110,111]. They are the workhorses of many digital music synthesis algorithms. Our C routines are modeled after [111]. Figure 16.1.16 depicts such an oscillator in computer music notation.

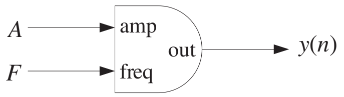


Fig. 16.1.16 Wavetable generator with amplitude and frequency control.

The amplitude and frequency inputs A and F do not have to be constant in time—they can be changing from one sampling instant to the next. In general, the generated signal will be given by:

$$\boxed{y(n)=\text{wavgen}(D,\mathbf{w},A(n),F(n),\&q)} \tag{16.1.52}$$

for $n = 0, 1, 2, \dots$, where $A(n)$ and $F(n)$ can themselves be generated as the outputs of other oscillators to provide amplitude and frequency modulation.

The length-D wavetable $\mathbf{w}$ can be filled with any waveform, such as sinusoids, linear combination of sinusoids of different harmonics, square, triangular, trapezoidal waves, and so on, as long as *one complete period* of the desired waveform fits into the full wavetable. Figure 16.1.17 shows one period of a square, triangular, and trapezoidal wave.

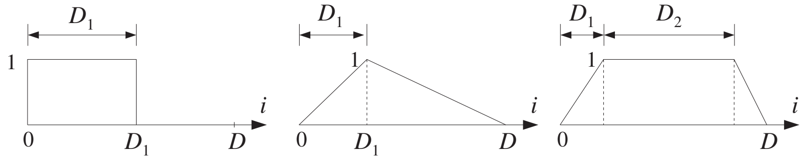


Fig. 16.1.17 Square, triangular, and trapezoidal waveforms.

The following C functions, `sine`, `square`, and `trapez` can be used to fill wavetables with such basic waveforms.

```
/* sine.c - sine wavetable of length D */

#include <math.h>

double sine(D, i)
```

```
int D, i;
{
       double pi = 4 * atan(1.0);

       return sin(2 * pi * i / D);
}


/* square.c - square wavetable of length D, with D1 ones */

double square(D1, i)
int D1, i;
{
       if (i < D1)
              return 1;
       else
              return 0;
}


/* trapez.c - trapezoidal wavetable: D1 rising, D2 steady */

double trapez(D, D1, D2, i)
int D, D1, D2, i;
{
       if (i < D1)
              return i/(double) D1;
       else
           if (i < D1+D2)
                return 1;
           else
                return (D - i)/(double) (D - D1 - D2);
}
```

To illustrate the usage of the wavetable generator routines, consider a wavetable $\mathbf{w}$ of length $D = 1000$ and fill it with one period of a sinusoid. The following program segment illustrates the reverse loading of the table using the function `sine`, and the generation of five sinusoids, shown in Fig. 16.1.18. The *same* wavetable is used by all sinusoids, but each is assigned its own offset index q that cycles around the wavetable according to a given frequency.

```
double *w;
w = (double *) calloc(D, sizeof(double));        use: D = 1000

q1 = q2 = q3 = q4 = q5 = 0;                      initialize qs

for (i=0; i<D; i++) {                            load wavetable with a sinusoid
       w[q1]   = sine(D, i);                     may need the cast w[(int)q1]
       gdelay2(D-1, 1.0, &q1);
       }

gdelay2(D-1, -m*D*F2, &q4);                      reset q4 = mDF2
gdelay2(D-1,  m*D*F2, &q5);                      reset q5 = −mDF2

for (n=0; n<Ntot; n++) {                         use: A = 1, Ntot = 1000
       y1[n] = wavgen(D, w, A, F1, &q1);         use: F1 = 1.0/D
       y2[n] = wavgen(D, w, A, F2, &q2);         use: F2 = 5.0/D
```

```
    y3[n] = wavgen(D, w, A, F3, &q3);          use: F3 = 10.5/D
    y4[n] = wavgen(D, w, A, F4, &q4);          use: F4 = F2
    y5[n] = wavgen(D, w, A, F5, &q5);          use: F5 = F2
    }
```

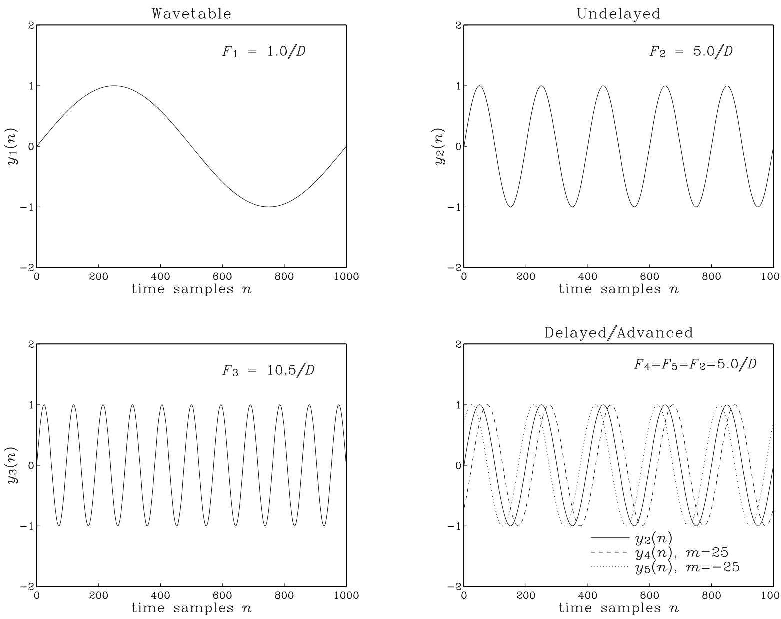


Fig. 16.1.18 Waveforms generated from a common wavetable.

The signal $y_1(n)$ is the sinusoid stored in the wavetable that becomes the source of all the other sinusoids. The first for-loop uses the offset index q_1 to load the wavetable. Upon exit from this loop, q_1 has cycled back to $q_1 = 0$. The frequency of $y_1(n)$ is one cycle in D samples, or,

$$F_1 = \frac{1}{D} = 0.001 \text{ cycles/sample}$$

and the corresponding shift is $c_1 = DF_1 = 1$. The signal $y_2(n)$ is generated from the same wavetable, but with frequency:

$$F_2 = \frac{5}{D} = 0.005 \text{ cycles/sample}$$

which corresponds to $c_2 = DF_2 = 5$ cycles in D samples. The wavetable is cycled over every five of its entries. The signal $y_3(n)$ is also generated from the same wavetable, but has frequency:

$$F_3 = \frac{10.5}{D} = 0.0105 \text{ cycles/sample}$$

which gives $c_3 = DF_3 = 10.5$, a non-integer value. The ten and a half cycles contained in the D samples can be seen in the figure. The wavetable is cycled over every 10.5 of its entries, and the output is obtained by the truncation method.

Finally, the last two signals $y_4(n)$ and $y_5(n)$ are the time-delayed and time-advanced versions of $y_2(n)$ by $m = 25$ samples, that is, $y_4(n)= y_2(n-25)$ and $y_5(n)= y_2(n+25)$. They are right- and left-shifted relative to $y_2(n)$ by one-eighth cycle, as can be seen in Fig. 16.1.18, because each F_2-cycle contains $1/F_2 = 200$ samples and therefore $m = 25$ corresponds to a $(1/8)$ of a cycle.

Because they have frequency F_2 and wavetable shift $c_2 = DF_2 = 5$, their effective starting offsets will be $q_4 = mc_2 = 25{\times}5 = 125$, and $q_5 = -mc_2 = -125$ (or, rather $q_5 = 1000 - 125 = 855$). These initial q-values are obtained by the two calls to `gdelay2` preceding the generation loop, with arguments $\mp mc_2$.

More complex waveforms can be generated by using several wavetables in combination. For example, Fig. 16.1.19 connects two wavetables together to implement *amplitude modulation.*

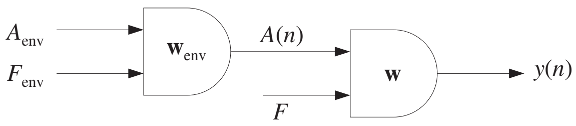


Fig. 16.1.19 Amplitude modulation.

The first generator $\mathbf{w}_{\text{env}}$ produces a time-varying envelope $A(n)$ that becomes the amplitude to the second generator whose wavetable $\mathbf{w}$ stores a copy of the desired signal, such as a note from an instrument. The envelope shape stored in the wavetable $\mathbf{w}_{\text{env}}$ could be triangular or trapezoidal, imitating instrument attack and decay. If we denote the main signal stored in $\mathbf{w}$ by $x(n)$, the configuration of Fig. 16.1.19 generates the modulated signal:

$$y(n)= A(n)x(n)$$

The amplitude input to the envelope generator A_{env} is a constant. Its frequency F_{env} is typically chosen such that the envelope cycles only over *one cycle* during the duration N_{tot} of the signal, that is,

$$\boxed{F_{\text{env}} = \frac{1}{N_{\text{tot}}}} \qquad (16.1.53)$$

As an example, consider the generation of a sinusoidal note of frequency $F = 0.01$ cycles/sample:

$$x(n)= \sin(2\pi Fn), \qquad n = 0,1,\dots,N_{\text{tot}}-1$$

with duration of $N_{\text{tot}} = 1000$ samples. The signal $x(n)$ is to be modulated by a triangular envelope whose attack portion is one-quarter its duration.

At a 44 kHz sampling rate, the frequency F would correspond to the 440 Hz note, A_{440}. Such a triangular envelope would be characteristic of a piano. Using wavetables of duration $D = 1000$, the following program segment illustrates the loading (in reverse order) of the wavetables with the appropriate waveforms, followed by the generation of the triangular envelope $A(n)$ and the modulated sinusoid $y(n)$. The truncation version, `wavgen`, of the generator routines was used:

```
double *w, *wenv, q, qenv;
w    = (double *) calloc(D, sizeof(double));      allocate wavetables
wenv = (double *) calloc(D, sizeof(double));      use: D = 1000

q = qenv = 0;                                     initialize offsets

for (i=0; i<D; i++) {                             load wavetables:
       w[q] = sine(D, i);                         may need the cast w[(int)q]
       wenv[qenv] = trapez(D, D/4, 0, i);         triangular envelope
       gdelay2(D-1, 1.0, &q);                     or, cdelay2(D-1, &q);
       gdelay2(D-1, 1.0, &qenv);
       }
                                                  use: Ntot = 1000 or 2000
Fenv = 1.0 / Ntot;                                envelope frequency

for (n=0; n<Ntot; n++) {
       A[n] = wavgen(D, wenv, Aenv, Fenv, &qenv);  use: Aenv = 1.0
       y[n] = wavgen(D, w, A[n], F, &q);           use: F = 0.01
       }
```

Figure 16.1.20 shows the two cases $N_{\rm tot} = 1000, 2000$. Because $F = 0.01$ cycles/sample, there are $FN_{\rm tot}$ cycles of the sinusoid in the duration of $N_{\rm tot}$, that is, $FN_{\rm tot} = 0.01{\times}1000 = 10$ cycles in the first case, and $FN_{\rm tot} = 0.01{\times}2000 = 20$ cycles in the second. For visual reference, the graphs also plot the triangular envelope $A(n)$ and its negative, $-A(n)$.

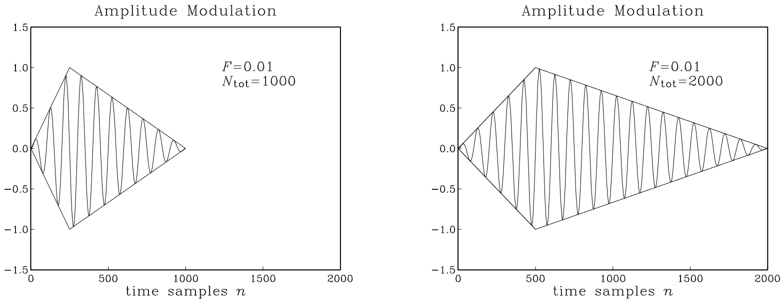


Fig. 16.1.20 Triangularly modulated sinusoid.

The triangular wave was generated from the trapezoidal function by setting $D_1 = D/4$ and $D_2 = 0$. For both values of $N_{\rm tot}$, the triangular envelope cycles only once, because of the choice (16.1.53) of its frequency. Note that the offset shift c corresponding to the frequency F will be $c = DF = 1000{\times}0.01 = 10$, whereas the shift for the envelope wavetable will be $c_{\rm env} = DF_{\rm env} = D/N_{\rm tot} = 1$ or 0.5 in the two cases.

Figure 16.1.21 shows another example, where the envelope signal was chosen to be varying sinusoidally about a constant value:

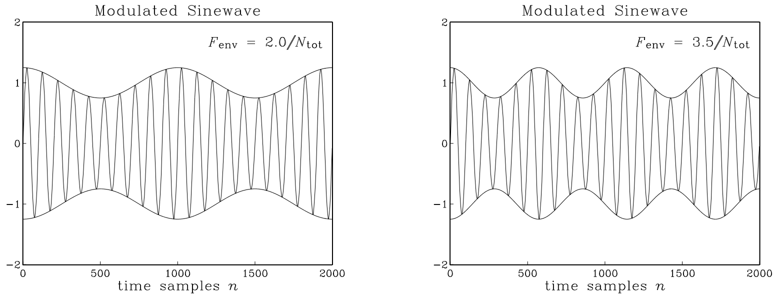


Fig. 16.1.21 Sinusoidally modulated sinusoid.

$$A(n)= 1 + 0.25\cos(2\pi F_{\rm env}n)$$

so that the generated waveform will be:

$$y(n)= A(n)x(n)= \big(1 + 0.25\cos(2\pi F_{\rm env}n)\big)\sin(2\pi Fn)$$

The envelope frequency was chosen to be $F_{\rm env} = 2/N_{\rm tot}$ for the first graph and $F_{\rm env} = 3.5/N_{\rm tot}$ for the second. These choices correspond to 2 and 3.5 envelope cycles in $N_{\rm tot}$ samples. With these values of $F_{\rm env}$, the generation part for this example was carried out by exactly the same for-loop as above. The initial loading of the wavetables was carried out by:

```
q = qenv = 0;                                    initialize offsets

for (i=0; i<D; i++) {                            load wavetables
       w[q] = sine(D, i);                        sinusoidal signal
       wenv[qenv] = 1 + 0.25 * sine(D, i);       sinusoidal envelope
       gdelay2(D-1, 1.0, &q);                    or, cdelay2(D-1, &q);
       gdelay2(D-1, 1.0, &qenv);
       }
```

In addition to amplitude modulation, we may introduce *frequency modulation* into the generated waveform. Figure 16.1.22 shows this case, where the first generator produces a periodic output with amplitude A_m and frequency F_m which is added to a carrier frequency F_c and the result becomes the frequency input to the second generator. For example, using a sinusoidal wavetable $\mathbf{w}_m$ will produce the frequency:

$$F(n)= F_c + A_m\sin(2\pi F_m n) \tag{16.1.54}$$

so that if the signal generator $\mathbf{w}$ is a unit-amplitude sinusoid, then the modulated output will be:

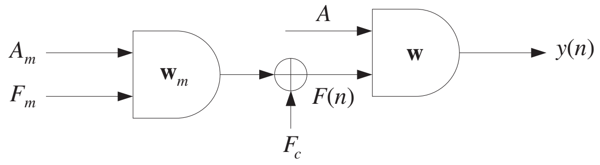


Fig. 16.1.22 Frequency modulation.

$$y(n) = \sin\big(2\pi F(n)n\big) \tag{16.1.55}$$

The following program segment illustrates the generation of four types of frequency modulated waveforms, shown in Fig. 16.1.23. The four cases can be obtained by uncommenting the applicable statements:

```
double *w, *wm;
w  = (double *) calloc(D, sizeof(double));
wm = (double *) calloc(D, sizeof(double));

q  = qm = 0;

for (i=0; i<D; i++) {                                  load wavetables
       w[q]   = sine(D, i);                            signals: y1(n), y2(n), y3(n)
    /* w[q]   = square(D/2, i); */                     signal: y4(n)
       gdelay2(D-1, 1.0, &q);

       wm[qm] = sine(D, i);                            signal: y1(n)
    /* wm[qm] = 2 * square(D/2, i) - 1; */             signal: y2(n)
    /* wm[qm] = trapez(D, D, 0, i);    */              signals: y3(n), y4(n)
       gdelay2(D-1, 1.0, &qm);
       }

for (n=0; n<Ntot; n++) {                               use: Ntot = 1000
       F[n] = Fc + wavgen(D, wm, Am, Fm, &qm);
       y[n] = wavgen(D, w, A, F[n], &q);               use: A = 1
       }
```

The lengths of the two wavetables $\mathbf{w}$ and $\mathbf{w}_m$ were $D = 1000$ and the signal duration $N_{\text{tot}} = 1000$. The signal $y_1(n)$ was a frequency modulated sinusoid of the form of Eq. (16.1.55) with signal parameters:

$$F_c = 0.02, \quad A_m = 0.5F_c, \quad F_m = 0.003$$

It might be thought of as a vibrato effect. The modulation frequency has $F_m N_{\text{tot}} =$ 3 cycles in the N_{tot} samples. The frequency $F(n)$ rises and falls between the limits $F_c - A_m \le F(n) \le F_c + A_m$, or $0.5F_c \le F(n) \le 1.5F_c$. The quantity $F(n)/F_c$ is also plotted in order to help visualize the effect of increasing and decreasing frequency.

The signal $y_2(n)$ is a sinusoid whose frequency is modulated by a square wave that switches between the values $F_c + A_m$ and $F_c - A_m$, where again $A_m = 0.5F_c$. The modulating square wave has frequency of 3 cycles in 1000 samples or $F_m = 0.003$. Note how the modulated signal $y_2(n)$ switches frequency more abruptly than $y_1(n)$.

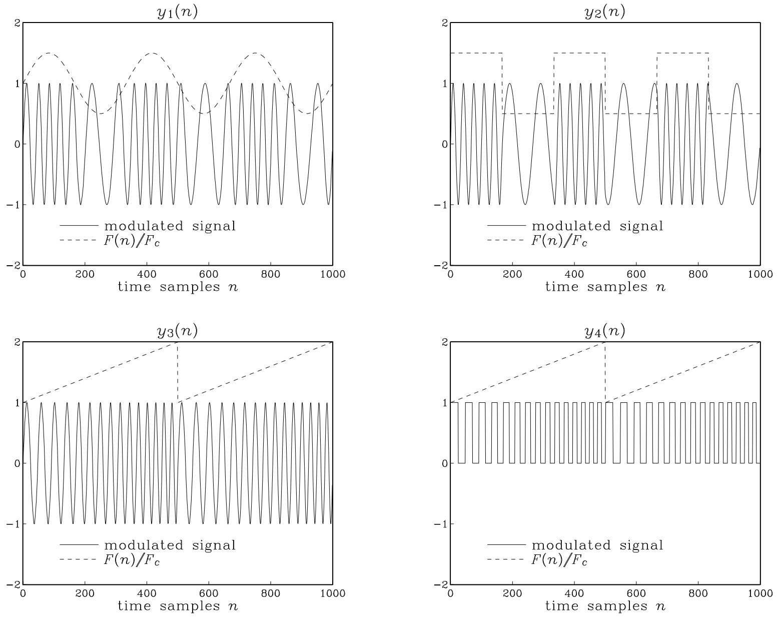


Fig. 16.1.23 Frequency modulated waveforms.

The signal $y_3(n)$ is a sinusoid whose frequency is linearly swept between the values $F_c \le F(n) \le F_c + A_m$, where here $A_m = F_c$ so that $F(n)$ doubles. It might be thought of as a portamento effect. The sawtooth generator was implemented with the function `trapez`, with arguments $D_1 = D$ and $D_2 = 0$. Its frequency was chosen to be 2 cycles in 1000 samples, or $F_m = 0.002$.

Finally, the signal $y_4(n)$ is a square wave, generated by `square` with $D_1 = D/2$, whose frequency is linearly swept between F_c and $2F_c$ with a modulation frequency of 2 cycles in 1000 samples, or $F_m = 0.002$.

Complex waveforms with rich sounds can be generated by combining amplitude and frequency modulation, as well as introducing such modulations on more than one level, for example, amplitude and/or frequency modulation of the amplitude generator in which A_{env} and F_{env} are themselves modulated by a third wavetable generator, and so on.

16.2 Digital Audio Effects

Audio effects, such as delay, echo, reverberation, comb filtering, flanging, chorusing, pitch shifting, stereo imaging, distortion, compression, expansion, noise gating, and equalization, are indispensable in music production and performance [131–161,163–173]. Some are also available for home and car audio systems.

Most of these effects are implemented using digital signal processors, which may reside in separate modules or may be built into keyboard workstations and tone generators. A typical audio effects signal processor is shown in Fig. 16.2.1.

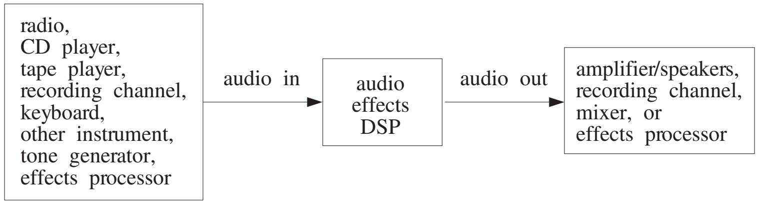


Fig. 16.2.1 Audio effects signal processor.

The processor takes in the “dry” analog input, produced by an instrument such as a keyboard or previously recorded on some medium, and samples it at an appropriate audio rate, such as 44.1 kHz (or less, depending on the effect). The sampled audio signal is then subjected to a DSP effects algorithm and the resulting processed signal is reconstructed into analog form and sent on to the next unit in the audio chain, such as a speaker system, a recording channel, a mixer, or another effects processor.

In all-digital recording systems, the sampling/reconstruction parts can be eliminated and the original audio input can remain in digitized form throughout the successive processing stages that subject it to various DSP effects or mix it with similarly processed inputs from other recording tracks.

In this section, we discuss some basic effects, such as delays, echoes, flanging, chorusing, reverberation, and dynamics processors. The design of equalization filters will be discussed in Chapters 11 and 12.

16.2.1 Delays, Echoes, and Comb Filters

Perhaps the most basic of all effects is that of *time delay* because it is used as the building block of more complicated effects such as reverb.

In a listening space such as a room or concert hall, the sound waves arriving at our ears consist of the *direct* sound from the sound source as well as the waves *reflected* off the walls and objects in the room, arriving with various amounts of time delay and attenuation.

Repeated multiple reflections result in the reverberation characteristics of the listening space that we usually associate with a room, hall, cathedral, and so on.

A *single reflection* or echo of a signal can be implemented by the following filter, which adds to the direct signal an attenuated and delayed copy of itself:

$$\boxed{y(n)=x(n)+ax(n-D)} \qquad \text{(echo filter)} \tag{16.2.1}$$

The delay D represents the round-trip travel time from the source to a reflecting wall and the coefficient a is a measure of the reflection and propagation losses, so that $|a|\le 1$. The transfer function and impulse response of this filter are:

$$H(z)=1+az^{-D}, \qquad h(n)=\delta(n)+a\delta(n-D) \tag{16.2.2}$$

Its block diagram realization is shown in Fig. 16.2.2. The frequency response is obtained from Eq. (16.2.2) by setting $z=e^{j\omega}$:

$$H(\omega)=1+ae^{-j\omega D}, \qquad |H(\omega)|=\sqrt{1+2a\cos(\omega D)+a^2} \tag{16.2.3}$$

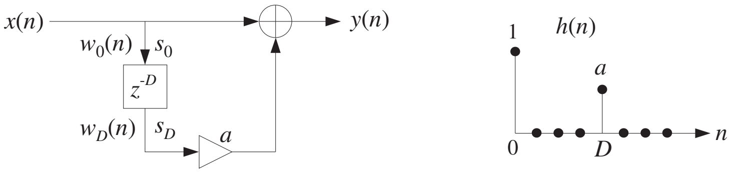


Fig. 16.2.2 Digital echo processor.

Such a filter acts as an FIR *comb filter* whose frequency response exhibits peaks at multiples of the fundamental frequency $f_1=f_s/D$. The zeros of the transfer function $H(z)$ are the solutions of the equation (assuming $0<a\le 1$):

$$1+az^{-D}=0 \quad \Rightarrow \quad z_k=\rho e^{\pi j(2k+1)/D}, \quad k=0,1,\dots,D-1 \tag{16.2.4}$$

where $\rho=a^{1/D}$. The magnitude response and the zero pattern are shown in Fig. 16.2.3, for the case $D=8$. If $a=1$, then $\rho=1$, and the zeros lie on the unit circle corresponding to exact zeros in the frequency response.

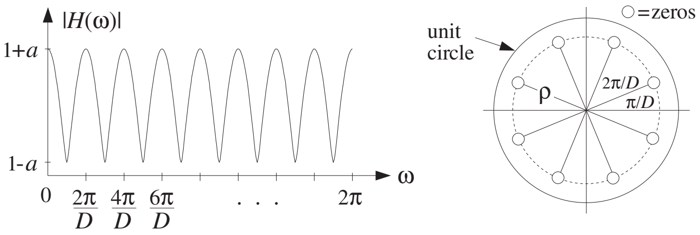


Fig. 16.2.3 FIR comb filter, with peaks at $\omega_k=2\pi k/D$, $k=0,1,\dots,D-1$.

At the dip frequencies $\omega_k=(2k+1)\pi/D$, we have $e^{j\omega_k D}=e^{j\pi}=-1$ giving $H(\omega_k)=1-a$. Between the dip frequencies, that is, at $\omega_k=2\pi k/D$, we have peaks with value $H(\omega_k)=1+a$, because $e^{j\omega_k D}=1$. In units of Hz, these peak frequencies are:

$$f_k=k\frac{f_s}{D}=kf_1\,, \qquad k=0,1,\dots,D-1 \tag{16.2.5}$$

The sample processing algorithm for this filter is given below, implemented with both a linear and circular delay line. As we mentioned in Chapter 4, for audio signals the delay D can be very large and therefore the circular delay line is more efficient. Denoting the $(D+1)$-dimensional delay-line buffer by $\mathbf{w} = [w_0, w_1, \dots, w_D]$, we have:

```
for each input sample x do:
    $y = x + aw_D$
    $w_0 = x$
    $\text{delay}(D, \mathbf{w})$
```

```
for each input sample x do:
    $s_D = \text{tap}(D, \mathbf{w}, p, D)$
    $y = x + as_D$
    $*p = x$
    $\text{cdelay}(D, \mathbf{w}, \&p)$
```

Note that the quantities w_D in the linear case and $s_D = \text{tap}(D, \mathbf{w}, p, D)$ in the circular one represent the Dth output of the tapped delay line, that is, the signal $x(n-D)$. Comb filters, like the above echo processor, arise whenever the direct signal is mixed with its delayed replicas. For example, instead of adding the echo we can subtract it, obtaining (with $a > 0$):

$$\boxed{y(n) = x(n) - ax(n-D)} \tag{16.2.6}$$

The transfer function and frequency response are now

$$H(z) = 1 - az^{-D}, \qquad H(\omega) = 1 - ae^{-j\omega D} \tag{16.2.7}$$

having peaks at $\omega_k = (2k+1)\pi/D$ and dips at $\omega_k = 2\pi k/D$, $k = 0, 1, \dots, D-1$. The magnitude response and zero pattern are shown in Fig. 16.2.4, for $D = 8$. Similarly, if we add three successive echoes, we obtain the filter:

$$\boxed{y(n) = x(n) + ax(n-D) + a^2x(n-2D) + a^3x(n-3D)} \tag{16.2.8}$$

Using the finite geometric series, we can express the transfer function as

$$H(z) = 1 + az^{-D} + a^2z^{-2D} + a^3z^{-3D} = \frac{1 - a^4z^{-4D}}{1 - az^{-D}} \tag{16.2.9}$$

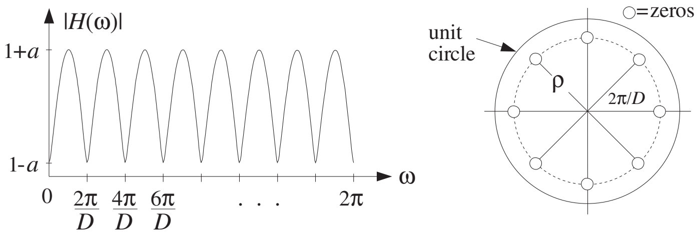


Fig. 16.2.4 Comb filter of Eq. (16.2.6), with dips at $\omega_k = 2\pi k/D$, $k = 0, 1, \dots, D-1$.

It follows that $H(z)$ vanishes at the zeros of the numerator which are not zeros of the denominator, that is,

$$z^{4D} = a^4, \quad \text{but} \quad z^D \neq a$$

or, equivalently, at

$$z_k = \rho e^{2\pi jk/4D}, \; k = 0, 1, \dots, 4D-1, \quad \text{but } k \text{ not a multiple of } 4$$

The filter has peaks at frequencies for which k is a multiple of 4, indeed, if $k = 4m$, $m = 0, 1, \dots, D-1$, then

$$\omega_k = \frac{2\pi k}{4D} = \frac{2\pi(4m)}{4D} = \frac{2\pi m}{D} \qquad \Rightarrow \qquad e^{j\omega_k D} = 1$$

and the filter's response takes on the maximum value $H(\omega_k) = 1 + a + a^2 + a^3$.

The magnitude response and zero pattern are shown in Fig. 16.2.5, for $D = 8$. The dips occur at the 32nd roots of unity, except at the 8th roots of unity at which there are peaks.

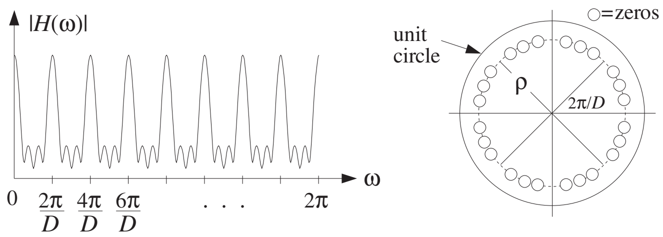


Fig. 16.2.5 Comb filter of Eq. (16.2.8), with peaks at $\omega_k = 2\pi k/D$, $k = 0, 1, \dots, D-1$.

Adding up an infinite number of successive echoes imitates the reverberating nature of a room and gives rise to an *IIR comb filter*:

$$y(n) = x(n) + ax(n-D) + a^2x(n-2D) + \cdots \tag{16.2.10}$$

which has impulse response:

$$h(n) = \delta(n) + a\delta(n-D) + a^2\delta(n-2D) + \cdots \tag{16.2.11}$$

and transfer function:

$$H(z) = 1 + az^{-D} + a^2z^{-2D} + \cdots$$

which can be summed by the geometric series into the form:

$$\boxed{H(z) = \frac{1}{1 - az^{-D}}} \qquad \text{(plain reverberator)} \tag{16.2.12}$$

The I/O equation (16.2.10) can then be recast recursively as

$$\boxed{y(n) = ay(n-D) + x(n)} \tag{16.2.13}$$

A block diagram realization is shown in Fig. 16.2.6. The feedback delay causes a unit impulse input to reverberate at multiples of D, that is, at $n = 0, D, 2D, \dots$. Such simple recursive comb filters form the elementary building blocks of more complicated reverb processors, and will be discussed further in Section 16.2.3.

The transfer function (16.2.12) has poles at $p_k = \rho e^{j\omega_k}$, $k = 0, 1, \dots, D-1$, where $\omega_k = 2\pi k/D$ and $\rho = a^{1/D}$. They are spaced equally around the circle of radius ρ, as shown in Fig. 16.2.7, for $D = 8$. At the pole frequencies ω_k, the frequency response develops peaks, just like the FIR comb of Fig. 16.2.3. Here, the sharpness of the peaks depends on how close to the unit circle the radius ρ is.

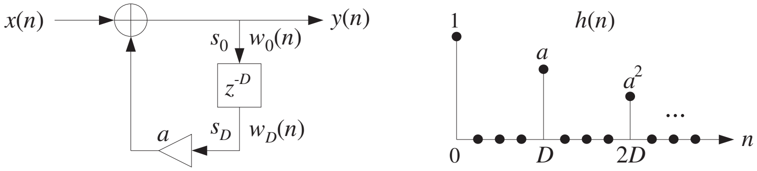


Fig. 16.2.6 Plain reverberator.

The repetition of the echoes every D samples corresponds to the fundamental repetition frequency of $f_1 = f_s/D$ Hz, or $\omega_1 = 2\pi/D$. In music performance, it is sometimes desired to lock the frequency of the decaying echoes to some external frequency, such as a drum beat. If f_1 is known, the proper value of D can be found from $D = f_s/f_1$, or the delay in seconds $T_D = DT = D/f_s = 1/f_1$.

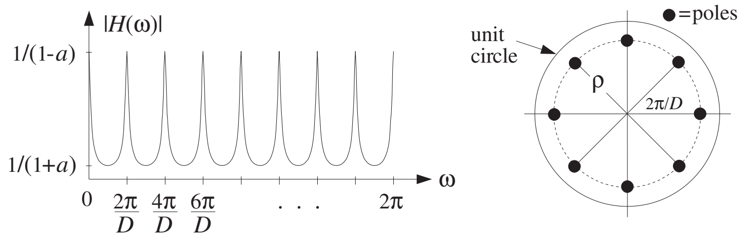


Fig. 16.2.7 IIR comb filter, with peaks at $\omega_k = 2\pi k/D$, $k = 0, 1, \dots, D-1$.

The sample processing algorithm for the realization of Fig. 16.2.6 can be given in terms of a linear or circular delay-line buffer, as follows:

```
for each input sample x do:
    $y = x + a w_D$
    $w_0 = y$
    delay$(D, \mathbf{w})$
```

```
for each input sample x do:
    $s_D = \text{tap}(D, \mathbf{w}, p, D)$
    $y = x + a s_D$
    $*p = y$
    cdelay$(D, \mathbf{w}, \&p)$
```

(16.2.14)

Note that, at each time instant, the output of the delay line is available and can be used to compute the filter's output y. The delay line cannot be updated until after y has been computed and fed back into the input of the delay. The quantities w_D and s_D represent the Dth tap output of the delay, that is, the signal $y(n-D)$.

The effective *time constant* for the filter response to decay below a certain level, say ϵ, can be obtained following the discussion of Section 6.3.2. At time $n = mD$ the impulse response has dropped to $\rho^n = \rho^{mD} = a^m$; therefore, the effective time constant $n_{\text{eff}} = m_{\text{eff}}D$ will be such that

$$\rho^{n_{\text{eff}}} = a^{m_{\text{eff}}} = \epsilon$$

which can be solved for m_{eff} and n_{eff}:

$$n_{\text{eff}} = m_{\text{eff}}D = \frac{\ln\epsilon}{\ln a}D = \frac{\ln\epsilon}{\ln\rho} \tag{16.2.15}$$

and in seconds:

$$\boxed{\tau_{\text{eff}} = n_{\text{eff}}T = \frac{\ln\epsilon}{\ln a}T_D} \tag{16.2.16}$$

where T is the sampling interval, such that $f_s = 1/T$, and $T_D = DT$ is the delay D in seconds. The so-called 60 dB reverberation time constant has $\epsilon = 10^{-3}$, which corresponds to a 60 dB attenuation of the impulse response.

16.2.2 Flanging, Chorusing, and Phasing

The value of the delay D in samples, or in seconds $T_D = DT$, can have a drastic effect on the perceived sound [135,136,141]. For example, if the delay is greater than about 100 milliseconds in the echo processor (16.2.1), the delayed signal can be heard as a quick repetition, a "slap". If the delay is less than about 10 msec, the echo blends with the direct sound and because only certain frequencies are emphasized by the comb filter, the resulting sound may have a hollow quality in it.

Delays can also be used to alter the *stereo image* of the sound source and are indispensable tools in stereo mixing. For example, a delay of a few milliseconds applied to one of the speakers can cause shifting and spreading of the stereo image. Similarly, a mono signal applied to two speakers with such a small time delay will be perceived in stereo.

More interesting audio effects, such as flanging and chorusing, can be created by allowing the delay D to *vary* in time [135,136,141]. For example, Eq. (16.2.1) may be replaced by:

$$\boxed{y(n) = x(n) + ax\big(n - d(n)\big)} \quad \text{(flanging processor)} \tag{16.2.17}$$

A *flanging effect* can be created by periodically varying the delay $d(n)$ between 0 and 10 msec with a low frequency such as 1 Hz. For example, a delay varying sinusoidally between the limits $0 \le d(n) \le D$ will be:

$$d(n) = \frac{D}{2}\big(1 - \cos(2\pi F_d n)\big) \tag{16.2.18}$$

where F_d is a low frequency, in units of [cycles/sample].

Its realization is shown in Fig. 16.2.8. The peaks of the frequency response of the resulting time-varying comb filter, occurring at multiples of f_s/d, and its notches at odd multiples of $f_s/2d$, will sweep up and down the frequency axis resulting in the characteristic whooshing type sound called flanging. The parameter a controls the *depth* of the notches. In units of [radians/sample], the notches occur at odd multiples of π/d.

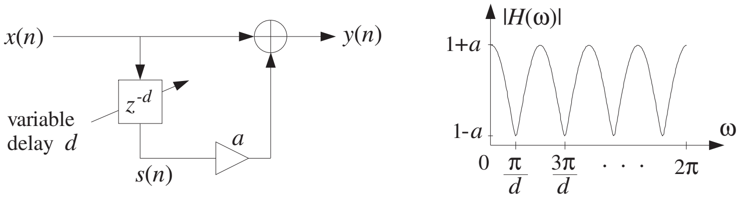


Fig. 16.2.8 Flanging effect, created with a periodically varying delay $d(n)$.

In the early days, the flanging effect was created by playing the music piece simultaneously through two tape players and alternately slowing down each tape by manually pressing the flange of the tape reel.

Because the variable delay d can take *non-integer* values within its range $0 \le d \le D$, the implementation of Eq. (16.2.17) requires the calculation of the output $x(n-d)$ of a delay line at such non-integer values. As we discussed in Section 16.1.3, this can be accomplished easily by truncation, rounding or linear interpolation.

Linear interpolation is the more accurate method, and can be implemented with the help of the following routine `tapi.c`, which is a generalization of the routine `tap` to non-integer values of d.

```
/* tapi.c - interpolated tap output of a delay line */

double tap();

double tapi(D, w, p, d)                      usage: sd = tapi(D, w, p, d);
double *w, *p, d;                            d = desired non-integer delay
int D;                                       p = circular pointer to w
{
       int i, j;
       double si, sj;

       i = (int) d;                          interpolate between s_i and s_j
       j = (i+1) % (D+1);                    if i = D, then j = 0; otherwise, j = i + 1

       si = tap(D, w, p, i);                 note, s_i(n) = x(n − i)
       sj = tap(D, w, p, j);                 note, s_j(n) = x(n − j)

       return si + (d - i) * (sj - si);
}
```

The input d must always be restricted to the range $0 \le d \le D$. Note that if d is one of the integers $d = 0, 1, \dots, D$, the routine's output is the same as the output of `tap`. The mod-$(D+1)$ operation in the definition of j is required to keep j within the array bounds $0 \le j \le D$, and is effective only when $d = D$, in which case the output is the content of the last register of the tapped delay line.

The following routine `tapi2.c` is a generalization of the routine `tap2`, which is implemented in terms of the offset index q instead of the circular pointer p, such that $p = w + q$.

```
/* tapi2.c - interpolated tap output of a delay line */

double tap2();

double tapi2(D, w, q, d)                     usage: sd = tapi2(D, w, q, d);
double *w, d;                                d = desired non-integer delay
int D, q;                                    q = circular offset index
{
       int i, j;
       double si, sj;

       i = (int) d;                          interpolate between s_i and s_j
       j = (i+1) % (D+1);                    if i = D, then j = 0; otherwise, j = i + 1

       si = tap2(D, w, q, i);                note, s_i(n) = x(n − i)
       sj = tap2(D, w, q, j);                note, s_j(n) = x(n − j)

       return si + (d - i) * (sj - si);
}
```

Linear interpolation should be adequate for low-frequency inputs, having maximum frequency much less than the Nyquist frequency. For faster varying inputs, more accurate interpolation methods can be used, designed by the methods of Chapter 14.

As an example illustrating the usage of `tapi`, consider the flanging of a plain sinusoidal signal of frequency $F = 0.05$ cycles/sample with length $N_{\text{tot}} = 200$ samples, so that there are $FN_{\text{tot}} = 10$ cycles in the 200 samples. The flanged signal is computed by

$$y(n) = \frac{1}{2}\big[x(n) + x\big(n - d(n)\big)\big] \tag{16.2.19}$$

with $d(n)$ given by Eq. (16.2.18), $D = 20$, and $F_d = 0.01$ cycles/sample, so that there are $F_d N_{\text{tot}} = 2$ cycles in the 200 samples.

The following program segment implements the calculation of the term $s(n) = x\big(n - d(n)\big)$ and $y(n)$. A delay-line buffer of maximal dimension $D + 1 = 21$ was used:

```
double *w, *p;
w = (double *) calloc(D+1, sizeof(double));
p = w;

for (n=0; n<Ntot; n++) {
    d = 0.5 * D * (1 - cos(2 * pi * Fd * n));      time-varying delay
    x = cos(2 * pi * F * n);                       input x(n)
    s = tapi(D, w, p, d);                          delay-line output x(n − d)
    y = 0.5 * (x + s);                             filter output
    *p = x;                                        delay-line input
    cdelay(D, w, &p);                              update delay line
    }
```

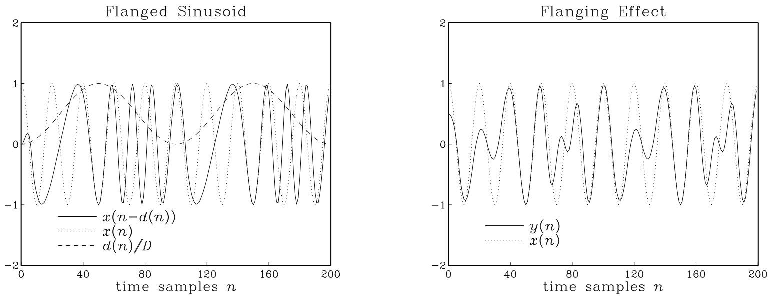


Fig. 16.2.9 Flanged sinusoidal signal.

Figure 16.2.9 shows the signals $x(n)$, $s(n)= x\big(n-d(n)\big)$, $y(n)$, as well as the time-varying delay $d(n)$ normalized by D.

Recursive versions of flangers can also be used that are based on the all-pole comb filter (16.2.13). The feedback delay D in Fig. 16.2.6 is replaced now by a variable delay d. The resulting flanging effect tends to be somewhat more pronounced than in the FIR case, because the sweeping comb peaks are sharper, as seen in Fig. 16.2.7.

Chorusing imitates the effect of a group of musicians playing the same piece simultaneously. The musicians are more or less synchronized with each other, except for small variations in their strength and timing. These variations produce the chorus effect. A digital implementation of chorusing is shown in Fig. 16.2.10, which imitates a chorus of three musicians.

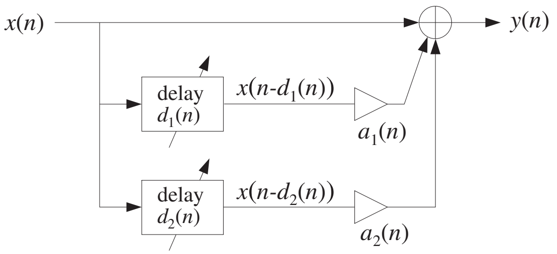


Fig. 16.2.10 Chorus effect, with randomly varying delays and amplitudes.

The small variations in the time delays and amplitudes can be simulated by varying them slowly and randomly [135,136]. A low-frequency random time delay $d(n)$ in the interval $0 \le d(n) \le D$ may be generated by

$$d(n)= D\big(0.5 + v(n)\big) \tag{16.2.20}$$

or, if the delay is to be restricted in the interval $D_1 \le d(n) < D_2$

$$d(n)= D_1 + (D_2 - D_1)\big(0.5 + v(n)\big) \tag{16.2.21}$$

The signal $v(n)$ is a zero-mean low-frequency random signal varying between $[-0.5, 0.5)$. It can be generated by the linearly interpolated generator routine `ranl` of Appendix A.2. Given a desired rate of variation F_{ran} cycles/sample for $v(n)$, we obtain the period $D_{\text{ran}} = 1/F_{\text{ran}}$ of the generator `ranl`.

As an example, consider again the signal $y(n)$ defined by Eq. (16.2.19), but with $d(n)$ varying according to Eq. (16.2.20). The input is the same sinusoid of frequency $F = 0.05$ and length $N_{\text{tot}} = 200$. The frequency of the random signal $v(n)$ was taken to be $F_{\text{ran}} = 0.025$ cycles/sample, corresponding to $N_{\text{tot}}F_{\text{ran}} = 5$ random variations in the 200 samples. The period of the periodic generator `ranl` was $D_{\text{ran}} = 1/F_{\text{ran}} = 40$ samples. The same program segment applies here, but with the change:

```
d = D * (0.5 + ranl(Dran, u, &q, &iseed));
```

where the routine parameters `u`, `q`, `iseed` are described in Appendix A.2.

Figure 16.2.11 shows the signals $x(n)$, $s(n)= x\big(n-d(n)\big)$, $y(n)$, as well as the quantity $d(n)/D$.

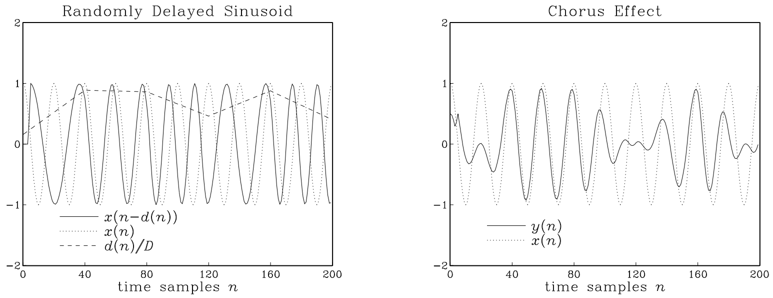


Fig. 16.2.11 Chorusing or doubling of sinusoidal signal.

Phasing or phase shifting is a popular effect among guitarists, keyboardists, and vocalists. It is produced by passing the sound signal through a *narrow notch filter* and combining a proportion of the filter's output with the direct sound.

The frequency of the notch is then varied in a controlled manner, for example, using a low-frequency oscillator, or manually with a foot control. The strong phase shifts that exist around the notch frequency combine with the phases of the direct signal and cause phase cancellations or enhancements that sweep up and down the frequency axis.

A typical overall realization of this effect is shown in Fig. 16.2.12. Multi-notch filters can also be used. The effect is similar to flanging, except that in flanging the sweeping notches are equally spaced along the frequency axis, whereas in phasing the notches can be unequally spaced and independently controlled, in terms of their location and width.

The magnitude and phase responses of a typical single-notch filter are shown in Fig. 16.2.13. Note that the phase response $\arg H(\omega)$ remains essentially zero, except in the vicinity of the notch where it has rapid variations.

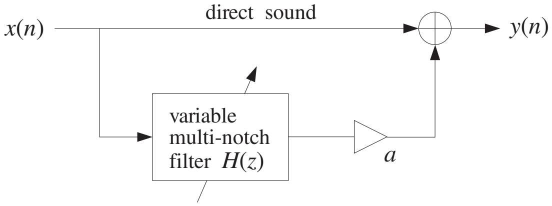


Fig. 16.2.12 Phasing effect with multi-notch filter.

In Section 6.4.3, we discussed simple methods of constructing notch filters. The basic idea was to start with the *notch polynomial* $N(z)$, whose zeros are at the desired notch frequencies, and place poles *behind* these zeros inside the unit circle, at some radial distance ρ. The resulting pole/zero notch filter was then $H(z)= N(z)/N(\rho^{-1}z)$.

Such designs are simple and effective, and can be used to construct the multi-notch filter of a phase shifter. Choosing ρ to be near unity gives very narrow notches. However, we cannot have complete and separate control of the widths of the different notches.

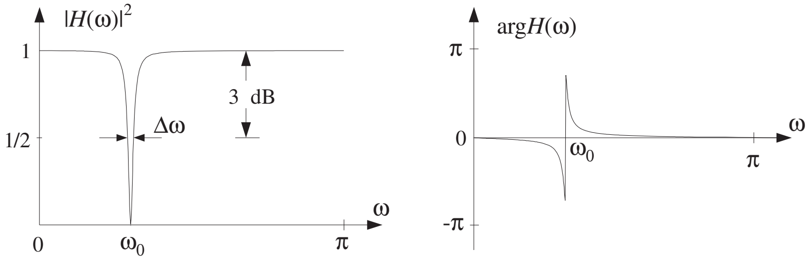


Fig. 16.2.13 Narrow notch filter causes strong phase shifts about the notch frequency.

A design method that gives precise control over the notch frequency and its 3-dB width is the bilinear transformation method, to be discussed in detail in Chapter 12. Using this method, a second-order single-notch filter can be designed as follows:

$$\boxed{H(z)= b\frac{1-2\cos\omega_0\, z^{-1}+z^{-2}}{1-2b\cos\omega_0\, z^{-1}+(2b-1)\,z^{-2}}} \tag{16.2.22}$$

where the filter parameter b is expressible in terms of the 3-dB width $\Delta\omega$ (in units of radians per sample) as follows:

$$\boxed{b=\frac{1}{1+\tan(\Delta\omega/2)}} \tag{16.2.23}$$

The Q-factor of a notch filter is another way of expressing the narrowness of the filter. It is related to the 3-dB width and notch frequency by:

$$Q=\frac{\omega_0}{\Delta\omega} \quad\Rightarrow\quad \Delta\omega=\frac{\omega_0}{Q} \tag{16.2.24}$$

Thus, the higher the Q, the narrower the notch. The transfer function (16.2.22) is normalized to unity gain at DC. The basic shape of $H(z)$ is that of Fig. 16.2.13. Because $|H(\omega)|$ is essentially flat except in the vicinity of the notch, several such filters can be *cascaded* together to create a multi-notch filter, with independently controlled notches and widths.

As an example, consider the design of a notch filter with notch frequency $\omega_0 = 0.35\pi$, for the two cases of $Q = 3.5$ and $Q = 35$. The corresponding 3-dB widths are in the two cases:

$$\Delta\omega=\frac{\omega_0}{Q}=\frac{0.35\pi}{3.5}=0.10\pi \qquad\text{and}\qquad \Delta\omega=\frac{\omega_0}{Q}=\frac{0.35\pi}{35}=0.01\pi$$

The filter coefficients are then computed from Eq. (16.2.23), giving the transfer functions in the two cases:

$$H(z)=0.8633\,\frac{1-0.9080z^{-1}+z^{-2}}{1-0.7838z^{-1}+0.7265z^{-2}}, \qquad (\text{for } Q=3.5)$$

$$H(z)=0.9845\,\frac{1-0.9080z^{-1}+z^{-2}}{1-0.8939z^{-1}+0.9691z^{-2}}, \qquad (\text{for } Q=35)$$

The magnitude squared and phase responses are shown in Fig. 16.2.14.

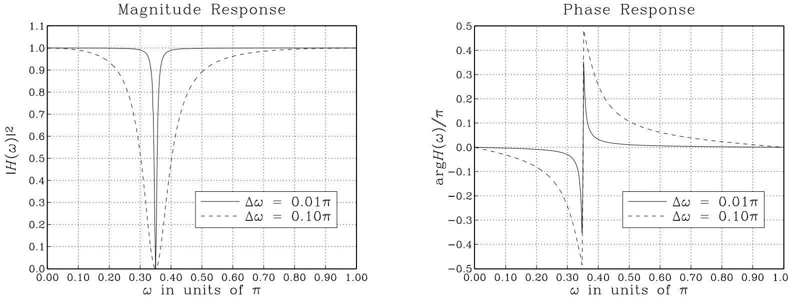


Fig. 16.2.14 Notch filters with $\omega_0 = 0.35\pi$, $Q = 3.5$ and $Q = 35$.

Given a time-varying notch frequency, say $\omega_0(n)$, and a possibly time-varying width $\Delta\omega(n)$, the filter coefficients in Eq. (16.2.22) will also be time-varying. The time-domain implementation of the filter can be derived using a particular realization, such as the canonical realization. For example, if the notch frequency sweeps sinusoidally between the values $\omega_1 \pm \omega_2$ at a rate ω_{sweep}, that is, $\omega_0(n)= \omega_1+\omega_2\sin(\omega_{\text{sweep}}n)$, then the following sample processing algorithm will determine the filter coefficients on the fly and use them to perform the filtering of the current input sample (here, $\Delta\omega$ and b remain fixed):

```
for each time instant n and input sample x do:
    compute current notch $\omega_0 = \omega_1 + \omega_2 \sin(\omega_{\text{sweep}} n)$
    $w_0 = bx + 2b\cos\omega_0\, w_1 - (2b-1)w_2$
    $y = w_0 - 2\cos\omega_0\, w_1 + w_2$
    $w_2 = w_1$
    $w_1 = w_0$
```

An alternative technique for designing multi-notch phasing filters was proposed by Smith [157]. The method uses a cascade of second-order allpass filters, each having a phase response that looks like that of Fig. 16.2.13 and changes by 180° at the notch. If the output of the allpass filter is added to its input, the 180° phase shifts will introduce notches at the desired frequencies.

The three effects of flanging, chorusing, and phasing are based on simple filter structures that are changed into *time-varying filters* by allowing the filter coefficients or delays to change from one sampling instant to the next.

The subject of *adaptive signal processing* [27] is also based on filters with time-varying coefficients. The time dependence of the coefficients is determined by certain design criteria that force the filter to adjust and optimize itself with respect to its inputs. The implementation of an adaptive algorithm is obtained by augmenting the sample processing algorithm of the filter by adding to it the part that adjusts the filter weights from one time instant to the next [45].

Adaptive signal processing has widespread applications, such as channel equalization, echo cancellation, noise cancellation, adaptive antenna systems, adaptive loudspeaker equalization, adaptive system identification and control, neural networks, and many others.

16.2.3 Digital Reverberation

The reverberation of a listening space is typically characterized by three distinct time periods: the direct sound, the early reflections, and the late reflections [131–160], as illustrated in Fig. 16.2.15.

The early reflections correspond to the first few reflections off the walls of the room. As the waves continue to bounce off the walls, their density increases and they disperse, arriving at the listener from all directions. This is the late reflection part.

The reverberation time constant is the time it takes for the room's impulse response to decay by 60 dB. Typical concert halls have time constants of about 1.8–2 seconds.

The sound quality of a concert hall depends on the details of its reverberation impulse response, which depends on the *relative locations* of the sound source and the listener. Therefore, simulating digitally the reverb characteristics of any given hall is an almost impossible task. As a compromise, digital reverb processors attempt to simulate a *typical* reverberation impulse response of a hall, and give the user the option of tweaking some of the parameters, such as the duration of the early reflections (the predelay time), or the overall reverberation time.

Other interesting reverb effects can be accomplished digitally that are difficult or impossible to do by analog means. For example, *gated reverb* is obtained by truncating the IIR response to an FIR one, as shown in Fig. 16.2.16, with a user-selectable gate time. This type of reverb is very effective with snare drums [142]. *Time-reversing* a gated response results in a *reverse reverb* that has no parallel in analog signal processing.

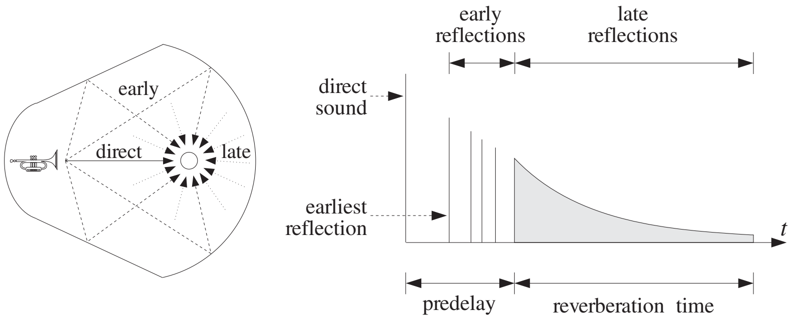


Fig. 16.2.15 Reverberation impulse response of a listening space.

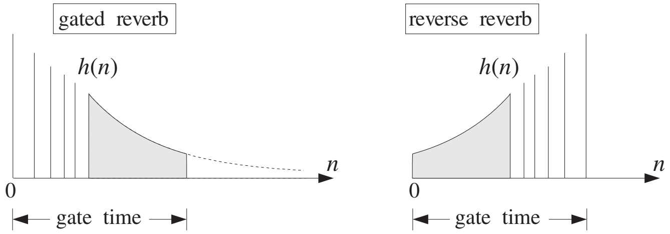


Fig. 16.2.16 Gated and reversed reverberation responses.

The plain reverb filter shown in Fig. 16.2.6 is too simple to produce a realistic reverberation response. However, as suggested by Schroeder [152], it can be used as the building block of more realistic reverb processors that exhibit the discrete early reflections and the diffuse late ones.

In most applications of DSP, we are interested in the steady state response of our filters. Reverberation is an exception. Here, it is the *transient response* of a hall that gives it its particular reverberation characteristics. The steady-state properties, however, do have an effect on the overall perceived sound.

The peaks in the steady-state spectrum of the plain reverb filter of Eq. (16.2.12), shown in Fig. 16.2.7, tend to accentuate those frequencies of the input signal that are near the peak frequencies. To prevent such coloration of the input sound, Schroeder also proposed [152] an *allpass* version of the plain reverberator that has a *flat* magnitude response for all frequencies:

$$\boxed{H(z) = \frac{-a + z^{-D}}{1 - az^{-D}}} \qquad \text{(allpass reverberator)} \qquad (16.2.25)$$

It has I/O difference equation:

$$\boxed{y(n)=ay(n-D)-ax(n)+x(n-D)} \tag{16.2.26}$$

Its frequency and magnitude responses are obtained by setting $z=e^{j\omega}$:

$$H(\omega)=\frac{-a+e^{-j\omega D}}{1-ae^{-j\omega D}} \quad\Rightarrow\quad |H(\omega)|=1, \text{ for all } \omega \tag{16.2.27}$$

The magnitude response is constant in ω because the numerator and denominator of $H(\omega)$ have the *same* magnitude, as can be seen from the simple identity:

$$|-a+e^{-j\omega D}|=\sqrt{1-2a\cos(\omega D)+a^2}=|1-ae^{-j\omega D}|$$

Although its magnitude response is flat, its transient response exhibits the same exponentially decaying pattern of echoes as the plain reverb. Indeed, the impulse response of Eq. (16.2.25) can be obtained by splitting $H(z)$ into the partial fraction expansion form:

$$H(z)=A+\frac{B}{1-az^{-D}} \tag{16.2.28}$$

where $A=-1/a$ and $B=(1-a^2)/a$. Expanding the B-term into its geometric series, gives

$$H(z)=(A+B)+B(az^{-D}+a^2z^{-2D}+a^3z^{-3D}+\cdots)$$

and taking inverse z-transforms leads to the impulse response:

$$h(n)=(A+B)\delta(n)+Ba\delta(n-D)+Ba^2\delta(n-2D)+\cdots \tag{16.2.29}$$

Figure 16.2.17 shows the *canonical* realization of Eq. (16.2.25) realized by a common delay z^{-D}. It also shows the parallel realization of Eq. (16.2.28), which was Schroeder's original realization [152].

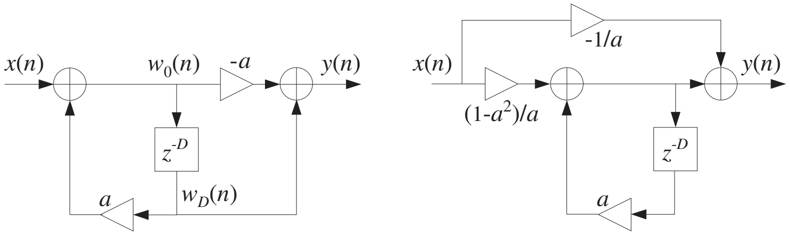


Fig. 16.2.17 Allpass reverberator in canonical and parallel form.

The sample processing algorithm of the canonical form, implemented with linear or circular delay lines, is given below:

```
for each input sample x do:
    $w_0 = x + aw_D$
    $y = -aw_0 + w_D$
    delay$(D, \mathbf{w})$
```

```
for each input sample x do:
    $s_D = \text{tap}(D, \mathbf{w}, p, D)$
    $s_0 = x + as_D$
    $y = -as_0 + s_D$
    $*p = s_0$
    cdelay$(D, \mathbf{w}, \&p)$
```

(16.2.30)

The circular delay versions of sample processing algorithms of the plain reverberator, Eq. (16.2.14), and the allpass reverberator, Eq. (16.2.30), can be implemented by the following C routines `plain.c` and `allpass.c`:

```
/* plain.c - plain reverberator with circular delay line */

double tap();
void cdelay();

double plain(D, w, p, a, x)                    usage: y=plain(D,w,&p,a,x);
double *w, **p, a, x;                          p is passed by address
int D;
{
       double y, sD;

       sD = tap(D, w, *p, D);                  Dth tap delay output
       y = x + a * sD;                         filter output
       **p = y;                                delay input
       cdelay(D, w, p);                        update delay line

       return y;
}
```

```
/* allpass.c - allpass reverberator with circular delay line */

double tap();
void cdelay();

double allpass(D, w, p, a, x)                  usage: y=allpass(D,w,&p,a,x);
double *w, **p, a, x;                          p is passed by address
int D;
{
       double y, s0, sD;

       sD = tap(D, w, *p, D);                  Dth tap delay output
       s0 = x + a * sD;
       y  = -a * s0 + sD;                      filter output
       **p = s0;                               delay input
       cdelay(D, w, p);                        update delay line

       return y;
}
```

The linear buffer `w` is $(D+1)$-dimensional, and the circular pointer p must be initialized to $p=w$, before the first call. The following program segment illustrates their usage:

```
double *w1, *p1;
double *w2, *p2;

w1 = (double *) calloc(D+1, sizeof(double));
w2 = (double *) calloc(D+1, sizeof(double));
p1 = w1; p2 = w2;

for (n=0; n<Ntot; n++) {
        y1[n] = plain(D, w1, &p1, a, x[n]);
        y2[n] = allpass(D, w2, &p2, a, x[n]);
        }
```

The plain and allpass reverberator units can be combined to form more realistic reverb processors. Schroeder's reverberator [152,131,135,146,140,148] consists of several plain units connected in parallel, which are followed by allpass units in cascade, as shown in Fig. 16.2.18. The input signal can also have a direct connection to the output, but this is not shown in the figure.

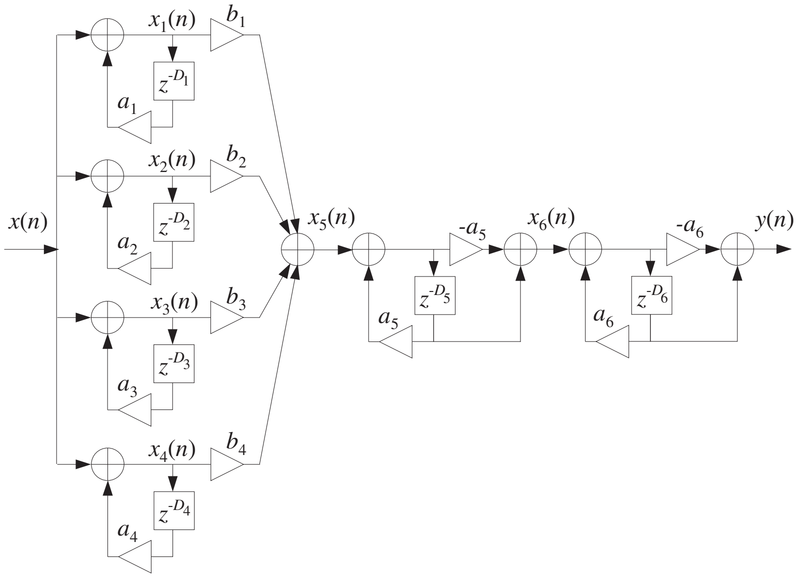


Fig. 16.2.18 Schroeder's reverb processor.

The implementation of the sample processing reverb algorithm can be carried out with the help of the routines `plain` and `allpass`. It is assumed that each unit has its own $(D_i{+}1)$-dimensional circular delay-line buffer $\mathbf{w}_i$ and corresponding circular pointer p_i:

```
for each input sample x do:
    $x_1 = \text{plain}(D_1, \mathbf{w}_1, \&p_1, a_1, x)$
    $x_2 = \text{plain}(D_2, \mathbf{w}_2, \&p_2, a_2, x)$
    $x_3 = \text{plain}(D_3, \mathbf{w}_3, \&p_3, a_3, x)$
    $x_4 = \text{plain}(D_4, \mathbf{w}_4, \&p_4, a_4, x)$
    $x_5 = b_1x_1 + b_2x_2 + b_3x_3 + b_4x_4$
    $x_6 = \text{allpass}(D_5, \mathbf{w}_5, \&p_5, a_5, x_5)$
    $y\ = \text{allpass}(D_6, \mathbf{w}_6, \&p_6, a_6, x_6)$
```

(16.2.31)

The different delays in the six units cause the density of the reverberating echoes to increase, generating an impulse response that exhibits the typical early and late reflection characteristics. Figure 16.2.19 shows the impulse response of the above filter for the following choices of parameters:

$$D_1 = 29,\ D_2 = 37,\ D_3 = 44,\ D_4 = 50,\ D_5 = 27,\ D_6 = 31$$

$$a_1 = a_2 = a_3 = a_4 = a_5 = a_6 = 0.75$$

$$b_1 = 1,\ b_2 = 0.9,\ b_3 = 0.8,\ b_4 = 0.7$$

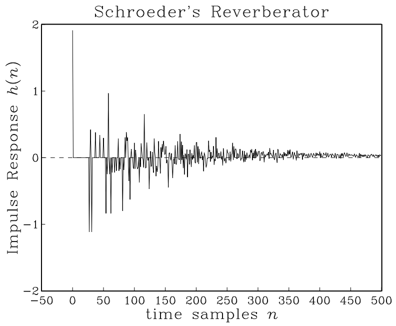


Fig. 16.2.19 Impulse response of Schroeder's reverberator.

Another variation [146,148] of the plain reverb filter of Fig. 16.2.6 is obtained by replacing the simple feedback multiplier a by a nontrivial *lowpass* filter $G(z)$, resulting in the transfer function:

$$\boxed{H(z) = \frac{1}{1 - z^{-D}G(z)}} \qquad \text{(lowpass reverberator)} \tag{16.2.32}$$

Figure 16.2.20 shows a realization. The presence of the lowpass filter in the feedback loop causes each echo to spread out more and more, resulting in a mellower and more diffuse reverberation response. To see this, expand $H(z)$ using the geometric series formula to get:

$$H(z)=1+z^{-D}G(z)+z^{-2D}G^2(z)+z^{-3D}G^3(z)+\cdots$$

giving for the impulse response $h(n)$:

$$h(n)=\delta(n)+g(n-D)+(g*g)(n-2D)+(g*g*g)(n-3D)+\cdots$$

where $g(n)$ is the impulse response of $G(z)$.

It follows that the first echo of the impulse response $h(n)$ at $n=D$ will have the shape of impulse response $g(n)$ the lowpass filter $G(z)$, and will be more spread out than just a single impulse. Similarly, the echo at $n=2D$ will be the impulse response of $G^2(z)$, which is the convolution $g*g$ of $g(n)$ with itself, and therefore it will be even more spread out than $g(n)$, and so on. The graphs of Fig. 16.2.22 illustrate these remarks.

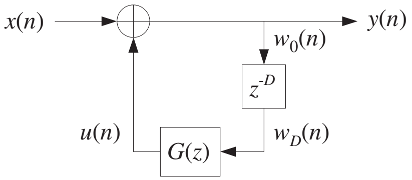


Fig. 16.2.20 Lowpass reverberator.

The feedback filter $G(z)$ can be FIR or IIR. It is described, in general, by the following Mth order transfer function, which also includes the FIR case:

$$G(z)=\frac{N(z)}{D(z)}=\frac{b_0+b_1z^{-1}+b_2z^{-2}+\cdots+b_Mz^{-M}}{1+a_1z^{-1}+a_2z^{-2}+\cdots+a_Mz^{-M}} \tag{16.2.33}$$

The filtering operation by $G(z)$ can be implemented by the canonical realization routine `can`. Assuming a $(D+1)$-dimensional circular buffer $\mathbf{w}$ for the delay D, and an $(M+1)$-dimensional linear delay-line buffer $\mathbf{v}=[v_0,v_1,\dots,v_M]$ for $G(z)$, we can write the sample processing algorithm of Eq. (16.2.32), as follows:

```
for each input sample x do:
    $u = \text{can}(M,\mathbf{a},M,\mathbf{b},\mathbf{v},w_D)$
    $y = x+u$
    $w_0 = y$
    $\text{delay}(D,\mathbf{w})$
```

```
for each input sample x do:
    $s_D = \text{tap}(D,\mathbf{w},p,D)$
    $u = \text{can}(M,\mathbf{a},M,\mathbf{b},\mathbf{v},s_D)$
    $y = x+u$
    $*p = y$
    $\text{cdelay}(D,\mathbf{w},\&p)$
```

(16.2.34)

where the input to the filter $G(z)$ is the Dth tap output w_D or s_D of the delay line. The following routine `lowpass.c` is an implementation using a circular delay line for D, and a linear delay line and the routine `can` for $G(z)$.

```
/* lowpass.c - lowpass reverberator with feedback filter G(z) */

double tap(), can();
void cdelay();

double lowpass(D, w, p, M, a, b, v, x)
double *w, **p, *a, *b, *v, x;                    v = state vector for G(z)
int D;                                             a, b, v are (M + 1)-dimensional
{
       double y, sD;

       sD = tap(D, w, *p, D);                     delay output is G(z) input
       y = x + can(M, a, M, b, v, sD);            reverb output
       **p = y;                                   delay input
       cdelay(D, w, p);                           update delay line

       return y;
}
```

As a simple example, consider the following first-order IIR filter [148] with transfer function:

$$\boxed{G(z)=\frac{b_0+b_1z^{-1}}{1+a_1z^{-1}}=\frac{0.3+0.15z^{-1}}{1-0.5z^{-1}}} \tag{16.2.35}$$

and weight vectors $\mathbf{a}=[1,a_1]=[1,-0.5]$ and $\mathbf{b}=[b_0,b_1]=[0.3,0.15]$.

The corresponding realization of the reverb processor Eq. (16.2.32) is shown in Fig. 16.2.21. The following program segment illustrates the usage `lowpass` for this example:

```
double *w, *p;
double v[2] = {0.0,  0.0};                            G(z) states
double a[2] = {1.0, -0.5};                            G(z) denominator
double b[2] = {0.3,  0.15};                           G(z) numerator

w = (double *) calloc(D+1, sizeof(double));
p = w;

for (n=0; n<Ntot; n++)
       y[n] = lowpass(D, w, &p, M, a, b, v, x[n]);    use M = 1
```

Figure 16.2.22 compares the reverberation responses of the plain reverb (16.2.12), allpass reverb (16.2.25), and lowpass reverb (16.2.32) with the loop filter of Eq. (16.2.35), with the parameter values $D=20$ and $a=0.75$.

The three inputs were an impulse, a length-5 square pulse, and a length-11 triangular pulse, that is,

$$\mathbf{x}=[1]$$

$$\mathbf{x}=[1,1,1,1,1]$$

$$\mathbf{x}=[0,1,2,3,4,5,4,3,2,1,0]$$

The duration of the inputs was chosen to be less than D so that the generated echoes do not overlap, except for the lowpass case in which the echoes become progressively

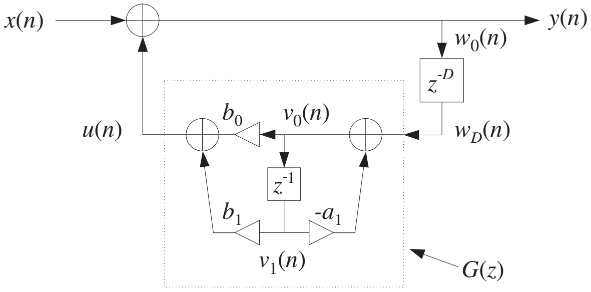


Fig. 16.2.21 Lowpass reverberator, with first-order feedback filter.

smoother (being successively lowpass filtered) and longer, and eventually will overlap as they decay.

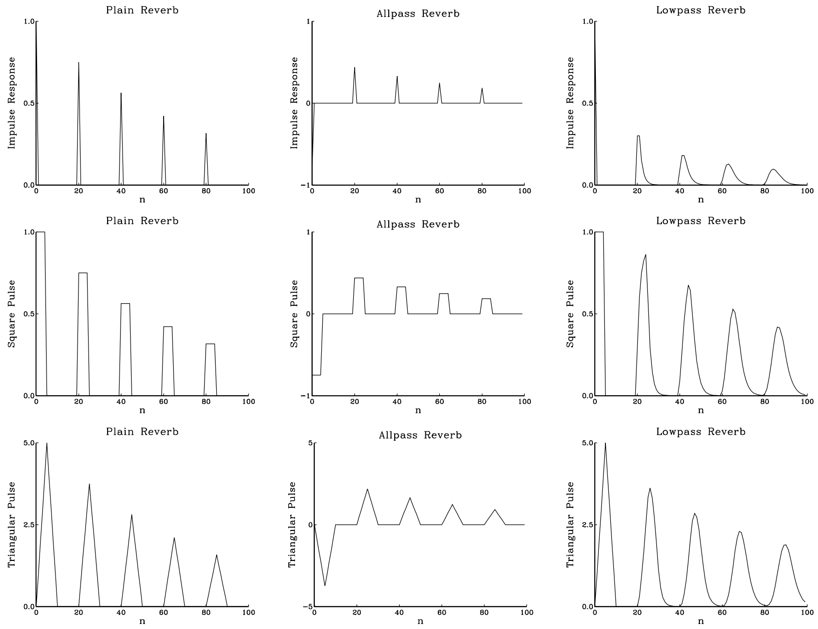


Fig. 16.2.22 Comparison of plain, allpass, and lowpass reverberators.

The plain and allpass reverberators have poles that are equidistant from the origin of the unit circle at radius $\rho = a^{1/D}$, and are equally spaced around the circle at the D root-of-unity angles $\omega_k = 2\pi k/D$, $k = 0, 1, \dots, D-1$. Therefore, all the poles have the *same* transient response time constants, as given by Eq. (16.2.15).

The reflectivity and absorptivity properties of the walls and air in a real room depend on frequency, with the higher frequencies decaying *faster* than the lower ones.

The lowpass reverberator Eq. (16.2.32) exhibits such frequency-dependent behavior. To understand it, consider the first-order example of Eq. (16.2.35). Its magnitude response $|G(\omega)|$ is shown in Fig. 16.2.23.

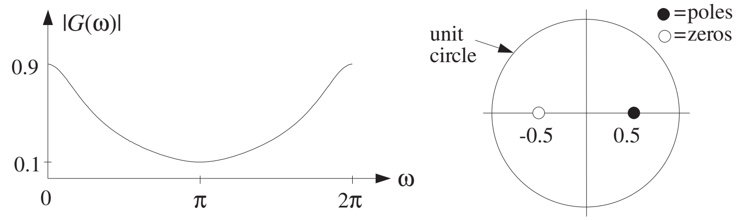


Fig. 16.2.23 Magnitude response of lowpass feedback filter $G(z)$.

The magnitude response and pole locations of the lowpass reverberator (16.2.32) are shown in Fig. 16.2.24. It can be seen that the poles are still approximately equally spaced around the circle, but the high-frequency poles have *shorter* radii and hence *shorter* time constants than the low-frequency ones.

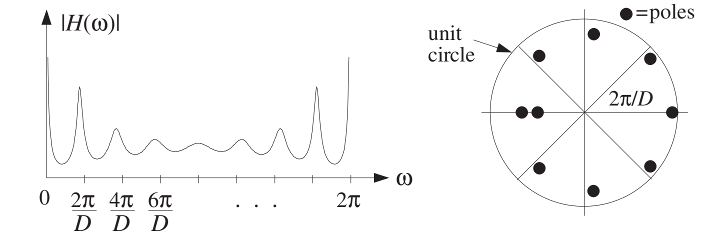


Fig. 16.2.24 Magnitude response of lowpass reverberator, for $D = 8$.

The pole locations of Eq. (16.2.32) are obtained as the roots of the denominator, that is, they are the solutions of

$$z^D = G(z) = \frac{N(z)}{D(z)} \tag{16.2.36}$$

For our example, $D(z)$ has order one and therefore Eq. (16.2.36) will have $D+1$ poles, say, p_i, $i = 1, 2, \dots, D+1$. Writing p_i in its polar form $p_i = \rho_i e^{j\omega_i}$, we have

$$\rho_i^D e^{j\omega_i D} = G(p_i) = |G(p_i)|e^{j\arg G(p_i)}$$

Defining the *phase delay* of the ith pole by

$$d_i = -\frac{\arg G(p_i)}{\omega_i}$$

we have

$$\rho_i^D e^{j\omega_i D} = |G(p_i)|e^{-j\omega_i d_i}$$

which can be separated into the two equations:

$$\rho_i^D = |G(p_i)|, \qquad e^{j\omega_i(D+d_i)} = 1$$

and give

$$\rho_i = |G(p_i)|^{1/D}, \qquad \omega_i = \frac{2\pi k_i}{D+d_i} \tag{16.2.37}$$

for some integer k_i.

Although these are coupled equations in the unknowns ρ_i, ω_i, we can see how the angles ω_i will be distributed around the unit circle, near the Dth roots of unity. Similarly, assuming ρ_i is near 1 and replacing $G(p_i)\simeq G(e^{j\omega_i}) = G(\omega_i)$, we have the approximation:

$$\boxed{\rho_i \simeq |G(\omega_i)|^{1/D}} \tag{16.2.38}$$

which by the *lowpass* nature of $G(\omega)$ implies that ρ_i will be *smaller* for higher frequencies ω_i and *larger* for lower ones, in qualitative agreement with the exact pole locations shown in Fig. 16.2.24. Using Eq. (16.2.15), we find for the exact and approximate ϵ-level time constants, in units of the delay time $T_D = TD$:

$$\boxed{\tau_i = \frac{\ln\epsilon}{D\ln\rho_i}T_D \simeq \frac{\ln\epsilon}{\ln|G(\omega_i)|}T_D} \tag{16.2.39}$$

It follows from Eq. (16.2.38) that the stability of the reverb filter $H(z)$, that is, $\rho_i < 1$, will be guaranteed if the feedback filter is normalized such that $|G(\omega)| < 1$, for all ω. Regardless of the above approximation, this condition implies stability by Nyquist's stability criterion or Rouché's theorem [39]. For our example of Eq. (16.2.35), we have $|G(\omega)| \leq 0.9$.

Besides the D poles that are approximately equally distributed around the unit circle, there is an extra one that essentially corresponds to the *zero* of the filter $G(z)$. Indeed, for that pole, say p, we have

$$p^D = G(p) = \frac{N(p)}{D(p)}$$

Because p is well inside the unit circle, if D is large, then $p^D \simeq 0$ and therefore, it corresponds to $N(p)\simeq 0$. For our example filter, this extra pole is near the $z = -0.5$ zero of the numerator filter $N(z) = 0.3 + 0.15z^{-1}$.

Table 16.2.1 shows for $D = 8$ the exact poles $p_i = \rho_i e^{j\omega_i}$ of Fig. 16.2.24, their frequencies ω_i and magnitudes ρ_i, as well as the approximate magnitudes given by Eq. (16.2.38), and the exact 60-dB ($\epsilon = 10^{-3}$) time constants τ_i.

The first D pole angles are approximately equal to the Dth root of unity angles. The approximation of Eq. (16.2.38) works well for all but the last pole, which is the one near the zero of $N(z)$.

$p_i = \rho_i e^{j\omega_i}$	ω_i/π	ρ_i	$\lvert G(\omega_i)\rvert^{1/D}$	τ_i/T_D
0.9888	0	0.9888	0.9869	76.594
$0.7282 \pm j0.6026$	± 0.2201	0.9452	0.9412	15.314
$0.1128 \pm j0.8651$	± 0.4587	0.8724	0.8715	6.326
$-0.4866 \pm j0.6303$	± 0.7093	0.7962	0.8047	3.789
-0.6801	1	0.6801	0.7499	2.240
-0.5174	1	0.5174	0.7499	1.310

Table 16.2.1 Reverberator poles and time constants, for $D = 8$.

An alternative way to understand the frequency dependence of the time constants is to look at the input-on and input-off transients and steady-state behavior of the filter $H(z)$ of Eq. (16.2.32). Fig. 16.2.25 compares the plain and lowpass reverberator transient outputs for a sinusoid that is turned on at $n = 0$ and off at $n = 150$. The filter parameters were $D = 30$, $a = 0.75$, and $G(z)$ was given by Eq. (16.2.35). The frequencies of the two sinusoids were $\omega = 0.2\pi$ and $\omega = \pi$ radians/cycle.

At the moment the input is cut off, there are D samples of the sinusoid stored in the delay line. As these samples recirculate around the feedback loop every D samples, they get attenuated effectively by the gain of the loop filter $|G(\omega)|$. For the lowpass reverberator, the loop gain is about 0.9 at low frequencies and 0.1 at high frequencies. Thus, the low-frequency sinusoid dies out slowly, whereas the high-frequency one dies out (and starts up) rapidly, leaving behind the slower but weaker low-frequency mode. For the plain reverberator, both the high- and low- frequency sinusoids die out with the *same* time constants.

Besides its use in reverberation effects, the lowpass reverberator filter (16.2.32) has also been used in computer music to model and synthesize guitar string and drum sounds [124–127]. The Karplus-Strong algorithm [124] for modeling plucked strings uses the following FIR lowpass feedback filter:

$$G(z) = \frac{1}{2}(1 + z^{-1}) \tag{16.2.40}$$

A guitar-string sound is generated by simulating the plucking of the string by initially filling the delay-line buffer $\mathbf{w} = [w_0, w_1, \dots, w_D]$ with zero-mean random numbers, and then letting the filter run with zero input. The value D of the delay is chosen to correspond to any desired fundamental frequency f_1, that is, $D = f_s/f_1$.

The recirculating block of random numbers gets lowpass filtered during each pass through the loop filter $G(z)$ and loses its high-frequency content. As a result, the high frequencies in the generated sound decay faster than the low frequencies, as is the case for natural plucked-string sounds.

Physical modeling of instruments is an active research area in computer music. Discrete-time models of the equations describing the physics of an instrument, such as linear or nonlinear wave equations, can be used to generate sounds of the instrument [124–130].

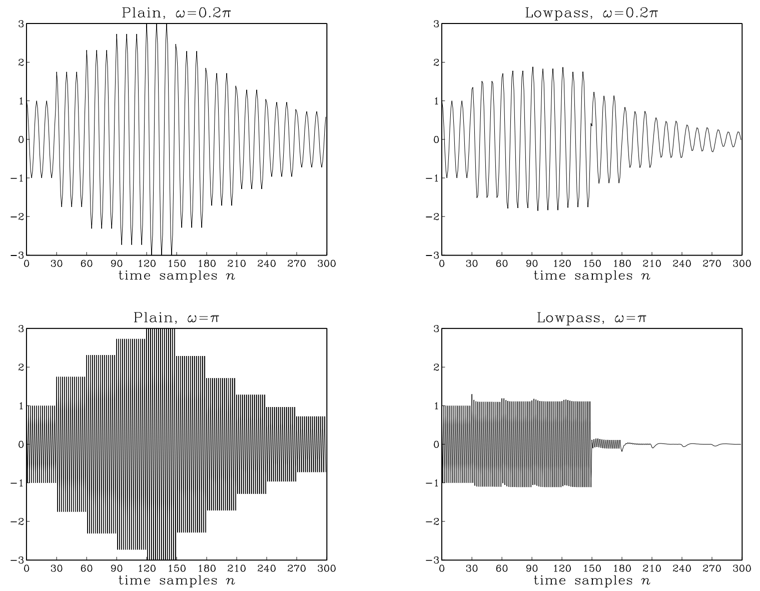


Fig. 16.2.25 Time constants of high- and low-frequency transients, for $D = 30$.

16.2.4 Multitap Delays

Most DSP audio effects processors have built-in a wide class of specialized multiple-delay type effects. They can be obtained from simple low-order FIR or IIR filters by replacing each single unit-delay z^{-1} by the progressively more general substitutions:

$$z^{-1} \longrightarrow z^{-D} \longrightarrow \frac{z^{-D}}{1-az^{-D}} \longrightarrow \frac{z^{-D}}{1-z^{-D}G(z)} \tag{16.2.41}$$

which represent a multiple delay, a ringing delay, and a lowpass ringing delay. As a first example, consider the plain ringing delay with transfer function:

$$\boxed{H(z)=\frac{z^{-D}}{1-az^{-D}}} \tag{16.2.42}$$

Expanding in powers of z^{-D}, we have

$$H(z)=z^{-D}+az^{-2D}+a^2z^{-3D}+\cdots$$

The corresponding impulse response will consist of the first delayed impulse $\delta(n-D)$, followed by its successive echoes of exponentially diminishing strength:

$$h(n)=\delta(n-D)+a\delta(n-2D)+a^2\delta(n-3D)+\cdots$$

This impulse response and a block diagram realization of Eq. (16.2.42) are shown in Fig. 16.2.26. This is basically the same as the plain reverberator of Fig. 16.2.6, but with the output taken *after* the delay, not before it. Its sample processing algorithm is a variation of Eq. (16.2.14):

```
for each input sample x do:
    $y = w_D$
    $w_0 = x + aw_D$
    delay$(D, \mathbf{w})$
```

```
for each input sample x do:
    $s_D = \text{tap}(D, \mathbf{w}, p, D)$
    $y = s_D$
    $*p = x + as_D$
    cdelay$(D, \mathbf{w}, \&p)$
```

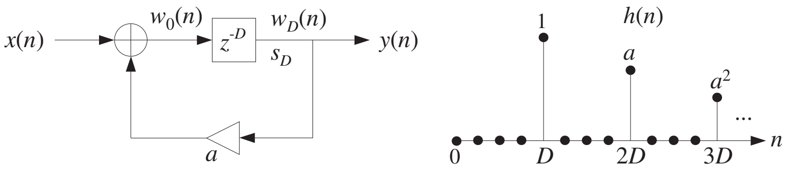


Fig. 16.2.26 Reverberating multi-delay.

As a second example, consider the following second-order FIR filter:

$$H(z)=b_0+b_1z^{-1}+b_2z^{-2}$$

The replacements (16.2.41) lead to the following three multi-delay filters, which are progressively more complicated:

$$H(z) = b_0 + b_1z^{-D_1} + b_2z^{-D_1}z^{-D_2}$$

$$H(z) = b_0 + b_1\left[\frac{z^{-D_1}}{1-a_1z^{-D_1}}\right] + b_2\left[\frac{z^{-D_1}}{1-a_1z^{-D_1}}\right]\left[\frac{z^{-D_2}}{1-a_2z^{-D_2}}\right]$$

$$H(z) = b_0 + b_1\left[\frac{z^{-D_1}}{1-z^{-D_1}G_1(z)}\right] + b_2\left[\frac{z^{-D_1}}{1-z^{-D_1}G_1(z)}\right]\left[\frac{z^{-D_2}}{1-z^{-D_2}G_2(z)}\right]$$

In the last two cases, the reverberating echoes from D_1 are passed into D_2 causing it to reverberate even more densely. Figure 16.2.27 shows the realization of the third case. Its sample processing algorithm can be stated as follows:

```
for each input sample x do:
    $y = b_0x + b_1w_{1D} + b_2w_{2D}$
    $u_2 = \text{can}(G_2, w_{2D})$
    $w_{20} = w_{1D} + u_2$
    delay$(D_2, \mathbf{w}_2)$
    $u_1 = \text{can}(G_1, w_{1D})$
    $w_{10} = x + u_1$
    delay$(D_1, \mathbf{w}_1)$
```

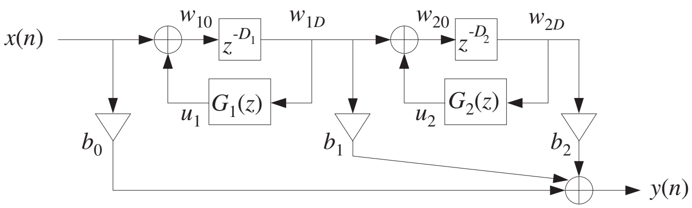


Fig. 16.2.27 Multi-delay effects processor.

where the statement $u_2 = \text{can}(G_2, w_{2D})$ denotes the generic filtering operation of the filter $G_2(z)$ whose input is w_{2D}, and similarly for $G_1(z)$.

Figure 16.2.28 shows the impulse response of such a multi-delay filter, computed by the above sample processing algorithm, with forward taps and delay values:

$$b_0 = 1, \quad b_1 = 0.8, \quad b_2 = 0.6$$

$$D_1 = 30, \quad D_2 = 40$$

and the two cases for the feedback filters:

$$G_1(z) = G_2(z) = 0.75 \qquad \text{(plain)}$$

$$G_1(z) = G_2(z) = \frac{0.3 + 0.15z^{-1}}{1 - 0.5z^{-1}} \qquad \text{(lowpass)}$$

The impulse response exhibits a few early reflections, followed by more dense ones, especially in the lowpass case where successive echoes get spread and overlap more and more with each other. Such multi-delay filters can also be used as preprocessors to reverb units for better modeling of the *early reflection* part of a reverberation response [135,146,140,153].

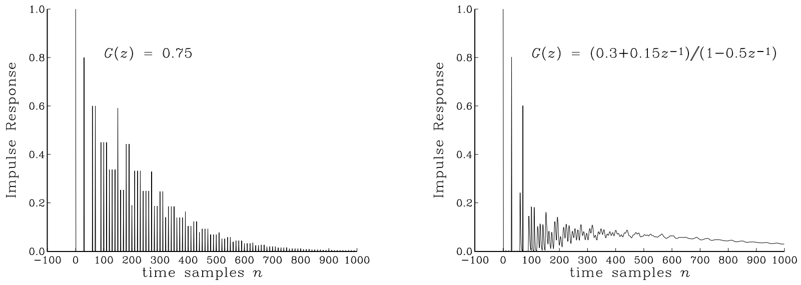


Fig. 16.2.28 Impulse response of plain and lowpass multi-delay.

As a third example, we can start with the simple second-order IIR filter:

$$H(z) = b_0 + \frac{b_1 z^{-1} + b_2 z^{-2}}{1 - a_1 z^{-1} - a_2 z^{-2}}$$

and replace each single delay z^{-1} by a multiple delay z^{-D}, getting the transfer function:

$$H(z) = b_0 + \frac{b_1 z^{-D_1} + b_2 z^{-D_1-D_2}}{1 - a_1 z^{-D_1} - a_2 z^{-D_1-D_2}} \tag{16.2.43}$$

Its realization is shown in Fig. 16.2.29. It may be thought of as a multitap delay line, tapped at delays D_1 and $D_1 + D_2$. The tap outputs are sent to the overall output and also fed back to the input of the delay line. The b_0 term represents the direct sound.

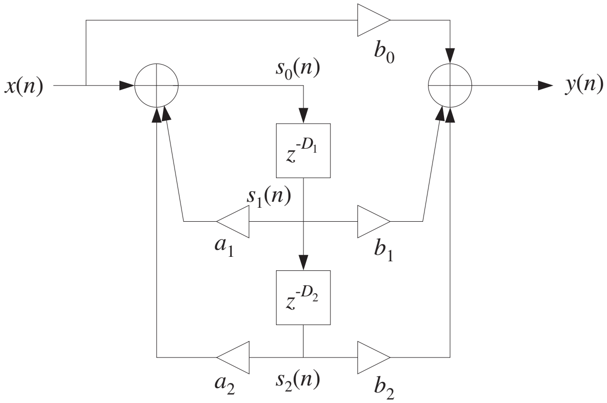


Fig. 16.2.29 Multitap delay effects processor.

Its sample processing algorithm can be implemented with a circular (D_1+D_2)-dimensional delay-line buffer $\mathbf{w}$ and pointer p, as follows:

```
for each input sample x do:
    $s_1 = \text{tap}(D_1 + D_2, \mathbf{w}, p, D_1)$
    $s_2 = \text{tap}(D_1 + D_2, \mathbf{w}, p, D_1 + D_2)$
    $y = b_0 x + b_1 s_1 + b_2 s_2$
    $s_0 = x + a_1 s_1 + a_2 s_2$
    $*p = s_0$
    $\text{cdelay}(D_1 + D_2, \mathbf{w}, \&p)$
```

One potential problem with this arrangement is that the feedback gains can render the filter unstable, if they are taken to be too large. For example, Fig. 16.2.30 shows the impulse response of the filter for the parameter choices

$$b_0 = 1, \quad b_1 = 0.8, \quad b_2 = 0.6$$

$$D_1 = 30, \quad D_2 = 40$$

and for the following two choices of feedback gains, one of which is stable and the other unstable:

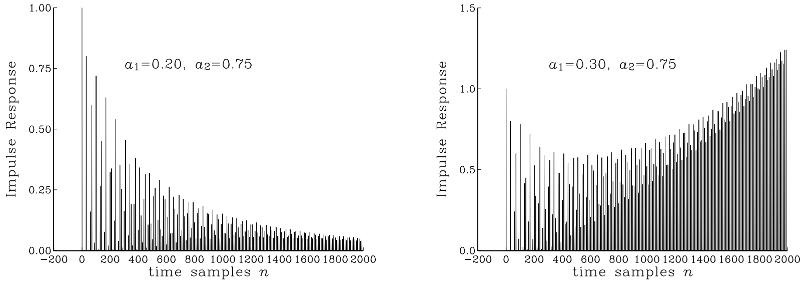


Fig. 16.2.30 Impulse response of multi-tap delay line.

$$
\begin{aligned}
a_1 &= 0.20, \quad a_2 = 0.75 \qquad \text{(stable)}\\
a_1 &= 0.30, \quad a_2 = 0.75 \qquad \text{(unstable)}
\end{aligned}
$$

The condition $|a_1|+|a_2| < 1$ guarantees stability by the Nyquist stability criterion or Rouché's theorem [39], because it ensures that $|G(\omega)| < 1$, where $G(z)= a_1 + a_2 z^{-D_2}$.

Typical DSP effects units include both types of delay effects shown in Figures 16.2.27 and 16.2.29, with five or more multiple delay segments and user-selectable feedback and feed-forward multipliers, and delay times D_i adjustable from 0–2000 msec; for example, see [156].

16.3 Dynamic Range Control

16.3.1 Compressors, Limiters, Expanders, and Noise Gates

Compressors, limiters, expanders, and gates have a wide variety of uses in audio signal processing [163–173]. Compressors attenuate strong signals; expanders attenuate weak signals. Because they affect the dynamic range of signals, they are referred to as *dynamics processors*.

Compressors are used mainly to *decrease* the dynamic range of audio signals so that they fit into the dynamic range of the playback or broadcast system; for example, for putting a recording on audio tape. But there are several other applications, such as announcers "ducking" background music, "de-essing" for eliminating excessive microphone sibilance, and other special effects [166].

Expanders are used for *increasing* the dynamic range of signals, for noise reduction, and for various special effects, such as reducing the sustain time of instruments [166].

A typical steady-state input/output relationship for a compressor or expander is as follows, in absolute and decibel units:

$$
y = y_0\left(\frac{x}{x_0}\right)^{\rho} \quad\Rightarrow\quad 20\log_{10}\left(\frac{y}{y_0}\right) = \rho\, 20\log_{10}\left(\frac{x}{x_0}\right) \tag{16.3.1}
$$

where x is here a constant input, x_0 a desired threshold, and ρ defines the compression or expansion ratio.[†] A compressor is effective only for $x \geq x_0$ and has $\rho < 1$, whereas an expander is effective for $x \leq x_0$ and has $\rho > 1$. Fig. 16.3.1 shows these relationships in dB, so that a 1 dB change in the input causes ρ dB change in the output, that is, ρ is the slope of the input/output straight lines. The hard-knee change in slope can be replaced by a soft-knee by quadratically interpolating between the two slope lines [170], but we will not implement this here.

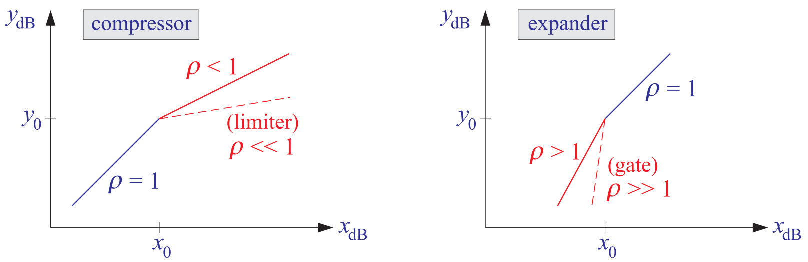


Fig. 16.3.1 Static input/output relationship of compressor or expander.

Typical practical values are $\rho = 1/4$–$1/2$ for compression, and $\rho = 2$–4 for expansion. *Limiters* are extreme forms of compressors that prevent signals from exceeding certain maximum thresholds; they have very small slope $\rho \ll 1$, for example, $\rho = 1/10$.

Noise gates are extreme cases of expanders that infinitely attenuate weak signals, and therefore, can be used to remove weak background noise; they have very large slopes $\rho \gg 1$, for example, $\rho = 10$.

16.3.2 Level Detectors and Gain Processors

The I/O equation (16.3.1) is appropriate only for constant signals. Writing, $y = Gx$, we see that the effective gain of the compressor is a nonlinear function of the input of the form, $G = G_0 x^{\rho-1}$. For time-varying signals, the gain must be computed from a *local average* of the signal which is representative of the signal's level.

A model of a compressor/expander is shown in Fig. 16.3.2. The level detector generates a *control signal* c_n that controls the gain g_n of the multiplier through a nonlinear gain processor.

Depending on the type of compressor, the control signal may be (1) the instantaneous *peak* value $|x_n|$, (2) the *envelope* of x_n, or (3) the *root-mean-square* value of x_n. A simple model of the envelope detector is as follows, with, $0 < \lambda < 1$,

$$
c_n = \lambda\, c_{n-1} + (1-\lambda)\,|x_n| \qquad \text{(level detector)} \tag{16.3.2}
$$

The difference equation for c_n acts as a *rectifier* followed by a simple first-order low-pass exponentially-weighted moving average (EMA) filter. The transfer function, impulse

[†] The inverse quantity, $R = 1/\rho$, is commonly called the "compression ratio," whereas ρ is the "slope".

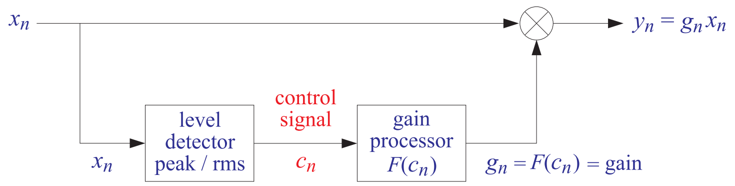


Fig. 16.3.2 Compressor/expander dynamics processor.

and unit-step responses of this filter are,

$$H(z) = \frac{1-\lambda}{1-\lambda z^{-1}}, \quad h_n = (1-\lambda)\lambda^n u(n), \quad (h*u)_n = (1-\lambda^{n+1})u(n) \tag{16.3.3}$$

Thus the filter output responds exponentially quickly to a level change, with an effective time constant, $t_{\text{eff}} = n_{\text{eff}} T_s$, where n_{eff} is the number of samples to converge to within ϵ of the final level, that is, $1 - \lambda^{n_{\text{eff}}} = 1 - \epsilon$, or,

$$\lambda^{n_{\text{eff}}} = \epsilon \quad \Rightarrow \quad n_{\text{eff}} = \frac{\ln\epsilon}{\ln\lambda} \quad \Rightarrow \quad \lambda = \epsilon^{1/n_{\text{eff}}} = \epsilon^{T_s/t_{\text{eff}}} \tag{16.3.4}$$

and ϵ is a user-definable parameter, such as, $\epsilon = 0.1, 0.01, 0.001, 0.05$, corresponding respectively to the so-called 20-dB, 40-dB, 60-dB, or 95% time constants. For these particular values of ϵ, Eq. (16.3.4) for calculating the filter parameter λ can be written in the approximate equivalent exponential forms,

$$\begin{aligned} \epsilon &= [0.1,\ 0.01,\ 0.001,\ 0.05] \approx [e^{-2.3},\ \ e^{-4.6},\ \ e^{-6.9},\ \ e^{-3}] \\ \lambda &= \left[e^{-2.3T_s/t_{\text{eff}}},\ \ e^{-4.6T_s/t_{\text{eff}}},\ \ e^{-6.9T_s/t_{\text{eff}}},\ \ e^{-3T_s/t_{\text{eff}}}\right] \end{aligned} \tag{16.3.5}$$

The time constant, t_{eff}, controls the time to rise or fall to a new input level. The time to rise to a level above the threshold (where the compressor is active) is called the *attack* time constant. The time to drop to a level below the threshold (where the compressor is inactive) is called the *release* time. In audio applications, t_{eff} is typically specified in milliseconds.

For $\lambda = 0$, Eq. (16.3.2) becomes an instantaneous peak detector. This case is useful when the compressor is used as a limiter. If in Eq. (16.3.2) the absolute value $|x_n|$ is replaced by its square, $|x_n|^2$, the control signal will track the *mean-square* value of the input. In this case, the quantity $\sqrt{c_n}$ will track the RMS level of the input.

The gain processor is a nonlinear function of the control signal imitating the I/O equation (16.3.1). For a compressor, we may define the gain function to be:

$$g = F(c) = \begin{cases} \left(\dfrac{c}{c_0}\right)^{\rho-1}, & \text{if } c \ge c_0 \\ 1, & \text{if } c \le c_0 \end{cases} \qquad \text{(compressor, limiter)} \tag{16.3.6}$$

where c_0 is a desired *threshold* and $\rho < 1$. For an expander, we have $\rho > 1$ and:

$$g = F(c) = \begin{cases} 1, & \text{if } c \ge c_0 \\ \left(\dfrac{c}{c_0}\right)^{\rho-1}, & \text{if } c \le c_0 \end{cases} \qquad \text{(expander, gate)} \tag{16.3.7}$$

Thus, the gain g_n and the final output signal y_n are computed as follows:

$$\boxed{\begin{aligned} g_n &= F(c_n) \\ y_n &= g_n x_n \end{aligned}} \tag{16.3.8}$$

Compressors/expanders are examples of *adaptive signal processing* systems, in which the filter coefficients (in this case, the gain G_n) are time-dependent and adapt themselves to the nature of the input signals [45]. The level detector (16.3.2) serves as the "adaptation" equation and its attack and release time constants are the "learning" time constants of the adaptive system; the parameter λ is called the "forgetting factor" of the system.

As a simulation example, consider a sinusoid of frequency $\omega_0 = 0.15\pi$ rads per sample whose amplitude changes to the three values $A_1 = 2$, $A_2 = 4$, and $A_3 = 0.5$ every 200 samples, as shown in Fig. 16.3.3, that is, $x_n = A_n \cos(\omega_0 n)$, with:

$$A_n = \begin{cases} A_1, & 0 \le n < 200 \\ A_2, & 200 \le n < 400 \\ A_3, & 400 \le n < 600 \end{cases}$$

or, more compactly,

$$A_n = A_1(u_n - u_{n-200}) + A_2(u_{n-200} - u_{n-400}) + A_3(u_{n-400} - u_{n-600})$$

A compressor is used with parameters $\lambda = 0.9$, $c_0 = 0.5$, and $\rho = 1/2$ (that is, 2:1 compression ratio). The output y_n is shown in Fig. 16.3.3; the control signal c_n and gain G_n in Fig. 16.3.4.

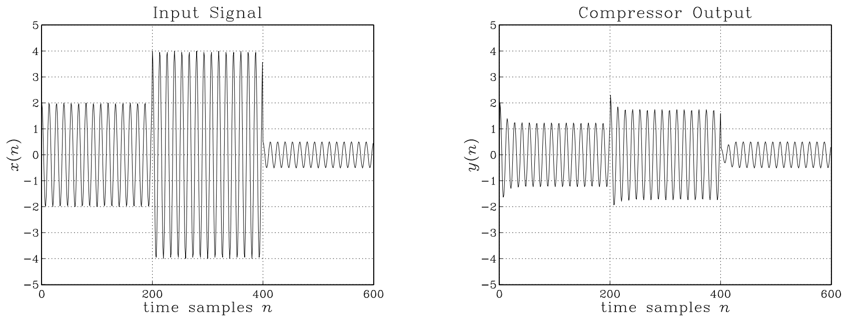


Fig. 16.3.3 Compressor input and output signals ($\rho = 1/2$, $\lambda = 0.9$, $c_0 = 0.5$).

The first two sinusoids A_1 and A_2 lie above the threshold and get compressed. The third one is left unaffected after the release time is elapsed. Although only the stronger signals are attenuated, the overall reduction of the dynamic range will be *perceived* as though the weaker signals also got amplified.

This property is the origin of the popular, but somewhat misleading, statement that compressors attenuate strong signals and amplify weak ones.

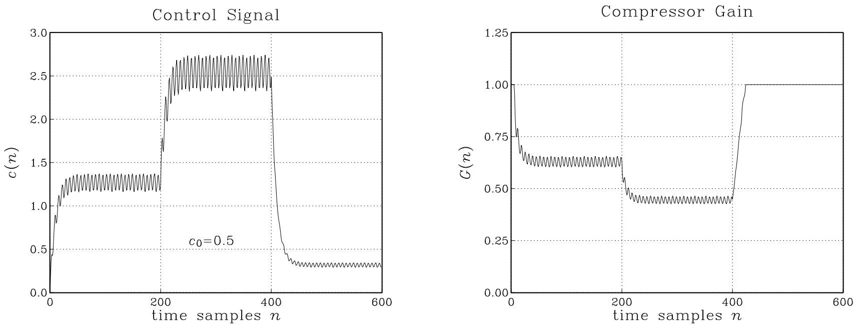


Fig. 16.3.4 Compressor control signal and gain ($\rho = 1/2$, $\lambda = 0.9$, $c_0 = 0.5$).

Jumping between the steady-state levels A_1 and A_2 corresponds to a 6 dB change. Because both levels get compressed, the output levels will differ by, $6\rho = 3$ dB.

16.3.3 Attack and Release Time Constants and Gain Smoothing

A more realistic compressor system is depicted in Fig. 16.3.5 that incorporates the following features:

1. It can accommodate different attack and release time constants, quantified by different filter λ-parameters, say, λ_a, λ_r.
2. The nonlinear gain, $g_n = F(c_n)$, is smoothed further by a lowpass filter, which can be taken to be either an FIR averager or another EMA filter like the detector, resulting in the smoothed gain, G_n.
3. An overall delay D may be introduced at the output that helps reduce transient overshoots, so that, $y_n = G_n x_{n-D}$.
4. Optionally, the gain-control signal c_n could be calculated not from the given input x_n, but from an alternative input, referred to as a "side-chain input", as would be the case in "ducking" applications, or by a bandpass-filtered version of x_n, as in "de-essing" applications.

The sample-by-sample processing algorithm may be summarized as follows:

```
for each audio sample $x_n$, do,
    1. calculate $c_n$, using $\lambda_a$ and $\lambda_r$
    2. calculate nonlinear gain, $g_n = F(c_n)$
    3. calculate smoothed gain, $G_n$
    4. calculate output, $y_n = G_n x_{n-D}$
```
(16.3.9)

These steps are outlined below. To accommodate different attack and release time constants, Eq. (16.3.2) may be replaced by the following version which switches from attack mode λ_a to release mode λ_r depending on the calculated level,

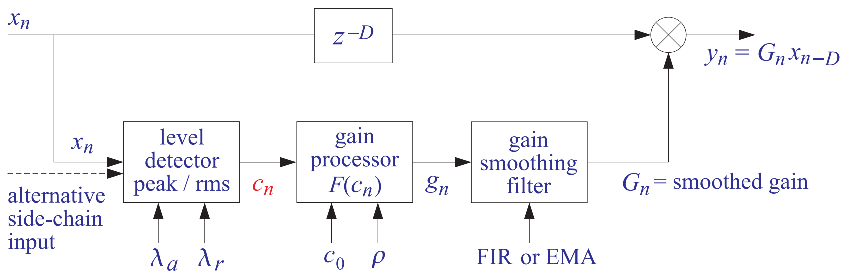


Fig. 16.3.5 Compressor/expander dynamics processor.

$$c_n = \begin{cases} \lambda_a c_{n-1} + (1-\lambda_a)\,|x_n|\,, & \text{if} \quad |x_n| \ge c_{n-1} \\ \lambda_r c_{n-1} + (1-\lambda_r)\,|x_n|\,, & \text{if} \quad |x_n| < c_{n-1} \end{cases} \tag{16.3.10}$$

By choosing $\lambda_a < \lambda_r$, the attack time constant will be shorter than the release one, allowing a quicker response at the onset of compression or expansion. Next, the gain g_n is computed from Eqs. (16.3.6) or (16.3.7), and then passed into the smoothing filter, which may taken to be either an FIR averager or an EMA filter of the form,

$$\begin{aligned} G_n &= \frac{1}{L}\big[g_n + g_{n-1} + \cdots + g_{n-L+1}\big] \qquad &\text{(FIR)} \\ G_n &= \lambda G_{n-1} + (1-\lambda)\,g_n \qquad &\text{(EMA)} \end{aligned} \tag{16.3.11}$$

The averager and EMA filters behave approximately equivalently if the parameters, L, λ, are chosen such that:[†]

$$L \approx \frac{1+\lambda}{1-\lambda} \quad \Leftrightarrow \quad \lambda \approx \frac{L-1}{L+1} \tag{16.3.12}$$

In some implementations [164,169], the EMA smoother is chosen to have different attack and release time constants as follows,

$$G_n = \begin{cases} \lambda_a G_{n-1} + (1-\lambda_a)\,g_n\,, & \text{if} \quad g_n \ge g_{n-1} \\ \lambda_r G_{n-1} + (1-\lambda_r)\,g_n\,, & \text{if} \quad g_n < g_{n-1} \end{cases} \tag{16.3.13}$$

Figure 16.3.6 shows the output signal from the previous example and the compressor gain using a seven-point FIR smoother, $L = 7$. The initial transients in G_n are caused by the input-on transients of the smoother.

Figure 16.3.7 shows the output signal and compressor gain of a limiter, which has a 10:1 compression ratio, $\rho = 1/10$, and uses also a seven-point smoother. The threshold was increased here to $c_0 = 1.5$, so that only A_2 lies above it and gets compressed.

Figure 16.3.8 shows an example of an expander, with parameters $\lambda = 0.9$, $c_0 = 0.5$, $\rho = 2$, and gain function computed by Eq. (16.3.7) and smoothed by a seven-point

[†] see, for example, Eq. (15.5.6)

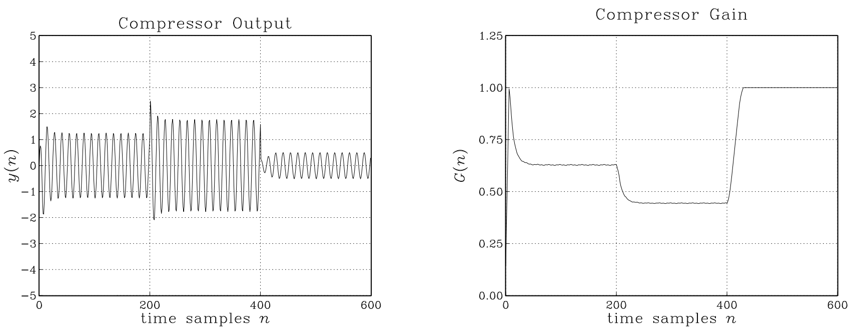


Fig. 16.3.6 Compressor output with smoothed gain ($\rho = 1/2$, $\lambda = 0.9$, $c_0 = 0.5$).

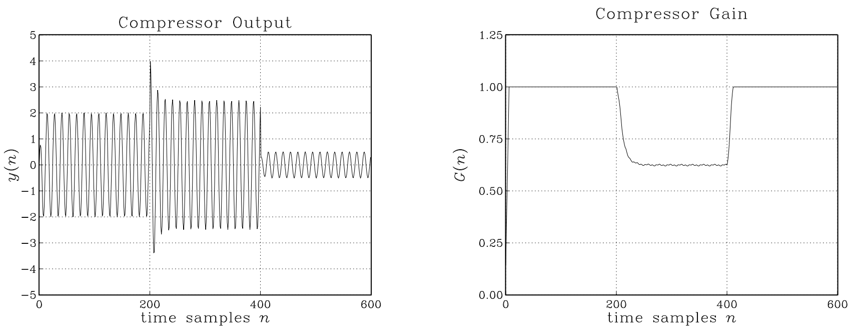


Fig. 16.3.7 Limiter output with smoothed gain ($\rho = 1/10$, $\lambda = 0.9$, $c_0 = 1.5$).

smoother. Only A_3 lies below the threshold and gets attenuated. This causes the overall dynamic range to increase. Although the expander affects only the weaker signals, the overall increase in the dynamic range is perceived as making the stronger signals louder and the weaker ones quieter.

Finally, Fig. 16.3.9 shows an example of a noise gate implemented as an expander with a 10:1 expansion ratio, $\rho = 10$, having the same threshold as Fig. 16.3.8. It essentially removes the sinusoid A_3, which might correspond to unwanted noise.

16.3.4 Computer Experiments

Experiment 1

a. Consider a sinusoid $x(t) = A\cos(2\pi f t)$. Show that its mean-square average over one period, and its absolute average are given by:

$$\overline{x^2(t)} = \frac{1}{2}A^2\,, \quad \overline{|x(t)|} = \frac{2}{\pi}A \tag{16.3.14}$$

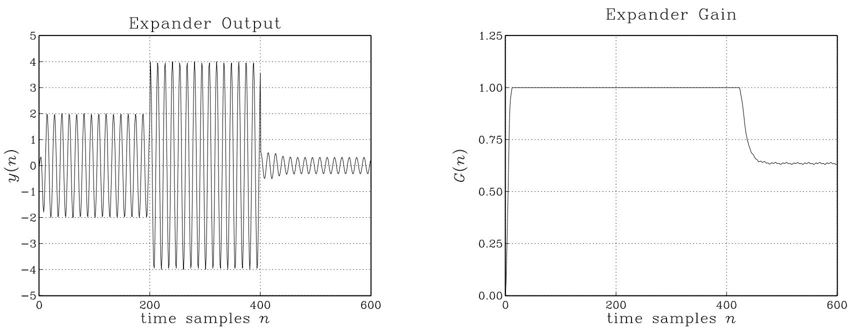


Fig. 16.3.8 Expander output and gain ($\rho = 2$, $\lambda = 0.9$, $c_0 = 0.5$).

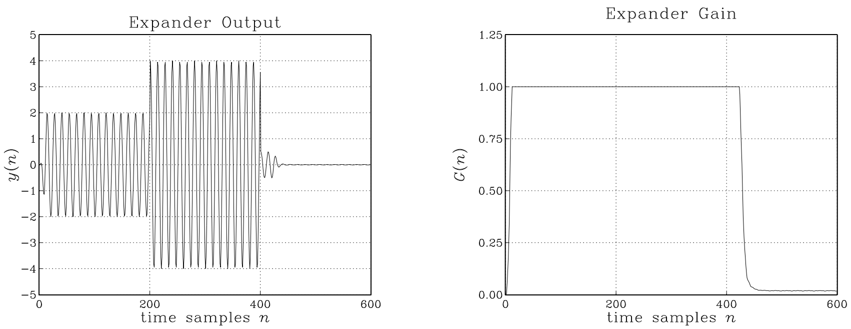


Fig. 16.3.9 Noise gate output and gain ($\rho = 10$, $\lambda = 0.9$, $c_0 = 0.5$).

These could be used as guides in choosing the compressor thresholds.

b. It is desired to design a digital dynamics processor based on the block diagram of Fig. 16.3.5 that operates at an 8 kHz sampling rate and has 20-dB attack and release time constants of 2 msec and 10 msec, respectively.

   Calculate the corresponding values of the forgetting factors λ_a, λ_r, and use them in Parts (c–g) of this project. In implementing the gain smoothing filter, choose its FIR length L or its EMA parameter λ based on the attack value of λ_a.

c. Generate a 75-msec long input signal $x(t)$ sampled at 8 kHz, consisting of three sinusoids of the following frequencies, amplitudes, and durations,

$$\begin{aligned}
f_1 &= 0.3\text{ kHz}\,, \quad A_1 = 2.0\,, \quad \text{duration,}\quad 0 \le t < 25\text{ msec}\\
f_2 &= 0.6\text{ kHz}\,, \quad A_2 = 4.0\,, \quad \text{duration,}\quad 25 \le t < 50\text{ msec}\\
f_3 &= 1.2\text{ kHz}\,, \quad A_3 = 0.5\,, \quad \text{duration,}\quad 50 \le t < 75\text{ msec}
\end{aligned}$$

   Calculate the mean-absolute values of the three levels. Plot $x(t)$ versus t.

d. Let x_n denote the time samples of $x(t)$ and use them as the input to Fig. 16.3.5. Using the parameters, $\rho = 1/3$, $c_0 = 1$, $D = 0$, calculate the corresponding compressed signal y_n using an FIR gain smoothing filter. Plot $y(t)$ vs. sampled t using the same scales as for $x(t)$.

   Moreover, in three separate graphs, plot versus t the control signal $c(t)$, the raw gain $g(t)$, and its smoothed version $G(t)$.

   Notes: Do not use the built-in function **filter** in this part, rather, implement the algorithm of Eq. (16.3.9) on a sample by sample basis.

   Although Eqs. (16.3.6) and (16.3.7) can be combined into a one-line anonymous vectorized MATLAB function, such function would generate NaN's if the control signal happened to be zero, $c = 0$. Therefore, it is best to write a separate function which handles such circumstance.

e. For the same input x_n, choose appropriate values for ρ, c_0 such that the compressor would act as a limiter that limits the f_2 signal, but not f_1 and f_3. You may use an FIR or EMA gain smoother in this part.

   Plot the corresponding output $y(t)$ using the same scales as for $x(t)$, and on separate graphs also plot the signals $c(t), g(t), G(t)$.

   For easy reference place on the $y(t)$ graph the values of ρ, c_0 that you used.

f. Next, choose appropriate values for ρ, c_0 and an FIR or EMA smoother, such that the dynamics processor would act as an expander that attenuates the f_3 component only, while leaving f_1, f_2 unaffected.

   Plot the corresponding output $y(t)$ using the same scales as for $x(t)$, and on separate graphs also plot the signals $c(t), g(t), G(t)$.

   For easy reference place on the $y(t)$ graph the values of ρ, c_0 that you used.

g. Repeat part (f) such that the dynamics processor would act as a noise gate suppressing the f_1 and f_3 components, but not f_2.

   Then, repeat this part, so that now the noise gate removes only f_3, but not f_1, f_2.

Some example graphs are depicted below.

Experiment 2 – Duckers

In this experiment, you will implement a dynamics processor acting as a "ducker", shown in Fig. 16.3.10. Please load the following wave files into MATLAB (music file is from Ch.9 files in Ref. [169]),

```
[xs,fs] = audioread(’speech.wav’);    % speech signal
[xm,fs] = audioread(’music.wav’);     % music signal
```

The signals $x_s(t)$ and $x_m(t)$ both have exactly the same duration of approximately 7 seconds, and the same sampling rate of $f_s = 44.1$ kHz. The actual speech contained in $x_s(t)$ is preceded and anteceded by 2 seconds of silence. Plot the signals $x_s(t)$ and $x_m(t)$ versus t, as well as the combined mixed signal, $x(t) = x_s(t) + x_m(t)$ (not shown in Fig. 16.3.10).

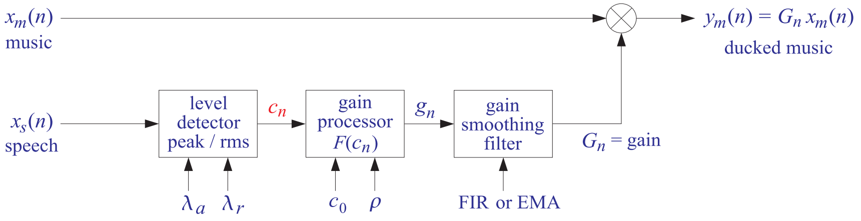


Fig. 16.3.10 Compressor/ducker dynamics processor.

In the dynamics processor of Fig. 16.3.10, the gain $G(t)$ to be applied to the music signal $x_m(t)$ is controlled by the side-chain input speech signal $x_s(t)$. The scaled music signal output is $y_m(t) = G(t)x_m(t)$. The compressor parameters are chosen such that $G(t)$, and hence $y_m(t)$, become very small whenever the speech signal $x_s(t)$ is present, thus, the music signal is "ducked" allowing a clearer hearing of the speech in the combined mixed output,

$$y(t) = x_s(t) + y_m(t) = x_s(t) + G(t)x_m(t) \tag{16.3.15}$$

Calculate the combined speech plus ducked music signal, $y(t) = x_s(t) + y_m(t)$, listen to it, and compare it with the unprocessed combined signal, $x(t) = x_s(t) + x_m(t)$.

Experiment with different values of the parameters $\rho, c_0, \lambda_a, \lambda_r$ until you are satisfied with the ducking result (you may use an EMA gain smoother here). As a starting point, you may want to choose c_0 to be somewhere between 40–60 dB below the maximum value of $x_s(t)$ and choose the attack time constant to be a few tens of milliseconds, and the release time constant, a few hundreds of milliseconds.

Once you are satisfied with the results, plot $y(t)$ and $y_m(t)$ versus t, as well as the control and gain signals, $c(t), g(t), G(t)$.

Save your processed speech plus ducked music signal $y(t)$ in a wave file. For your reference, the following wave files are included containing the signals, $x(t) = x_s(t) + x_m(t)$, and, $y_m(t)$ and $y(t) = x_s(t) + y_m(t)$, which are also depicted in the example graphs below (Fig. 16.3.17).

$$\begin{aligned} x(t) &= x_s(t) + x_m(t)\,, && \texttt{'speech+music.wav'} \\ y_m(t) &= G(t)x_m(t)\,, && \texttt{'ducked music.wav'} \\ y(t) &= x_s(t) + y_m(t)\,, && \texttt{'speech+ducked.wav'} \end{aligned}$$

16.3.5 Example Graphs

16.4 Problems

16.1 It is desired to generate the periodic sequence $\mathbf{h} = [0, 1, 2, 0, 1, 2, 0, 1, 2, \dots]$ of period three. Determine the filter $H(z)$ whose impulse response is $\mathbf{h}$.

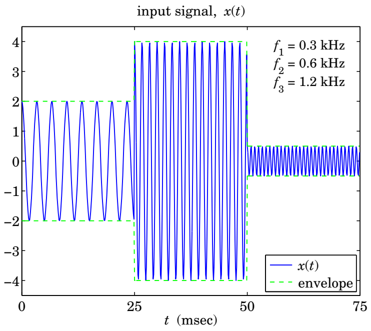


Fig. 16.3.11 Input signal x_n.

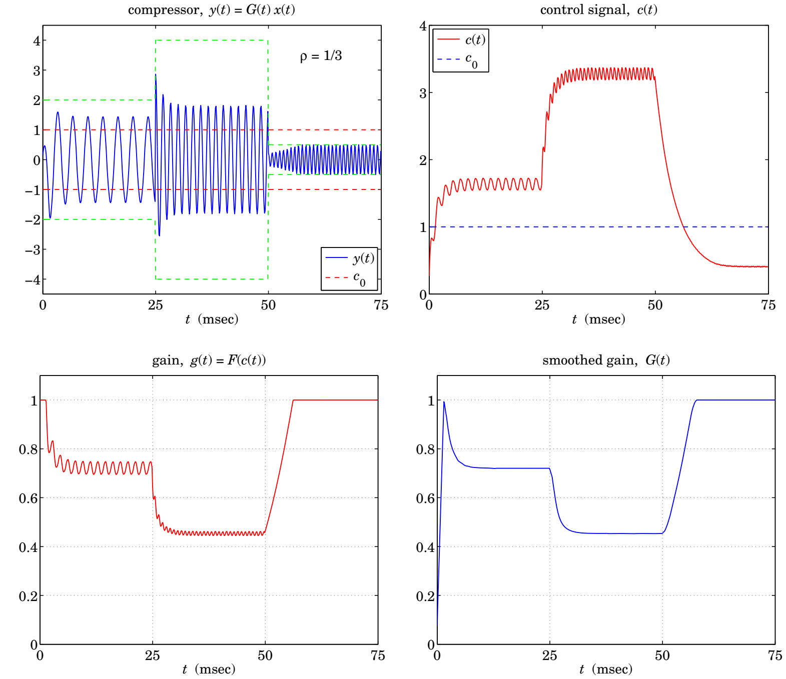


Fig. 16.3.12 Compressor, $\rho = 1/3$, $c_0 = 1$, using FIR smoother.

a. Realize the filter in its direct and canonical forms. Write the corresponding sample processing algorithms for generating the periodic sequence. Crank the algorithms for a total of 9 iterations, making a list of the values of the internal states and output of the filter.

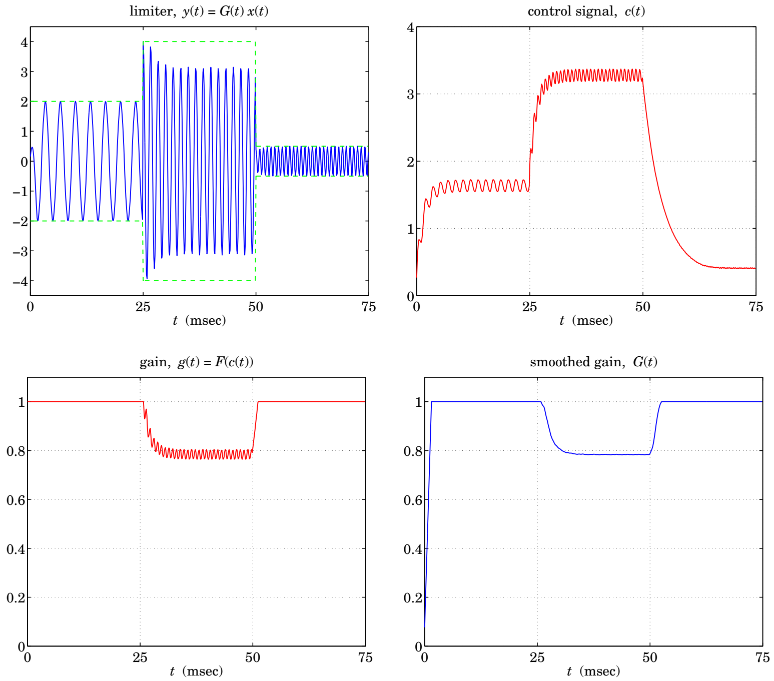


Fig. 16.3.13 Limiter.

b. For the direct form realization, rewrite the generation algorithm in its circular-buffer form of Eq. (16.1.20) or (16.1.22), and initialized by Eq. (16.1.19).

Iterate the algorithm 15 times, making a table of the internal states $\mathbf{w}$, the output y, the circular pointer index q, and indicating the buffer entry that holds the current output for each iteration. Why did we choose to iterate 15 times? Do you observe the repetition period of the buffer entries?

16.2 Consider the filter $H(z) = \dfrac{1 + 2z^{-1} + 3z^{-2} - 4z^{-3} - 5z^{-4}}{1 - z^{-5}}$. What is its periodic causal impulse response? Realize the filter in its direct and canonical forms.

a. For each realization, write the corresponding sample processing algorithm for generating the periodic impulse response. Crank the algorithm for a total of 15 iterations, making a list of the values of the internal states and output of the filter.

b. For the direct form realization, iterate the generation algorithm in its circular buffer form, making a table as in Problem 16.1(b). How many iterations are needed before we observe the repetition of the buffer entries?

16.3 The eight waveform samples:

$$\mathbf{b} = [b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7]$$

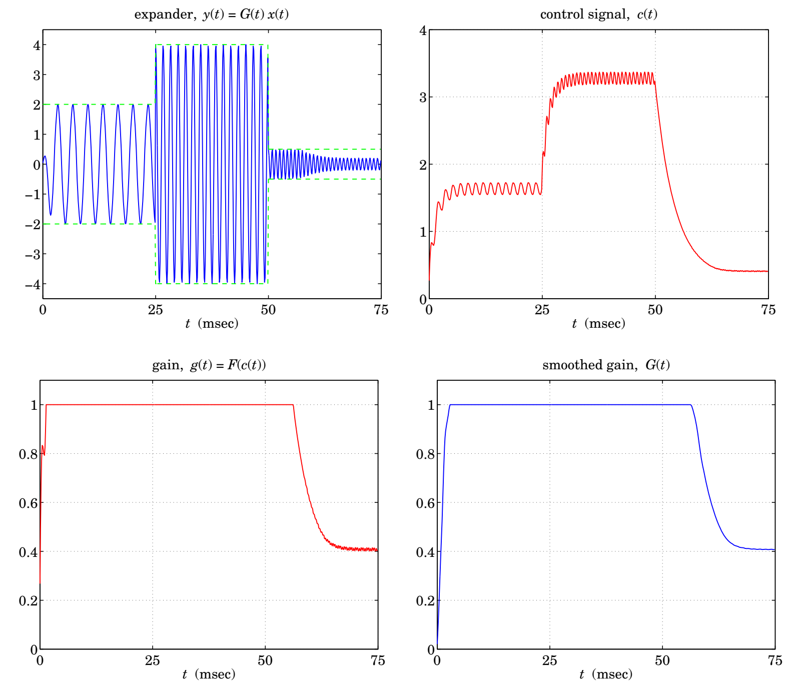


Fig. 16.3.14 Expander.

are stored in reverse order in the eight-dimensional circular wavetable:

$$\mathbf{w} = [w_0, w_1, w_2, w_3, w_4, w_5, w_6, w_7]$$

It is desired to generate a periodic subsequence of period $d = 5$. Determine this subsequence when the output is obtained by the four methods of: (a) truncating down, (b) truncating up, (c) rounding, and (d) linear interpolation.

16.4 Repeat Problem 16.3 when the subsequence has period $d = 6$.

16.5 The waveform samples $\mathbf{b} = [1, 2, 3, 4, 5, 6, 7, 8]$ are stored (in reverse order) into an eight-dimensional circular wavetable $\mathbf{w}$. It is desired to use the wavetable to generate a periodic subsequence of period 3. Determine this subsequence when the output is obtained by the four approximations of: (a) truncating down, (b) truncating up, (c) rounding, and (d) linear interpolation.

16.6 Repeat Problem 16.5, for generating a subsequence of period 5. Repeat for a subsequence of period 6.

16.7 *Computer Experiment: Wavetable Generators.* Using the wavetable generator `wavgen`, write a C program to reproduce all the graphs of Fig. 16.1.18.

Then, repeat using the rounding and interpolation versions of the wavetable generator, `wavgenr` and `wavgeni`. Compare the outputs of the three generator types.

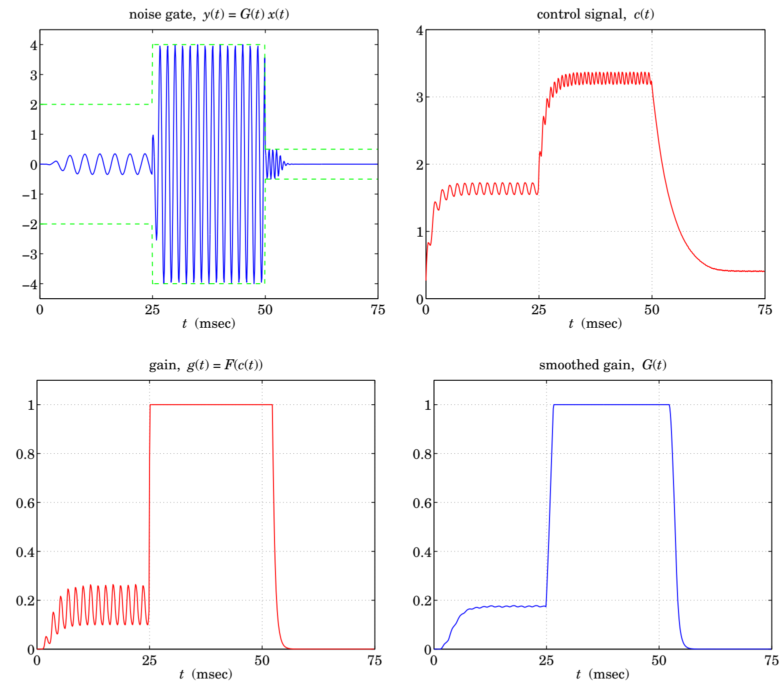


Fig. 16.3.15 Noise gate suppresing f_1 and f_3.

16.8 *Computer Experiment: Wavetable Amplitude and Frequency Modulation.* Write a program to reproduce all the graphs of Figures 16.1.20–16.1.23.

16.9 Consider the four comb filters:

$$y(n) = x(n) + x(n-8), \qquad y(n) = x(n) + x(n-8) + x(n-16)$$
$$y(n) = x(n) - x(n-8), \qquad y(n) = x(n) - x(n-8) + x(n-16)$$

Determine their transfer functions and their impulse responses. Place their zeros on the z-plane relative to the unit circle. Sketch their magnitude responses. How are they similar or different? Draw their canonical realization forms using 8-fold delays z^{-8}. Write the corresponding sample processing algorithms both in their linear and circular-buffer versions.

16.10 *Computer Experiment: Flanging and Chorusing.* Write a C program to reproduce the graphs of Figures 16.2.9 and 16.2.11.

Repeat the chorusing experiment using the following model for the chorus processor, shown in Fig. 16.2.10:

$$y(n) = \frac{1}{3}\Big[x(n) + a_1(n)x\big(n - d_1(n)\big) + a_2(n)x\big(n - d_2(n)\big)\Big]$$

where $d_1(n)$ and $d_2(n)$ are generated as in Eq. (16.2.20) by the low-frequency noise routine `ran1` of Appendix A.2 using two different seeds. The amplitudes $a_1(n)$, $a_2(n)$ are also low-frequency random numbers with unity mean.

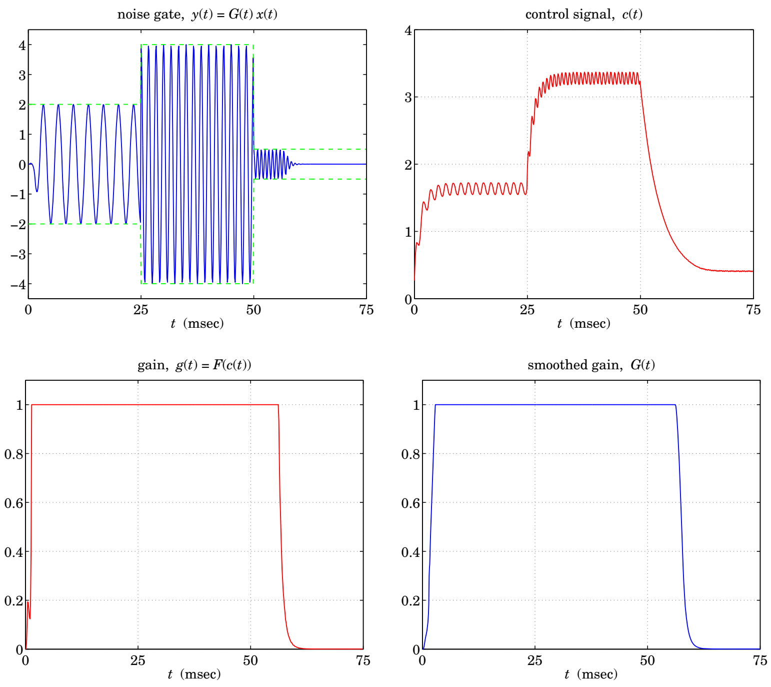


Fig. 16.3.16 Noise gate suppresing f_3 only.

Repeat the flanging experiment using the recursive flanging processor:

$$y(n) = ay\big(n - d(n)\big) + x(n)$$

where $a = 0.8$. State the processing algorithm in this case, using a circular buffer for the feedback delay line and the routine `tapi` to interpolate between buffer entries.

16.11 *Computer Experiment: Reverberation Examples.* Using the circular-buffer reverberator routines `plain`, `allpass`, `lowpass`, write a C program to reproduce all the graphs of Fig. 16.2.22. [*Caution:* Use different circular buffers for the three reverb filters.]

16.12 *Computer Experiment: Schroeder's Reverberator.* Write a C program that implements Schroeder's reverberator shown in Fig. 16.2.18 and uses the sample processing algorithm (16.2.31). Iterate the sample processing algorithm for $0 \le n \le 500$ and reproduce the impulse response shown in Fig. 16.2.19.

16.13 Consider the lowpass reverberator shown in Fig. 16.2.21. Write *explicitly* all the difference equations required for its time-domain implementation. Then, write the corresponding sample processing algorithm, with the D-fold delay implemented circularly.

16.14 Consider the lowpass reverberator $H(z)$ of Eq. (16.2.32) with the first-order feedback filter (16.2.35). Let p_i, A_i, $i = 1, 2, \dots, D+1$ be the poles and residues of the $H(z)$, that is,

$$H(z) = \frac{1}{1 - z^{-D}G(z)} = \sum_{i=1}^{D+1} \frac{A_i}{1 - p_i z^{-1}}$$

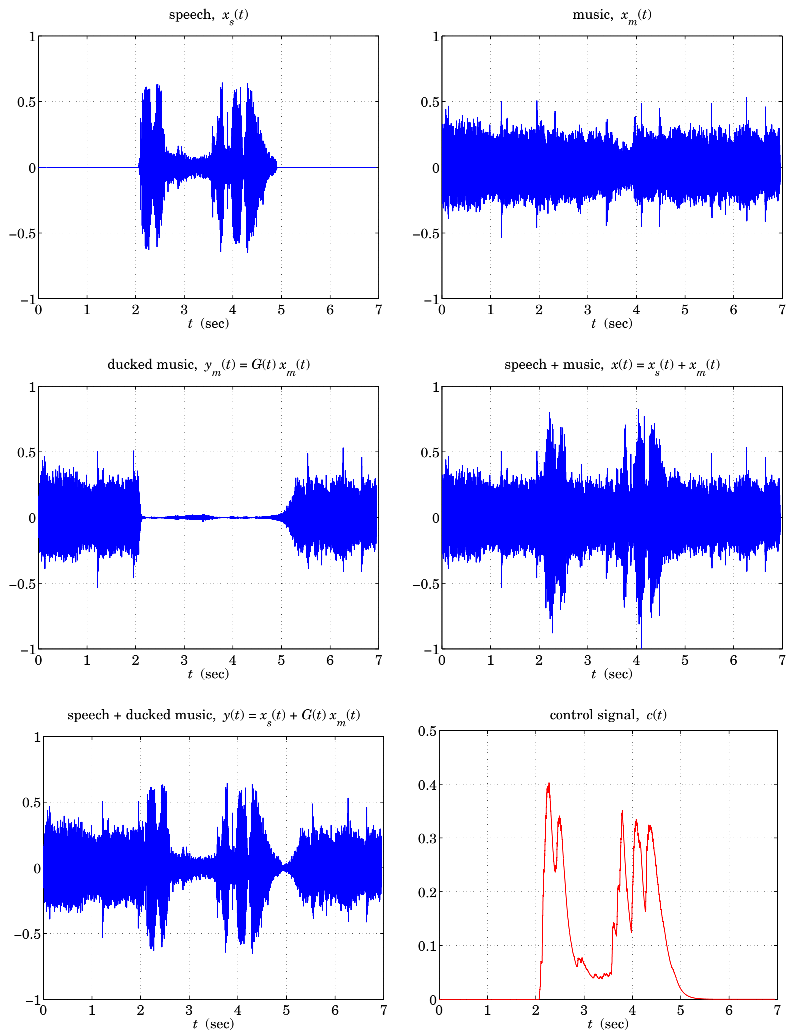


Fig. 16.3.17 Speech, music, speech + ducked music.

Assume that all p_i are inside the unit circle. Note that if $b_1 = 0$, then there are only D poles. Suppose a sinusoid of frequency ω and duration L is applied to the input:

$$x(n) = e^{j\omega n}\big(u(n) - u(n - L)\big)$$

Show that the output signal will be given by:

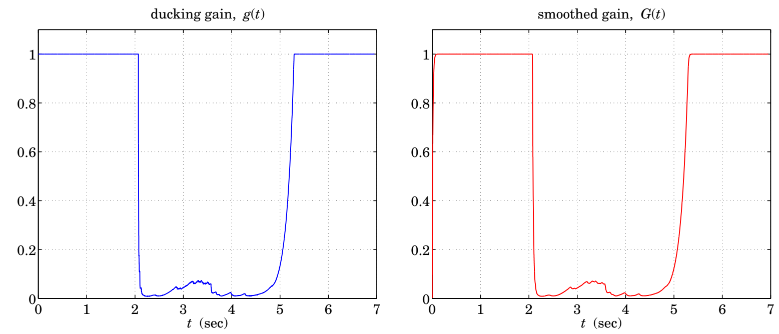


Fig. 16.3.18 Control signal for ducked music, and gains.

$$y(n)=H(\omega)e^{j\omega n}\big(u(n)-u(n-L)\big)+\sum_{i=1}^{D+1}B_i p_i^n\big(u(n)-e^{j\omega L}p_i^{-L}u(n-L)\big)$$

where $B_i = p_i A_i/(p_i - e^{j\omega})$, $i = 1, 2, \dots, D+1$. See also Problem 6.29.

16.15 *Computer Experiment: Reverberator Time Constants.* Reproduce all the graphs of Figure 16.2.25 by iterating the sample processing algorithms of the plain and lowpass reverberators. The input is defined as:

$$x(n)=\cos(\omega n)\big(u(n)-u(n-150)\big)$$

with $\omega = 0.2\pi$ and $\omega = \pi$. Generate similar graphs also for the following frequencies: $\omega = 0.4\pi$, 0.6π, 0.8π, and 0.99π.

For the lowpass cases, verify that the output obtained by iterating the sample processing algorithm agrees with (the real part of) the analytical expression given in Problem 16.14. For this part, you will need to use MATLAB to calculate the poles p_i, residues A_i, B_i, and evaluate the expression for $y(n)$, for $0 \le n \le 299$.

16.16 *Computer Experiment: Karplus-Strong String Algorithm.* The Karplus-Strong algorithm for generating plucked-string sounds [124–126] is defined by the lowpass reverberator filter of Eq. (16.2.32) with feedback filter $G(z) = (1+z^{-1})/2$. It was described in Section 16.2.3.

For the two delay values $D = 25, 50$, initialize the delay-line buffer by filling it with zero-mean random numbers, for example, $w[i] = \text{ran}(\&\text{iseed}) - 0.5$, for $i = 0, 1, \dots, D$. Then, run the sample processing algorithm (16.2.34) with zero input $x(n) = 0$, for $0 \le n \le 499$. Plot the resulting output signals $y(n)$.

The harshness of the initial plucking of the string is simulated by the initial random numbers stored in the delay line. As these random numbers recirculate the delay line, they get lowpass filtered by $G(z)$, thus losing their high-frequency content and resulting in a decaying signal that is dominated basically by the frequency $f_1 = f_s/D$.

16.17 A prototypical delay effect usually built into commercial audio DSP effects processors is given by the transfer function:

$$H(z)=c+b\frac{z^{-D}}{1-az^{-D}}$$

where c represents the direct sound path. Draw a block diagram of this filter using only one D-fold delay z^{-D}. Write the difference equations describing it and translate them into a sample processing algorithm implemented with a circular buffer.

16.18 *Computer Experiment: Plain and Lowpass Reverberating Delays.* The basic building blocks of many multi-delay effects are the following plain and lowpass reverberating delays:

$$H(z)=\frac{z^{-D}}{1-az^{-D}}\,,\qquad H(z)=\frac{z^{-D}}{1-z^{-D}G(z)}$$

where $G(z)$ is a lowpass feedback filter. Draw the block diagrams of these filters and write their sample processing algorithms implementing z^{-D} circularly. Then, translate the algorithms into C routines, say `plaindel.c` and `lpdel.c`. How do they differ from the routines `plain` and `lowpass` of Section 16.2.3?

16.19 *Computer Experiment: Multi-Delay Effects.* Commercial audio DSP effects processors have built-in multi-delay effects obtained by cascading several basic reverberating delay of the type of Problem 16.18; for example, see Ref. [156].

A typical example was shown in Fig. 16.2.27. Write a C program that implements this block diagram. The program must make use of the two routines `plaindel` and `lpdel` that you wrote in the previous problem.

Note, that you will need to use two circular buffers $\{\mathbf{w}_1, \mathbf{w}_2\}$ and their circular pointers $\{p_1, p_2\}$, for the two delays.

Using this program, and the parameter values that were used in Fig. 16.2.28, compute and plot the outputs of the filter, for $0 \le n \le 2000$, for the two inputs:

$$x(n)=\delta(n)\,,\qquad x(n)=u(n)-u(n-100)$$

16.20 *Computer Experiment: Multi-Tap Delay Effects.* In the electronic music community, a multi-tap delay is usually defined to have both feed forward and feedback paths, as well as a direct sound path, with user-adjustable gains; for example, see Ref. [156].

Write a C routine that implements the circular-buffer version of the sample processing algorithm of the multitap delay line shown in Fig. 16.2.29. The inputs to the routine should be the current input audio sample x, the values of the forward taps $\{b_0, b_1, b_2\}$, feedback taps $\{a_1, a_2\}$, delay values $\{D_1, D_2\}$, and the (D_1+D_2)-dimensional delay-line buffer $\mathbf{w}$ and its associated circular pointer p.

Using this routine, and the parameter values that were used for the stable case of Fig. 16.2.30, compute and plot the outputs of the filter, for $0 \le n \le 1000$, for the two inputs:

$$x(n)=\delta(n)\,,\qquad x(n)=u(n)-u(n-200)$$

16.21 Show that the condition $|a_1| + |a_2| < 1$ is sufficient to guarantee the stability of the multitap delay line filter of Eq. (16.2.43). [*Hint:* Work with the pole equation $z^{D_1+D_2} = a_1 z^{D_2} + a_2$.]

16.22 Stereo delay effects can be accomplished by the block diagram of Fig. 16.4.1. Two basic delays of the type of Problem 16.18 are used in the left and right channels and are coupled by introducing cross-feedback coefficients, such that the reverberating output of one is fed into the input of the other; for example, see Ref. [156]. Show that the input/output relationships can be expressed in the z-domain as:

$$Y_L(z)=H_{LL}(z)X_L(z)+H_{LR}(z)X_R(z)$$
$$Y_R(z)=H_{RL}(z)X_L(z)+H_{RR}(z)X_R(z)$$

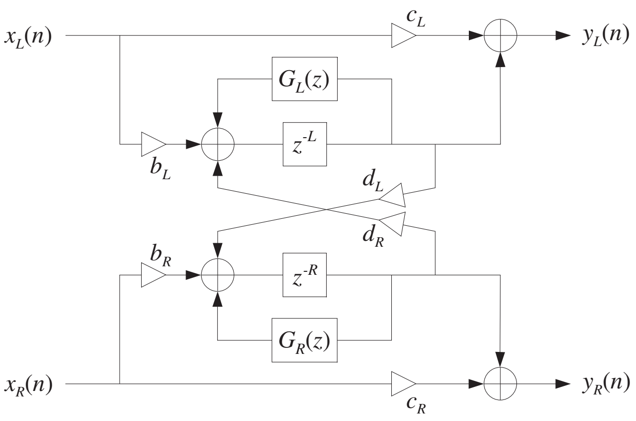


Fig. 16.4.1 Stereo delay effects processor.

Determine the direct and cross-transfer functions $H_{LL}(z)$, $H_{LR}(z)$, $H_{RL}(z)$, $H_{RR}(z)$, in terms of the indicated multipliers and feedback filters $G_L(z)$, $G_R(z)$. What conclusions do you draw in the special cases: (1) $d_L = 0$, $d_R \neq 0$; (2) $d_L \neq 0$, $d_R = 0$; (3) $d_L = 0$, $d_R = 0$?

Consider the case of the plain feedback filters: $G_L(z) = a_L$, $G_R(z) = a_R$. Introduce two delay-line buffers $\mathbf{w}_L$ and $\mathbf{w}_R$ for the indicated delays z^{-L} and z^{-R} and write the *difference equations* describing the time-domain operation of the block diagram. Then, translate the difference equations into a sample processing algorithm that transforms each input stereo pair $\{x_L, x_R\}$ into the corresponding output stereo pair $\{y_L, y_R\}$. Implement the delays circularly; therefore, you will also need to introduce two circular pointers $\{p_L, p_R\}$.

16.23 *Computer Experiment: Stereo Delay Effects.* Write a C routine that implements the stereo sample processing algorithm of the previous problem. Using this routine, compute and plot the left and right output signals $y_L(n)$, $y_R(n)$, for $n = 0, 1, \dots, 299$, for the case when there is only a left input pulse of duration 5, that is,

$$x_L(n) = u(n) - u(n-5), \qquad x_R(n) = 0$$

Use $L = 30$ and $R = 70$ for the left and right delays, and the multiplier values:

$$a_L = a_R = 0.6, \quad b_L = b_R = 1, \quad c_L = c_R = 0, \quad d_L = d_R = 0.3$$

Identify on your graphs the origin of the various length-5 pulses that appear in the outputs. Next, repeat the experiment using $d_L = 0.3$, $d_R = 0$, so that only the left output is fed into the right input. Again, identify the origin of the pulses in your outputs.

16.24 *Computer Experiment: Compressors and Limiters.* Consider the compressor and limiter presented in Figures 16.3.3—16.3.7.

a. Reproduce these graphs. Is it better to apply the smoothing filter to the output of the gain processor $f(c_n)$, rather than to its input c_n?

b. Given a sinusoid $x(n) = A\cos(\omega_0 n)$, calculate its theoretical mean absolute value $\overline{|x_n|}$ and its rms value $(\overline{|x_n|^2})^{1/2}$, both averaged over one period of the sinusoid.

Are the steady-state values of the control signal in the above graphs consistent with the theoretical values calculated here? In your program in (a), include the numerical calculation of the mean absolute values of the three output sinusoids, averaged over the three length-200 segments. Are these averages consistent with the given compression ratio?

c. Redo the graphs in (a), but without using any smoothing filter.

d. Repeat part (a) using a 3:1 compression ratio, $\rho = 1/3$ and then a 4:1 ratio.

e. Repeat part (a) using a delay of $d = 40$ samples in the direct signal path, as described in Section 16.3.1. Too much of such a delay can introduce a "backward playing" quality into the output. Can you observe this?

Repeat using a delay $D = 40$ in the level detector's input (but not in the direct signal path).

f. Repeat part (a) using a seven-point smoother, but with filter parameter $\lambda = 0.99$. Repeat with $\lambda = 0.2$. Do you observe the effect on the attack and release time constants.?

16.25 *Computer Experiment: Expanders and Gates.* Consider the expander and gate of Figures 16.3.8 and 16.3.9.

a. Redo these graphs using no additional smoothing filter, and then, redo them using a seven-point smoother.

b. Repeat part (a) using a 3:1 expansion ratio, $\rho = 3$.

c. Repeat part (a) using a 2:1 expansion ratio, $\rho = 2$, but moving the threshold higher to the value $c_0 = 1.5$. What happens to the three sinusoids in this case? What happens in the case of the noise gate?

17

High-Order Digital Parametric Equalizers

17.1 Overview

Digital parametric audio equalizers are commonly implemented as biquadratic filters [283–297], as we discussed them in Sec. 12.4. In some circumstances, it might be of interest to use equalizer designs based on high-order filters. Such designs can provide flatter passbands and sharper bandedges at the expense of higher computational cost.

In this section, based on [329], we present a family of digital equalizers and shelving filters derived from high-order Butterworth, Chebyshev, and elliptic lowpass analog prototypes and obtain explicit design equations for the filter coefficients in terms of the desired peak gain, peak or cut frequency, bandwidth, and bandwidth gain. We discuss frequency-shifted transposed, normalized-lattice, and minimum roundoff-noise state-space realization structures, as well as structures that allow the independent control of center frequency, gain, and bandwidth. The design equations apply equally well to ordinary lowpass, highpass, bandpass, and bandstop filters.

High-order equalizers have been considered previously by Moorer [327] who used a conformal mapping method based on elliptic functions to map a first-order lowpass digital shelving filter into a high-order elliptic equalizer, and by Keiler and Zölzer [328] who obtained a fourth-order equalizer based on a second-order analog Butterworth prototype. Our elliptic designs are essentially equivalent to Moorer's, but we follow a direct approach that closely parallels the conventional analog filter design methods and can be applied equally well to all three filter types, Butterworth, Chebyshev, and elliptic.

We start by designing a high-order analog lowpass shelving filter that meets the given gain and bandwidth specifications. The analog filter is then transformed into a digital lowpass shelving filter using the bilinear transformation. Finally, the digital shelving filter is transformed into a peaking equalizer centered at the desired peak frequency using a lowpass-to-bandpass z-domain transformation [321,322].

One starts by designing a lowpass analog shelving equalizer filter having a magnitude response of the form:

$$|H(\Omega)|^2 = \frac{G^2 + G_0^2\varepsilon^2 F_N^2(w)}{1+\varepsilon^2 F_N^2(w)}\,, \quad w = \frac{\Omega}{\Omega_B}$$

and from the zeros and poles of that expression, one determines the corresponding analog transfer function, and then transforms that into a digital bandpass equalizer. The quantity Ω_B is an effective bandwidth frequency that corresponds to a desired bandwidth level G_B (such as the 3-dB level). The parameter ε is calculated from the condition that $F_N(w) = 1$ at $\Omega = \Omega_B$:

$$\frac{G^2 + G_0^2\varepsilon^2}{1+\varepsilon^2} = G_B^2 \quad \Rightarrow \quad \varepsilon = \sqrt{\frac{G^2 - G_B^2}{G_B^2 - G_0^2}}$$

Here, G is the peak or cut gain, and G_0 a reference gain, typically chosen to be $G_0 = 1$ for cascaded equalizers. All of the designs of the conventional lowpass, highpass, and bandpass analog filters that we discussed in Sec. 13.5 correspond to the special case of $G = 1, G_0 = 0$.

A MATLAB toolbox of functions for the design and implementation of such parametric equalizers is available in [330], and included in ISP2e. The toolbox may also be used to design ordinary lowpass, highpass, bandpass, and bandstop filters, as an alternative to the methods discussed in this chapter.

17.2 General Considerations

The design specifications for the digital equalizer are the quantities $\{G, G_0, G_B, f_0, \Delta f, f_s\}$, that is, the peak or cut gain G, the reference gain G_0 (usually set equal to unity), the bandwidth gain G_B, the peak or cut frequency f_0 in Hz, the bandwidth Δf measured at level G_B, and the sampling rate f_s. These are illustrated in Fig. 17.2.1 for the Butterworth case. In the elliptic case, an additional gain, G_s, needs to be specified, as discussed in Section 17.4. The bandwidth is related to the left and right bandedge frequencies f_1, f_2 by $\Delta f = f_2 - f_1$. It is convenient to work with the normalized digital frequencies in units of radians per sample:

$$\omega_0 = \frac{2\pi f_0}{f_s}\,, \quad \Delta\omega = \frac{2\pi\Delta f}{f_s}\,, \quad \omega_1 = \frac{2\pi f_1}{f_s}\,, \quad \omega_2 = \frac{2\pi f_2}{f_s} \tag{17.2.1}$$

The starting point of the design method is an equivalent analog lowpass shelving filter, illustrated in Fig. 17.2.1, that has the same gain specifications as the desired equalizer, but with peak frequency centered at $\Omega = 0$ and bandedge frequencies at $\pm\Omega_B$.

The analog filter may be transformed directly to the desired digital equalizer by the bandpass transformation between the s and z planes [321]:

$$s = \frac{1 - 2\cos\omega_0\, z^{-1} + z^{-2}}{1 - z^{-2}} \tag{17.2.2}$$

The corresponding frequency mapping between $s = j\Omega$ and $z = e^{j\omega}$ is found from (17.2.2) to be:

$$\Omega = \frac{\cos\omega_0 - \cos\omega}{\sin\omega} \tag{17.2.3}$$

where $\omega = 2\pi f/f_s$ and f is the physical frequency in Hz. The requirement that the bandedge frequencies ω_1, ω_2 map onto $\pm\Omega_B$ gives the conditions:

$$\frac{\cos\omega_0 - \cos\omega_1}{\sin\omega_1} = -\Omega_B\,, \quad \frac{\cos\omega_0 - \cos\omega_2}{\sin\omega_2} = \Omega_B \tag{17.2.4}$$

These may be solved for ω_0 and Ω_B in terms of ω_1 and ω_2:

$$\Omega_B = \tan\left(\frac{\Delta\omega}{2}\right), \qquad \tan^2\left(\frac{\omega_0}{2}\right) = \tan\left(\frac{\omega_1}{2}\right)\tan\left(\frac{\omega_2}{2}\right) \tag{17.2.5}$$

where $\Delta\omega = \omega_2 - \omega_1$. Equivalently, we have:

$$\cos\omega_0 = \frac{\sin(\omega_1 + \omega_2)}{\sin\omega_1 + \sin\omega_2} \tag{17.2.6}$$

Conversely, Eqs. (17.2.4) may be solved for ω_1 and ω_2 in terms of ω_0 and $\Delta\omega$:

$$e^{j\omega_1} = \frac{c_0 + j\sqrt{\Omega_B^2 + s_0^2}}{1 + j\Omega_B}, \qquad e^{j\omega_2} = \frac{c_0 + j\sqrt{\Omega_B^2 + s_0^2}}{1 - j\Omega_B} \tag{17.2.7}$$

where $\Delta\omega$ enters through $\Omega_B = \tan(\Delta\omega/2)$. Extracting the real parts of Eq. (17.2.7), we obtain:

$$\cos\omega_1 = \frac{c_0 + \Omega_B\sqrt{\Omega_B^2 + s_0^2}}{\Omega_B^2 + 1}, \qquad \cos\omega_2 = \frac{c_0 - \Omega_B\sqrt{\Omega_B^2 + s_0^2}}{\Omega_B^2 + 1} \tag{17.2.8}$$

where we introduced the shorthand notation $c_0 = \cos\omega_0$ and $s_0 = \sin\omega_0$. Eqs. (17.2.7) have the proper limits as $\omega_0 \to 0$ and $\omega_0 \to \pi$, resulting in the cutoff frequencies (measured at level G_B) of the digital lowpass and highpass shelving equalizers:

$$\begin{aligned} &\omega_0 = 0\,, && \omega_1 = 0\,, && \omega_2 = \Delta\omega\,, && \text{(LP shelf)} \\ &\omega_0 = \pi\,, && \omega_1 = \pi - \Delta\omega\,, && \omega_2 = \pi\,, && \text{(HP shelf)} \end{aligned} \tag{17.2.9}$$

The magnitude responses of the high-order analog lowpass shelving Butterworth, Chebyshev, and elliptic prototype filters that we consider here are taken to be:

$$|H_a(\Omega)|^2 = \frac{Ghere^2 + G_0^2\varepsilon^2 F_N^2(w)}{1 + \varepsilon^2 F_N^2(w)} \tag{17.2.10}$$

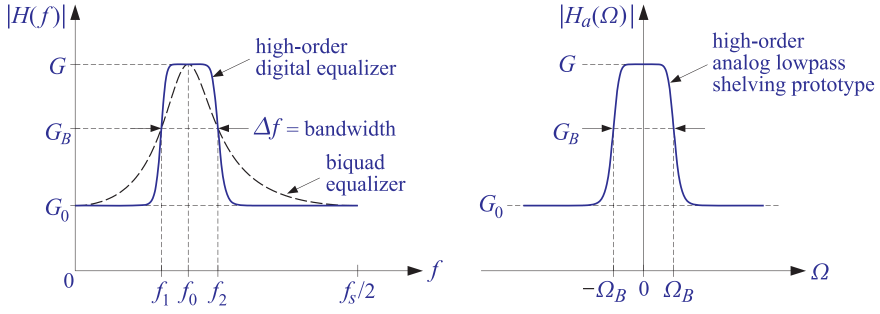


Fig. 17.2.1 Specifications of high-order equalizer and the equivalent lowpass analog prototype.

where N is the analog filter order, ε is a constant, and $F_N(w)$ is a function of the normalized frequency $w = \Omega/\Omega_B$ given by:

$$F_N(w) = \begin{cases} w^N, & \text{Butterworth} \\ C_N(w), & \text{Chebyshev, type-1} \\ 1/C_N(w^{-1}), & \text{Chebyshev, type-2} \\ \mathrm{cd}(NuK_1, k_1), \quad w = \mathrm{cd}(uK, k), & \text{Elliptic} \end{cases} \tag{17.2.11}$$

where $C_N(x)$ is the order-N Chebyshev polynomial, that is, $C_N(x) = \cos(N\cos^{-1}x)$, and $\mathrm{cd}(x, k)$ is the Jacobian elliptic function cd with modulus k and real quarter-period K. The parameters k and k_1 are defined in Section 17.4.

We note that the definition of $F_N(w)$ for the Chebyshev-2 case is slightly different from that of Eq. (13.1.5) because of the different normalization frequency w and different definition of the ε parameter.

In all four cases, the function $F_N(w)$ is normalized such that $F_N(1) = 1$. The requirement that the bandwidth gain be equal to G_B at the frequencies $\Omega = \pm\Omega_B$ gives a condition from which the constant ε may be determined. Setting $\Omega = \Omega_B$ in Eq. (17.2.10), we obtain:

$$|H_a(\Omega_B)|^2 = \frac{G^2 + G_0^2\varepsilon^2}{1 + \varepsilon^2} = G_B^2 \quad \Leftrightarrow \quad \varepsilon = \sqrt{\frac{G^2 - G_B^2}{G_B^2 - G_0^2}} \tag{17.2.12}$$

The analog transfer function $H_a(s)$ corresponding to Eq. (17.2.10) is constructed by finding the left-hand s-plane zeros of the numerator and denominator of (17.2.10) and pairing them in conjugate pairs. By construction, $H_a(s)$, and hence the equalizer transfer function, will have minimum phase. This is a desirable property because our designs imply that the transfer function of a cut by the same amount as a boost will be the inverse of the corresponding boost transfer function. In terms of its s-plane zeros and poles, $H_a(s)$ may be written in the factored form:

$$H_a(s) = H_0\left[\frac{1 - s/z_0}{1 - s/p_0}\right]^r \prod_{i=1}^{L}\left[\frac{(1 - s/z_i)(1 - s/z_i^*)}{(1 - s/p_i)(1 - s/p_i^*)}\right] \tag{17.2.13}$$

where L is the number of analog second-order sections, related to the analog filter order by $N = 2L + r$, where $r = 0$, if N is even, and $r = 1$, if N is odd. The notation $[F]^r$ means that the factor F is present if $r = 1$ and absent if $r = 0$. The quantity H_0 is the gain at $\Omega = 0$ (and at the peak frequency $\omega = \omega_0$) and is given in terms of G or G_B as follows:

$$H_0 = \begin{cases} G, & \text{Butterworth and Chebyshev-2} \\ G^r G_B^{1-r}, & \text{Chebyshev-1 and Elliptic} \end{cases} \tag{17.2.14}$$

The zeros z_0, z_i and poles p_0, p_i are given below for all four filter types. We will use Eq. (17.2.13) for the Butterworth, Chebyshev-2, and elliptic designs. For Chebyshev type-1 designs, it is more convenient to use the following form:

$$H_a(s) = H_\infty\left[\frac{z_0 - s}{p_0 - s}\right]^r \prod_{i=1}^{L}\left[\frac{(z_i - s)(z_i^* - s)}{(p_i - s)(p_i^* - s)}\right] \tag{17.2.15}$$

where H_∞ is the gain at $\Omega = \infty$ (and at $\omega = 0$ and $\omega = \pi$) given by:

$$H_\infty = \begin{cases} G_0, & \text{Butterworth and Chebyshev-1} \\ G_0^r G_B^{1-r}, & \text{Chebyshev-2} \\ G_0^r G_s^{1-r}, & \text{Elliptic} \end{cases} \tag{17.2.16}$$

The conventional lowpass filters are obtained as special cases of Eqs. (17.2.10)–(17.2.16) in the limit $G_0 = 0, G = 1$. For realization purposes, it proves convenient to implement the transformation (17.2.2) in two stages by first transforming the analog lowpass shelving filter into a digital lowpass shelving filter using the ordinary bilinear transformation, and then transforming that into the bandpass peaking equalizer. This two-step process is expressed by writing Eq. (17.2.2) in the form [321,322]:

$$s = \frac{1-\hat{z}^{-1}}{1+\hat{z}^{-1}} = \frac{1-2c_0\, z^{-1}+z^{-2}}{1-z^{-2}} \quad \Leftrightarrow \quad \hat{z}^{-1} = \frac{z^{-1}(c_0 - z^{-1})}{1-c_0 z^{-1}} \tag{17.2.17}$$

Such transformations have been used in the design of the biquadratic equalizer [222] and bandpass and bandstop filters with variable characteristics [284,324,325].

Under the lowpass transformation from s to $\hat{z}$, the factored form of Eq. (17.2.13) results in a digital lowpass shelving filter of order N that is a cascade of first- and second-order sections in the variable $\hat{z}$. Then, the lowpass-to-bandpass transformation from $\hat{z}$ to z will yield the bandpass equalizer, centered at ω_0, as a cascade of second- and fourth-order sections in the variable z, with a net filter order of $2N$.

Thus, the designed equalizer transfer function can be expressed in terms of the variable s, or the variable $\hat{z}$, or the variable z, in the following equivalent cascaded forms:

$$\begin{aligned} H(z) &= \left[\frac{B_{00}+B_{01}s}{A_{00}+A_{01}s}\right]^r \prod_{i=1}^{L}\left[\frac{B_{i0}+B_{i1}s+B_{i2}s^2}{A_{i0}+A_{i1}s+A_{i2}s^2}\right] \\ &= \left[\frac{\hat{b}_{00}+\hat{b}_{01}\hat{z}^{-1}}{1+\hat{a}_{01}\hat{z}^{-1}}\right]^r \prod_{i=1}^{L}\left[\frac{\hat{b}_{i0}+\hat{b}_{i1}\hat{z}^{-1}+\hat{b}_{i2}\hat{z}^{-2}}{1+\hat{a}_{i1}\hat{z}^{-1}+\hat{a}_{i2}\hat{z}^{-2}}\right] \\ &= \left[\frac{b_{00}+b_{01}z^{-1}+b_{02}z^{-2}}{1+a_{01}z^{-1}+a_{02}z^{-2}}\right]^r \prod_{i=1}^{L}\left[\frac{b_{i0}+b_{i1}z^{-1}+b_{i2}z^{-2}+b_{i3}z^{-3}+b_{i4}z^{-4}}{1+a_{i1}z^{-1}+a_{i2}z^{-2}+a_{i3}z^{-3}+a_{i4}z^{-4}}\right] \end{aligned} \tag{17.2.18}$$

When $N = 1$, the r-factor is identical to the conventional biquad equalizer [283–297]. For the special cases of lowpass and highpass digital shelving filters, we have $c_0 = \pm 1$, and Eq. (17.2.17) reduces to $\hat{z}^{-1} = \pm z^{-1}$, and the quartic sections are not applicable.

The algebraic relations among the coefficients of Eq. (17.2.18) are straightforward and given below. For the s to $\hat{z}^{-1}$ transformation, the first- and second-order section coefficients are:

$$\begin{array}{ll} D_0 = A_{00}+A_{01} & D_i = A_{i0}+A_{i1}+A_{i2} \\[4pt] \hat{b}_{00} = \dfrac{B_{00}+B_{01}}{D_0} & \hat{b}_{i0} = \dfrac{B_{i0}+B_{i1}+B_{i2}}{D_i} \\[8pt] \hat{b}_{01} = \dfrac{B_{00}-B_{01}}{D_0} & \hat{b}_{i1} = \dfrac{2(B_{i0}-B_{i2})}{D_i} \\[8pt] \hat{a}_{01} = \dfrac{A_{00}-A_{01}}{D_0} & \hat{b}_{i2} = \dfrac{B_{i0}-B_{i1}+B_{i2}}{D_i} \\[8pt] & \hat{a}_{i1} = \dfrac{2(A_{i0}-A_{i2})}{D_i} \\[8pt] & \hat{a}_{i2} = \dfrac{A_{i0}-A_{i1}+A_{i2}}{D_i} \end{array} \tag{17.2.19}$$

For the $\hat{z}^{-1}$ to z^{-1} transformation, we obtain the second- and fourth-order coefficients:

$$\begin{array}{ll} b_{00} = \hat{b}_{00} & b_{i0} = \hat{b}_{i0} \\ b_{01} = c_0(\hat{b}_{01}-\hat{b}_{00}) & b_{i1} = c_0(\hat{b}_{i1}-2\hat{b}_{i0}) \\ b_{02} = -\hat{b}_{01} & b_{i2} = (\hat{b}_{i0}-\hat{b}_{i1}+\hat{b}_{i2})c_0^2-\hat{b}_{i1} \\ a_{01} = c_0(\hat{a}_{01}-1) & b_{i3} = c_0(\hat{b}_{i1}-2\hat{b}_{i2}) \\ a_{02} = -\hat{a}_{01} & b_{i4} = \hat{b}_{i2} \\ & a_{i1} = c_0(\hat{a}_{i1}-2) \\ & a_{i2} = (1-\hat{a}_{i1}+\hat{a}_{i2})c_0^2-\hat{a}_{i1} \\ & a_{i3} = c_0(\hat{a}_{i1}-2\hat{a}_{i2}) \\ & a_{i4} = \hat{a}_{i2} \end{array} \tag{17.2.20}$$

17.3 Poles and Zeros

The construction of the poles and zeros of the parametric equalizer is slightly different from the constructions of Sec. 13.5 and 13.7. The zeros and poles of the analog shelving filter $H_a(s)$ are constructed by by finding the roots of the numerator and denominator of Eq. (17.2.10), that is, solving:

$$\begin{aligned} &\text{for the zeros:} \quad G^2 + G_0^2\varepsilon^2 F_N^2(w) = 0 \\ &\text{for the poles:} \quad 1 + \varepsilon^2 F_N^2(w) = 0 \end{aligned} \tag{17.3.1}$$

or, equivalently,

$$\begin{aligned} F_N(w) &= \pm j\,\frac{G}{G_0\varepsilon} \\ F_N(w) &= \pm j\,\frac{1}{\varepsilon} \end{aligned} \tag{17.3.2}$$

For the *Butterworth case*, we have, $F_N(w) = w^N$, which leads to the following left-hand s-plane zeros and poles:

$$z_0 = -\frac{g\beta}{g_0}, \quad z_i = \frac{g\beta}{g_0}(-s_i + jc_i), \quad p_0 = -\beta, \quad p_i = \beta(-s_i + jc_i) \tag{17.3.3}$$

for $i = 1, 2, \dots, L$, where we introduced the parameters:

$$g = G^{1/N}, \quad g_0 = G_0^{1/N}, \quad \beta = \varepsilon^{-1/N}\Omega_B = \varepsilon^{-1/N}\tan\left(\frac{\Delta\omega}{2}\right) \tag{17.3.4}$$

$$s_i = \sin\phi_i\,, \quad c_i = \cos\phi_i\,, \quad \phi_i = \frac{(2i-1)\pi}{2N}\,, \quad i = 1, 2, \dots, L \tag{17.3.5}$$

Eq. (17.4.1) was obtained by multiplying out the first-order zero and pole factors and distributing the gain $H_0 = G = g^N$ over the N first-order sections of Eq. (17.2.13).

For the *type-1 Chebyshev case*, we have, $F_N(w) = C_N(w)$, and the left-hand s-plane zeros and poles are found to be:

$$\begin{aligned} z_0 &= -\Omega_B \sinh u\,, \quad z_i = j\Omega_B\cos(\phi_i - ju) = \Omega_B(-s_i\sinh u + jc_i\cosh u) \\ p_0 &= -\Omega_B \sinh v\,, \quad p_i = j\Omega_B\cos(\phi_i - jv) = \Omega_B(-s_i\sinh v + jc_i\cosh v) \end{aligned} \tag{17.3.6}$$

where $i = 1, 2, \dots, L$, and s_i, c_i, ϕ_i are the same as in Eq. (17.3.5). The quantities u, v are given by:

$$\begin{aligned} e^u &= g_0^{-1}\beta\,, \quad \beta = \left(G\varepsilon^{-1} + G_B\sqrt{1+\varepsilon^{-2}}\right)^{1/N} \\ e^v &= \alpha = \left(\varepsilon^{-1} + \sqrt{1+\varepsilon^{-2}}\right)^{1/N} \end{aligned} \tag{17.3.7}$$

where $g_0 = G_0^{1/N}$. The transfer function (17.4.6) was obtained by inserting (17.3.6) into Eq. (17.2.15) and distributing the gain, $H_\infty = G_0 = g_0^N$, over the N first-order sections.

For the *type-2 Chebyshev case*, we have, $F_N(w) = 1/C_N(1/w)$, which leads to zeros and poles that are essentially the inverses of those of Eq. (17.3.6). For $i = 1, 2, \dots, L$, we have:

$$\begin{aligned} z_0^{-1} &= -\Omega_B^{-1}\sinh u\,, \quad z_i^{-1} = j\Omega_B^{-1}\cos(\phi_i - ju) = \Omega_B^{-1}(-s_i\sinh u + jc_i\cosh u) \\ p_0^{-1} &= -\Omega_B^{-1}\sinh v\,, \quad p_i^{-1} = j\Omega_B^{-1}\cos(\phi_i - jv) = \Omega_B^{-1}(-s_i\sinh v + jc_i\cosh v) \end{aligned} \tag{17.3.8}$$

where s_i, c_i, ϕ_i are the same as in Eq. (17.3.5), and the quantities u, v are defined by:

$$\begin{aligned} e^u &= g^{-1}\beta\,, \quad \beta = \left(G_0\varepsilon + G_B\sqrt{1+\varepsilon^2}\right)^{1/N} \\ e^v &= \alpha = \left(\varepsilon + \sqrt{1+\varepsilon^2}\right)^{1/N} \end{aligned} \tag{17.3.9}$$

where $g = G^{1/N}$. Inserting these into Eq. (17.2.13) and distributing the gain, $H_0 = G = g^N$, over the N first-order sections, we obtain the analog transfer function (17.4.12).

We note that in both Chebyshev cases, the shelving zeros z_i (or their inverses in type-2) as well as the poles p_i, lie on an ellipse on the s-plane, while in the Butterworth case they lie on a circle.

In the *elliptic case*, we have, $F_N(w) = \mathrm{cd}(uK_1, k_1)$, with, $w = \mathrm{cd}(uK, k)$. Assuming initially that $G \neq 0$ and $G_0 \neq 0$, the resulting left-hand s-plane zeros and poles of $H_a(s)$ in Eq. (17.2.13) are given as follows, for $i = 1, 2, \dots, L$:

$$z_i = j\Omega_B\,\mathrm{cd}\big((u_i - ju_0)K, k\big)\,, \quad p_i = j\Omega_B\,\mathrm{cd}\big((u_i - jv_0)K, k\big) \tag{17.3.10}$$

where the $u_i = (2i-1)/N$ are the same as in Eq. (13.3.9), and u_0, v_0 are real-valued and are the solutions of the equations:

$$\mathrm{sn}(ju_0NK_1, k_1) = j\frac{G}{G_0\varepsilon}\,, \quad \mathrm{sn}(jv_0NK_1, k_1) = j\frac{1}{\varepsilon} \tag{17.3.11}$$

If N is odd, there is an additional real-valued zero and pole obtained from Eq. (17.3.10) by setting $u_i = 1$ (which corresponds to the index $i = L+1$):

$$\begin{aligned} z_0 &= j\Omega_B\,\mathrm{cd}\big((1 - ju_0)K, k\big) = j\Omega_B\,\mathrm{sn}(ju_0K, k) \\ p_0 &= j\Omega_B\,\mathrm{cd}\big((1 - jv_0)K, k\big) = j\Omega_B\,\mathrm{sn}(jv_0K, k) \end{aligned} \tag{17.3.12}$$

where we used the identity [318]: $\mathrm{cd}(K - x, k) = \mathrm{sn}(x, k)$. The evaluation of the elliptic functions **cd** and **sn** and their inverses can be carried out efficiently by means of the Landen transformation described previously in Sec. 13.4.

Working with the normalized frequency,

$$w_i = \frac{z_i}{j\Omega_B} = \mathrm{cd}\big((u_i - ju_0)K, k\big)$$

we may verify the root condition (17.3.2):

$$\begin{aligned} F_N(w_i) &= \mathrm{cd}\big((u_i - ju_0)NK_1, k_1\big) = \mathrm{cd}\big((2i-1)K_1 - ju_0NK_1, k_1\big) \\ &= (-1)^i\,\mathrm{sn}(ju_0NK_1, k_1) = \pm j\frac{G}{G_0\varepsilon} \end{aligned}$$

where we used the property [318],

$$\mathrm{cd}\big((2i-1)K_1 + x, k_1\big) = (-1)^i\,\mathrm{sn}(x, k_1)$$

which is valid for integer i. Similarly, for odd N we have

$$\mathrm{cd}\big((1 - ju_0)NK_1, k_1\big) = \mathrm{sn}(ju_0NK_1, k_1) = \frac{jG}{G_0\varepsilon}$$

The two special cases $G_0 = 0$ and $G = 0$ must be treated separately because they lead to the values $z_0 = \infty$ and $z_0 = 0$ in Eq. (17.3.12). When $G_0 = 0$, Eq. (17.2.10) implies that the zeros of $H_a(s)$ coincide with the poles of $F_N(w)$, which were defined in Eq. (13.3.9). Thus, the conjugate zeros are:

$$z_i = j\Omega_B(k\zeta_i)^{-1}\,, \quad \zeta_i = \mathrm{cd}(u_iK, k) \tag{17.3.13}$$

The same conclusion can also be drawn by noting that when $G_0 = 0$ the solution of Eq. (17.3.11) is, $ju_0NK_1 = jK_1'$, that is, it corresponds to a pole of the $\mathrm{sn}(x, k_1)$ function. But because of the degree equation, we also have, $ju_0K = jK'$, which is a pole of the $\mathrm{sn}(x, k)$ function. Therefore, $z_0 = \infty$ and the zero factor, $(1 - s/z_0)$, of $H_a(s)$ may be replaced by unity. The expression (17.3.10) for z_i reduces to (17.3.13) for this value of u_0, provided we use the identity,

$$\mathrm{cd}(x - jK', k) = \frac{1}{k\,\mathrm{cd}(x, k)}$$

When $G = 0$, the zeros of $H_a(s)$ coincide with the zeros of $F_N(w)$ given by Eq. (13.3.9), but there is an extra zero at $z_0 = 0$ for the odd-N case. The factor $(1 - s/z_0)$ of Eq. (17.2.13) must be handled as a limiting case as $G \to 0$.

Using the Taylor series expansion $\mathrm{sn}(x,k)\simeq x$, which is valid for small x, it follows that when G is small, the solution of Eq. (17.3.11) for u_0, and the zero z_0 of Eq. (17.3.12), are given approximately by:

$$ju_0NK_1 \simeq j\frac{G}{G_0\varepsilon} \;\Rightarrow\; u_0 \simeq \frac{G}{G_0\varepsilon NK_1}, \quad z_0 = j\Omega_B\,\mathrm{sn}(ju_0K,k)\simeq -\Omega_B u_0K = -\frac{\Omega_B GK}{G_0\varepsilon NK_1}$$

Because in the odd-N case the overall gain in Eq. (17.2.13) is $H_0 = G$, it follows that the first-order factor $H_0(1 - s/z_0)$ of the transfer function will have a finite limit as $G \to 0$:

$$H_0(1-s/z_0)\simeq G + G\frac{s\,G_0\varepsilon NK_1}{\Omega_B GK} \to \frac{s}{\Omega_B}G_0\varepsilon\frac{NK_1}{K}$$

Thus, in the odd-N case, the first-order numerator factor of Eq. (17.2.13) takes the following forms:

$$H_0(1-s/z_0) = \begin{cases} G(1-s/z_0), & \text{if } G_0 \neq 0, G\neq 0 \\ G, & \text{if } G_0 = 0, G \neq 0 \\ (s/\Omega_B)(G_0\varepsilon)(NK_1/K), & \text{if } G_0\neq 0, G = 0\end{cases} \tag{17.3.14}$$

For the even-N case, we have $H_0 = G_B$, per Eq. (17.2.14). Similarly, the conjugate zeros z_i, $i = 1,2,\dots,L$, are given as follows, for both even and odd N:

$$z_i = \begin{cases} j\Omega_B\,\mathrm{cd}\big((u_i - ju_0)K,k\big), & \text{if } G_0\neq 0, G\neq 0 \\ j\Omega_B\,(k\zeta_i)^{-1}, & \text{if } G_0 = 0, G\neq 0 \\ j\Omega_B\,\zeta_i, & \text{if } G_0 \neq 0, G = 0\end{cases} \tag{17.3.15}$$

The case $G_0 = 0, G\neq 0$ corresponds to the conventional designs of analog lowpass elliptic filters of Sec. 13.5. In the following sections, we present the design equations for the coefficients of Eqs. (17.2.18) in the Butterworth, the two Chebyshev, and the elliptic cases.

17.4 Butterworth, Chebyshev, and Elliptic Designs

Butterworth Designs

Using Eq. (17.3.3) for the Butterworth zeros and poles, we obtain the following expression for the analog transfer function (13.5.1) in the Butterworth case:

$$H_a(s) = \left[\frac{g\beta + g_0 s}{\beta + s}\right]^r\prod_{i=1}^{L}\left[\frac{g^2\beta^2 + 2gg_0s_i\beta s + g_0^2s^2}{\beta^2 + 2s_i\beta s + s^2}\right] \tag{17.4.1}$$

where we defined the parameters:

$$g = G^{1/N},\quad g_0 = G_0^{1/N},\quad \beta = \varepsilon^{-1/N}\Omega_B = \varepsilon^{-1/N}\tan\left(\frac{\Delta\omega}{2}\right) \tag{17.4.2}$$

$$s_i = \sin\phi_i\,,\quad \phi_i = \frac{(2i-1)\pi}{2N}\,,\quad i = 1,2,\dots,L \tag{17.4.3}$$

The parameter ε is given by Eq. (17.2.12) and Ω_B by Eq. (17.2.5). Using the coefficient transformations given above, we find the coefficients of the digital lowpass shelving filter (17.2.18):

$$\begin{aligned}
D_0 &= \beta + 1 & D_i &= \beta^2 + 2s_i\beta + 1\\
\hat{b}_{00} &= (g\beta + g_0)/D_0 & \hat{b}_{i0} &= (g^2\beta^2 + 2gg_0s_i\beta + g_0^2)/D_i\\
\hat{b}_{01} &= (g\beta - g_0)/D_0 & \hat{b}_{i1} &= 2(g^2\beta^2 - g_0^2)/D_i\\
\hat{a}_{01} &= (\beta - 1)/D_0 & \hat{b}_{i2} &= (g^2\beta^2 - 2gg_0s_i\beta + g_0^2)/D_i\\
& & \hat{a}_{i1} &= 2(\beta^2 - 1)/D_i\\
& & \hat{a}_{i2} &= (\beta^2 - 2s_i\beta + 1)/D_i
\end{aligned} \tag{17.4.4}$$

The coefficients of the second and fourth-order sections of the bandpass equalizer (17.2.18) are:

$$\begin{aligned}
D_0 &= \beta + 1 & D_i &= \beta^2 + 2s_i\beta + 1\\
b_{00} &= (g_0 + g\beta)/D_0 & b_{i0} &= (g^2\beta^2 + 2gg_0s_i\beta + g_0^2)/D_i\\
b_{01} &= -2g_0c_0/D_0 & b_{i1} &= -4c_0(g_0^2 + gg_0s_i\beta)/D_i\\
b_{02} &= (g_0 - g\beta)/D_0 & b_{i2} &= 2\big(g_0^2(1 + 2c_0^2) - g^2\beta^2\big)/D_i\\
a_{01} &= -2c_0/D_0 & b_{i3} &= -4c_0(g_0^2 - gg_0s_i\beta)/D_i\\
a_{02} &= (1 - \beta)/D_0 & b_{i4} &= (g^2\beta^2 - 2gg_0s_i\beta + g_0^2)/D_i\\
& & a_{i1} &= -4c_0(1 + s_i\beta)/D_i\\
& & a_{i2} &= 2(1 + 2c_0^2 - \beta^2)/D_i\\
& & a_{i3} &= -4c_0(1 - s_i\beta)/D_i\\
& & a_{i4} &= (\beta^2 - 2s_i\beta + 1)/D_i
\end{aligned} \tag{17.4.5}$$

When $N = 1$, we have $g = G$, $g_0 = G_0$, $\beta = \varepsilon^{-1}\tan(\Delta\omega/2)$, and the second-order section coefficients $\{b_{00}, b_{01}, b_{02}, a_{01}, a_{02}\}$ become identical to those of the conventional biquadratic equalizer, for example, in the form given in Sec. 12.4.

The $N = 2$ case corresponds to the second-order shelving filters discussed in [327,295] and used in [328] to design a fourth-order equalizer. We note also that Eqs. (17.4.1)–(17.4.5) have the proper limits in the ordinary resonator/bandpass and notch/bandstop cases $G_0 = 0, G = 1$ and $G_0 = 1, G = 0$.

Chebyshev Type-1 Designs

For the Chebyshev designs, the bandwidth $\Delta\omega$ and gain level G_B define the extent of the equiripple passband in the type-1 case, or the onset of the equiripple stopband in the type-2 case.

Therefore, for the type-1 case, G_B must be chosen to be very close to G in order to achieve a flat passband, and for the type-2 case, it must be very close to G_0 to achieve a flat stopband. These remarks are illustrated in Fig. 17.4.1. For the type-1 case, the resulting analog transfer function takes the form:

$$H_a(s)=\left[\frac{b\Omega_B+g_0s}{a\Omega_B+s}\right]^r\prod_{i=1}^{L}\left[\frac{(b^2+g_0^2c_i^2)\Omega_B^2+2g_0bs_i\Omega_Bs+g_0^2s^2}{(a^2+c_i^2)\Omega_B^2+2as_i\Omega_Bs+s^2}\right] \tag{17.4.6}$$

where we defined $g_0=G_0^{1/N}$ and:

$$b=g_0\sinh u=\frac{1}{2}(\beta-g_0^2\beta^{-1}),\quad a=\sinh v=\frac{1}{2}(\alpha-\alpha^{-1}) \tag{17.4.7}$$

$$e^u=g_0^{-1}\beta,\quad \beta=\left(G\varepsilon^{-1}+G_B\sqrt{1+\varepsilon^{-2}}\right)^{1/N},\quad e^v=\alpha=\left(\varepsilon^{-1}+\sqrt{1+\varepsilon^{-2}}\right)^{1/N} \tag{17.4.8}$$

$$s_i=\sin\phi_i,\quad c_i=\cos\phi_i,\quad \phi_i=\frac{(2i-1)\pi}{2N},\quad i=1,2,\dots,L \tag{17.4.9}$$

The choice of these parameters allows a graceful passage to the limit $G_0=0$, $G=1$, which is relevant in designing ordinary lowpass and bandpass filters. The digital lowpass shelving filter coefficients of Eq. (17.2.18) are found to be:

$$\begin{aligned}
D_0&=a\Omega_B+1 & D_i&=(a^2+c_i^2)\Omega_B^2+2as_i\Omega_B+1\\
\hat b_{00}&=(b\Omega_B+g_0)/D_0 & \hat b_{i0}&=\big((b^2+g_0^2c_i^2)\Omega_B^2+2g_0bs_i\Omega_B+g_0^2\big)/D_i\\
\hat b_{01}&=(b\Omega_B-g_0)/D_0 & \hat b_{i1}&=2\big((b^2+g_0^2c_i^2)\Omega_B^2-g_0^2\big)/D_i\\
\hat a_{01}&=(a\Omega_B-1)/D_0 & \hat b_{i2}&=\big((b^2+g_0^2c_i^2)\Omega_B^2-2g_0bs_i\Omega_B+g_0^2\big)/D_i\\
&& \hat a_{i1}&=2\big((a^2+c_i^2)\Omega_B^2-1\big)/D_i\\
&& \hat a_{i2}&=\big((a^2+c_i^2)\Omega_B^2-2as_i\Omega_B+1\big)/D_i
\end{aligned} \tag{17.4.10}$$

and using Eq. (17.2.20), we obtain the bandpass equalizer coefficients:

$$\begin{aligned}
D_0&=a\Omega_B+1 & D_i&=(a^2+c_i^2)\Omega_B^2+2as_i\Omega_B+1\\
b_{00}&=(g_0+b\Omega_B)/D_0 & b_{i0}&=\big((b^2+g_0^2c_i^2)\Omega_B^2+2g_0bs_i\Omega_B+g_0^2\big)/D_i\\
b_{01}&=-2g_0c_0/D_0 & b_{i1}&=-4c_0(g_0^2+g_0bs_i\Omega_B)/D_i\\
b_{02}&=(g_0-b\Omega_B)/D_0 & b_{i2}&=2\big(g_0^2(1+2c_0^2)-(b^2+g_0^2c_i^2)\Omega_B^2\big)/D_i\\
a_{01}&=-2c_0/D_0 & b_{i3}&=-4c_0(g_0^2-g_0bs_i\Omega_B)/D_i\\
a_{02}&=(1-a\Omega_B)/D_0 & b_{i4}&=\big((b^2+g_0^2c_i^2)\Omega_B^2-2g_0bs_i\Omega_B+g_0^2\big)/D_i\\
&& a_{i1}&=-4c_0(1+as_i\Omega_B)/D_i\\
&& a_{i2}&=2\big(1+2c_0^2-(a^2+c_i^2)\Omega_B^2\big)/D_i\\
&& a_{i3}&=-4c_0(1-as_i\Omega_B)/D_i\\
&& a_{i4}&=\big((a^2+c_i^2)\Omega_B^2-2as_i\Omega_B+1\big)/D_i
\end{aligned} \tag{17.4.11}$$

Chebyshev Type-2 Designs

In the type-2 Chebyshev case, the analog shelving filter transfer function of Eq. (13.5.1), constructed from the corresponding left-hand s-plane zeros and poles given by Eq. (17.3.8), is found to be:

$$H_a(s)=\left[\frac{g\Omega_B+bs}{\Omega_B+as}\right]^r\prod_{i=1}^{L}\left[\frac{g^2\Omega_B^2+2gbs_i\Omega_Bs+(b^2+g^2c_i^2)s^2}{\Omega_B^2+2as_i\Omega_Bs+(a^2+c_i^2)s^2}\right] \tag{17.4.12}$$

where we set $g=G^{1/N}$ and defined:

$$b=g\sinh u=\frac{1}{2}(\beta-g^2\beta^{-1}),\quad a=\sinh v=\frac{1}{2}(\alpha-\alpha^{-1}) \tag{17.4.13}$$

with s_i,c_i,ϕ_i given by Eq. (17.4.9), and the quantities u,v defined by:

$$e^u=g^{-1}\beta,\quad \beta=\left(G_0\varepsilon+G_B\sqrt{1+\varepsilon^2}\right)^{1/N},\quad e^v=\alpha=\left(\varepsilon+\sqrt{1+\varepsilon^2}\right)^{1/N} \tag{17.4.14}$$

The form of Eq. (17.4.12) facilitates the limit $G=0$, $G_0=1$, which describes ordinary notch/bandstop filters. The coefficients of the corresponding digital lowpass shelving filter are:

$$\begin{aligned}
D_0&=\Omega_B+a & D_i&=\Omega_B^2+2as_i\Omega_B+a^2+c_i^2\\
\hat b_{00}&=(g\Omega_B+b)/D_0 & \hat b_{i0}&=(g^2\Omega_B^2+2gbs_i\Omega_B+b^2+g^2c_i^2)/D_i\\
\hat b_{01}&=(g\Omega_B-b)/D_0 & \hat b_{i1}&=2(g^2\Omega_B^2-b^2-g^2c_i^2)/D_i\\
\hat a_{01}&=(\Omega_B-a)/D_0 & \hat b_{i2}&=(g^2\Omega_B^2-2gbs_i\Omega_B+b^2+g^2c_i^2)/D_i\\
&& \hat a_{i1}&=2(\Omega_B^2-a^2-c_i^2)/D_i\\
&& \hat a_{i2}&=(\Omega_B^2-2as_i\Omega_B+a^2+c_i^2)/D_i
\end{aligned} \tag{17.4.15}$$

and the coefficients of the bandpass equalizer:

$$\begin{aligned}
D_0&=\Omega_B+a & D_i&=\Omega_B^2+2as_i\Omega_B+a^2+c_i^2\\
b_{00}&=(b+g\Omega_B)/D_0 & b_{i0}&=(g^2\Omega_B^2+2gbs_i\Omega_B+b^2+g^2c_i^2)/D_i\\
b_{01}&=-2bc_0/D_0 & b_{i1}&=-4c_0(b^2+g^2c_i^2+gbs_i\Omega_B)/D_i\\
b_{02}&=(b-g\Omega_B)/D_0 & b_{i2}&=2\big((b^2+g^2c_i^2)(1+2c_0^2)-g^2\Omega_B^2\big)/D_i\\
a_{01}&=-2ac_0/D_0 & b_{i3}&=-4c_0(b^2+g^2c_i^2-gbs_i\Omega_B)/D_i\\
a_{02}&=(a-\Omega_B)/D_0 & b_{i4}&=(g^2\Omega_B^2-2gbs_i\Omega_B+b^2+g^2c_i^2)/D_i\\
&& a_{i1}&=-4c_0(a^2+c_i^2+as_i\Omega_B)/D_i\\
&& a_{i2}&=2\big((a^2+c_i^2)(1+2c_0^2)-\Omega_B^2\big)/D_i\\
&& a_{i3}&=-4c_0(a^2+s_i^2-as_i\Omega_B)/D_i\\
&& a_{i4}&=(\Omega_B^2-2as_i\Omega_B+a^2+c_i^2)/D_i
\end{aligned} \tag{17.4.16}$$

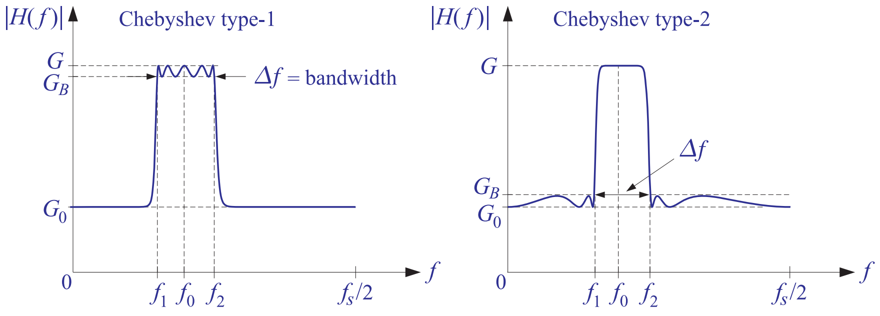


Fig. 17.4.1 Bandwidth specifications of Chebyshev type-1 and type-2 equalizers.

We note that for both Chebyshev cases, the filter order $N = 1$ corresponds to the conventional biquadratic equalizer.

Elliptic Designs

In this section we adapt the conventional elliptic filter design methods of Sec. 13.5 to the equalizer problem. We follow the notational conventions and computational algorithms of Ref. [309]. The required elliptic function moduli k, k_1 may be determined in terms of the given filter specifications by the procedure described below.

The use of the elliptic function **cd** (instead of usual **sn**) in the definition of Eq. (17.2.11) applies to both the even and odd values of the filter order N. The elliptic function moduli k, k_1 and the filter order N are required to satisfy the following degree equation:

$$N\frac{K'}{K} = \frac{K'_1}{K_1} \tag{17.4.17}$$

where K, K' and K_1, K'_1 are the quarter periods corresponding to the moduli k, k_1 and defined in terms of the complete elliptic integrals [313,317,318] by $K = K(k)$, $K' = K(k')$, $K_1 = K(k_1)$, and $K'_1 = K(k'_1)$, where k', k'_1 are the complementary moduli $k' = (1-k^2)^{1/2}$ and $k'_1 = (1-k_1^2)^{1/2}$.

A consequence of the degree equation is that $F_N(w) = \mathrm{cd}(NuK_1, k_1)$ is a rational function of $w = \mathrm{cd}(uK, k)$ given as follows (and normalized such that $F_N(1) = 1$):

$$F_N(w) = [w]^r \prod_{i=1}^{L}\left[\left(\frac{w^2 - \zeta_i^2}{1 - w^2k^2\zeta_i^2}\right)\left(\frac{1 - k^2\zeta_i^2}{1 - \zeta_i^2}\right)\right] \tag{17.4.18}$$

where $N = 2L + r$, and ζ_i and $(k\zeta_i)^{-1}$ are the zeros and poles of $F_N(w)$, where:

$$\zeta_i = \mathrm{cd}(u_iK, k)\,, \quad u_i = \frac{2i-1}{N}\,, \quad i = 1, 2, \dots, L \tag{17.4.19}$$

Because the elliptic designs are equiripple in both the passband and stopband, the specifications of the equalizer must be modified by adding a gain G_s that defines the level of the equiripple stopband. These specifications and those of the equivalent analog lowpass shelving filter are shown in Fig. 17.4.2.

The gain G_B defines the equiripple passband, which extends over the $\pm\Omega_B$ interval for the shelving filter. The equiripple stopband begins at a frequency $\Omega_s > \Omega_B$ that defines the elliptic modulus $k = \Omega_B/\Omega_s$. At the normalized frequency $w_1 = \Omega_s/\Omega_B = 1/k$, we have $F_N(w_1) = 1/k_1$. Indeed, the condition that $w_1 = \mathrm{cd}(uK, k) = 1/k$ is satisfied with $u = jK'/K$, that is, $\mathrm{cd}(uK, k) = \mathrm{cd}(jK', k) = 1/k$, which is a standard property of the cd elliptic function [317,318]. Then, the same property and the degree equation (17.4.17) imply that:

$$F_N(w_1) = \mathrm{cd}(NuK_1, k_1) = \mathrm{cd}(jNK_1K'/K, k_1) = \mathrm{cd}(jK'_1, k_1) = 1/k_1 \tag{17.4.20}$$

Using Eq. (17.4.20), the requirement that the gain be equal to G_s at $w = w_1$ gives the condition:

$$|H_a(\Omega_s)|^2 = \frac{G^2 + G_0^2\varepsilon^2/k_1^2}{1 + \varepsilon^2/k_1^2} = G_s^2 \quad \Leftrightarrow \quad k_1 = \frac{\varepsilon}{\varepsilon_s}\,, \quad \varepsilon_s = \sqrt{\frac{G^2 - G_s^2}{G_s^2 - G_0^2}} \tag{17.4.21}$$

Thus, the elliptic moduli k, k_1 are given as follows in terms of the shelving filter specifications:

$$k = \frac{\Omega_B}{\Omega_s}\,, \quad k_1 = \frac{\varepsilon}{\varepsilon_s} \tag{17.4.22}$$

Because of the degree equation, any two of the parameters N, G_s, Ω_s, or equivalently, N, k, k_1, will determine the third. For the equalizer problem, it is convenient to fix N and G_s, with G_s chosen to be very close to G_0 in order to achieve a flat stopband. Then, from the degree equation we may determine the parameter k and, hence, the value of the stopband edge frequency Ω_s.

An exact solution of the degree equation can be derived by using the property of Eq. (17.4.20). Setting $w_1 = 1/k$ and $F_N(w_1) = 1/k_1$ in Eq. (17.4.18), we obtain the formula for k_1:

$$k_1 = k^N \prod_{i=1}^{L} \mathrm{sn}^4(u_iK, k) \tag{17.4.23}$$

where we used the property: $(1-\zeta_i^2)/(1-k^2\zeta_i^2) = \mathrm{sn}^2(u_iK, k)$. Noting the invariance [312] of the degree equation under the substitutions $k \to k'_1$ and $k_1 \to k'$, we also obtain the exact solution for k in terms of N, k_1, expressed via the complementary moduli k', k'_1:

$$k' = (k'_1)^N \prod_{i=1}^{L} \mathrm{sn}^4(u_iK'_1, k'_1) \tag{17.4.24}$$

Eqs. (17.4.23) and (17.4.24), known as the "modular equations," were derived first by Jacobi in his original treatise on elliptic functions [313] and have been used since in the context of elliptic filter design [307,311,312].

The degree equation can also be solved approximately, and accurately, by working with the nomes q, q_1 corresponding to the moduli k, k_1. Exponentiating Eq. (17.4.17), we have:

$$q_1 = q^N \quad \Leftrightarrow \quad q = q_1^{1/N} \tag{17.4.25}$$

where $q = e^{-\pi K'/K}$ and $q_1 = e^{-\pi K'_1/K_1}$. Once q has been calculated from N and q_1, the

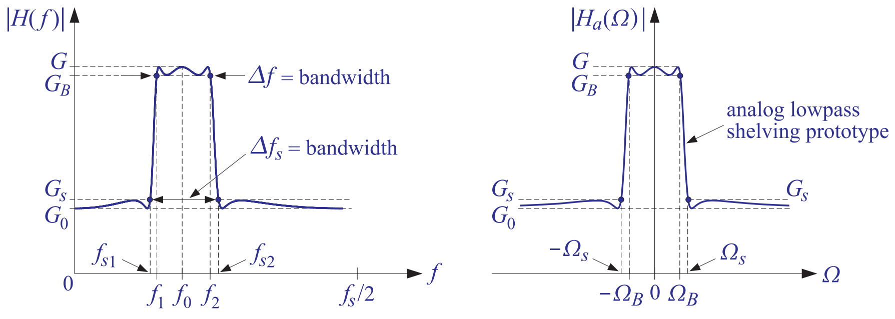


Fig. 17.4.2 Design specifications of elliptic equalizer and corresponding shelving filter.

modulus k can be determined from the series expansion [317]:

$$k = 4\sqrt{q}\left(\frac{\sum_{m=0}^{\infty} q^{m(m+1)}}{1 + 2\sum_{m=1}^{\infty} q^{m^2}}\right)^2 \tag{17.4.26}$$

which converges very fast. For example, keeping only the terms up to $m = 7$, gives a very accurate approximation.

The shelving filter transfer function $H_a(s)$ is constructed by Eq. (17.2.13), where the poles p_0, p_i are given by Eqs. (17.3.10)–(17.3.12) and the zeros z_0, z_i by Eqs. (17.3.14)–(17.3.15). The expressions for the zeros take into account the special cases $G = 1, G_0 = 0$ and $G = 0, G_0 = 1$.

Once $H_a(s)$ is determined from its zeros and poles, it may be transformed to the digital equalizer forms of Eq. (17.2.18) using the bilinear transformations. The required coefficient transformations are given by Eqs. (17.2.19) and (17.2.20). The resulting digital filter coefficients do not have any easily stated analytical form.

From the calculated value of $\Omega_s = \Omega_B/k$, the equalizer's bandwidth, $\Delta\omega_s = 2\pi\Delta f_s/f_s$, at the stopband level G_s can be derived by inverting the relationship $\Omega_s = \tan(\Delta\omega_s/2)$. The left and right stopband edge frequencies can be calculated from Eq. (17.2.8) with Ω_s replacing Ω_B:

$$\cos\omega_{s1} = \frac{c_0 + \Omega_s\sqrt{\Omega_s^2 + s_0^2}}{\Omega_s^2 + 1}\,, \quad \cos\omega_{s2} = \frac{c_0 - \Omega_s\sqrt{\Omega_s^2 + s_0^2}}{\Omega_s^2 + 1} \tag{17.4.27}$$

We note that the type-1 Chebyshev designs correspond to the limit $\Omega_s \to \infty, G_s \to G_0$, or, $k = k_1 = 0$. In this limit, the quarter periods become $K = K_1 = \pi/2$, the elliptic function cd tends to an ordinary cosine, $w = \mathrm{cd}(uK, k) = \cos(u\pi/2)$, and the function $F_N(w) = \mathrm{cd}(NuK_1, k_1) = \cos(Nu\pi/2)$ becomes equal to the Nth order Chebyshev polynomial $C_N(w)$, and the $\zeta_i = \mathrm{cd}(u_iK, k) = \cos(u_i\pi/2)$ become its roots.

17.5 Order Determination

We saw in the elliptic case that the order N and the stopband level G_s were enough to fix the rest of the design parameters, and in particular, the bandwidth Δf_s at G_s. Conversely, if Δf, Δf_s and the levels G_B, G_s are specified independently, then, the moduli k, k_1 are fixed and may not necessarily satisfy the degree equation (17.4.17) with an integer N. In this case, one may calculate:

$$N = \frac{K_1'/K_1}{K'/K} \tag{17.5.1}$$

and round it up to the next integer. Then, either k_1 needs to be recalculated from Eq. (17.4.23), or k from Eq. (17.4.24). Given that N is a decreasing function of k_1 and an increasing function of k, it follows that the resulting specifications will be slightly improved. In the first case, k_1 will slightly decrease implying that either ε decreases or ε_s increases, and hence, either G_B gets closer to G, or G_s gets closer to G_0. In the second

case, k will slightly increase, implying that Ω_s will get smaller, resulting in a narrower bandwidth Δf_s.

A similar determination of the order N can be carried out in the Butterworth and Chebyshev cases. One must specify a secondary bandwidth specification, such as Δf_s at G_s, as illustrated in Fig. 17.4.2. Defining the parameters k, k_1 exactly as in Eq. (17.4.22), and using the condition that $F_N(w_1) = 1/k_1$ at $w_1 = 1/k$, we obtain the following degree equations. For the Butterworth case:

$$k_1 = k^N \quad \Rightarrow \quad N = \frac{\ln k_1}{\ln k} \tag{17.5.2}$$

For the type-1 Chebyshev case, we have $C_N(1/k) = 1/k_1$, or,

$$\cosh\big(N\cosh^{-1}(1/k)\big) = 1/k_1 \quad \Rightarrow \quad N = \frac{\cosh^{-1}(1/k_1)}{\cosh^{-1}(1/k)} \tag{17.5.3}$$

For the type-2 Chebyshev case, because G_B was chosen to be close to G_0, the secondary bandwidth level G_s must be chosen to be very close to G, thus corresponding to a narrower bandwidth Δf_s than Δf. This implies that $k = \Omega_B/\Omega_s > 1$ and also $k_1 = \varepsilon/\varepsilon_s > 1$. The degree equation in this case is $C_N(k) = k_1$, or,

$$\cosh(N\cosh^{-1}k) = k_1 \quad \Rightarrow \quad N = \frac{\cosh^{-1}k_1}{\cosh^{-1}k} \tag{17.5.4}$$

The inequality $\varepsilon_s < \varepsilon$ for the type-2 case can be seen from the identity:

$$\varepsilon_s^2 - \varepsilon^2 = \frac{(G_B^2 - G_s^2)(G^2 - G_0^2)}{(G_s^2 - G_0^2)(G_B^2 - G_0^2)} \tag{17.5.5}$$

where for a boost, we must have $G > G_s > G_B > G_0$, and for a cut, $G < G_s < G_B < G_0$. For the Butterworth, type-1 Chebyshev, and elliptic cases, we always have $\varepsilon_s > \varepsilon$, because for a boost we must have $G > G_B > G_s > G_0$, and for a cut, $G < G_B < G_s < G_0$.

17.6 Bandwidth

The bandwidth levels G_B and G_s may be chosen arbitrarily, as long as they satisfy the basic inequalities (with the roles of G_B and G_s reversed in the Chebyshev-2 case):

$$\begin{aligned} &G > G_B > G_s > G_0 \quad \text{(boost)} \\ &G < G_B < G_s < G_0 \quad \text{(cut)} \end{aligned} \tag{17.6.1}$$

If the boost gain is more than 3 dB above the reference G_0, one may choose G_B to be 3 dB below the peak, $G_B^2 = G^2/2$, or, alternatively, 3 dB above the reference, $G_B^2 = 2G_0^2$. Other choices that respect the inequalities (17.6.1) are the geometric and arithmetic means [296]:

$$G_B^2 = GG_0\,, \quad G_B^2 = \frac{1}{2}(G^2 + G_0^2) \tag{17.6.2}$$

The corresponding values of ε defined by Eq. (17.2.12) are in these cases:

$$\varepsilon = \sqrt{\frac{G}{G_0}}\,, \quad \varepsilon = 1 \tag{17.6.3}$$

A more general definition is the weighted arithmetic mean:

$$G_B^2 = \frac{G^2 + \alpha^2 G_0^2}{1+\alpha^2} \quad \Rightarrow \quad \varepsilon = \alpha \tag{17.6.4}$$

where α is an arbitrary constant. The geometric mean choice implies that a boost and a cut by equal and opposite gains in dB will cancel exactly [292]. On the other hand, as we discuss in the next section, the weighted arithmetic mean makes possible a generalization of the Regalia-Mitra realization [288] that allows the independent control of the filter coefficients by the equalizer's center frequency f_0, bandwidth Δf, and gain G.

Regardless of the choice of G_B, and for all four filter types, it can be shown that a boost and a cut by gains G and G^{-1}, with bandwidth levels G_B and G_B^{-1}, and with the same center frequency and bandwidth, will cancel each other. Consider the boost and the cut defined by the gains:

$$\begin{aligned} &G > G_B > G_s > G_0 && \text{(boost)} \\ &G^{-1} < G_B^{-1} < G_s^{-1} < G_0^{-1} && \text{(cut)} \end{aligned} \tag{17.6.5}$$

From the definition (17.2.12), it follows that:

$$\varepsilon_{\text{cut}} = \sqrt{\frac{G^{-2} - G_B^{-2}}{G_B^{-2} - G_0^{-2}}} = \frac{G_0}{G}\sqrt{\frac{G^2 - G_B^2}{G_B^2 - G_0^2}} = \frac{G_0}{G}\varepsilon_{\text{boost}} \tag{17.6.6}$$

This implies that the root conditions (17.3.2) for the zeros and poles will exchange roles, and therefore, $z_{i,\text{cut}} = p_{i,\text{boost}}$ and $p_{i,\text{cut}} = z_{i,\text{boost}}$. The corresponding analog, and hence, the digital transfer functions will become inverses of each other:

$$H_{\text{cut}}(z) = \frac{1}{H_{\text{boost}}(z)} \tag{17.6.7}$$

The elliptic modulus k_1 remains invariant under this change because the quantity ε_s also changes in the same way as in Eq. (17.6.6), and therefore, the ratio $\varepsilon/\varepsilon_s$ remains unchanged.

The bandwidth $\Delta\omega = \omega_2 - \omega_1$ is given in linear frequency scale and enters the design equations, for all filter types, through the quantity $\Omega_B = \tan(\Delta\omega/2)$. If the bandwidth is to be specified in octaves, then it may be mapped to the linear $\Delta\omega$ in the following way.

Because the quantities $\Omega_i = \tan(\omega_i/2)$, $i = 0, 1, 2$, are related through $\Omega_1\Omega_2 = \Omega_0^2$, that is, through Eq. (17.2.5), we may set $\Omega_2 = 2^{B/2}\Omega_0$ and $\Omega_1 = 2^{-B/2}\Omega_0$, where B plays the role of an equivalent analog octave bandwidth. Using some trigonometric identities, we obtain the following expression for $\Delta\omega$ in terms of B:

$$\Omega_B = \tan\left(\frac{\Delta\omega}{2}\right) = \sin\omega_0 \sinh\left(\frac{\ln 2}{2}B\right) \tag{17.6.8}$$

The true bandwidth in octaves is defined by $b = \log_2(\omega_2/\omega_1)$, or, $2^b = \omega_2/\omega_1$. Replacing $\omega_2 = 2\arctan(\Omega_2) = 2\arctan(2^{B/2}\Omega_0)$, and similarly for ω_1, we obtain the following "bandwidth equation" relating B, b, and $\Omega_0 = \tan(\omega_0/2)$ [331]:

$$2^b = \frac{\arctan(2^{B/2}\Omega_0)}{\arctan(2^{-B/2}\Omega_0)} \tag{17.6.9}$$

In order to map the given octave bandwidth b to the linear one, one must solve Eq. (17.6.9) for B and substitute it in (17.6.8). By expanding (17.6.9) to first order in b and B, Bristow-Johnson obtained the following approximate solution [292]:

$$B = \frac{\omega_0}{\sin\omega_0}\,b \tag{17.6.10}$$

This approximation works very well for low frequencies ω_0, as well as for high ω_0 and narrow b. For any values of ω_0 and b, Eq. (17.6.9) may be solved iteratively, with Eq. (17.6.10) serving as the starting point. By rearranging (17.6.9) in the form $2^{B/2} = \Omega_0/\tan\big(2^{-b}\arctan(2^{B/2}\Omega_0)\big)$, we obtain the following convergent iteration, initialized at $B_0 = B$ given by (17.6.10):

$$2^{B_{n+1}/2} = \frac{\Omega_0}{\tan\big(2^{-b}\arctan(2^{B_n/2}\Omega_0)\big)}\,, \quad n = 0, 1, 2, \dots \tag{17.6.11}$$

At large values of ω_0 where the approximation (17.6.10) is not as good, the convergence is very fast, requiring only two or three iterations; the convergence is slow at small ω_0, but then the approximation (17.6.10) is good and there is no need for the iteration. The calculated physical bandwidth at the nth iteration may be defined through $2^{b_n} = \arctan(2^{B_n/2}\Omega_0)\,/\,\arctan(2^{-B_n/2}\Omega_0)$. The iteration error $|b_n - b|$ decreases essentially exponentially with the iteration index n. This can be seen as follows. Assuming that B_n is near the desired solution B of (17.6.9), and linearizing the recursion (17.6.11) about B, we obtain the following solution for the errors $\Delta B_n = B_n - B$:

$$\Delta B_n = \text{const}\cdot(-a)^n\,, \quad a = \frac{(2^B + \Omega_0^2)\arctan(2^{-B/2}\Omega_0)}{(1 + 2^B\Omega_0^2)\arctan(2^{B/2}\Omega_0)} \tag{17.6.12}$$

where the quantity a can be shown to be less than unity for all values of B and Ω_0, and a decreasing function of B ($a \lesssim 1$ at low ω_0, which explains the slow convergence in that case.) Thus, ΔB_n decreases exponentially, and so does the error $|b_n - b|$, since it follows $|\Delta B_n|$.

Once B has been calculated from b and ω_0, it may be used in (17.6.8) to obtain the linear bandwidth $\Delta\omega$ and, from it, the actual bandedge frequencies ω_1, ω_2 through Eqs. (17.2.8), or from $\omega_{2,1} = 2\arctan(2^{\pm B/2}\Omega_0)$. The calculated bandedge frequencies will always lie within the Nyquist interval, for all values of ω_0 and b. However, it must be emphasized that, although ω_1, ω_2 are b-octaves apart, they will not necessarily lie symmetrically at $\pm b/2$ octaves about ω_0, and may result in a very asymmetric band, especially at large ω_0s.

For the Chebyshev and elliptic cases, it may be desirable to be able to design the filters based on a more standard definition of the bandwidth, such as the 3-dB width, yet preserving the flatness of the passband and stopband controlled by the gains G_B and G_s. This issue has been discussed in [332]. In general terms, the problem is to compute the design bandwidth $\Delta\omega$ at the level G_B from a given bandwidth $\Delta\omega_b$ at an arbitrary intermediate level G_b, such that $G_0 < G_s < G_b < G_B < G$. For a given order N, the required bandwidth $\Delta\omega$ and the design parameter Ω_B can be determined from $\Delta\omega_b$ as follows:

$$\Omega_B = \tan\left(\frac{\Delta\omega}{2}\right) = \frac{1}{w_b}\tan\left(\frac{\Delta\omega_b}{2}\right) \tag{17.6.13}$$

where the normalized frequency w_b is is the solution of the equation:

$$F_N(w_b)=\frac{\varepsilon_b}{\varepsilon}\,,\quad \varepsilon_b=\sqrt{\frac{G^2-G_b^2}{G_b^2-G_0^2}} \tag{17.6.14}$$

Eq. (17.6.14) was obtained from the equivalent magnitude condition:

$$\frac{G^2+G_0^2\varepsilon^2F_N^2(w_b)}{1+\varepsilon^2F_N^2(w_b)}=G_b^2 \tag{17.6.15}$$

The solution of (17.6.14) is straightforward. For example, for the type-1 Chebyshev case, one may solve $\cosh(Nu)=\varepsilon_b/\varepsilon$ for u and then calculate $w_b=\cosh(u)$. Similarly, for the elliptic case, using the inverse of the **cd** elliptic function, one may solve $\text{cd}(NuK_1,k_1)=\varepsilon_b/\varepsilon$ for u and compute $w_b=\text{cd}(uK,k)$, where k_1 is fixed from the levels G_B and G_s, and k is calculated from N,k_1 using the degree equation. Once the bandwidth $\Delta\omega$ is determined, it may be used to complete the filter design. If the G_b-bandwidth is given in octaves, then it can be converted to linear frequency scale by applying Eqs. (17.6.8)–(17.6.11) to $\Delta\omega_b$ instead of $\Delta\omega$.

If the filter order N is not given, but rather both $\Delta\omega,\Delta\omega_b$ at the levels G_B,G_b are given, then, the quantities ε_b and $w_b=\tan(\Delta\omega_b/2)/\tan(\Delta\omega/2)$ are fixed and the filter order may be determined by solving Eq. (17.6.14) for N as in Sect. 17.5. This is straightforward for the Butterworth and Chebyshev cases.

The elliptic case is a bit more difficult because the third level G_s must also be fixed independently. The following trial-and-error approach works well: for each successive filter order $N=1,2,\dots$, calculate k from N,k_1 using the degree equation, then solve $\text{cd}(NuK_1,k_1)=\varepsilon_b/\varepsilon$ for u, calculate the error $e=\text{cd}(uK,k)-w_b$, and keep the first N for which e becomes negative (it always starts from positive values provided that $w_b<\varepsilon_b/\varepsilon$, which is easily met for practical specifications.)

If the filter is designed with the computed N and the given $\Delta\omega$, then the resulting G_b-width will be slightly narrower than $\Delta\omega_b$. If the width $\Delta\omega_b$ is to be matched exactly, the resulting $\Delta\omega$, obtained from Eqs. (17.6.13)–(17.6.14) using the computed N, will be slightly wider than specified.

17.7 Realizations

The digital equalizer transfer function $H(z)$ given by Eq. (17.2.18) may be realized as the cascade of the fourth-order sections in (17.2.18), or alternatively, as the cascade of the frequency-shifted second-order lowpass shelving filter sections in (17.2.18), in which each unit delay $\hat{z}^{-1}$ is replaced by the lowpass to bandpass transformation of Eq. (17.2.17), that is,

$$s=\frac{1-\hat{z}^{-1}}{1+\hat{z}^{-1}}=\frac{1-2c_0z^{-1}+z^{-2}}{1-z^{-2}}\quad\Leftrightarrow\quad \boxed{\hat{z}^{-1}=\frac{z^{-1}(c_0-z^{-1})}{1-c_0z^{-1}}}$$

We consider briefly the frequency-shifted versions of the transposed, normalized-lattice [338–340], and minimum-noise state-space [342–346] realizations of Eq. (17.2.18).

The latter two are known to have excellent numerical properties, at the expense of effectively doubling the number of filter coefficients. Such realizations may be appropriate under stringent filter specifications, such as very low center frequencies or rapidly-varying equalizer parameters [340].

The *normalized lattice* realization of the bandpass transformation (17.2.17) is shown in Fig. 17.7.1. It has the expected limit (without requiring any pole/zero cancellations) in the lowpass and highpass shelving filter cases $\omega_0=0$ and $\omega_0=\pi$. Other realizations of (17.2.17) are, of course, possible that require fewer operations, such as, for example, the one-multiplier form of [222,325]. However, they lack the scaling and L_2-normalization properties of the normalized lattice.

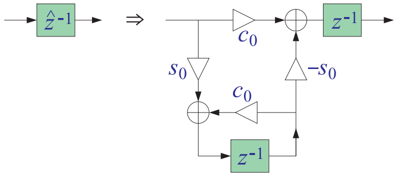


Fig. 17.7.1 Normalized lattice realization of $\hat{z}^{-1}=\dfrac{z^{-1}(c_0-z^{-1})}{1-c_0z^{-1}}$.

The *transposed* (of the direct-form II) realization of the second-order sections of Eq. (17.2.18), after each delay $\hat{z}^{-1}$ has been replaced by Fig. 17.7.1, is shown in Fig. 17.7.2. The figure represents the transfer function:

$$\frac{B(\hat{z})}{A(\hat{z})}=\frac{\hat{b}_0+\hat{b}_1\hat{z}^{-1}+\hat{b}_2\hat{z}^{-2}}{1+\hat{a}_1\hat{z}^{-1}+\hat{a}_2\hat{z}^{-2}}\,,\quad \hat{z}^{-1}=\frac{(c_0-z^{-1})z^{-1}}{1-c_0z^{-1}} \tag{17.7.1}$$

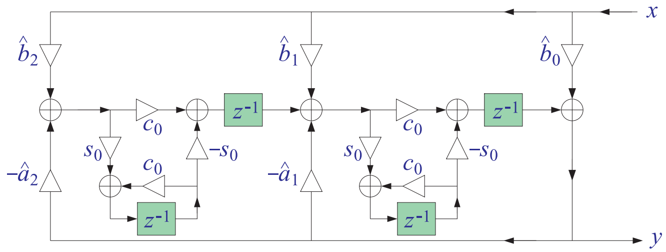


Fig. 17.7.2 Frequency-shifted transposed realization of the filter sections of Eq. (17.2.18).

The *normalized lattice* realization of Eq. (17.7.1) is shown in Fig. 17.7.3. The reflection and transmission coefficients are constructed as follows [338–340]:

$$\gamma_1=\frac{\hat{a}_1}{1+\hat{a}_2}\,,\quad \gamma_2=\hat{a}_2\,,\quad \tau_1=\sqrt{1-\gamma_1^2}\,,\quad \tau_2=\sqrt{1-\gamma_2^2} \tag{17.7.2}$$

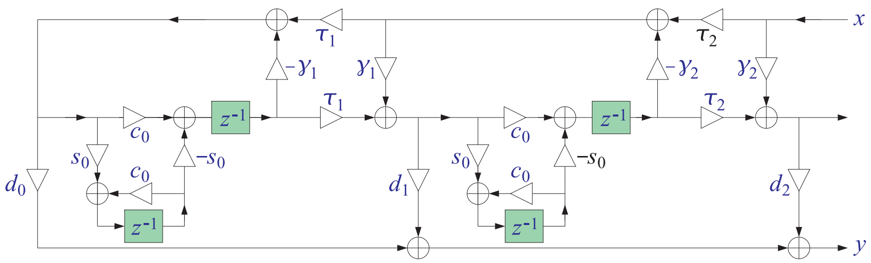


Fig. 17.7.3 Frequency-shifted normalized lattice realization of the sections of Eq. (17.2.18).

The ladder coefficients d_0, d_1, d_2 are the solutions of the triangular system:

$$\begin{bmatrix} 1 & \gamma_1 & \hat{a}_2 \\ 0 & 1 & \hat{a}_1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d_0\tau_1\tau_2 \\ d_1\tau_2 \\ d_2 \end{bmatrix} = \begin{bmatrix} \hat{b}_0 \\ \hat{b}_1 \\ \hat{b}_2 \end{bmatrix} \tag{17.7.3}$$

The first-order factor of Eq. (17.2.18) is obtained by setting $\hat{a}_2 = \hat{b}_2 = 0$, or equivalently, $\gamma_2 = 0$, $d_2 = 0$, and $\tau_2 = 1$, which amounts to deleting the γ_2 lattice section.

Computationally, Eqs. (17.7.2) and (17.7.3) are simple to use. Explicit expressions for the lattice filter parameters can be given in the Butterworth and Chebyshev cases. For example, for the first-order Butterworth factor of Eq. (17.2.18), we find:

$$\gamma_1 = \frac{\beta - 1}{\beta + 1}, \quad d_0 = \frac{\sqrt{\beta}(g + g_0)}{\beta + 1}, \quad d_1 = \frac{g\beta - g_0}{\beta + 1} \tag{17.7.4}$$

Eq. (17.7.4) can also be used to implement the conventional *biquadratic* equalizer in its lattice form. For the ith second-order Butterworth factor of (17.2.18), we have:

$$\gamma_1 = \frac{\beta^2 - 1}{\beta^2 + 1}, \quad \gamma_2 = \frac{\beta^2 - 2s_i\beta + 1}{\beta^2 + 2s_i\beta + 1} \tag{17.7.5}$$

$$\begin{aligned}
d_0 &= \frac{\sqrt{\beta}(g + g_0)\big[(1 - \beta^2)(g - g_0) + 2s_i\beta(g + g_0)\big]}{(\beta^2 + 2s_i\beta + 1)\sqrt{2s_i(\beta^2 + 1)}} \\
d_1 &= \frac{\sqrt{2\beta}(g + g_0)\big[g\beta(1 + \beta s_i) - g_0(\beta + s_i)\big]}{(\beta^2 + 2s_i\beta + 1)\sqrt{s_i(\beta^2 + 1)}} \\
d_2 &= \frac{g^2\beta^2 - 2gg_0s_i\beta + g_0^2}{\beta^2 + 2s_i\beta + 1}
\end{aligned} \tag{17.7.6}$$

Optimum state-space realizations that have minimum roundoff noise under fixed-point arithmetic, and under an L_2-scaling constraint for the internal states, are well-known [342–344]. Explicit design equations for the case of second-order sections with complex-conjugate poles have been given by Barnes [345] and Bomar [346].

One may use such optimum realizations for each second-order section of Eq. (17.2.18), and replace the delays $\hat{z}^{-1}$ by Fig. 17.7.1. The resulting state-space realization of Eq. (17.7.1) is shown in Fig. 17.7.4, where the indicated state vectors **s** and **w** are two-dimensional. The corresponding time-domain description is:

$$\begin{aligned}
y(n) &= C\mathbf{s}(n) + Dx(n) \\
\mathbf{s}(n+1) &= c_0\big[A\mathbf{s}(n) + Bx(n)\big] - s_0\mathbf{w}(n) \\
\mathbf{w}(n+1) &= s_0\big[A\mathbf{s}(n) + Bx(n)\big] + c_0\mathbf{w}(n)
\end{aligned} \tag{17.7.7}$$

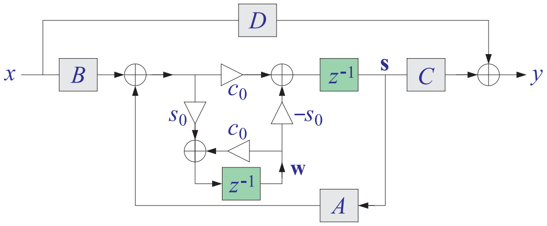
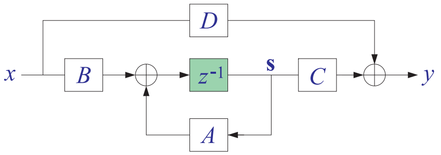


Fig. 17.7.4 Frequency-shifted state-space realization of the filter sections of Eq. (17.2.18).

The *ABCD* parameters were given in Sec. 7.4 and are repeated in Appendix-1 of this section. For the odd-N case, the first-order factor in $\hat{z}^{-1}$ is described by Eq. (17.7.7) with one-dimensional *ABCD* parameters.

For the special case of the ordinary *biquadratic equalizer* ($N = 1$), the first-order factor in $\hat{z}^{-1}$ may be regarded as a second-order factor in z^{-1} and realized directly in its optimum second-order state-space form. We have derived the following explicit form of the optimum second-order state-space parameters in this case:



$$\boxed{\begin{aligned} y(n) &= C\mathbf{s}(n) + Dx(n) \\ \mathbf{s}(n+1) &= A\mathbf{s}(n) + Bx(n) \end{aligned}} \tag{17.7.8}$$

where,

$$\begin{aligned}
A &= \frac{1}{1+\beta}\begin{bmatrix} c_0 & \beta + s_0 \\ \beta - s_0 & c_0 \end{bmatrix}, \\
B &= \frac{\sqrt{2\beta}}{1+\beta}\begin{bmatrix} \sqrt{1 - s_0} \\ -\sigma\sqrt{1 + s_0} \end{bmatrix} \\
C &= \frac{\sqrt{2\beta}(G - G_0)}{2(1+\beta)}\big[\sigma\sqrt{1 + s_0},\, -\sqrt{1 - s_0}\big], \\
D &= \frac{G_0 + G\beta}{1 + \beta}
\end{aligned} \tag{17.7.9}$$

where, $\beta = \varepsilon^{-1}\tan(\Delta\omega/2)$, and, $\sigma = \mathrm{sign}(c_0)$, with the unusual convention that $\mathrm{sign}(0) = 1$ The corresponding first-order lowpass and highpass shelving filters obtained in the limits $\omega_0 = 0$ and $\omega_0 = \pi$ are described by the one-dimensional state-space parameters:

$$A = \pm\frac{1-\beta}{1+\beta}\,, \quad B = \frac{2\sqrt{\beta}}{1+\beta}\,, \quad C = \pm\frac{\sqrt{\beta}(G-G_0)}{1+\beta}\,, \quad D = \frac{G_0+G\beta}{1+\beta} \tag{17.7.10}$$

17.8 Decoupled Realizations

The realizations shown in Figs. 17.7.2–17.7.4 are partially decoupled in the sense that the dependence on the center frequency ω_0 resides only in the multipliers c_0, s_0, whereas the dependence on the bandwidth and gain resides in the other coefficients.

For the Butterworth and the two Chebyshev cases, it is possible to generalize the Regalia-Mitra realizations [288] in which the dependence on the bandwidth, gain, and center frequency is completely decoupled into separate filter coefficients.

Such decoupling is possible [296] only if the bandwidth level is defined according the weighted arithmetic mean of Eq. (17.6.4). Then, the constant $\varepsilon = \alpha$ is independent of the peak gain G and hence the parameter β of Eq. (17.4.2) depends only on the bandwidth $\Delta\omega$.

For the first-order factor of Eq. (17.2.18), the decoupled realization is obtained by a rearrangement of the first-order normalized lattice filter as shown in Fig. 17.8.1, where the coefficients d_0, d_1 are the solutions of the system:

$$\begin{bmatrix} 1 & \hat{a}_1 \\ \hat{a}_1 & 1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \end{bmatrix} = \begin{bmatrix} \hat{b}_0 \\ \hat{b}_1 \end{bmatrix} \tag{17.8.1}$$

Eq. (17.8.1) is equivalent to expanding the corresponding first-order transfer function in the form:

$$\frac{B(\hat{z})}{A(\hat{z})} = \frac{\hat{b}_0 + \hat{b}_1\hat{z}^{-1}}{1+\hat{a}_1\hat{z}^{-1}} = d_0 + d_1\frac{A^R(\hat{z})}{A(\hat{z})} \tag{17.8.2}$$

where $A^R(\hat{z})$ is the reverse of the polynomial $A(\hat{z})$. For the Butterworth first-order filter coefficients given by Eq. (17.4.4), we find:

$$\gamma_1 = \frac{\beta-1}{\beta+1}\,, \quad d_0 = \frac{g+g_0}{2}\,, \quad d_1 = \frac{g-g_0}{2} \tag{17.8.3}$$

Similarly, for the type-1 Chebyshev case, we have:

$$\gamma_1 = \frac{a\Omega_B - 1}{a\Omega_B + 1}\,, \quad d_0 = \frac{1}{2}\left(\frac{b}{a} + g_0\right)\,, \quad d_1 = \frac{1}{2}\left(\frac{b}{a} - g_0\right) \tag{17.8.4}$$

Thus, the coefficients d_0, d_1 depend only on the gain, and the coefficients γ_1, τ_1, only on the bandwidth. The second-order factors in Eq. (17.2.18) also admit a decoupled realization, but at the expense of doubling the number of delays. Fig. 17.8.2 shows this realization, where the coefficients d_0, d_1, d_2 are the solutions of the system:

$$\begin{bmatrix} 1 & \hat{a}_2 & 1 \\ \hat{a}_1 & \hat{a}_1 & 2 \\ \hat{a}_2 & 1 & 1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2\tau_1\tau_2 \end{bmatrix} = \begin{bmatrix} \hat{b}_0 \\ \hat{b}_1 \\ \hat{b}_2 \end{bmatrix} \tag{17.8.5}$$

This is equivalent to expanding the second-order transfer function (17.7.1) in the form:

$$\frac{B(\hat{z})}{A(\hat{z})} = d_0 + d_1\frac{A^R(\hat{z})}{A(\hat{z})} + d_2\frac{\tau_1\tau_2(1+\hat{z}^{-1})^2}{A(\hat{z})} \tag{17.8.6}$$

For the ith second-order Butterworth factors of Eq. (17.4.4), we have:

$$d_0 = \frac{1}{2}g_0(g_0+g)\,, \quad d_1 = \frac{1}{2}g_0(g_0-g)\,, \quad d_2 = (g^2-g_0^2)\sqrt{\frac{\beta(\beta^2+1)}{32s_i}} \tag{17.8.7}$$

with the reflection coefficients given by Eq. (17.7.5). For the type-1 Chebyshev case, we have:

$$\gamma_1 = \frac{(a^2+c_i^2)\Omega_B^2 - 1}{(a^2+c_i^2)\Omega_B^2 + 1}\,, \quad \gamma_2 = \frac{(a^2+c_i^2)\Omega_B^2 - 2as_i\Omega_B + 1}{(a^2+c_i^2)\Omega_B^2 + 2as_i\Omega_B + 1}$$

$$d_0 = \frac{1}{2}g_0\left(g_0 + \frac{b}{a}\right)\,, \quad d_1 = \frac{1}{2}g_0\left(g_0 - \frac{b}{a}\right)\,, \quad d_2 = (b^2 - g_0^2a^2)\sqrt{\frac{((a^2+c_i^2)\Omega_B^2+1)\Omega_B}{32s_ia(a^2+c_i^2)}}$$

Replacing the unit delays $\hat{z}^{-1}$ by Fig. 17.7.1, and splitting the multiplier d_2 into two factors, one depending on g and the other on Ω_B, we obtain a realization of the equalizer that allows the independent control of the gain, bandwidth, and center frequency. In the Chebyshev cases, one must choose $\alpha \ll 1$ for type-1 and $\alpha \gg 1$ for type-2 in Eq. (17.6.4), in order to achieve flat passbands and stopbands, respectively.

The main limitation of such decoupled realizations is the restrictive definition of the bandwidth level G_B. Thus, the transposed, normalized-lattice, and state-space realizations are more flexible.

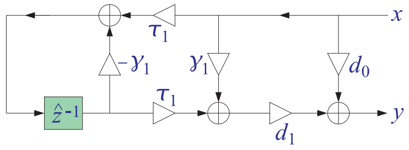


Fig. 17.8.1 Decoupled realization of the first-order factor of Eq. (17.2.18).

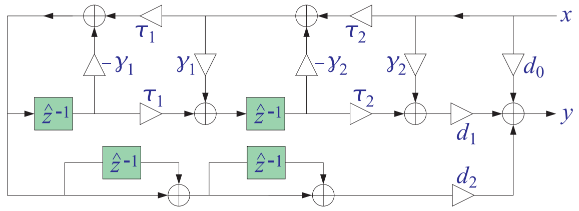


Fig. 17.8.2 Decoupled realization of the second-order factors of Eq. (17.2.18).

17.9 Design Examples

Figures 17.9.1–17.9.4 show the magnitude response of the cascade of four equalizer filters: a lowpass shelf, a boost, a cut, and a highpass shelf, designed according to the four filter types with analog filter orders of $N = 4$ and $N = 5$. The center frequencies, bandwidths, and boost gains (relative to $G_0 = 0$ dB) were taken to be:

$$\begin{array}{lll} f_1 = 0\text{ kHz}\,, & \Delta f_1 = 1\text{ kHz}\,, & G_1 = 9\text{ dB} \\ f_2 = 4\text{ kHz}\,, & \Delta f_2 = 2\text{ kHz}\,, & G_2 = 12\text{ dB} \\ f_3 = 9\text{ kHz}\,, & \Delta f_3 = 2\text{ kHz}\,, & G_3 = -6\text{ dB} \\ f_4 = 20\text{ kHz}\,, & \Delta f_4 = 4\text{ kHz}\,, & G_4 = 6\text{ dB} \end{array} \tag{17.9.1}$$

where all gains must be converted from dB to absolute units before used in the design equations. The sampling rate was 40 kHz.

For the Butterworth case, shown in Fig. 17.9.1, the bandwidth gains were chosen to be 3 dB below (or above, for the cut case) the peak gains, that is:

$$G_{B1} = 6\text{ dB}\,,\quad G_{B2} = 9\text{ dB}\,,\quad G_{B3} = -3\text{ dB}\,,\quad G_{B4} = 3\text{ dB} \tag{17.9.2}$$

The bullet dots on the graphs show the center and bandedge frequencies computed by Eq. (17.2.8). The conventional biquad equalizers (first order for the shelves), designed with the same specifications, are also shown in Fig. 17.9.1, both individually (dotted lines) and as their overall cascaded response (dashed line). Because of the slow rolloffs of the individual sections, the biquad cascaded response no longer meets the required specifications.

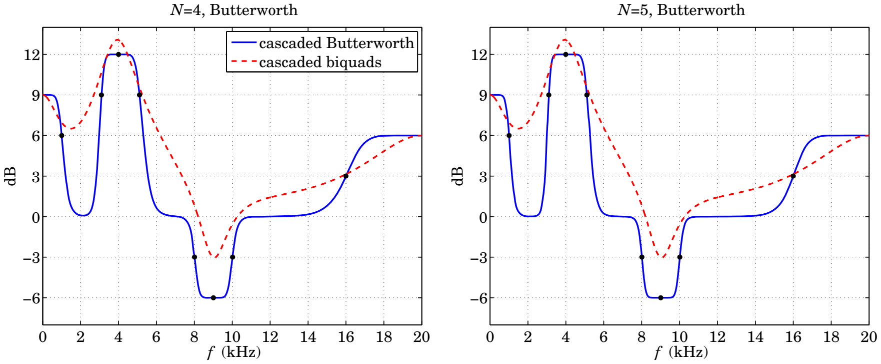


Fig. 17.9.1 Butterworth designs.

For the type-1 Chebyshev cases, shown in Fig. 17.9.2, we have kept the same center frequencies, bandwidths, and peak gains, but in order to achieve flat passbands, we have chosen the bandwidth gains to be 0.01 dB below the peak gains, that is,

$$G_{B1} = 8.99\text{ dB}\,,\quad G_{B2} = 11.99\text{ dB}\,,\quad G_{B3} = -5.99\text{ dB}\,,\quad G_{B4} = 5.99\text{ dB} \tag{17.9.3}$$

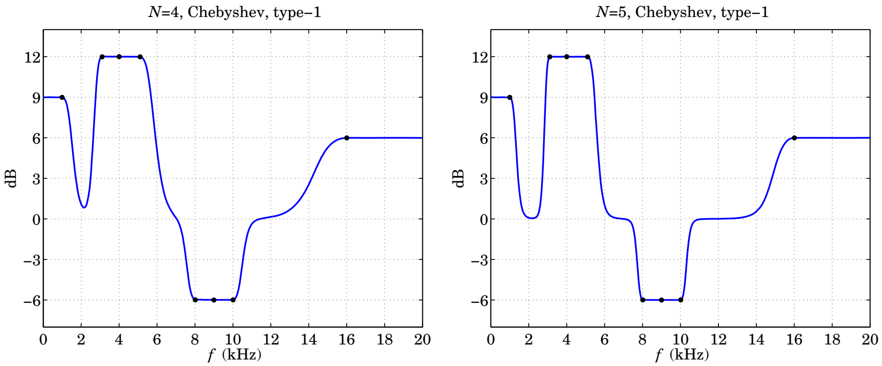


Fig. 17.9.2 Chebyshev type-1 designs.

As a result, the passbands are very flat, but the only way to flatten the stopbands is to increase the rolloff rate by choosing larger values of the filter order N. For the type-2 Chebyshev cases, shown in Fig. 17.9.3, in order to achieve flat stopbands, we have chosen the bandwidth gains to be 0.01 dB above the 0-dB reference gain G_0:

$$G_{B1} = 0.01\text{ dB}\,,\quad G_{B2} = 0.01\text{ dB}\,,\quad G_{B3} = -0.01\text{ dB}\,,\quad G_{B4} = 0.01\text{ dB} \tag{17.9.4}$$

The bandwidths near the reference gain line have the assumed values, but the equalizer peaks or cuts become narrower, with their width increasing with the filter order N. Fig. 17.9.4 shows the elliptic case, in which both the passband and stopband bandwidth gains were chosen to be 0.01 dB below the peaks and reference, that is, denoting the stopband gains by G_s:

$$\begin{array}{llll} G_{B1} = 8.99\text{ dB}\,, & G_{B2} = 11.99\text{ dB}\,, & G_{B3} = -5.99\text{ dB}\,, & G_{B4} = 5.99\text{ dB} \\ G_{s1} = 0.01\text{ dB}\,, & G_{s2} = 0.01\text{ dB}\,, & G_{s3} = -0.01\text{ dB}\,, & G_{s4} = 0.01\text{ dB} \end{array} \tag{17.9.5}$$

The elliptic case combines the benefits of the type-1 and type-2 Chebyshev cases and achieves both flat passbands and stopbands.

To assess the accuracy of the elliptic function computations using a fixed number of Landen iterations, we have computed the percentage error in the overall cascaded frequency response and found that it is less than 0.1 percent if four iterations are used and less than 10^{-5} percent for five iterations, as compared to the case of maximum precision in which the tolerance was defined to be the machine epsilon. Thus, fixing the number of Landen iterations to five makes the implementation of the elliptic case only slightly more complicated than the Chebyshev cases.

Fig. 17.9.5 shows the same example, but redesigned so that the Chebyshev and elliptic cases have the same 3-dB widths as the Butterworth case. The bandwidths listed in Eq. (17.9.1) were taken to represent the 3-dB widths relative to the peak gains, that is, corresponding to the levels:

$$G_{b1} = 6\text{ dB}\,,\quad G_{b2} = 9\text{ dB}\,,\quad G_{b3} = -3\text{ dB}\,,\quad G_{b4} = 3\text{ dB} \tag{17.9.6}$$

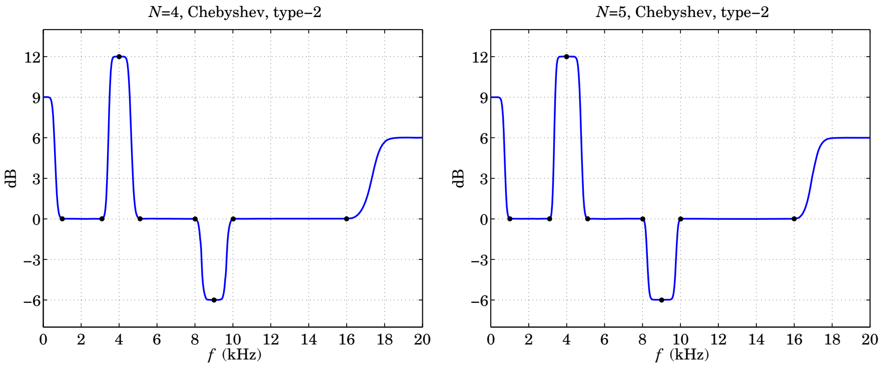


Fig. 17.9.3 Chebyshev type-2 designs.

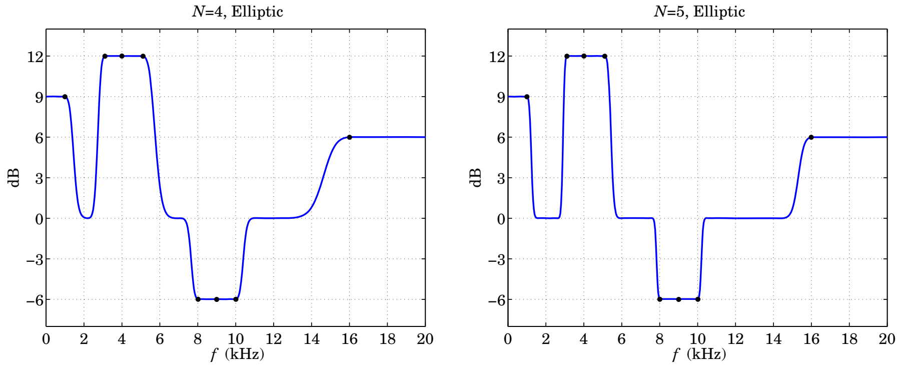


Fig. 17.9.4 Elliptic designs.

The gains G_B and G_s were still defined by Eqs. (17.9.3)–(17.9.5). The 3-dB widths were remapped to the bandwidths at the G_B design levels using Eqs. (17.6.13)–(17.6.14). The analog filter order was $N = 4$.

We have compared also the performance of the canonical (direct-form-II), transposed, normalized lattice, and state-space realizations under some extreme filter settings with rapidly changing parameters. We used the same benchmark example discussed by Moorer [340], and applied Butterworth, Chebyshev, and elliptic equalizers of orders $N = 1$–10 designed with the following gain specifications:

Butterworth:	$G = 18$ dB,	$G_B = 15$ dB	
Chebyshev-1:	$G = 18$ dB,	$G_B = 17.99$ dB	
Chebyshev-2:	$G = 18$ dB,	$G_B = 0.01$ dB	
Elliptic:	$G = 18$ dB,	$G_B = 17.99$ dB,	$G_s = 0.01$ dB

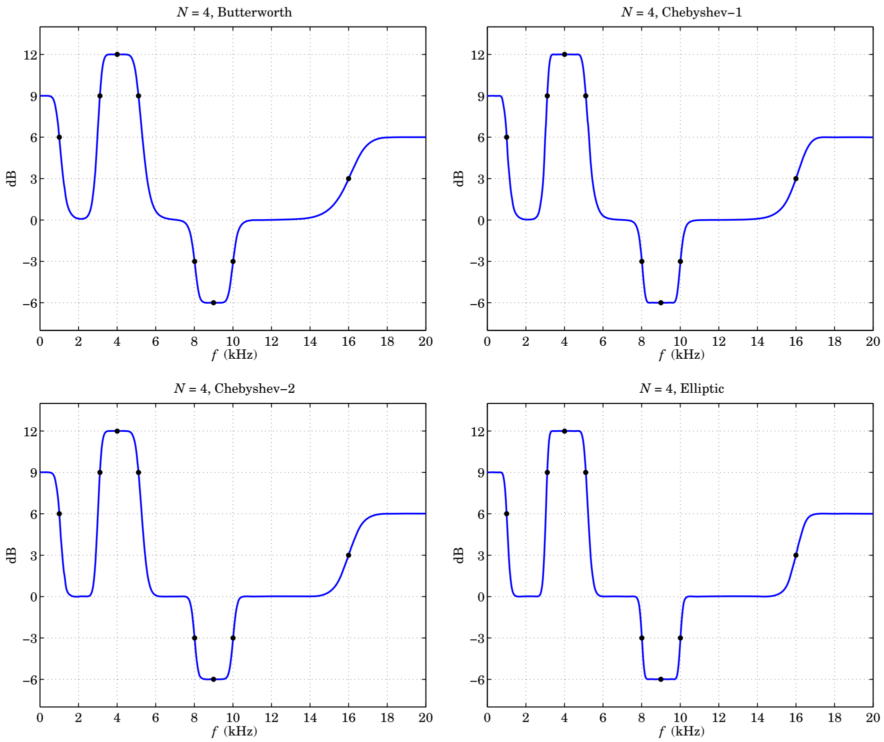


Fig. 17.9.5 Designs with common 3-dB widths.

For the first 1000 time samples, the center frequency and bandwidth (at level G_B) were fixed at 44.1 Hz and 22.05 Hz, respectively; for the next 2000 samples, the center frequency was ramped up linearly to 441 Hz and the bandwidth to 220.5 Hz; and for the last 1000 samples they were kept fixed at 441 Hz and 220.5 Hz, respectively. The sampling rate was 44.1 kHz.

The input was a 4000-long vector of uniform random numbers in the range $[0, 1)$. The resulting outputs from the four realizations are shown in Fig. 17.9.6 for the elliptic design with $N = 5$. The graphs on the right column of the figure show the responses to a step-input of amplitude equal to 0.5, which corresponds to the mean of the random input.

The transposed realization was implemented as the cascade of the realizations of Fig. 17.7.2. The canonical realization was the transposed of Fig. 17.7.2. The normalized-lattice and the state-space realizations were implemented by cascading the realizations of Figs. 17.7.3 and 17.7.4, respectively.

We observe that the transposed, lattice, and state-space realizations yield comparable results. The outputs differ only during the middle period when the filter is time-varying and the realizations are not equivalent. Consistent with Moorer's observations [340], the normalized lattice output is visually indistinguishable from that of the state-space case—the two output signals differing by less that 0.2 percent. As expected, the canonical form suffers from larger oscillations during the middle period due to its unscaled internal states.

The results from the Butterworth and Chebyshev designs and for orders $N = 1$–10 were comparable to those of Fig. 17.9.6. We have also varied the sampling rate up to 96 kHz and/or lowered all center frequencies with similar results. We looked specifically at the case when the initial center frequency was set to zero (a shelving filter) for the first 1000 samples. The canonical realization tended to deteriorate as the center frequencies got lower, resulting in large oscillations during the middle period; but the other realizations remained robust.

We also studied the performance of the transposed realizations of the fourth-order factors of Eq. (17.2.18) and found that they were mostly well-behaved, but deteriorated at zero center frequencies, in fact, becoming unstable due to coefficient roundoff errors that pushed some of the poles outside the unit circle.

As a final example, we considered the behavior of the different realizations as the equalizer was being turned on and its gain and bandwidth were time-varying. The equalizer had a fixed center frequency of 400 Hz. The sampling rate was 44.1 kHz. The input was a 3000-sample long unit-amplitude sinusoid of frequency of 400 Hz. For the first

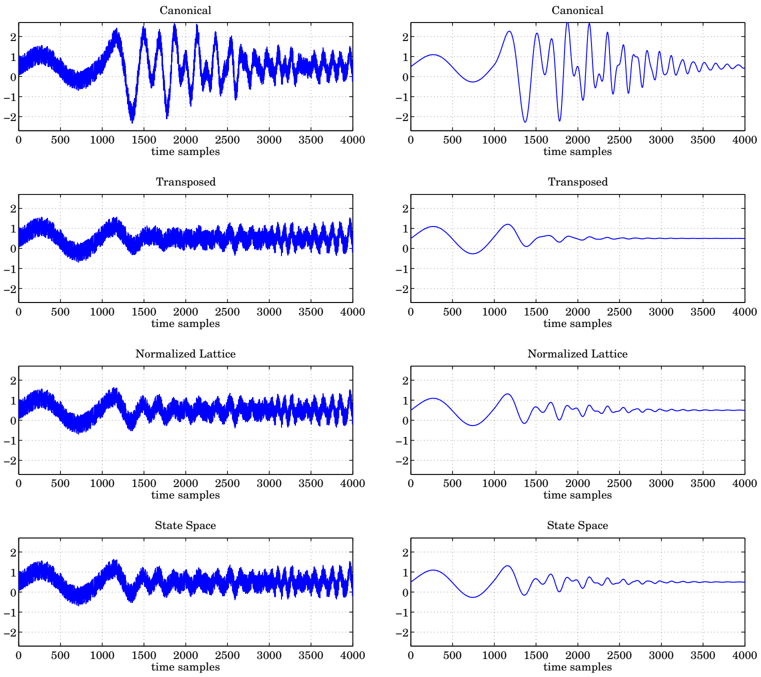


Fig. 17.9.6 Response of equalizer with time-varying center frequency and bandwidth.

1000 samples, the equalizer was off; for the next 1000 samples, it was turned on with its bandwidth changing linearly from 20 Hz to 100 Hz and its peak gain changing from 0 dB to 18 dB; for the last 1000 samples, the bandwidth was fixed at 100 Hz and the gain at 18 dB.

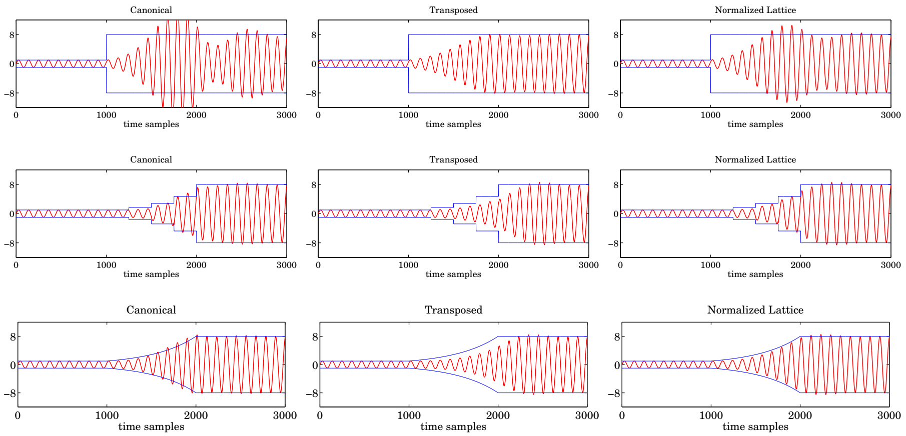


Fig. 17.9.7 Sinusoidal response of equalizer with time-varying gain.

Fig. 17.9.7 shows the outputs from the canonical, transposed, and normalized lattice realizations of an elliptic design with $N = 5$ and stopband level $G_s = 0.01$ dB. The bandwidth level G_B was taken to be 0.01 dB below the peak gain G (when these definitions could not be made because G was too small, we defined $G_B = \sqrt{G}$, and $G_s = \sqrt{G_B}$.) The state-space realization is not shown as it always produced virtually the same output as the lattice.

In the first row of graphs, the gain was switched on instantaneously to 18 dB (G=8 in absolute units); in the second row, it was turned on gradually in four steps that were linearly spaced in dB between 0 and 18 dB; and in the third row, the gain was increased continuously, varying linearly in dB. The gain curves have been superimposed on the graphs. Similar results were observed in the Butterworth and Chebyshev cases.

The gradual turning on of the equalizer [333–336] had the beneficial effect of eliminating undesirable overshoots (even two intermediate steps had the same positive effect.) Although the transposed realization was somewhat more sluggish in following the changing gain than the lattice, its lower computational cost and good numerical behavior make it a good choice for the implementation of high-order equalizers.

We also carried out the experiments of Figs. 17.9.6 and 17.9.7 using the decoupled realizations of Figs. 17.8.1 and 17.8.2 and found that they had virtually identical performance as the normalized lattice.

17.10 Appendix-1 State-Space Realizations

Given the numerator and denominator coefficients of a second-order transfer function of the form of Eq. (17.7.1) with complex-conjugate poles, the optimum minimum roundoff-error state-space realization is constructed by the following steps [345]:

$$
\begin{aligned}
&\sigma = -\frac{\hat{a}_1}{2}\,, \quad \omega = \sqrt{\hat{a}_2 - \frac{\hat{a}_1^2}{4}}\,, \quad p = \sigma + j\omega \\
&q_1 = \hat{b}_1 - \hat{b}_0\hat{a}_1\,, \quad q_2 = \hat{b}_2 - \hat{b}_0\hat{a}_2 \\
&\alpha_r = \frac{q_1}{2}\,, \quad \alpha_i = -\frac{q_1\sigma + q_2}{2\omega}\,, \quad \alpha = \alpha_r + j\alpha_i \\
&P = \frac{|\alpha|}{1 - |p|^2}\,, \quad Q = \mathrm{Im}\left[\frac{\alpha}{1 - p^2}\right], \quad k = \sqrt{\frac{P+Q}{P-Q}} \\
&B_1 = \sqrt{\frac{|\alpha| - \alpha_i}{P - Q}}\,, \quad B_2 = -\mathrm{sign}(\alpha_r)\sqrt{\frac{|\alpha| + \alpha_i}{P + Q}} \\
&C_1 = \frac{\alpha_r}{B_1}\,, \quad C_2 = \frac{\alpha_r}{B_2}
\end{aligned}
\tag{17.10.1}
$$

which define the $ABCD$ state-space parameters:

$$
A = \begin{bmatrix} \sigma & \omega k \\ -\omega/k & \sigma \end{bmatrix}, \quad B = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}, \quad C = [C_1, C_2]\,, \quad D = \hat{b}_0
\tag{17.10.2}
$$

In the special case when $\alpha_r = 0$ and $\alpha_i > 0$, we have:

$$
B_1 = 0\,, \quad B_2 = -\sqrt{\frac{2|\alpha|}{P+Q}}\,, \quad C_1 = \sqrt{2|\alpha|(P-Q)}\,, \quad C_2 = 0
\tag{17.10.3}
$$

and in the case, $\alpha_r = 0$ and $\alpha_i < 0$:

$$
B_1 = \sqrt{\frac{2|\alpha|}{P-Q}}\,, \quad B_2 = 0\,, \quad C_1 = 0\,, \quad C_2 = -\sqrt{2|\alpha|(P+Q)}
\tag{17.10.4}
$$

The condition that the poles be conjugate pairs is equivalent to the reality of the quantity ω. This condition is guaranteed by the bilinear transformation construction of the factors of Eq. (17.2.18). For the first-order factor of (17.2.18), we may define $B_1 = (1 - \hat{a}_1^2)^{1/2}$ and:

$$
A = \begin{bmatrix} -\hat{a}_1 & 0 \\ 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} B_1 \\ 0 \end{bmatrix}, \quad C = [q_1/B_1, 0]\,, \quad D = \hat{b}_0
\tag{17.10.5}
$$

17.11 Appendix-2 High-Order Analog Equalizer Design

The design of analog equalizers requires only some minor changes. The lowpass analog shelving filter of Eq. (13.5.1) is designed exactly as before, but with the design parameter $\Omega_B = \Delta\omega$, where $\Delta\omega = 2\pi\Delta f$ is the desired bandwidth in radians per second.

The shelving filter is transformed into an analog bandpass equalizer by the s-domain frequency transformation:

$$
s \to s + \frac{\omega_0^2}{s}
\tag{17.11.1}
$$

where $\omega_0 = 2\pi f_0$ is the desired center frequency in rads/second. Eq. (17.11.1) turns the first- and second-order sections of Eq. (13.5.1) into second- and fourth-order sections in s. The bandedge frequencies are calculated by

$$
\omega_{2,1} = \pm\frac{\Delta\omega}{2} + \left(\omega_0^2 + \frac{\Delta\omega^2}{4}\right)^{1/2}
$$

They satisfy the relationships,

$$
\omega_0^2 = \omega_1\omega_2\,, \quad \Delta\omega = \omega_2 - \omega_1
$$

In octaves, we have,

$$
\omega_1 = 2^{-B/2}\omega_0\,, \quad \omega_2 = 2^{B/2}\omega_0\,, \quad \Delta\omega = 2\omega_0\sinh\bigl(B\ln(2)/2\bigr)
$$

For a lowpass shelving filter, one must use, $\Omega_B = \omega_c$, where, $\omega_c = 2\pi f_c$, is the cutoff frequency defined at level G_B. For the highpass case, one must start the lowpass design with, $\Omega_B = 1/\omega_c$, and apply the highpass transformation, $s \to 1/s$.

17.12 Appendix-3 MATLAB Functions

We developed a set of MATLAB functions for implementing the designs and filtering operations discussed here. These functions, as well as the scripts used to generate all of the examples of Sect. 17.9, may be downloaded from the author's web page [330]. The set does not require any additional toolboxes and contains the following functions:

`hpeq`	high-order parametric equalizer design, Sections 2–5
`hpeqex0,1,2`	examples illustrating the usage of hpeq
`blt`	LP-to-BP bilinear transformation, Eqs. (17.2.17), (17.2.19)–(17.2.20)
`bandedge`	bandedge frequencies, Eqs. (17.2.8) and (17.4.27)
`hpeqord`	determine filter order from specifications, Eqs. (17.5.1)–(17.5.4)
`octbw`	octave to linear bandwidth calculation, Eqs. (17.6.8)–(17.6.11)
`hpeqbw`	bandwidth remapping, Eqs. (17.6.13)–(17.6.14)
`fresp`	frequency response of cascaded sections, Eqs. (17.2.18)
`dir2latt`	direct-form to normalized lattice coefficients, Eqs. (17.7.2)–(17.7.3)
`dir2state`	direct-form to state-space parameters, Eqs. (17.10.1)–(17.10.5)
`dir2decoup`	direct-form to decoupled realization, Eqs. (17.8.1)–(17.8.7)
`transpfilt`	filtering in cascaded transposed form, Fig. 17.7.2
`nlattfilt`	filtering in cascaded normalized lattice form, Fig. 17.7.3
`df2filt`	filtering in direct-form-II realized by the transposed of Fig. 17.7.2
`statefilt`	filtering in cascaded state-space form, Fig. 17.7.4
`decoupfilt`	filtering in cascaded decoupled form, Figs. 17.8.1–17.8.2
`stpeq`	state-space biquad parametric equalizer, Eq. (17.7.9)
`landen`	Landen transformation, Eq. (13.4.1)

`cde,acde`	cd elliptic function and its inverse, Eqs. (13.4.6)–(13.4.7)
`sne,asne`	sn elliptic function and its inverse, Eqs. (13.4.6)–(13.4.7)
`cne,dne`	cn and dn elliptic functions (for real arguments)
`ellipk`	complete elliptic integral $K(k)$, Eq. (13.4.4)
`ellipdeg`	exact solution of degree equation (k from N, k_1), Eq. (17.4.24)
`ellipdeg1`	exact solution of degree equation (k_1 from N, k), Eq. (17.4.23)
`ellipdeg2`	solution of degree equation using nomes, Eq. (17.4.26)
`elliprf`	elliptic rational function, Eq. (17.4.18)

In addition, there are some scripts for testing the filtering algorithms of Sect. 17.7 and the convergence of the iteration (17.6.11). Moreover, the functions allow the design of *analog* parametric equalizers and shelving filters:

```
hpeq_a
hpeqord_a
bandedge_a
hpeqbw_a
hpeqex1_a
resp_a
```

18

STFT and Phase Vocoder

18.1 Introduction

The DTFT is defined, in general, for an infinitely-long signal,

$$X(\omega)=\sum_{n=-\infty}^{\infty}x(n)e^{-j\omega n}\,,\qquad \omega=\frac{2\pi f}{f_s}$$

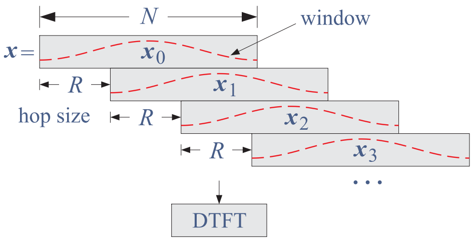
In many practical applications, such as speech or music processing, the DTFT of such long signal is essentially meaningless. For example, we may think of an hour-long recording of a person who speaks for a while, then sings for a while, then plays some music for a while, then speaks again. Even though the DTFT of such long signal is computable, it would not convey useful information about the frequency content of the signal since the nature of the signal keeps changing from speech to singing to music and so on.

A more useful quantity would be to compute the Fourier transforms of successive short time-segments of the signal that more closely capture the changing frequency content over time. This is the *time-dependent, or, short-time Fourier transform* (STFT).

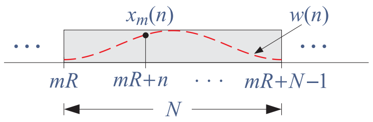
In this chapter, we introduce the STFT briefly, and discuss its use as a general signal processing system and, in particular, its application to the *phase vocoder,* used for time-scale and pitch-scale modification of speech and audio signals [375–390].

18.2 Short-Time Fourier Transform

The short-time Fourier transform (STFT) is defined by dividing the input signal $x(n)$ into successive overlapping length-N blocks, shifted relative to each other by R samples (the hop size), then windowing each block by an appropriate length-N window, $w(n)$, and taking the DTFT of each block, as show below,



$$\boxed{X(\omega, mR)=\sum_{n=0}^{N-1} x(mR+n)\,w(n)\,e^{-j\omega n}}$$



where $R \le N$ and the N time samples within the mth segment being transformed are,

$$x_m(n) = x(mR+n)\,w(n)\,, \quad n = 0, 1, \dots, N-1$$

The discrete STFT is obtained by replacing the above DTFTs by N-point DFTs, that is, evaluating them at the N DFT frequencies,

$$\omega_k = \frac{2\pi k}{N}\,, \quad k = 0, 1, \dots, N-1$$

Thus, setting, $X_{k,m} \equiv X(\omega_k, mR)$, we define the STFT,

$$\boxed{X_{k,m} = \sum_{n=0}^{N-1} x(mR+n)\,w(n)\,e^{-j\omega_k n} = \sum_{n=0}^{N-1} x_m(n)\,e^{-j\omega_k n}} \quad \text{(STFT)} \qquad (18.2.1)$$

$$k = 0, 1, \dots, N-1\,, \quad m = 0, 1, \dots, M$$

The STFT can be visualized as an $N\times(M+1)$-dimensional matrix whose columns are the N-point DFTs of the time segments, and the number of segments is $M+1$. Given an input signal of length L_x, that is, $x(n)$, $n = 0, 1, \dots, L_x - 1$, the number of segments can be calculated as follows, and then, prior to calculating the STFT, the signal $x(n)$ can be extended by padding enough zeros at its end until all frames have length N,

$$\begin{aligned} M &= \text{floor}\left(\frac{L_x}{R}\right) \\ L_{\text{ext}} &= MR + N \end{aligned} \quad \Rightarrow \quad x_{\text{ext}}(n) = \begin{cases} x(n)\,, & 0 \le n \le L_x - 1 \\ 0\,, & L_x \le n \le L_{\text{ext}} - 1 \end{cases} \qquad (18.2.2)$$

We will assume that this extension has been made and denote the extended signal by $x(n)$. Thus, the STFT matrix X can be constructed by taking the DFTs of the columns of the matrix of frames or segments, X_{frames}, whose mth column represents the mth time segment $x_m(n)$,

$$X_{\text{frames}} = [\mathbf{x}_0,\, \mathbf{x}_1,\, \dots, \mathbf{x}_m,\, \dots, \mathbf{x}_M]\,, \quad \mathbf{x}_m = \begin{bmatrix} x_m(0) \\ \vdots \\ x_m(n) \\ \vdots \\ x_m(N-1) \end{bmatrix}$$

$$X = \text{DFT}(X_{\text{frames}}) \equiv [\text{DFT}(\mathbf{x}_0),\, \text{DFT}(\mathbf{x}_1),\, \dots,\, \text{DFT}(\mathbf{x}_M)]$$

In MATLAB, all the DFTs can be computed with a single FFT call, acting column-wise on the columns of X_{frames},

$$X = \text{fft}(X_{\text{frames}}) = [\text{fft}(\mathbf{x}_0),\, \text{fft}(\mathbf{x}_1),\, \dots,\, \text{fft}(\mathbf{x}_M)]$$

Assembling the overlapping frames into the frame matrix, X_{frames}, can be done conveniently with the help of the MATLAB function, **buffer**. But prior to calling the **fft** function, each column of X_{frames} must be windowed by the chosen window function $w(n)$ — this operation can also be done efficiently in MATLAB, as we discuss below.

18.3 Spectrograms

The STFT can be displayed as a *spectrogram*, that is, plotting $X_{k,m}$ (usually in dB) as a 2D intensity plot or as 3D surface plot versus the frequency index k and versus the time frame index m, or, given a sampling rate $f_s = 1/T_s$, plotting versus the frequency and time variables,

$$\begin{aligned} f_k &= \frac{kf_s}{N}\,, \quad k = 0, 1, \dots, N-1 \\ t_m &= mRT_s = \frac{mR}{f_s}\,, \quad m = 0, 1, \dots M \end{aligned} \qquad (18.3.1)$$

Example 18.3.1: As an example, consider the following continuous-time signal consisting of three segments, varying like a chirp signal over the first segement with instantaneous frequency, $\dot{f}(t) = f_1 + \alpha t$, then consisting of a sum of two sinusoids of frequencies f_2, f_3 over the second segment, and finally varying again like a chirp over the third segment,

$$x(t) = \begin{cases} \cos(2\pi f_1 t + \pi\alpha t^2)\,, & 0 \le t \le T_0 \\ \cos(2\pi f_2 t) + \cos(2\pi f_3 t)\,, & T_0 < t \le 2T_0 \\ \cos(2\pi f_1 t + \pi\alpha t^2)\,, & 2T_0 < t \le 3T_0 \end{cases}$$

with parameter values and sampling rate,

$$\begin{aligned} &T_0 = 1000 \text{ sec} \\ &f_1 = 1 \text{ Hz}\,, \quad f_2 = 2 \text{ Hz}\,, \quad f_3 = 3 \text{ Hz}\,, \quad \alpha = \frac{f_1}{T_0}\ \frac{\text{Hz}}{\text{sec}} \\ &f_s = 10 \text{ Hz}\,, \quad T_s = \frac{1}{f_s} = 0.1 \text{ sec} \end{aligned}$$

The spectrogram was be computed with the help of the ISP2e function, **stftgram**, discussed in Sec. 18.11. MATLAB's built-in function, **spectrogram**, can also be used. The 2D intensity and 3D surface plots displaying the time variation of the frequency content of the signal are shown on Fig. 18.3.1, generated by the MATLAB code,

```
fs = 10; Ts = 1/fs;             % sampling rate
T0 = 1000;                      % segment period, seconds
f1 = 1; f2 = 2; f3 = 3;         % frequencies, Hz
a = f1/T0;                      % chirp parameter, Hz/sec

x = @(t) cos(2*pi*f1*t + pi*a*t.^2) .* (t>=0 & t<T0) + ...
      (cos(2*pi*f2*t) + cos(2*pi*f3*t)) .* (t>=T0 & t<2*T0) + ...
       cos(2*pi*f1*t*0 + pi*a*t.^2) .* (t>=2*T0);

R = 20; N = 256;                        % spectrogram parameters

tn = 0 : Ts : 3*T0;                     % sampling times
xn = x(tn);                             % sampled signal

[t,f,S] = stftgram(xn,R,N,fs);          % spectrogram,

% S is in dB, S = 20*log10(|X|) and normalized to unity maximum
% only the positive half of the f's is plotted

figure; surf(t/T0,f,S,'edgecolor','none');      % 2D intensity plot
axis tight; view(0,90); colormap(jet);
xlabel('{\itt / T}_0'); ylabel('{\itf}  (Hz)');
xaxis(0,3, 0:3); yaxis(0,5, 0:5);
title('spectrogram - 2D intensity plot');

figure; surf(t/T0,f,S,'edgecolor','none');      % 3D surface plot
view(-40, 50); colormap(jet);
xlabel('{\itt / T}_0'); ylabel('{\itf}  (Hz)'); zlabel('dB');
xaxis(0,3, 0:3); yaxis(0,5, 0:5); zaxis(-200,0, -200:100:0);
title('spectrogram - 3D surface plot');
```

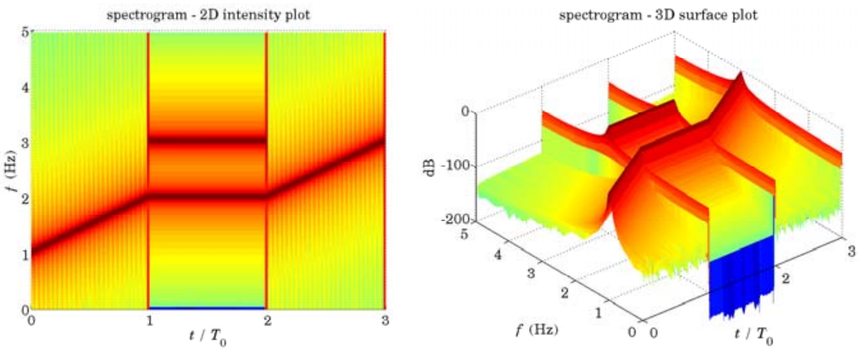


Fig. 18.3.1 Spectrogram example.

18.4 Inverse STFT and OLA Reconstruction

The inverse STFT (ISTFT) can be obtained by performing the inverse DFT, reconstructing the mth segment,

$$x(mR+n)w(n)=x_m(n)=\frac{1}{N}\sum_{k=0}^{N-1}X_{k,m}\,e^{j\omega_k n}\qquad(18.4.1)$$

$$0\le n\le N-1\,,\quad m=0,1,\dots,M$$

However, solving for $x(mR+n)$ requires division by $w(n)$, which is typically very small near its end points, and this would cause the amplification of even a small amount of noise that might be present. For this reason, a better reconstruction procedure is by the overlap-add (OLA) method, that is, aligning the inverse DFTs $x_m(n)$ according to their absolute timing, starting at $n=mR$ for the m segment, and then adding them up,

$$\boxed{y(n)=\sum_{m=-\infty}^{\infty}x_m(n-mR)}\qquad\text{(ISTFT, OLA reconstruction)}\qquad(18.4.2)$$

It can be shown [375], that for many windows and many practical choices for R, the signal $y(n)$ is equal to $x(n)$ up to a constant factor that depends on the choice of window and R.

But even if such window property, known as the *constant-overlap-add* (COLA) property, is not completely valid, one can still reconstruct $x(n)$ exactly by noting that $y(n)$ is related to $x(n)$, by

$$y(n)=x(n)\,\tilde{w}(n)$$

where $\tilde{w}(n)$ is the overlapped-added version of the window,

$$\tilde{w}(n)=\sum_{m=-\infty}^{\infty}w(n-mR)\qquad(18.4.3)$$

Thus, even if $\tilde{w}(n)$ is not constant in n, we can still solve for $x(n)$ by,

$$y(n)=x(n)\,\tilde{w}(n)=\sum_{m=-\infty}^{\infty}x_m(n-mR)\quad\Rightarrow\quad x(n)=\frac{\displaystyle\sum_{m=-\infty}^{\infty}x_m(n-mR)}{\displaystyle\sum_{m=-\infty}^{\infty}w(n-mR)}\qquad(18.4.4)$$

Since $\tilde{w}(n)$ is periodic in n with period of R samples, it can be expanded in its R-point discrete Fourier series,

$$\boxed{\begin{aligned}\tilde{w}(n)&=\sum_{m=-\infty}^{\infty}w(n-mR)=\frac{1}{R}\sum_{r=0}^{R-1}W(\omega_r)\,e^{j\omega_r n}\,,\quad \omega_r=\frac{2\pi r}{R}\\ W(\omega_r)&=\sum_{n=0}^{N-1}w(n)\,e^{-j\omega_r n}=\text{DTFT of }w(n)\text{ evaluated at }\omega=\omega_r\end{aligned}}\qquad(18.4.5)$$

Thus, the condition for the COLA property is that,

$$W(\omega_r)=0\,,\quad r=1,2,\dots,R-1 \tag{18.4.6}$$

so that only the $r=0$, or $\omega_r=0$, term is present in Eq. (18.4.5), resulting into the constant value,

$$\tilde{w}(n)=\frac{W(0)}{R}$$

See Sec. 18.11 for a demonstration of the COLA condition (18.4.6). Some windows with a “good” COLA behavior are the Hanning and Bartlett windows [375],

$$\text{Hanning:}\quad w(n)=0.5-0.5\cos\left(\frac{2\pi n}{N-1}\right),\quad n=0,1,\dots,N-1$$

$$\text{Bartlett:}\quad w(n)=1-\frac{|2n-N+1|}{N-1},\quad n=0,1,\dots,N-1$$

and the modified Hanning [387],

$$w(n)=0.5-0.5\cos\left(\frac{2\pi n}{N}\right),\quad n=0,1,\dots,N-1$$

18.5 STFT-Based Signal Processing System

A more general signal processing system based on the STFT is depicted in Fig. 18.5.1, which uses different hop sizes R_a, R_s for the analysis and synthesis parts. The following steps are carried out:

a. The input signal $x(n)$ is extended to the analysis length,

$$L_a=MR_a+N$$

as in Eq. (18.2.2), and the output signal $y(n)$ is initialized to zero over its expected synthesis length, $L_s=MR_s+N$,

$$y(n)=0\,,\quad n=0,1,\dots,L_s-1$$

b. The STFT $X_{k,m}$ of the input $x(n)$ is computed relative to the analysis hop size R_a,

$$X_{k,m}=\sum_{n=0}^{N-1}x(mR_a+n)\,w(n)\,e^{-j\omega_k n}$$
$$0\le k\le N-1\,,\quad 0\le m\le M \tag{18.5.1}$$

c. Next, $X_{k,m}$ is modified according to some transformation, such as filtering, noise-reduction, gain control, fading, or phase modification as in the phase vocoder, resulting in an output STFT, say, $Y_{k,m}$.

d. Then, the inverse STFT of $Y_{k,m}$ is computed, and each frame is windowed by another length-N window, which is usually the same as the analysis window $w(n)$,

$$y_m(n)=w(n)\cdot\frac{1}{N}\sum_{k=0}^{N-1}Y_{k,m}\,e^{j\omega_k n}\,,\quad 0\le n\le N-1 \tag{18.5.2}$$

e. The resulting windowed segments are overlapped-added with respect to the synthesis hop R_s to obtain the synthesized transformed output $y(n)$,

$$y(n)=\sum_{m=-\infty}^{\infty}y_m(n-mR_s) \tag{18.5.3}$$

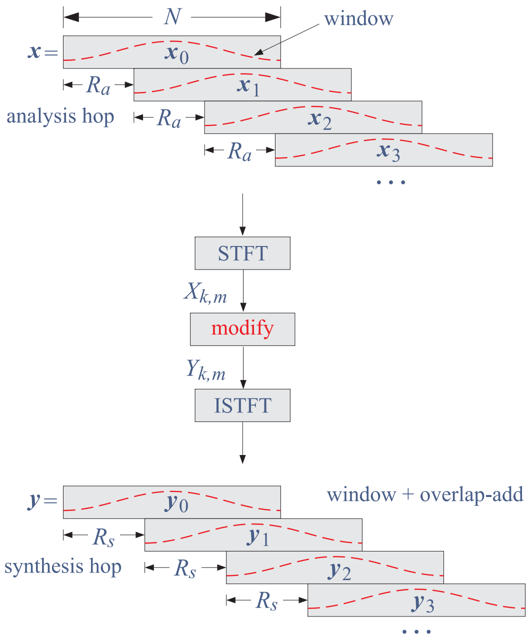


Fig. 18.5.1 STFT signal processing system.

18.6 STFT Computation

The STFT can be computed efficiently in MATLAB with a single FFT call as follows. Assuming that $x(n)$ has been extended to length, $L_a=MR_a+N$, then with the help of the built-in MATLAB function **buffer**, the signal $x(n)$ can be rearranged into an $N\times(M+1)$ matrix whose columns are the time frames, $X_{\text{buff}}=[\mathbf{x}_0\,,\mathbf{x}_1\,,\dots,\mathbf{x}_M]$, i.e., $X_{\text{buff}}(n,m)=x_m(n)$.

Then, the N-point FFT of that matrix will generate, after windowing, the FFTs of all the columns, resulting in the STFT matrix X,

$$\boxed{\begin{aligned} X_{\text{buff}} &= \mathbf{buffer}(\mathbf{x},\ N,\ N-R_a,\ \texttt{'nodelay'}) \\ W &= \mathbf{repmat}(\mathbf{w},\,1,\,M+1) = \text{replicated window} \\ X &= \mathbf{fft}(W.{*}X_{\text{buff}},\,N) \end{aligned}} \qquad \text{(STFT)} \qquad (18.6.1)$$

where $\mathbf{w}$ is the N-dimensional column vector of the window, w_n, $n = 0, 1, \dots, N-1$,

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{N-1} \end{bmatrix}$$

and W is its replication into an $N\times(M+1)$ matrix so that it can be multiplied point-wise by X_{buff}. The replication operation was implemented with the built-in function, **repmat**, but it can also be implemented as follows, assuming $\mathbf{w}$ is a column vector,

$$W = \mathbf{w}(:,\,\text{ones}(1, M+1)) = \underbrace{\begin{bmatrix} w_0 & w_0 & \cdots & w_0 \\ w_1 & w_1 & \cdots & w_1 \\ \vdots & \vdots & \vdots & \vdots \\ w_{N-1} & w_{N-1} & \cdots & w_{N-1} \end{bmatrix}}_{M+1 \text{ columns}}$$

The $N - R_a$ argument of the **buffer** function indicates the amount of overlap of successive segments, while the `'nodelay'`, argument implies that the first segment does not have any zeros padded in front of it.

Once X is computed, it may be subjected to a transformation, $Y = f(X)$, resulting in the output STFT matrix Y, which also has dimension $N\times(M+1)$. Its inverse can be carried out by a single IFFT call, resulting in the output matrix of time-frames,

$$\boxed{\begin{aligned} &Y_{\text{buff}} = \mathbf{ifft}(Y, N) \\ &y_m(n) = Y_{\text{buff}}(n, m) = n\text{th element of } m\text{th column} \end{aligned}} \qquad \text{(ISTFT)} \qquad (18.6.2)$$

The ISTFT overlap-add operation of Eq. (18.4.2) may be implemented efficiently by the following iteration that reconstructs $y(n)$ segment-by-segment while windowing,

$$\boxed{\begin{array}{l} \text{for } m = 0, 1, 2, \dots, M \\ \quad \text{for } n = 0, 1, \dots, N-1 \\ \qquad y(mR_s + n) = y(mR_s + n) + y_m(n)\,w(n) \end{array}} \qquad \text{(OLA reconstruction)} \qquad (18.6.3)$$

where the n-loop can be vectorized and we must initialize $y(n)$ to zero, that is, $y(n) = 0$, for $n = 0, 1, \dots, L_s - 1$, where, $L_s = MR_s + N$.

The ISP function, **ola**, listed below incorporates the above steps. By default, it assumes no windowing, but any pre-windowing of segments can be added easily, as seen below.

```
% ---------------------------------------------------------------------
%
% y = ola(Y,R);         % OLA reconstruction
%
% Y = Nx(M+1) matrix of column/frames to be overlap-added by hop size R
% R = hop size, must be 0 < R <= N, R=N (no-overlap)
%
% y = column vector of overlap-added columns
%
% notes: N = column/frame size
%        M+1 = no. frames
%        length(y) = L = R*M + N
%
% if each frame is to be pre-windowed before overlap, then, do,
%
%     w = ...                 % define Nx1 vector of window samples
%     W = repmat(w,1,M+1);  % replicated Nx(M+1) window matrix
%     y = ola(W.*Y,R);      % overlap-add the windowed frames

function y = ola(Y,R)

[N,M1] = size(Y); M = M1-1;

L = R*M + N;
y = zeros(L,1);   % pre-allocate

n = (1:N)';

for m = 0:M-1
    y(m*R + n) = y(m*R + n) + Y(:,m+1);
end

% ---------------------------------------------------------------------
```

In summary, the complete algorithm for the STFT-based signal processing system is as follows in MATLAB-like notation, given hop sizes R_a, R_s, and assuming the same length-N analysis and synthesis window $w(n)$, and extending $x(n)$ to length L_a,

$$\boxed{\begin{aligned} &X_{\text{buff}} = \mathbf{buffer}(\mathbf{x},\ N,\ N-R_a,\ \texttt{'nodelay'}) = \text{input time frames} \\ &W = \mathbf{repmat}(\mathbf{w},\,1,\,M+1) = \text{replicated window} \\ &X = \mathbf{fft}(W.{*}X_{\text{buff}},\,N) = \text{input STFT} \\ &Y = f(X) = \text{desired modification, output STFT} \\ &Y_{\text{buff}} = \mathbf{ifft}(Y, N) = \text{output time frames} \\ &y = \mathbf{ola}(W.{*}Y_{\text{buff}},\,R_s) = \text{OLA reconstruction with windowing} \end{aligned}} \qquad (18.6.4)$$

where the operation, “$.*$”, denotes element-wise multiplication.

18.7 Phase Vocoder

The phase vocoder is an example of such STFT signal processing system, realized by Eqs. (18.6.1)–(18.6.4). The transformation step, $X_{k,m} \to Y_{k,m}$, to be carried out between Eqs. (18.6.1) and (18.6.2) is, in its simplest form, a modification of the phase of $X_{k,m}$, while preserving its magnitude. We have in polar form,

$$\begin{aligned} X_{k,m} &= |X_{k,m}|\, e^{j\Phi_{k,m}} \\ Y_{k,m} &= |Y_{k,m}|\, e^{j\Psi_{k,m}} \end{aligned} \tag{18.7.1}$$

where the magnitudes are preserved,

$$|Y_{k,m}| = |X_{k,m}| \tag{18.7.2}$$

and the output phases $\Psi_{k,m}$ are computed recursively from the input phases $\Phi_{k,m}$, as follows,

```
phase modification algorithm

for $k = 0, 1, \dots, N-1$,
    $\omega_k = \dfrac{2\pi k}{N}$
    $\Psi_{k,0} = \Phi_{k,0}$
    for $m = 1, 2, \dots, M$,
        $\Delta\omega_{k,m} = \dfrac{1}{R_a} \cdot \Big[\Phi_{k,m} - \Phi_{k,m-1} - R_a\,\omega_k\Big]_{\text{mod}\,2\pi}$
        $\omega_{k,m} = \omega_k + \Delta\omega_{k,m}$
        $\Psi_{k,m} = \Psi_{k,m-1} + R_s\,\omega_{k,m}$
```

(18.7.3)

where, the k-loop may be vectorized, and the notation, $[x]_{\text{mod}\,2\pi}$, stands for the *phase unwrapping* of the angle x modulo 2π, that is, adding to, or subtracting from, x enough multiples of 2π until it lies in the standardized angle interval, $-\pi \le x \le \pi$. It can be implemented easily by the vectorized anonymous MATLAB function, **mod2pi**(x),

```
mod2pi = @(x) mod(x+pi, 2*pi) - pi;      % usage: y = mod2pi(x)
```

In summary, the complete phase vocoder algorithm is described by Eqs. (18.6.1), (18.7.2), (18.7.3), (18.6.2), and (18.6.3), in that order. The justification of the algorithm (18.7.3) is given in Sec. 18.9.

18.8 Time-Scale Modification

The main application of the phase vocoder is in time-scale and pitch-scale modification of speech and audio signals. In time-scale modification, one wishes to replay an audio signal at a faster or slower speed without altering its frequency content, as for example, in playing a piano piece faster where same keys (i.e. generated frequencies) are played, but at a faster speed.

By choosing different hop sizes R_a, R_s, the duration of the output signal $y(n)$ can be made longer or shorter than that of the input $x(n)$, depending on whether $R_s > R_a$ or $R_s < R_a$, respectively.

The purpose of the phase modification equations (18.7.2) and (18.7.3) is to preserve the frequency content of the signal under such change in duration. We may define the speed-up and time-stretching factors by,

$$\begin{aligned} &\boxed{r = \frac{R_a}{R_s}} \quad \text{(speed-up factor)} \\ &\boxed{\frac{1}{r} = \frac{R_s}{R_a}} \quad \text{(time-stretching factor)} \end{aligned} \tag{18.8.1}$$

so that $R_s = R_a/r$, and $r > 1$ corresponds to a faster rendition and shorter duration of the signal, and $r < 1$, slower rendition and longer duration. If we specify R_a, r, then we may calculate R_s by rounding, $R_s = \text{round}(R_a/r)$, or conversely, as is preferred in practice, if we specify, R_s, r, then, $R_a = \text{round}(rR_s)$,

$$R_s = \text{round}\left(\frac{R_a}{r}\right), \quad \text{or, alternatively,} \quad R_a = \text{round}(rR_s)$$

18.9 Phase Vocoder Model

Here, we provide a simplified justification of the phase vocoder algorithm, based on the discussion in [385]. Given a sinusoidal signal of varying amplitude and varying phase,

$$x(t) = A(t)\, e^{j\Phi(t)}$$

the *instantaneous* analog frequency is defined as the time-derivative of the phase:

$$\Omega(t) = \dot{\Phi}(t) = \frac{d\Phi(t)}{dt}$$

Considering the signal values at two nearby time instants, t and $t + \Delta t$, we may expand the phase to *first-order* in Δt and approximate the phase and the instantaneous frequency as,

$$\begin{aligned} \Phi(t+\Delta t) &= \Phi(t) + \dot{\Phi}(t)\,\Delta t = \Phi(t) + \Omega(t)\,\Delta t \\ \Omega(t+\Delta t) &= \Omega(t) \end{aligned} \tag{18.9.1}$$

The phase vocoder is based on the implicit assumption that the signal is a sum of such sinusoidal terms with varying amplitudes and phases,

$$x(t) = \sum_i A_i(t)\, e^{j\Phi_i(t)}, \quad \Omega_i(t) = \dot{\Phi}_i(t) \tag{18.9.2}$$

The main objective of the phase vocoder algorithm is to ensure that the instantaneous frequencies contained in the signal are *preserved* in going from the overlapped analysis frames at hop size R_a to the overlapped synthesis frames at hop size R_s. Ideally, the STFTs of the input and output frames would be:

$$X_{k,m} = \sum_{n=0}^{N-1} x(mR_a + n)\, w(n)\, e^{-j\omega_k n}$$

$$Y_{k,m} = \sum_{n=0}^{N-1} x(mR_s + n)\, w(n)\, e^{-j\omega_k n}$$

where ω_k are the DFT frequencies,

$$\omega_k = \frac{2\pi k}{N}\,, \quad k = 0, 1, \dots, N-1$$

and, we assumed that the signal $x(t)$ was sampled at a rate $f_s = 1/T$, so that the time intervals that correspond to the length-N window of the mth frame are as follows, for the analysis and synthesis cases,

$$t^a_{mn} = (mR_a + n)T = t^a_m + nT \qquad \text{(analysis)}$$

$$t^s_{mn} = (mR_s + n)T = t^s_m + nT \qquad \text{(synthesis)}$$

where, $t^a_m = mR_aT$, and, $t^s_m = mR_sT$, are the beginning times of the mth segments. It follows that the sampled signal for the analysis frames, $x(mR_s + n)$, would be according to the sinusoidal model of Eq. (18.9.2),

$$x(t^a_m + nT) = \sum_i A_i(t^a_m + nT)\, e^{j\Phi_i(t^a_m + nT)}$$

Assuming a small enough sampling interval T, we may expand the signal phases using the approximation of Eq. (18.9.1). Assuming also that the signal amplitudes vary slowly across each windowed frame, we obtain the following approximations, over the length-N window of the mth frame,

$$A_i(t^a_m + nT) \approx A_i(t^a_m)\,, \quad 0 \le n \le N-1$$

$$\Phi_i(t^a_m + nT) = \Phi_i(t^a_m) + (nT)\,\Omega_i(t^a_m)$$

Defining the digital instantaneous frequencies in [rads/sample],

$$\omega_i(t^a_m) = \Omega_i(t^a_m)\, T$$

then, the phase approximations can be written as,

$$\Phi_i(t^a_m + nT) = \Phi_i(t^a_m) + n\,\omega_i(t^a_m)$$

so that the signal can be approximated as follows within the mth frame,

$$x(t^a_m + nT) \approx \sum_i A_i(t^a_m)\, e^{j\Phi_i(t^a_m) + jn\,\omega_i(t^a_m)} \tag{18.9.3}$$

Inserting Eq. (18.9.3) into the analysis STFT, we have,

$$\begin{aligned}
X_{k,m} &= \sum_{n=0}^{N-1} x(mR_a + n)\, w(n)\, e^{-j\omega_k n} \\
&= \sum_{n=0}^{N-1} \sum_i A_i(t^a_m)\, e^{j\Phi_i(t^a_m) + jn\,\omega_i(t^a_m)}\, w(n)\, e^{-j\omega_k n} \\
&= \sum_i A_i(t^a_m)\, e^{j\Phi_i(t^a_m)} \sum_{n=0}^{N-1} e^{jn\,\omega_i(t^a_m)} e^{-jn\,\omega_k}\, w(n)
\end{aligned}$$

where the summation over n is recognized to be the frequency-shifted DTFT of the window $w(n)$,

$$\hat{W}(\omega) = \sum_{n=0}^{N-1} e^{-j\omega n} w(n) = \text{DTFT of } w(n)$$

$$\hat{W}\big(\omega_k - \omega_i(t^a_m)\big) = \sum_{n=0}^{N-1} e^{jn\,\omega_i(t^a_m)} e^{-jn\,\omega_k}\, w(n)$$

Because we always assume that the window $w(n)$ is real-valued and symmetric about its middle, it follows that its DTFT can be factored into a real and even function of ω, and a phase part corresponding to a delay by $(N-1)/2$ samples,

$$\hat{W}(\omega) = W(\omega)\, e^{-j\omega(N-1)/2}\,, \quad W(\omega) = W(-\omega) = \text{real-valued}$$

for example, we have for the rectangular window,

$$\hat{W}(\omega) = \frac{\sin(\omega N/2)}{\sin(\omega/2)}\, e^{-j\omega(N-1)/2}$$

It follows that the STFT of the analysis frames will be,

$$X_{k,m} = \sum_i A_i(t^a_m)\, e^{j\Phi_i(t^a_m)}\, W\big(\omega_i(t^a_m) - \omega_k\big)\, e^{j[\omega_i(t^a_m) - \omega_k](N-1)/2} \tag{18.9.4}$$

For large N, the function $W(\omega)$ is highly concentrated about $\omega = 0$, and therefore, only that sinusoidal term i whose instantaneous frequency $\omega_i(t^a_m)$ falls within the kth DFT bin, that is, $\omega_i(t^a_m) \approx \omega_k$, will effectively contribute to the above sum. Therefore, we can keep approximately only the $i = k$ term,

$$X_{k,m} \approx A_k(t^a_m)\, e^{j\Phi_k(t^a_m)}\, W\big(\omega_k(t^a_m) - \omega_k\big)\, e^{j[\omega_k(t^a_m) - \omega_k](N-1)/2} \tag{18.9.5}$$

In this expression, the frequency, $\omega_k(t^a_m)$, is not equal to ω_k but it is nearby, i.e., we can introduce a deviation from ω_k to be determined,

$$\omega_{k,m} \equiv \omega_k(t^a_m) = \omega_k + \Delta\omega_{k,m} \tag{18.9.6}$$

In order to obtain the phase of the STFT, that is, in polar form,

$$X_{k,m} = |X_{k,m}|\,e^{j\Phi_{k,m}}$$

we observe that Eq. (18.9.5) is already separated into real factors and phases, so that up to multiples of 2π we must have,

$$\Phi_{k,m} = \Phi_k(t_m^a) + \big[\omega_k(t_m^a) - \omega_k\big]\,(N-1)/2 \tag{18.9.7}$$

Our objective, eventually, is to determine the instantaneous frequencies, $\omega_k(t_m^a)$, from the computed STFT phases $\Phi_{k,m}$. To do so, we consider how these phases change from frame to frame, i.e., in going from time, $t_{m-1}^a = (m-1)R_a T$, to time, $t_m^a = mR_a T$. But since, we have,

$$t_m^a = t_{m-1}^a + R_a T$$

we may use the approximate expansion of Eq. (18.9.1) to write,

$$\Phi_k(t_m^a) = \Phi_k(t_{m-1}^a + R_a T) \approx \Phi_k(t_{m-1}^a) + R_a T\,\Omega_k(t_m^a)\,, \quad \text{or,}$$

$$\Phi_k(t_m^a) \approx \Phi_k(t_{m-1}^a) + R_a\,\omega_k(t_m^a)$$

and also from Eq. (18.9.1), we have approximately, $\omega_k(t_{m-1}^a) = \omega_k(t_m^a)$, so that,

$$\begin{aligned}
\Phi_{k,m} &= \Phi_k(t_m^a) + \big[\omega_k(t_m^a) - \omega_k\big]\,(N-1)/2 \\
&= \Phi_k(t_{m-1}^a) + R_a\,\omega_k(t_m^a) + \big[\omega_k(t_m^a) - \omega_k\big]\,(N-1)/2 \\
&= \Phi_k(t_{m-1}^a) + \big[\omega_k(t_{m-1}^a) - \omega_k\big]\,(N-1)/2 + R_a\,\omega_k(t_m^a) \\
&= \Phi_{k,m-1} + R_a\,\omega_k(t_m^a)\,, \quad \text{or,}
\end{aligned}$$

$$\Phi_{k,m} = \Phi_{k,m-1} + R_a\,\omega_k(t_m^a)$$

$$\Phi_{k,m} = \Phi_{k,m-1} + R_a\,(\omega_k + \Delta\omega_{k,m})$$

thus, up to multiples of 2π,

$$\boxed{\begin{aligned}
&\Phi_{k,m} = \Phi_{k,m-1} + R_a\,(\omega_k + \Delta\omega_{k,m}) \\
&R_a\,\Delta\omega_{k,m} = \Phi_{k,m} - \Phi_{k,m-1} + R_a\,\omega_k
\end{aligned}} \tag{18.9.8}$$

In a similar fashion, we can show that the synthesis STFTs,

$$Y_{k,m} = |Y_{k,m}|\,e^{j\Psi_{k,m}}$$

will satisfy similar recursions which will ensure that the instantaneous frequencies are preserved:

$$\Psi_{k,m} = \Psi_{k,m-1} + R_s\,(\omega_k + \Delta\omega_{k,m}) \tag{18.9.9}$$

Therefore, if we can solve Eq. (18.9.8) for $\Delta\omega_{k,m}$, we can reconstruct the output STFTs recursively. To solve for, $\Delta\omega_{k,m}$, we note that in the equation,

$$R_a\,\Delta\omega_{k,m} = \Phi_{k,m} - \Phi_{k,m-1} + R_a\,\omega_k$$

the right-hand side, being a phase, is defined up to a multiple of 2π. Thus, phase-unwrapping it modulo-2π to fall within the standard interval $[-\pi, \pi]$, we obtain,

$$R_a\,\Delta\omega_{km} = \Big[\Phi_{k,m} - \Phi_{k,m-1} - R_a\,\omega_k\Big]_{\text{mod}\,2\pi}$$

thus,

$$\boxed{\Delta\omega_{k,m} = \frac{1}{R_a}\cdot\Big[\Phi_{k,m} - \Phi_{k,m-1} - R_a\,\omega_k\Big]_{\text{mod}\,2\pi}} \tag{18.9.10}$$

Putting all the above steps together, we arrive at the phase modification algorithm (18.7.3).

18.10 Pitch-Scale Modification

Pitch shifting refers to scaling all frequencies by a factor r, that is, replacing, $f \to rf$, without altering the duration of the signal, as for example in playing a piano piece an octave higher. This is not the same as frequency translation in which all frequencies are shifted by a fixed amount, $f \to f + f_0$.

If we have an audio signal $x(n)$ recorded at a sampling rate f_s and we replay it a rate that is r times faster, $f_s' = rf_s$, then both the duration and the pitch will be altered, with the duration becoming r times shorter, and the pitch becoming r times higher—this is known as the “chipmunk” effect, used for example in children’s cartoons.

In order to perform pitch shifting without changing the duration of the signal, we may combine a phase vocoder time-scale modification with a resampling operation. In other words, we may “chipmunk” the signal and then correct its duration with a phase vocoder.

For example, suppose $r > 1$, and we resample $x(n)$ at the rate f_s/r, and play it back at f_s/r, then it would sound like the original, but if we play it back at $f_s = r\cdot f_s/r$, then it would have a pitch r times higher but it will also have r times shorter duration. Thus, if we follow this operation with a phase vocoder with a time-stretching factor r, or equivalently, speed-up factor $1/r$, we would restore the original duration, while not affecting the already pitch-shifted frequencies.

Alternatively, we may reverse these operations. First we apply a phase vocoder with speed-up factor of $1/r$ or stretching factor r. Now the signal will have r times longer duration, but its pitch will not have shifted if played at rate f_s. If we now resample this at the rate f_s/r and played it a rate f_s/r, it would sound the same as that longer version, but if we play it back at $f_s = r\cdot f_s/r$, it will be pitch-shifted by a factor of r, and its duration will be scaled down by a factor of r to the original duration.

The two alternative approaches are depicted below in Fig. 18.10.1, where it is advantageous to apply the top version when $r > 1$, and the bottom, when $r < 1$.

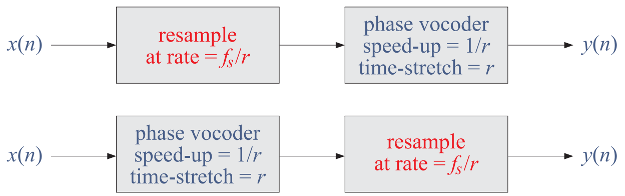


Fig. 18.10.1 Pitch shifting by a factor of r.

18.11 Computer Experiments

Phase Vocoder Design

Write a MATLAB function, **phvoc**, to implement a phase vocoder, with syntax,

```
y = phvoc(x, r, Ra, N);

% x  = input audio signal
% r  = speed-up factor
% Ra = analysis hop size, synthesis hop size Rs = round(Ra/r)
% N  = FFT frame length
%
% y = output signal
```

It must be structured to implement Eqs. (18.6.1), (18.7.2), (18.7.3), (18.6.2), and (18.6.3), and must consist internally of the following three parts, which call three subfunctions, **stft**, **phmap**, **istft**,

```
X = stft(x,Ra,N);        % STFT of input

Y = phmap(X,r,Ra,N);     % phase remapping

y = istft(Y,Rs,N);       % ISTFT/OLA
```

These subfunctions can be embedded inside **phvoc**, or reside in separate M-files. In the phase vocoder context, the Hanning window is preferred for both the analysis and synthesis parts. It is defined as follows for length N,

$$w(n) = 0.5 - 0.5\cos\left(\frac{2\pi n}{N-1}\right), \quad n = 0, 1, \dots, N-1$$

Your **stft** can be implemented using Eq. (18.6.1), or, alternatively, you can use a for-loop that runs over the overlapping frames, i.e.,

```
n = 1:N;                               % relative frame indices
for m=0:M
   ta = m*Ra;                          % beginning of m-th frame
   X(m+1,:) = fft(w.*x(ta+n), N);      % w,x(ta+n) must be length-N rows
end
```

Your **istft** can be done with a similar loop, i.e.,

```
n = 1:N;                               % relative frame indices
for m=0:M
   ts = m*Rs;                          % beginning of m-th frame
   ym = w.*real(ifft(Y(m+1,:)));       % IDFT of m-th frame & window
   y(ts + tN) = y(ts + tN) + ym;       % overlap-add
end
```

a. To test your function, load in your main program an audio file of your choice, sampled at 44.1 kHz, and save a 10-second portion of one channel that includes both music and vocals, and process it through your function **phvoc** for the following values of the speed-up factor, $r = 1.4$ and $r = 0.7$, corresponding to playing the signal 40% faster, or, 30% slower, respectively,

```
r = 1.4;

[x,FS] = audioread('your_audio_file');

% x = save a 10-sec portion that includes music and vocals

y = phvoc(x,r,Ra,N);    % experiment with different Ra,N

audiowrite('phvoc.wav', [x, zeros(1,FS), y], FS);

% save 'phvoc.wav' as part of your report
```

For each r, concatenate the input and the output, separated by 1-sec delay, and save the result in a wave file, or, if you wish, you may create a single wave file that concatenates the input and the outputs for the two values of r.

b. Repeat part (a) for the speed-up factor $r = 0.7$.

Pitch-Scale Modification

For this part, please write a MATLAB function, **pitchmod**, that performs pitch-scale modification by a factor r. For simplicity, you may realize it by the top alternative shown in Fig. 18.10.1, for both cases, $r > 1$ and $r < 1$. The function should have usage:

```
y = pitchmod(x, r, Ra, N)

% x  = input audio signal
% r  = pitch scaling factor
% Ra = analysis hop
% N  = FFT frame length
%
% y = output signal
```

To implement the resampling operation, you may use the built-in MATLAB function, **resample**, which requires r to be a rational number, $r = p/q$, where p, q are integers. We may approximate r or $1/r$ in such rational form using the built-in function, **rat**. Thus, your MATLAB function, **pitchmod** could be structured to do the following:

```
% [p,q] = rat(1/r);
% pass p,q into resample
% pass output of resample into phvoc
```

a. To test your function, load the included wave file, **flute2.wav**, with the command,

```
[x,FS] = audioread('flute2.wav');   x = x(:,1).';
```

and send it through your **pitchmod** function using parameters $r = 2$, $R_a = 256$, $N = 2048$. Concatenate the pitch-shifted output $y(n)$ with the input $x(n)$ and save it in a wave file to become part of your report,

```
audiowrite('pitchmod.wav', [x, zeros(1,FS), y], FS);
```

To hear the pitch-shifted result, run the command,

```
soundsc([x, zeros(1,FS), y], FS);
```

To see the pitch-shifted result, compute a 4096-point FFT of $x(n)$ and plot its (normalized to unity maximum) spectrum $|X(f)|$ over the frequency range, $0 \le f \le 1800$ Hz, and observe that it is dominated by harmonics of a fundamental. Determine the frequency in Hz of the fundamental harmonic. What key does it correspond to on an ABC musical scale?

[*Note:* MATLAB's FFT function will truncate the signal to length 4096 before computing the spectrum, but that is good enough here to demonstrate the pitch-shifting property.]

Repeat the spectrum plot for the pitch-shifted output $y(n)$, and also, determine the frequency of its fundamental harmonic, which should be. roughly equal to r times the original one. Compare the two spectrum plots and note how the pitch-shifted one is essentially a scaled version of the original one by a factor of r. Generate also the spectrograms of the *full length* signals $x(n)$ and $y(n)$, as described in the Appendix. Some example graphs are shown in Sec. 18.11.

b. Repeat all the questions of part (a) for the scale factor, $r = 1/2$.

c. In order to visualize the operations depicted in Fig. 18.10.1, load the attached MAT file, **x4.mat**, which consists of a four-millisecond portion of the above flute waveform, sampled at a rate of 44.1 kHz. The following command loads that signal in the variable x4,

```
load x4.mat
```

Plot this signal versus time in the range $0 \le t \le 4$ msec using a stem plot.

Then, resample this signal at the rate f_s/r with $r = 2$, and plot it again versus $0 \le t \le 4$. You will observe that it looks like the original, except the spacing between the time samples is r times larger.

Next, replay this resampled signal at the rate, $f_s = r \cdot f_s/r$, and plot it again versus $0 \le t \le 4$. This is the "chipmunked" version of the original.

Finally, pass the latter signal into your **phvoc** function with a time-stretching factor r, or speed-up factor $1/r$, and plot the result again versus $0 \le t \le 4$. Because of the short lengths of the signals, use the following parameters for the vocoder, $R_a = 4$ and $N = 32$.

Note how the resulting signal has the same 4-msec length as the original, but it appears to be varying r times faster.

Some example graphs are included in Sec. 18.11.

COLA Property

In this part, the objective is to check the COLA property. Consider the Hanning window with length $N = 128$ and its overlapped version with $M = 10$, defined by,

$$\tilde{w}(n) = \sum_{m=0}^{M} w(n - mR) \tag{18.11.1}$$

and the following values of the hop size R,

$$R = \frac{N}{2}\,, \quad \frac{3N}{8}\,, \quad \frac{N}{3}\,, \quad \frac{N}{4}$$

For each value of R, calculate and plot $\tilde{w}(n)$ over $0 \le n \le 800$, and moreover, calculate and plot the magnitude of the corresponding R-point DFT based on Eq. (18.4.5), normalized to unity maximum. Again, some example graphs are included in Sec. 18.11.

Do-It-Yourself Spectrogram

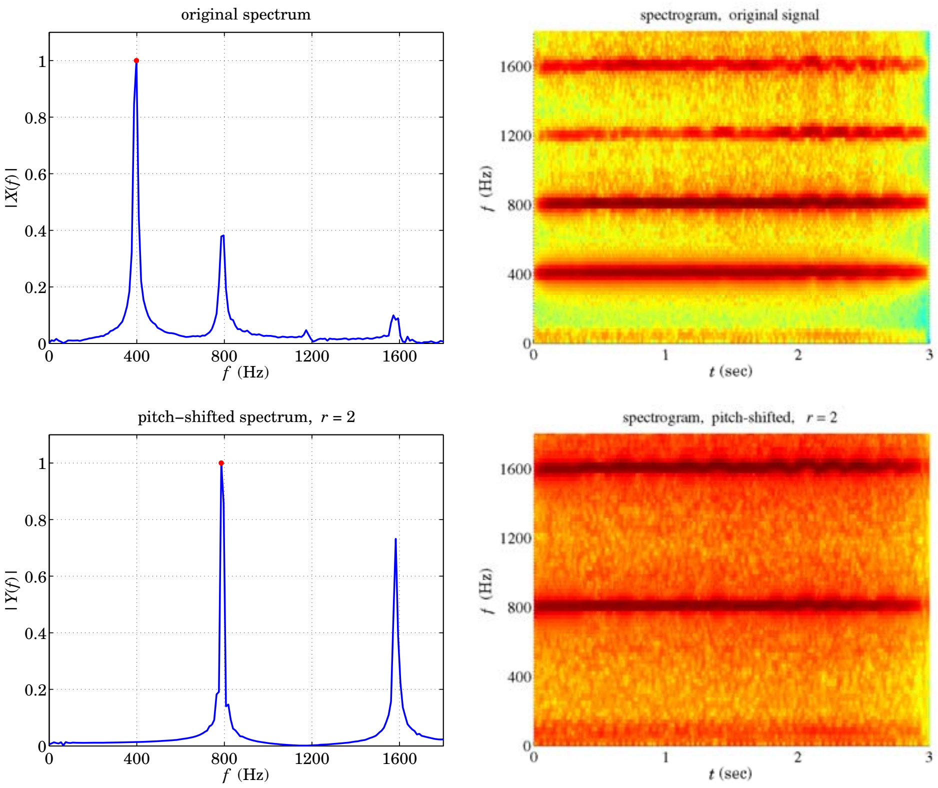
You can use your STFT function to do your own version of a spectrogram. The following code segment illustrates the procedure on the flute waveform example. All three spectrograms shown at the end were produced in a similar fashion.

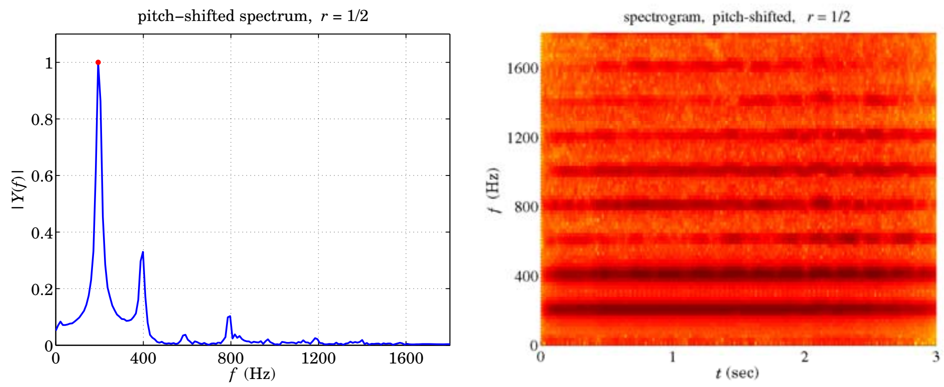
```
R = 256;                       % hop size
N  = 2048;                     % FFT length

[x,fs] = audioread('flute2.wav'); x = x(:,1).';

X = stft(x,R,N).';             % X assumed to have size Nx(M+1)
M = size(X,2) - 1;

k = 0:N/2;                     % positive half of Nyquist interval
f = k*fs/N;                    % 0 <= f <= fs/2
t = (0:M)*R/fs;                % M+1 time frames, hop interval R*Ts = R/fs

Xmag = abs(X(k+1,:));          % extract positive-frequency half
Xmag = Xmag/max(max(Xmag));    % normalize to unity maximum
```

```
S = 20*log10(Xmag);            % dB

surf(t,f,S,’edgecolor’,’none’);    % spectrogram, t horizontal, f vertical

colormap(jet); colorbar;
axis tight; view(0,90);
xlabel(’{\itt} (sec)’);
ylabel(’{\itf}  (Hz)’);
ylim([0,1800]);                % show range 0 <= f <= 1800 Hz
```

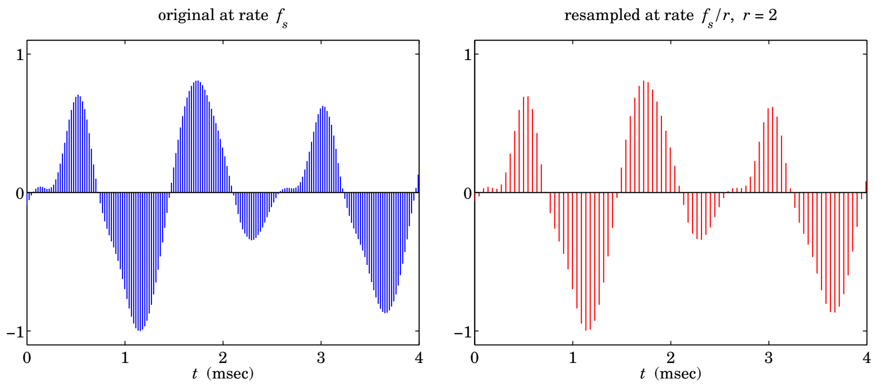
Example Graphs

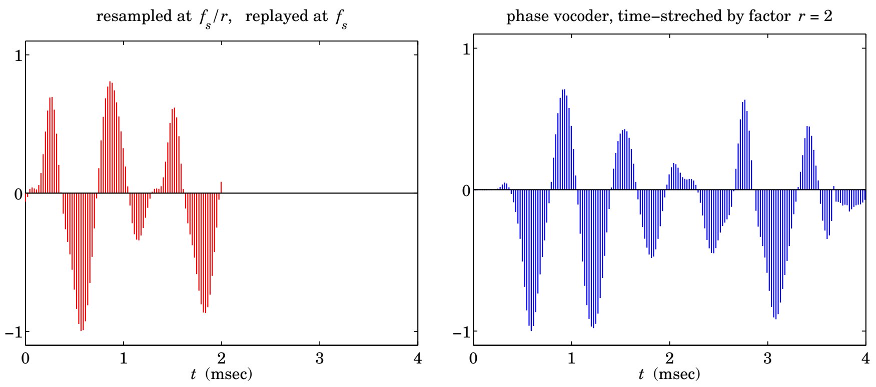
For the **flute2.wav** DTFT spectrum and spectrogram



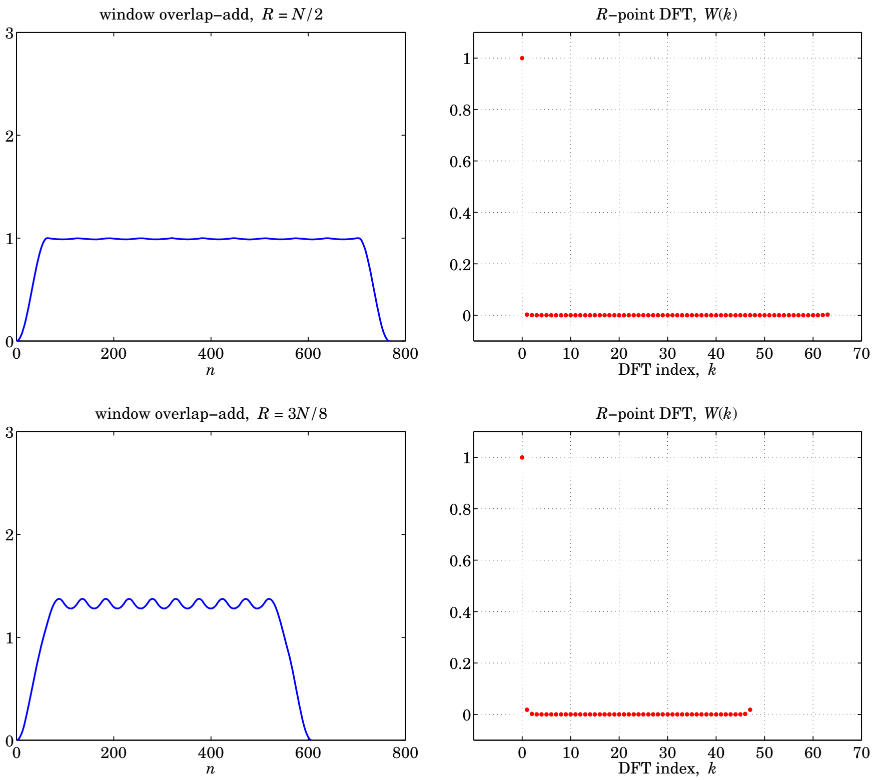


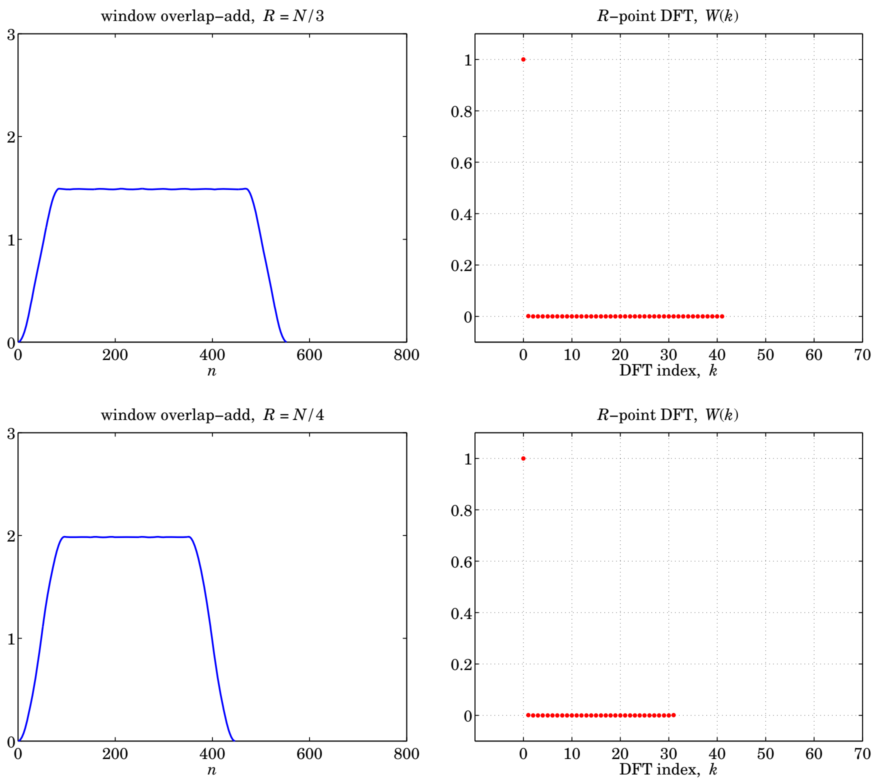
For the resampled versions of the **x4.mat** portion of the **flute2.wav** waveform





For illustrating the COLA property





19

DCT, MDCT, and Data Compression

19.1 DCT and MDCT Compression Systems

Like the DFT, the *discrete cosine transform* (DCT) belongs to the family of *discrete orthogonal transforms* and is widely used for image and audio compression. It was originally proposed in [394] and has been studied very extensively. For example, the JPEG image compression standard is based on the DCT.

The *modified DCT* (MDCT) is not quite an orthogonal transform, but through the use of the related *time-domain aliasing cancellation* (TDAC) property, it shares the same remarkable data compression properties as the DCT. The discrete wavelet transform (DWT) and its variants also are used in data compression, for example, the JPEG-2000 image compression system has replaced the original DCT-based JPEG standard. We present an introduction to the DWT in Chap. 20.

In this Chapter, we study the data compression properties of the DCT, implement a DCT compression system in MATLAB, and apply it to audio and image data. In addition, a compression system based on the MDCT and its inverse (IMDCT) is studied and implemented in MATLAB, including the related TDAC property, which is key in most current audio compression systems, such as MP3, AAC, WMA, Vorbis, and others. Some related references on this material are, [391–412]. A unified discussion of several discrete orthogonal transforms may be found in [396].

The DCT takes as input a length-N real-valued signal vector, $\mathbf{x}$, assumed here to be a column, and produces a length-N real-valued vector of DCT values, $\mathbf{C}$, while the inverse DCT reverses the process, recovering the original signal vector,

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix} \xrightarrow{\text{DCT}} \mathbf{C} = \begin{bmatrix} C_0 \\ C_1 \\ \vdots \\ C_{N-1} \end{bmatrix} \xrightarrow{\text{IDCT}} \mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix} \tag{19.1.1}$$

In a typical DCT-based data compression system, a large fraction (e.g., 80-90%) of the DCT coefficients C_k are dropped, retaining only a few of the most significant ones, which are then quantized for storage or transmission. The signal recovered from the

few retained DCT coefficients—while not identically equal to the original one—is close enough to the original to be perceptually indistinguishable from it.

Such process is referred to as *lossy compression* since the recovered signal is slightly different from the original one as, for example, in MP3 audio or JPEG images. In audio and image applications, the DCT coefficient quantization process takes into account the psychophysical properties of the hearing or visual system, and is beyond the scope of this chapter.

Fig. 19.1.1 shows a simplified DCT compression system [391] in which a long input signal is divided into contiguous length-N blocks or frames, the N-point DCT of each frame is computed and compressed, then, the corresponding inverse DCTs of the frames are computed and the recovered signal blocks are concatenated together to form the output signal.

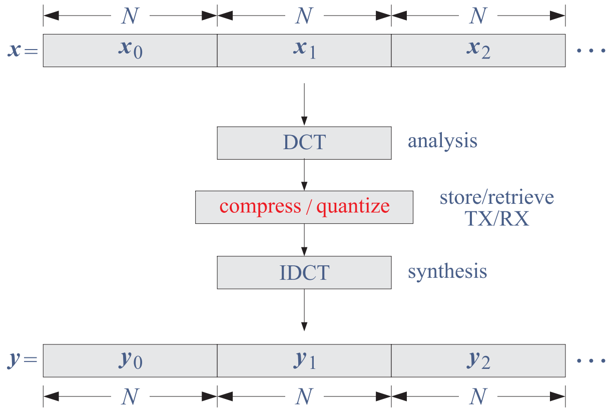


Fig. 19.1.1 DCT data compression system.

To avoid possible artifacts that may be introduced by the blocking process, a more refined MDCT-based approach divides the input signal into overlapping blocks.

Fig. 19.1.2 shows a typical MDCT-based compression system that divides the input into blocks that are 50% overlapping, then windows each block, and calculates its MDCT, compresses the MDCT coefficients, and takes the inverse MDCT, windows the resulting blocks again, and finally overlaps and adds the results.

The window functions must be chosen properly, that is, satisfying the so-called *Princen-Bradley conditions*, so that the overlap/add operation correctly implements the time-domain aliasing cancellation (TDAC) property that allows the faithful reconstruction of the input signal.

MDCT-based compression systems are used in current audio compression formats, such as AAC and WMA. In Sec. 19.6, we discuss the implementation of such a system in MATLAB.

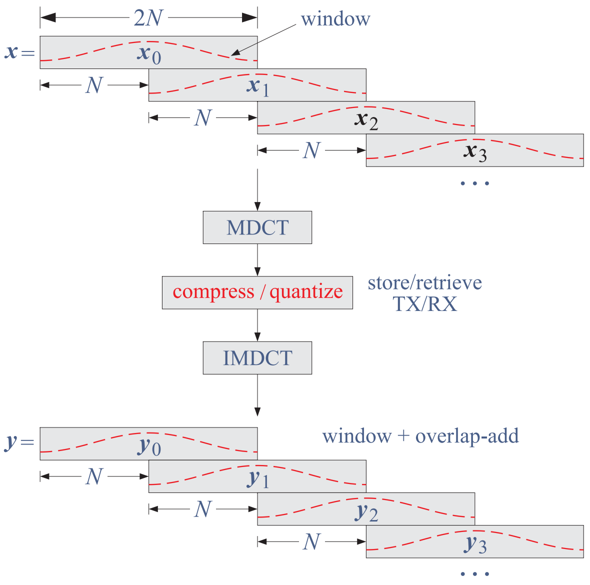


Fig. 19.1.2 MDCT/TDAC signal compression system.

19.2 Discrete Cosine Transform

There exist eight versions of the DCT, see Ref. [392], but the type-2 is the most widely used and is the default version in MATLAB's **dct** function,[†] and can be implemented efficiently using an FFT. With reference to Eq. (19.1.1), the type-2 DCT is defined as follows,

$$\boxed{C_k = \sum_{n=0}^{N-1} x_n \cos\left(\frac{\pi k}{N}\left(n+\tfrac{1}{2}\right)\right), \quad k = 0,1,\dots,N-1} \quad \text{(DCT)} \qquad (19.2.1)$$

and its inverse,

$$\boxed{x_n = \frac{1}{N}C_0 + \frac{2}{N}\sum_{k=1}^{N-1} C_k \cos\left(\frac{\pi k}{N}\left(n+\tfrac{1}{2}\right)\right), \quad n = 0,1,\dots,N-1} \quad \text{(IDCT)} \quad (19.2.2)$$

The DCT pair satisfies the following Parseval-like identity,

$$\boxed{\sum_{n=0}^{N-1} x_n^2 = \frac{1}{N}\left[C_0^2 + 2\sum_{k=1}^{N-1} C_k^2\right]} \quad \text{(Parseval)} \qquad (19.2.3)$$

[†] recent versions of MATLAB include the first four DCT types

In MATLAB, and other literature, a normalized version of the DCT coefficients is used, defined as follows, where δ_k denotes the Kronecker delta,

$$D_k = \frac{1}{s_k}C_k\,, \quad \text{where} \quad s_k = \sqrt{\frac{N}{2}\left(\delta_k + 1\right)}\,, \quad k = 0,1,\dots,N-1 \qquad (19.2.4)$$

so that,

$$s_0 = \sqrt{N}\,, \qquad s_k = \sqrt{\frac{N}{2}}\,, \quad k = 1,2,\dots,N-1$$

$$D_0 = \sqrt{\frac{1}{N}}\,C_0\,, \quad D_k = \sqrt{\frac{2}{N}}\,C_k\,, \quad k = 1,2,\dots,N-1$$

With this definition, Eqs. (19.2.1)–(19.2.3) read as follows,

$$\boxed{D_k = \frac{1}{s_k}\sum_{n=0}^{N-1} x_n \cos\left(\frac{\pi k}{N}\left(n+\tfrac{1}{2}\right)\right), \quad k = 0,1,\dots,N-1} \quad \text{(DCT)} \qquad (19.2.5)$$

$$\boxed{x_n = \sum_{k=0}^{N-1} D_k \frac{1}{s_k}\cos\left(\frac{\pi k}{N}\left(n+\tfrac{1}{2}\right)\right), \quad n = 0,1,\dots,N-1} \quad \text{(IDCT)} \qquad (19.2.6)$$

$$\boxed{\sum_{n=0}^{N-1} x_n^2 = \sum_{k=0}^{N-1} D_k^2} \quad \text{(Parseval)} \qquad (19.2.7)$$

The above relationships can be understood more simply by expressing them in matrix form. Let us define the $N\times N$ matrix of DCT coefficients by its kn matrix element,

$$B_{kn} = \cos\left(\frac{\pi k}{N}\left(n+\tfrac{1}{2}\right)\right), \quad \begin{array}{l} k = 0,1,\dots,N-1 \\ n = 0,1,\dots,N-1 \end{array} \qquad (19.2.8)$$

Then, with reference to the column vectors of Eq. (19.1.1), the forward DCT and its inverse can be written in matrix form as follows,

$$\mathbf{C} = B\mathbf{x} \quad \Rightarrow \quad \mathbf{x} = B^{-1}\mathbf{C}$$

Define the diagonal matrix of the scale factors s_k,

$$S = \text{diag}\left([s_0, s_1, \dots, s_{N-1}]\right) = \begin{bmatrix} s_0 & 0 & 0 & \cdots & 0 \\ 0 & s_1 & 0 & \cdots & 0 \\ 0 & 0 & s_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & s_{N-1} \end{bmatrix}$$

Then, it can be shown that the matrix B satisfies the following orthogonality-like property from which its inverse can be determined,

$$\begin{aligned} BB^T &= S^2 = \text{diag}\left([s_0^2, s_1^2, \dots, s_{N-1}^2]\right) \\ B^{-1} &= B^T S^{-2} = B^T\,\text{diag}\left([s_0^{-2}, s_1^{-2}, \dots, s_{N-1}^{-2}]\right) \end{aligned} \qquad (19.2.9)$$

Thus, the forward and inverse DCTs can be written in the following matrix forms, which are exactly equivalent to Eqs. (19.2.1) and (19.2.2), and also to (19.2.5) and (19.2.6) since, $\mathbf{D} = S^{-1}\mathbf{C}$,

$$\boxed{\begin{aligned} \mathbf{C} &= B\mathbf{x} \\ \mathbf{x} &= B^T S^{-2}\mathbf{C} \end{aligned}} \quad \Rightarrow \quad \boxed{\begin{aligned} \mathbf{D} &= S^{-1}B\mathbf{x} = A\mathbf{x} \\ \mathbf{x} &= B^T S^{-1}\mathbf{D} = A^T\mathbf{D} \end{aligned}} \tag{19.2.10}$$

where we defined the rescaled DCT matrix,

$$A = S^{-1}B \tag{19.2.11}$$

which is *orthogonal*, that is,

$$AA^T = I_N \quad \Rightarrow \quad A^{-1} = A^T \tag{19.2.12}$$

Indeed, we have from Eq. (19.2.9),

$$AA^T = S^{-1}BB^T S^{-1} = S^{-1}S^2 S^{-1} = I_N$$

The normalized DCT matrix A can be computed for any N by the following one-line anonymous MATLAB function:

```
A = @(N) sqrt(2/N) * [ones(1,N)/sqrt(2); cos(pi*(1:N-1)'*((0:N-1)+1/2)/N)];
```

Although the above matrix formulation is very efficient in MATLAB for moderate sizes (i.e., $N \le 1024$), there is an even faster implementation based on computing the N-point DCT vector $\mathbf{C}$ from a related $2N$-point FFT. The computational steps are summarized as follows.[†] Define the extended signal and DCT column vectors of length-$2N$ obtained by appending to $\mathbf{x}$ its reversed version, and similarly, appending its negative reversed version to the DCT,

$$\begin{aligned} \mathbf{y} &= [x_0, x_1, \dots, x_{N-1}, x_{N-1}, \dots, x_1, x_0]^T \\ \mathbf{C}^{\text{ext}} &= [C_0, C_1, \dots, C_{N-2}, C_{N-1}, 0, -C_{N-1}, -C_{N-2}, \dots, -C_1]^T \end{aligned} \tag{19.2.13}$$

Then, it can be shown that the $2N$-point DFT of $\mathbf{y}$ is related to the vector $\mathbf{C}^{\text{ext}}$ by,

$$\begin{aligned} Y_k &= 2e^{j\pi k/2N} C_k^{\text{ext}}\,, \quad k = 0, 1, \dots, 2N-1\,, \quad \text{or,} \\ C_k^{\text{ext}} &= \frac{1}{2} e^{-j\pi k/2N} Y_k\,, \quad k = 0, 1, \dots, 2N-1 \end{aligned} \tag{19.2.14}$$

where we have the $2N$-point DFT pair,

$$\begin{aligned} Y_k &= \sum_{n=0}^{2N-1} y_n e^{-2\pi jk/2N}\,, \quad k = 0, 1, \dots, 2N-1 \\ y_n &= \frac{1}{2N} \sum_{n=0}^{2N-1} Y_k e^{2\pi jk/2N}\,, \quad n = 0, 1, \dots, 2N-1 \end{aligned}$$

[†] See, for example, Problem 10.26 in Chap. 10.

This leads to the following efficient computational algorithm for the forward DCT:

$$\boxed{\begin{array}{ll} \text{(a)} & \text{extend the data vector } \mathbf{x} \text{ to } \mathbf{y} \text{ as in Eq. (19.2.13)} \\ \text{(b)} & \text{compute the FFT of } \mathbf{y}\text{, that is, } Y_k,\ k = 0, 1, \dots, 2N-1 \\ \text{(c)} & \text{construct the extended } C_k^{\text{ext}} \text{ vector as in Eq. (19.2.14), and} \\ \text{(d)} & \text{retain the first } N \text{ elements, } C_k = C_k^{\text{ext}},\ k = 0, 1, \dots, N-1 \\ \text{(e)} & \text{renormalize the result, } D_k = C_k/s_k,\ k = 0, 1, \dots, N-1 \end{array}} \tag{19.2.15}$$

For the inverse DCT, the above steps are entirely reversible, that is, starting with D_k, $k = 0, 1, \dots, N-1$, first undo the normalization scale factors, $C_k = s_k D_k$, then form the length-$2N$ extended vector C_k^{ext} as in Eq. (19.2.13), and evaluate the DFT values Y_k as in Eq. (19.2.14), then, perform an inverse $2N$-point FFT to recover $\mathbf{y}$ and retain its the first N elements, $x_n = y_n$, $n = 0, 1, \dots, N-1$.

19.3 DCT Compression System

To clarify the operations shown in Fig. 19.1.1, and the possible methods of compressing the DCT coefficients, we discuss a small example, implemented in MATLAB. Define a length-40 signal, shown below in Fig. 19.3.1, and divided it up in 4 frames of length $N = 10$, with the help of the **buffer** function, and compute the DCT of all frames,[†]

```
L = 40; t = (0:L-1)/L;    %
x = sin(10*t.^2) + 2*t;   % length L=40

N = 10;                   % frame length
X = buffer(x,N);          % 10x4 matrix of frames
D = dct(X);               % 10x4 matrix of DCT coefficients

%                     X                                           D
%  --------------------------------------------------------------------------
%       0   1.0851   1.5985   0.8883          1.2623   4.9797   2.6349   6.5811
%  0.0562   1.2362   1.4259   1.2766         -0.9433  -0.7086   1.2451  -1.2616
%  0.1250   1.3833   1.2163   1.7165          0.1282  -0.3166   0.4891  -1.4116
%  0.2062   1.5205   0.9861   2.1495   DCT  -0.0959  -0.0124  -0.0916   0.1412
%  0.2998   1.6408   0.7575   2.5086   ==>   0.0286  -0.0736   0.1147  -0.2160
%  0.4056   1.7365   0.5577   2.7305         -0.0310  -0.0039  -0.0281   0.0275
%  0.5231   1.7996   0.4164   2.7699          0.0108  -0.0280   0.0432  -0.0768
%  0.6515   1.8224   0.3622   2.6134         -0.0125  -0.0016  -0.0111   0.0097
%  0.7894   1.7986   0.4175   2.2892          0.0041  -0.0106   0.0163  -0.0286
%  0.9349   1.7241   0.5943   1.8686         -0.0035  -0.0004  -0.0031   0.0026
```

We consider two methods of compression. In method-1, we pick a compression factor, $r < 1$, which defines the number of DCT coefficients to be kept from each length-N frame,

$$N_r = \text{round}(rN)\,, \qquad r_{\text{actual}} = \frac{N_r}{N} \tag{19.3.1}$$

then, sort the absolute values of the DCT coefficients in each column in descending order, and keep the highest N_r of them, setting the rest of the coefficients to zero.

[†] like the **fft** function, the **dct** function computes at once the DCT of all the columns of the input.

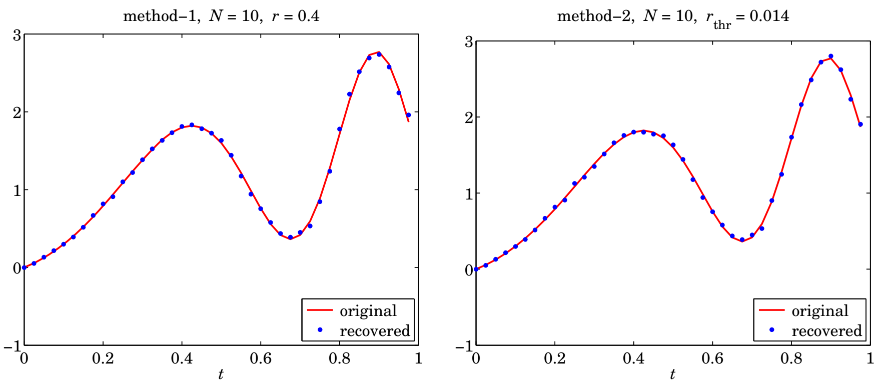


Fig. 19.3.1 Original and compressed signals, with method-1 (left) and method-2.

Because of the rounding process, the actual realized compression factor, r_{actual}, may be slightly different from r. The sorting and construction of the new DCT coefficient matrix could be done as follows,

```
M = size(D,2);                        % no. of frames
Nr = round(r*N);                      % no. of kept coefficients per frame
[~, Ir] = sort(abs(D),'descend');     % sort column-wise, descending order
Ir = Ir(1:Nr,:);                      % Ir = Nr x M matrix of sorting indices
C = zeros(size(D));                   % kept DCT coefficients
for m = 1:M                           % rebuild DCT from the kept coefficients
    Dr(:,m) = D(Ir(:,m),m);           % Dr = Nr x M matrix, sorted coefficients
    C(Ir(:,m),m) = Dr(:,m);           % C = new DCT coefficients
end
```

In a storage/retrieval or transmitting/receiving system, one would store or transmit both the index and coefficient matrices, I_r, D_r, in order to be able to re-position the kept coefficients in their original order [391], therefore, the compression factor would be, $2r$, in this case. For example, with, $r = 0.4$, we have the sorted coefficients and sorting indices,

```
%      Ir      |                  Dr
%  ------------|--------------------------------------
%  1  1  1  1  |  1.2623    4.9797    2.6349    6.5811
%  2  2  2  3  | -0.9433   -0.7086    1.2451   -1.4116
%  3  3  3  2  |  0.1282   -0.3166    0.4891   -1.2616
%  4  5  5  5  | -0.0959   -0.0736    0.1147   -0.2160
```

In method-2, we pick a threshold factor, $r_{\text{thr}} < 1$, which defines a threshold value D_{thr} for the DCT coefficients below which the coefficients are discarded, and construct a new DCT matrix that only has coefficients such that, $|D| \ge D_{\text{thr}}$, where,

$$D_{\text{thr}} = r_{\text{thr}} \cdot |D|_{\max} \tag{19.3.2}$$

This method can be implemented by the example code:

```
Dthr = r_thr * max(max(abs(D)));      % DCT threshold
I = find(abs(D) < Dthr);              % indices of DCT coeffs to be zeroed
C = D;                                % start with C = D
C(I) = 0;                             % discard coefficients below Dthr
r_actual = 1-length(I)/(N*M);         % realized compression factor
```

Applying the two methods to the DCT matrix D of the above example, we obtain the following new DCT matrices C, where we used, $r = 0.4$, for method-1, and, $r_{\text{thr}} = 0.014$, for method-2 (chosen such that both methods achieve the same actual compression ratio, $r_{\text{actual}} = 0.4$.)

```
%   method-1             C              | method-2             C
% -------------------------------------|-----------------------------------
%  1.2623    4.9797    2.6349    6.5811  |  1.2623    4.9797    2.6349    6.5811
% -0.9433   -0.7086    1.2451   -1.2616  | -0.9433   -0.7086    1.2451   -1.2616
%  0.1282   -0.3166    0.4891   -1.4116  |  0.1282   -0.3166    0.4891   -1.4116
% -0.0959         0         0         0  | -0.0959         0         0    0.1412
%       0   -0.0736    0.1147   -0.2160  |       0         0    0.1147   -0.2160
%       0         0         0         0  |       0         0         0         0
%       0         0         0         0  |       0         0         0         0
%       0         0         0         0  |       0         0         0         0
%       0         0         0         0  |       0         0         0         0
%       0         0         0         0  |       0         0         0         0
```

We note that for method-1 there are, $N_r = rN = 0.4 \cdot 10 = 4$, coefficients per frame, but that number is variable in method-2 for which the computed threshold was, $D_{\text{thr}} = r_{\text{thr}}|D|_{\max} = 0.014 \cdot 6.5811 = 0.0921$, with all coefficients with magnitudes less than that set to zero.

The actual compression factor, representing the fraction *non-zero* DCT coefficients, was the same in the two cases. The final reconstructed output was obtained by performing an inverse DCT on C and concatenating the resulting frames, with the results plotted in Fig. 19.3.1,

```
Y = idct(C);     % IDCT of all frames
y = Y(:);        % concatenate frames
y = y(1:L);      % make x,y lengths equal (in case buffer had extended X)

plot(t,x,'r-', t,y,'b.');
```

The approach also works with images, using a 2D-DCT (which is equivalent to taking the 1D-DCT of each column followed by the 1D-DCT of each row of the image). For example, the following MATLAB code based on the MATLAB documentation of the **dct2** function [393] compresses an image with an effective 40% compression ratio,

```
X = imread('cameraman.tif');          % read image, 256x256 matrix
D = dct2(X);                          % compute its 2D-DCT

Dmax = max(max(abs(D)));              % Dmax = 30393.4687
Dth = 10;                             % select a threshold
rth = Dth/Dmax;                       % with threshold factor,
                                      % rth = 3.2902e-04

C = D;
C(abs(D )< Dth) = 0;                  % compressed DCT
```

```
ra = length(find(C))/prod(size(C))    % actual compression ratio,
                                      % ra = 26617/65536 = 0.4061
Y = idct2(C);                         % inverse 2D-DCT

figure; imshowpair(X,Y,'montage')     % display images side by side
```

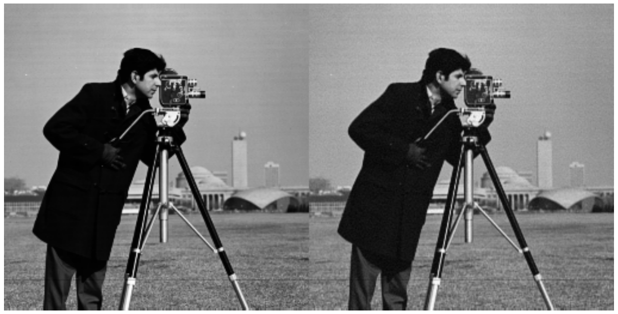
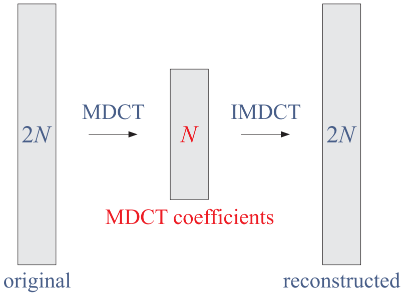


Fig. 19.3.2 Original and compressed images.

The JPEG image compression standard uses similar DCT operations, but applied to 8x8 sub-blocks of the image, and employing a standardized quantization scheme [398,399]. JPEG has been replaced by JPEG2000, which uses wavelet compression [401,402].

19.4 MDCT and Time-Domain Aliasing Cancellation

The modified DCT (MDCT) is not quite an orthogonal or invertible transform as it transforms a length-$2N$ data block into a length-N vector of MDCT coefficients, while the inverse MDCT (IMDCT) transforms the length-N vector of MDCT coefficients back to a length-$2N$ data block, which is not quite equal to the original block.



However, because the MDCT is used in blocks that are 50% overlapping, the reconstruction error introduced in one block is cancelled by the error introduced by the next block—a property referred to as *time-domain aliasing cancellation* (TDAC)—so that the original signal is reconstructed correctly.

See Refs. [403]–[410] for a review of the properties of the MDCT and its fast implementation. Here, we consider only its matrix formulation, which is fast enough for our purposes.

The N-point MDCT D_k of a $2N$-point signal x_n, and the corresponding $2N$-point inverse MDCT y_n are defined as follows,

$$\boxed{\begin{aligned} &\text{(MDCT):}\quad D_k = \sum_{n=0}^{2N-1} x_n \cos\left(\frac{\pi}{N}\left(k+\tfrac{1}{2}\right)\left(n+\tfrac{1}{2}+\tfrac{1}{2}N\right)\right), \quad k = 0,1,\dots,N-1 \\ &\text{(IMDCT):}\quad y_n = \frac{1}{N}\sum_{k=0}^{N-1} D_k \cos\left(\frac{\pi}{N}\left(k+\tfrac{1}{2}\right)\left(n+\tfrac{1}{2}+\tfrac{1}{2}N\right)\right), \quad n = 0,1,\dots,2N-1 \end{aligned}} \tag{19.4.1}$$

The precise relationship of the reconstructed signal block y_n to the original one x_n is given below, being expressed more simply by splitting the input and reconstructed length-$2N$ blocks into their upper and lower length-N sub-blocks,

$$\mathbf{x} = \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix} \xrightarrow{\text{MDCT}} \mathbf{D} \xrightarrow{\text{IMDCT}} \mathbf{y} = \begin{bmatrix} \frac{1}{2}(\mathbf{a}-\mathbf{a}_R) \\ \frac{1}{2}(\mathbf{b}+\mathbf{b}_R) \end{bmatrix} \tag{19.4.2}$$

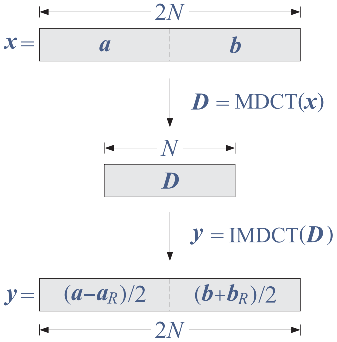
where $\mathbf{a}_R, \mathbf{b}_R$ denote the *reversed* vectors, e.g., for $N = 4$,

$$\mathbf{a} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \quad\Rightarrow\quad \mathbf{a}_R = \begin{bmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = J\mathbf{a}$$

where, defining the $N\times N$ reversing matrix J having ones along its anti-diagonal line, one may think of $\mathbf{a}_R$ as the result of the matrix operation, $\mathbf{a}_R = J\mathbf{a}$. Thus, introducing also the $N\times N$ identity matrix I, one may write Eq. (19.4.2) as,

$$\mathbf{x} = \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix} \xrightarrow{\text{MDCT}} \mathbf{D} \xrightarrow{\text{IMDCT}} \mathbf{y} = \begin{bmatrix} \frac{1}{2}(I-J)\mathbf{a} \\ \frac{1}{2}(I+J)\mathbf{b} \end{bmatrix} = \begin{bmatrix} \frac{1}{2}(I-J) & 0 \\ 0 & \frac{1}{2}(I+J) \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix} \tag{19.4.3}$$

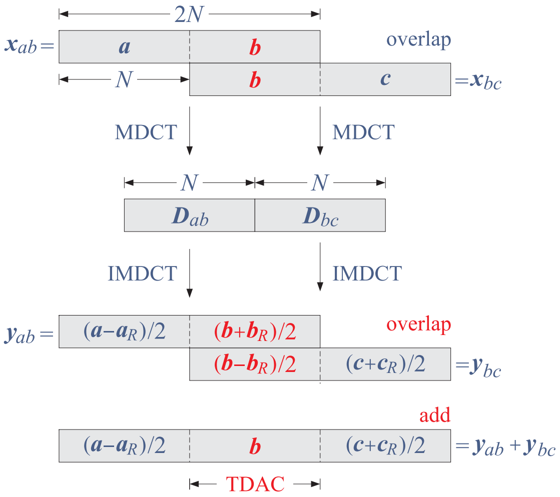
Pictorially, we have for one block,



Applying this property to two blocks that are 50% overlapping, we observe that if the overlapping reconstructed blocks are added, the second half of the first block is corrected by the first half of the second block, recovering the overlapping portion of the original blocks, that is,

$$\frac{1}{2}(\mathbf{b}+\mathbf{b}_R)+\frac{1}{2}(\mathbf{b}-\mathbf{b}_R)=\mathbf{b} \tag{19.4.4}$$

This is precisely the time-domain aliasing cancellation (TDAC) property.



$$\boxed{\begin{aligned}\mathbf{D}_{ab}&=\mathrm{MDCT}(\mathbf{x}_{ab})\\ \mathbf{D}_{bc}&=\mathrm{MDCT}(\mathbf{x}_{bc})\\ \mathbf{y}_{ab}&=\mathrm{IMDCT}(\mathbf{D}_{ab})\\ \mathbf{y}_{bc}&=\mathrm{IMDCT}(\mathbf{D}_{bc})\end{aligned}}$$

For a long signal that is split into several such 50% overlapping blocks, as shown in Fig. 19.1.2, the entire signal is recovered correctly, with the exception of the first and last length-N portions that are not overlapping—these can be fixed by padding N zeros at the beginning and end of the original signal prior to MDCT processing. Alternatively, the last and first N outputs can be replaced by NaN's or by the original input samples.

The fundamental result of Eq. (19.4.3) can be derived by considering the matrix formulation of the MDCT/IMDCT. Let us define the $N\times 2N$ MDCT transformation matrix by its kn matrix element,

$$F_{kn}=\cos\left(\frac{\pi}{N}\left(k+\tfrac{1}{2}\right)\left(n+\tfrac{1}{2}+\tfrac{1}{2}N\right)\right),\qquad \begin{aligned}k&=0,1,\dots,N-1\\ n&=0,1,\dots,2N-1\end{aligned} \tag{19.4.5}$$

Then, the MDCT and IMDCT transformations (19.4.1) can be expressed in matrix form,

$$\boxed{\begin{aligned}\mathbf{D}&=F\mathbf{x}\\ \mathbf{y}&=\frac{1}{N}F^T\mathbf{D}\end{aligned}}\quad\Rightarrow\quad \mathbf{y}=\frac{1}{N}F^TF\mathbf{x} \tag{19.4.6}$$

We may split F into its two $N\times N$ submatrices, $F=[A,B]$, defined by their matrix elements,

$$\begin{aligned}A_{kn}&=F_{kn}=\cos\left(\frac{\pi}{N}\left(k+\tfrac{1}{2}\right)\left(n+\tfrac{1}{2}+\tfrac{1}{2}N\right)\right)\\ B_{kn}&=F_{k,n+N}=\cos\left(\frac{\pi}{N}\left(k+\tfrac{1}{2}\right)\left(n+\tfrac{1}{2}+\tfrac{3}{2}N\right)\right)\end{aligned}\quad\text{with}\quad \begin{aligned}k&=0,1,\dots,N-1\\ n&=0,1,\dots,N-1\end{aligned} \tag{19.4.7}$$

Then, it can be shown that the submatrices A,B satisfy the relationships,

$$\boxed{\begin{aligned}\frac{1}{N}A^TA&=\frac{1}{2}(I-J)\\ \frac{1}{N}B^TB&=\frac{1}{2}(I+J)\\ A^TB&=B^TA=0\end{aligned}} \tag{19.4.8}$$

And, these imply Eq. (19.4.3) because,

$$\frac{1}{N}F^TF=\frac{1}{N}\begin{bmatrix}A^T\\ B^T\end{bmatrix}[A,B]=\frac{1}{N}\left[\begin{array}{c|c}A^TA & A^TB\\ \hline B^TA & B^TB\end{array}\right]=\left[\begin{array}{c|c}\frac{1}{2}(I-J) & 0\\ \hline 0 & \frac{1}{2}(I+J)\end{array}\right]$$

19.5 Princen-Bradley Windows

As depicted in Fig. 19.1.2, each frame of length-$2N$ of the input and reconstructed output is windowed with a length-$2N$ window in order to reduce the blocking effects near the endpoints of the frames. Such windows are chosen to be symmetric about their middle with their second length-N half being the reversed version of their first half,

$$\hat{\mathbf{w}}=[\underbrace{w_0,w_1,\dots,w_{N-1}}_{\mathbf{w}},\underbrace{w_{N-1},\dots,w_1,w_0}_{\mathbf{w}_R}]$$

and moreover, in order to preserve the TDAC property, the windows must satisfy the following so-called *Princen-Bradley condition* expressed in terms of the length-$2N$ window $\hat{w}_n$, or, in terms of its half portion, w_n, for, $n=0,1,\dots,N-1$,

$$\hat{w}_n^2+\hat{w}_{n+N}^2=2\quad\Rightarrow\quad w_n^2+w_{N-1-n}^2=2 \tag{19.5.1}$$

Define the diagonal matrices of the windows,

$$\hat{W}=\mathrm{diag}(\hat{\mathbf{w}})=\left[\begin{array}{c|c}W & 0\\ \hline 0 & W_R\end{array}\right],\quad W=\mathrm{diag}(\mathbf{w}),\quad W_R=\mathrm{diag}(\mathbf{w}_R)=JWJ$$

for example, for $N=4$, and $\mathbf{w}=[w_0,w_1,w_2,w_3]$, we have,

$$W = \begin{bmatrix} w_0 & 0 & 0 & 0 \\ 0 & w_1 & 0 & 0 \\ 0 & 0 & w_2 & 0 \\ 0 & 0 & 0 & w_3 \end{bmatrix}$$

$$W_R = \begin{bmatrix} w_3 & 0 & 0 & 0 \\ 0 & w_2 & 0 & 0 \\ 0 & 0 & w_1 & 0 \\ 0 & 0 & 0 & w_0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} w_0 & 0 & 0 & 0 \\ 0 & w_1 & 0 & 0 \\ 0 & 0 & w_2 & 0 \\ 0 & 0 & 0 & w_3 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Then, the windowing operation on a length-$2N$ block, assumed to be a column, can be expressed as a matrix multiplication by the diagonal window matrices,

$$\mathbf{x} = \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix} \quad \Rightarrow \quad \hat{W}\mathbf{x} = \begin{bmatrix} W\mathbf{a} \\ W_R\mathbf{b} \end{bmatrix}$$

The combined operations of windowing an input block, computing its MDCT followed by an IMDCT, and windowing the result with the same window are as follows,

$$\mathbf{x} \xrightarrow{\text{window}} \hat{W}\mathbf{x} = \begin{bmatrix} W\mathbf{a} \\ W_R\mathbf{b} \end{bmatrix} \xrightarrow{\text{MDCT/IMDCT}} \begin{bmatrix} \frac{1}{2}(W\mathbf{a} - W_R\mathbf{a}_R) \\ \frac{1}{2}(W_R\mathbf{b} + W\mathbf{b}_R) \end{bmatrix} = \mathbf{y}$$

$$\xrightarrow{\text{window}} \hat{W}\mathbf{y} = \begin{bmatrix} \frac{1}{2}(W^2\mathbf{a} - WW_R\mathbf{a}_R) \\ \frac{1}{2}(W_R^2\mathbf{b} + W_RW\mathbf{b}_R) \end{bmatrix}$$

where we made use of the properties, $W_R = JWJ$ and $J^2 = I$, which imply,

$$(W\mathbf{a})_R = JW\mathbf{a} = JWJJ\mathbf{a} = W_R\mathbf{a}_R$$

$$(W_R\mathbf{b})_R = JW_R\mathbf{b} = J^2WJ\mathbf{b} = W\mathbf{b}_R$$

When two 50% overlapping blocks are subjected to these operations and added, then the overlapping portions (i.e., second half of the first block and first half of the second) will combine as in Eq. (19.4.4), resulting in,

$$\frac{1}{2}(W_R^2\mathbf{b} + W_RW\mathbf{b}_R) + \frac{1}{2}(W^2\mathbf{b} - WW_R\mathbf{b}_R) = \frac{1}{2}(W^2 + W_R^2)\mathbf{b} \tag{19.5.2}$$

where $\mathbf{b}_R$ term is cancelled because the diagonal matrices W, W_R commute. Thus, in order to guarantee the TDAC property, the W matrix must satisfy the following condition, which is equivalent to the Princen-Bradley condition of Eq. (19.5.1),

$$\frac{1}{2}(W^2 + W_R^2) = I \tag{19.5.3}$$

A couple of window examples that satisfy this property are as follows, with the first being used in MP3 and MPEG-2 AAC formats, and the second in Vorbis [410],

$$\begin{aligned} &\text{(sine):} \quad \hat{w}_n = \sqrt{2}\,\sin\left(\frac{\pi}{2N}\left(n + \tfrac{1}{2}\right)\right), \quad n = 0, 1, \dots, 2N-1 \\ &\text{(vorbis):} \quad \hat{w}_n = \sqrt{2}\,\sin\left[\frac{\pi}{2}\sin^2\left(\frac{\pi}{2N}\left(n + \tfrac{1}{2}\right)\right)\right], \quad n = 0, 1, \dots, 2N-1 \end{aligned} \tag{19.5.4}$$

A more general procedure for constructing such windows, which is discussed in [409], begins with a typical length-$(N+1)$ window, say, f_k, $k = 0, 1, \dots, N$, that is symmetric about its middle (i.e., about $\frac{1}{2}N$), such as a Kaiser or a Hamming window, and constructs the first length-N half of the $\hat{w}_n$ window by forming the square-root of the cumulative sum of f_k,

$$\hat{w}_n = w_n = \left[\frac{2}{S}\sum_{k=0}^{n} f_k\right]^{1/2}, \quad n = 0, 1, \dots, N-1, \quad \text{where } S \equiv \sum_{k=0}^{N} f_k \tag{19.5.5}$$

then, the second length-N half is taken to be the reversed version of w_n, obtained by replacing n by $N - 1 - n$ in Eq. (19.5.5),

$$\hat{w}_{n+N} = w_{N-1-n} = \left[\frac{2}{S}\sum_{m=0}^{N-1-n} f_m\right]^{1/2} = \left[\frac{2}{S}\sum_{k=n+1}^{N} f_{N-k}\right]^{1/2}, \quad n = 0, 1, \dots, N-1$$

where we changed summation variables from m to $N - k$. Exploiting the assumed symmetry of the f_k window, i.e., $f_{N-k} = f_k$, we obtain,

$$\hat{w}_{n+N} = w_{N-1-n} = \left[\frac{2}{S}\sum_{k=n+1}^{N} f_k\right]^{1/2}, \quad n = 0, 1, \dots, N-1 \tag{19.5.6}$$

Together Eqs. (19.5.5) and (19.5.6) define a length-$2N$ window that satisfies the Princen-Bradley condition (19.5.1), indeed,

$$\hat{w}_n^2 + \hat{w}_{n+N}^2 = \frac{2}{S}\left[\sum_{k=0}^{n} f_k + \sum_{k=n+1}^{N} f_k\right] = \frac{2}{S}\left[\sum_{k=0}^{N} f_k\right] = \frac{2}{S}S = 2$$

An example of such window is the so-called *Kaiser-Bessel derived* (KBD) window, which is used in AAC and Dolby AC-3 formats—it is generated by the above procedure from an ordinary Kaiser window of length $(N+1)$ and shape parameter β,[†]

$$f_k = I_0\left(\beta\sqrt{1 - \left(\frac{k - N/2}{N/2}\right)^2}\right), \quad k = 0, 1, \dots, N \tag{19.5.7}$$

Similarly, a Hamming window would have,[‡]

$$f_k = 0.54 - 0.46\cos\left(\frac{2\pi k}{N}\right), \quad k = 0, 1, \dots, N \tag{19.5.8}$$

19.6 Derivations and Computer Experiments

Derivations

1. Using the finite geometric series, prove the following identity valid for any integer N and real-valued parameter α,

$$\sum_{n=0}^{N-1}\cos\left(\frac{\alpha}{N}\left(n + \tfrac{1}{2}\right)\right) = \frac{\sin\alpha}{2\sin\left(\dfrac{\alpha}{2N}\right)} \tag{19.6.1}$$

[†] the usual $I_0(\beta)$ denominator is not necessary since f_k gets rescaled by S.

[‡] A Kaiser window with $\beta = 5.2$ is good approximation to Hamming window.

Hints: Note the following,

$$
\begin{aligned}
&\sum_{n=0}^{N-1} z^n = \frac{z^N - 1}{z - 1} \\
&\cos\theta = \mathrm{Re}\big[e^{j\theta}\big] \\
&e^{j\theta} - 1 = 2je^{j\theta/2}\sin(\theta/2) \\
&\sin(\alpha) = 2\sin(\alpha/2)\cos(\alpha/2)
\end{aligned}
$$

Moreover, show the special limiting values, for $\alpha = 2\pi Nm$, with arbitrary integer m,

$$
\sum_{n=0}^{N-1}\cos\Big(\frac{\alpha}{N}\big(n+\tfrac{1}{2}\big)\Big)\bigg|_{a=2\pi Nm} = \frac{\sin\alpha}{2\sin\Big(\dfrac{\alpha}{2N}\Big)}\Bigg|_{a=2\pi Nm} = N(-1)^m \tag{19.6.2}
$$

Then, show the following identity for real α, β,

$$
\sum_{n=0}^{N-1}\cos\Big(\frac{\alpha}{N}\big(n+\tfrac{1}{2}\big)\Big)\cos\Big(\frac{\beta}{N}\big(n+\tfrac{1}{2}\big)\Big) = \frac{\sin(\alpha+\beta)}{4\sin\Big(\dfrac{\alpha+\beta}{2N}\Big)} + \frac{\sin(\alpha-\beta)}{4\sin\Big(\dfrac{\alpha-\beta}{2N}\Big)} \tag{19.6.3}
$$

2. To prove the matrix equation (19.2.9) for the DCT matrix B, work with the km matrix element $(BB^T)_{km}$, and using the identities of the previous question show that,

$$
(BB^T)_{km} = \sum_{n=0}^{N-1} B_{kn}B_{mn} = \frac{N}{2}\Big[\delta(k+m) + \delta(k-m)\Big], \qquad \begin{aligned} k &= 0,1,\dots,N-1 \\ m &= 0,1,\dots,N-1 \end{aligned} \tag{19.6.4}
$$

Then, explain why this is equivalent to,

$$
(BB^T)_{km} = s_k^2\,\delta(k-m), \qquad \begin{aligned} k &= 0,1,\dots,N-1 \\ m &= 0,1,\dots,N-1 \end{aligned} \tag{19.6.5}
$$

3. Consider the $N\times N$ MDCT matrices A, B defined in Eq. (19.4.7). In order to prove the relationships (19.4.8), use the results of Eqs. (19.6.1)–(19.6.3) to show the following expressions for the matrix elements,

$$
\begin{aligned}
\frac{1}{N}(A^TA)_{nm} &= \frac{1}{2}\Big[\delta(n-m) - \delta(n+m-N+1)\Big] \\
\frac{1}{N}(B^TB)_{nm} &= \frac{1}{2}\Big[\delta(n-m) + \delta(n+m-N+1)\Big] \\
(A^TB)_{nm} &= 0
\end{aligned} \tag{19.6.6}
$$

for $n = 0,1,\dots,N-1$, and, $m = 0,1,\dots,N-1$. Explain why the matrix elements of the $N\times N$ reversing matrix J are,

$$
J_{nm} = \delta(n+m-N+1), \quad n,m = 0,1,\dots,N-1
$$

Finally, explain why Eqs. (19.6.6) are equivalent to Eqs. (19.4.8).

Computer Experiment – Fast DCT and IDCT functions

Write two MATLAB functions, **dctf** and **idctf**, that implement the fast DCT algorithm listed in Eq. (19.2.15), with usage,

```
D = dctf(X);    % forward DCT
X = idctf(D);   % inverse DCT

% X = NxM matrix of frames
% D = NxM matrix of DCT coefficients
```

where the DCT is performed on each column of the $N\times M$ input matrix of signal frames, with the mth column of D holding the DCT of the mth column of X. Internally, your functions must contain a *single* FFT/IFFT call that handles all the frames at once. For this purpose, you may use the built-in FFT/IFFT functions that act column-wise.

Create a test M-file that tests your functions on a 1024 × 100 random matrix X and compares the results, as well as the computational speed, with the built-in functions, **dct** and **idct**. In addition, your test file should compare the results and speed of the computations implemented via the DCT matrix A using the anonymous MATLAB function $A(N)$ defined in Sec. 19.2. In that case, the DCT and IDCTs of all the frames can be computed simply by the single commands, $D = AX$ and $X = A^TD$.

Computer Experiment – DCT compression system

Write a MATLAB function, **dctcompr**, that implements the compression scheme depicted in Fig. 19.1.1 that incorporates the two compression methods outlined in Sec. 19.3, with usage,

```
[y,ra] = dctcompr(x,N,r,method);

% x = signal to be compressed
% N = frame length
% r = compression ratio or threshold factor (r<1)
% method = 1,2, compression method, default is 1
%
% y = compressed signal, same length as x
% ra = actual compression ratio achieved
```

In this function, you may use either the dct/idct built-in functions, or your own fast versions from the previous question. The function should put together the following operations,

$$\boxed{\begin{array}{l} X = \textbf{buffer}(\mathbf{x}, N) \\ D = \textbf{dct}(X) \\ C = f(D, r) = \text{method 1 or 2} \\ Y = \textbf{idct}(C) \\ \mathbf{y} = \text{concatenate } Y \text{ columns} \end{array}} \tag{19.6.7}$$

Write an M-file that tests your function on the example of Sec. 19.3, and, moreover, it applies the function to a short audio signal, e.g., of duration of 3-4 sec, sampled at 44.1 kHz. Run your program with smaller and smaller values of r (for method 1) until you can hear audible distortions. Include your test file and description of your results with your report.

Computer Experiment – DCT Basis Functions

The inverse DCT of Eq. (19.2.6) expresses a length-N signal x_n as a linear combination of N co-sinusoidal basis functions, that is,

$$x_n = \sum_{k=0}^{N-1} D_k F_k(n)\,, \quad n = 0, 1, \dots, N-1 \tag{19.6.8}$$

where the basis functions $F_k(n)$ are defined as follows, for $k = 0, 1, \dots, N-1$,

$$F_k(n) = \frac{1}{s_k}\cos\left(\frac{\pi k}{N}\left(n + \tfrac{1}{2}\right)\right)\,, \quad n = 0, 1, \dots, N-1 \tag{19.6.9}$$

To gain an understanding of what these functions look like, compute and plot all of them for the cases $N = 8$ and $N = 16$. You may plot them as staircase plots for clarity, as shown in Fig. 19.6.1 at the end.

Computer Experiment – Princen-Bradley Windows

Write a MATLAB function, **pbwin.m**, that generates the three types of Princen-Bradley windows mentioned in Sec. 19.5, namely, sine, vorbis, and KBD windows. It should have usage:

```
% w = pbwin(N,type,beta);
%
% N = window half-length
% type = 0,1,2,3, for rectangular, sine, vorbis, KBD
% beta = Kaiser shape parameter when type=3
%
% w = length-2N window, column vector
```

To help you debug your functions, here are their values for $N = 4$ and $\beta = 5$, see also Fig. 19.6.2 at the end,

```
%   sine     vorbis      KBD
% ----------------------------
%  0.2759    0.0845    0.1836
%  0.7857    0.6591    0.7356
%  1.1759    1.2512    1.2078
%  1.3870    1.4117    1.4023
%  1.3870    1.4117    1.4023
%  1.1759    1.2512    1.2078
%  0.7857    0.6591    0.7356
%  0.2759    0.0845    0.1836
```

and for each, we can verify the Princen-Bradley condition,

```
% w(1:N).^2 + w(N+1:2*N).^2
%
% ans =
%    2.0000
%    2.0000
%    2.0000
%    2.0000
```

Computer Experiment – MDCT and IMDCT Functions

Write two functions, **mdct** and **imdct**, that implement the forward and inverse MDCT transformations defined in Eq. (19.4.1), with usage:

```
D = mdct(X);     % forward MDCT
Y = imdct(D);    % inverse MDCT

% X = (2N)xM matrix of frames
%
% D = NxM matrix of MDCT coefficients
%
% Y = (2N)xM matrix of frames
```

You may use the matrix form of Eq. (19.4.6), so that the MDCTs and IMDCTs of a $(2N)\times M$ matrix of frames, X, would be simply,

$$D = FX\,, \quad Y = \frac{1}{N}F^T D$$

where the $(2N)\times N$ matrix F is given by Eq. (19.4.5), and should be constructed in a fully vectorized way, i.e., no loops.

Computer Experiment – MDCT/TDAC compression system

Write a MATLAB function, **mdctcompr**, that uses your functions **mdct**, **imdct**, **pbwin**, and implements the MDCT/TDAC compression scheme depicted in Fig. 19.1.2. It must have usage,

```
[y,ra] = mdctcompr(x,N,rth,win,beta);

% x = signal to be compressed
% N = MDCT length, frame length = 2*N
% rth = compression threshold, rth<1, (rth=0 for no compression)
% win = 0,1,2,3, for rectangular, sine, vorbis, KBD windows
% beta = Kaiser shape parameter when win = 3
%
% y = compressed signal
% ra = actual compression ratio achieved
```

Your function must incorporate the following operations,

$X = \mathbf{buffer}(\mathbf{x}, 2N, N, \text{‘}\texttt{nodelay}\text{’})$	% create 50% overlapping frames
$\mathbf{w} = \mathbf{pbwin}(N, \texttt{win}, \beta)$	% length-$2N$ window (column)
$W = \mathbf{repmat}(w, 1, M)$	% replicate window, M = number of frames
$D = \mathbf{mdct}(W.\!*X)$	% MDCT of windowed frames
$C = f(D, r_{\text{thr}})$	% compress MDCT using threshold method
$Y = W.\!*\,\mathbf{imdct}(C)$	% inverse MDCT followed by windowing
$\mathbf{y} = \mathbf{ola}(Y, N)$	% overlap/add frames
$\mathbf{y} = \mathbf{y}(1:\text{length}(\mathbf{x}))$	% make $\mathbf{x}, \mathbf{y}$ lengths equal

(19.6.10)

For simplicity, use only the threshold compression method (i.e., method-2 of Sec. 19.3) that discards all MDCT coefficients that are below a threshold, i.e., $|D| < r_{\text{thr}} \cdot |D|_{\max}$.

The OLA operation is the same as the one that we considered in the phase vocoder case, except here we must use frames of length $2N$ and hop size of N samples. As a reminder, given the $(2N)\times M$ output frame matrix Y, the following partially vectorized loop will reconstruct the signal $\mathbf{y}$ from Y,

```
N = size(Y,1)/2;    % Y has size 2N x M
M = size(Y,2);      % number of frames

L = N*M + N;        % length of y, before made equal to length(x)

y = zeros(L,1);     % pre-allocate y

n = (1:2*N)';       % work with column vectors

for m = 0:M-1
    y(m*N + n) = y(m*N + n) + Y(:,m+1);
end
```

Please carry out the following experiments:

1. Test your function on the same audio file that you used for the DCT case, but adjust the threshold r_{thr} in order to achieve an actual compression ratio of about $r_a = 0.15$–0.20.

2. Consider 100 samples of the same signal example of Sec. 19.3,

```
t = 0:0.01:0.99;           % 100 time instants in the interval [0,1)
x = sin(10*t.^2) + 2*t;    % signal samples
```

Using the parameters $N = 20$, $r_{\text{thr}} = 0.005$, and a KBD window with shape parameter $\beta = 15$, calculate the compressed output signal $\mathbf{y}$ and the achieved compression ratio r_a, and plot both $\mathbf{x}$ and $\mathbf{y}$ versus t on the same graph. The TDAC property will be effective for the entire length of the signal but for the first N and last N output samples.

Repeat by using the higher threshold, $r_{\text{thr}} = 0.05$, which means less accuracy since more DCT coefficients are thrown out. See some example graphs in Fig. 19.6.3 at the end that also show the overlapping frames.

3. To help you debug your **mdctcompr** function and to visualize the TDAC property, consider the following length-28 signal,

```
x = (1:28)';
```

Using $N = 4$ and a KBD window with $\beta = 6$, divide $\mathbf{x}$ into six 50% overlapping frames of length $2N = 8$, then window each frame, perform its MDCT followed immediately by an IMDCT, and window again each output frame. List columnwise the input $\mathbf{x}$, as well as the windowed/IMDCTs, and overlap-add all columns to obtain the final reconstructed output, $\mathbf{y}$, displayed in the last column, which should agree exactly with $\mathbf{x}$ except for the first 4 and last 4 outputs.

```
%   x      y1       y2       y3       y4       y5       y6       y
% ---------------------------------------------------------------
%   1   -0.34        0        0        0        0        0   -0.34
%   2   -0.80        0        0        0        0        0   -0.80
%   3    1.39        0        0        0        0        0    1.39
%   4    3.88        0        0        0        0        0    3.88
%   5    5.65    -0.65        0        0        0        0       5
%   6    7.53    -1.53        0        0        0        0       6
%   7    4.34     2.66        0        0        0        0       7
%   8    0.49     7.51        0        0        0        0       8
%   9       0     9.97    -0.97        0        0        0       9
%  10       0    12.27    -2.27        0        0        0      10
%  11       0     7.07     3.93        0        0        0      11
%  12       0     0.86    11.14        0        0        0      12
%  13       0        0    14.28    -1.28        0        0      13
%  14       0        0    17.00    -3.00        0        0      14
%  15       0        0     9.80     5.20        0        0      15
%  16       0        0     1.24    14.76        0        0      16
%  17       0        0        0    18.59    -1.59        0      17
%  18       0        0        0    21.73    -3.73        0      18
%  19       0        0        0    12.53     6.47        0      19
%  20       0        0        0     1.61    18.39        0      20
%  21       0        0        0        0    22.91    -1.91      21
%  22       0        0        0        0    26.46    -4.46      22
%  23       0        0        0        0    15.26     7.74      23
%  24       0        0        0        0     1.99    22.01      24
%  25       0        0        0        0        0    27.22   27.22
%  26       0        0        0        0        0    31.20   31.20
%  27       0        0        0        0        0    17.99   17.99
%  28       0        0        0        0        0     2.36    2.36
```

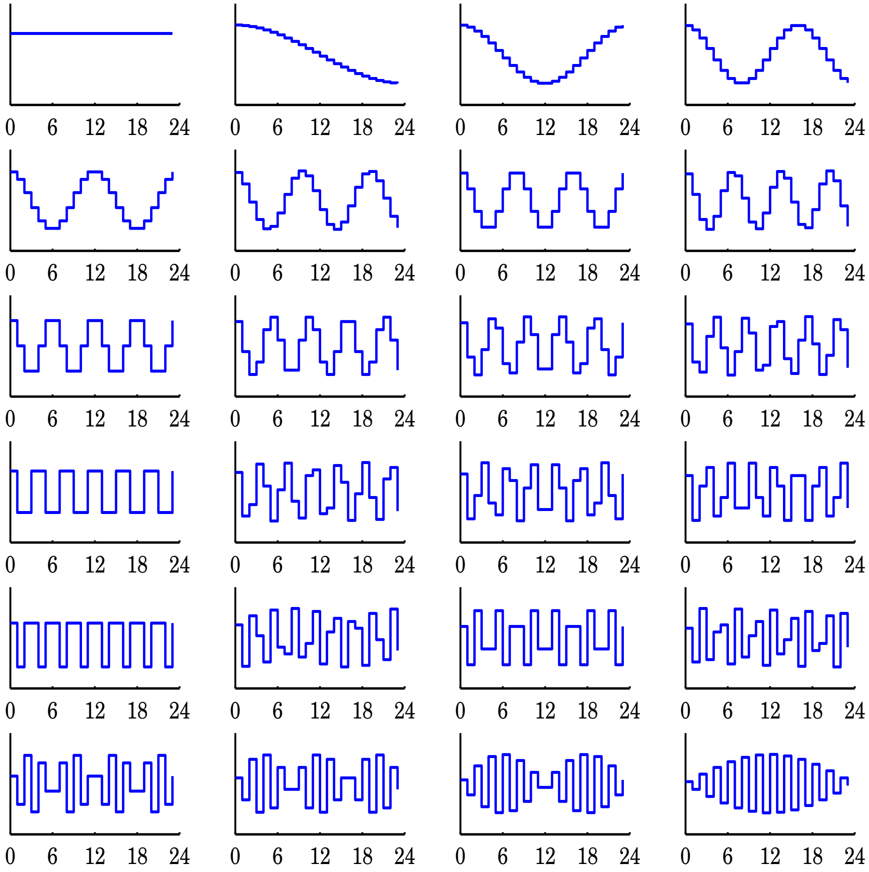


Fig. 19.6.1 DCT basis functions, $N = 24$.

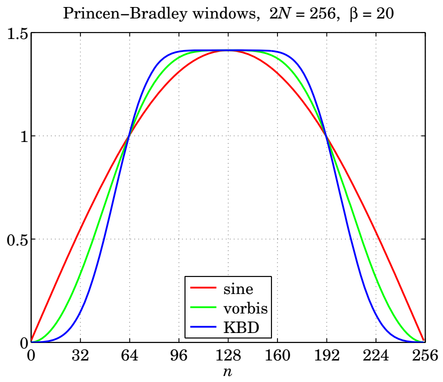


Fig. 19.6.2 Princen-Bradley windows.

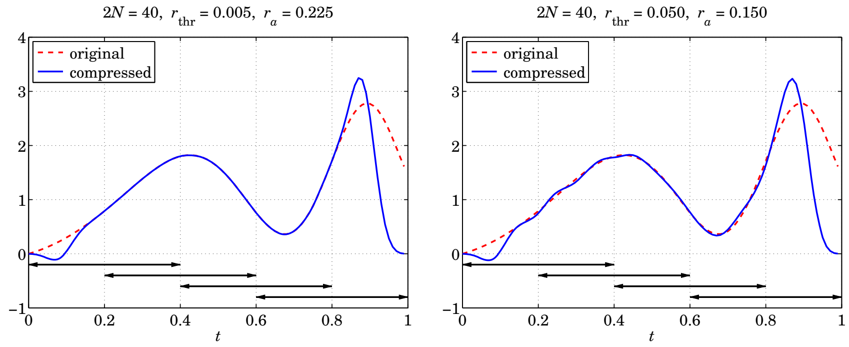


Fig. 19.6.3 MDCT/TDAC compression example with KBD window and four overlapping frames.

20 Discrete Wavelet Transforms

Over the past two decades, wavelets have become useful signal processing tools for signal representation, compression, and denoising [413–581]. There exist several books on the subject [413–434], and several tutorial reviews [435–456]. The theory of wavelets and multiresolution analysis is by now very mature [457–509] and has been applied to a remarkably diverse range of applications, such as image compression and coding, JPEG2000 standard, FBI fingerprint compression, audio signals, numerical analysis and solution of integral equations, electromagnetics, biomedical engineering, astrophysics, turbulence, chemistry, infrared spectroscopy, power engineering, economics and finance, bioinformatics, characterization of long-memory and fractional processes, and statistics with regression and denoising applications [510–581].

In this chapter,[†] we present a short review of wavelet concepts, such as multiresolution analysis, dilation equations, scaling and wavelet filters, filter banks, discrete wavelet transforms in matrix and convolutional forms, wavelet denoising, and undecimated wavelet transforms. Our discussion emphasizes computational aspects.

20.1 Multiresolution Analysis

Wavelet multiresolution analysis expands a time signal into components representing different scales—from a coarser to a finer resolution. Each term in the expansion captures the signal details at a particular scale level. The expansion is defined in terms of a sequence of nested closed subspaces V_j of the space L2R of square integrable functions on the real line $\mathbb{R}$:

$$\cdots \subset V_{-2} \subset V_{-1} \subset V_0 \subset V_1 \subset V_2 \subset \cdots \subset \mathrm{L2}R \tag{20.1.1}$$

The space V_j approximates a signal at a scale j with a resolution of 2^{-j} time units. Roughly speaking, if T_0 is the sampling time interval in subspace V_0, then the sampling interval in V_j will be $T_j = 2^{-j}T_0$, which is coarser if $j < 0$, and finer if $j > 0$. The union

[†] adapted from the author's book on *Applied Optimum Signal Processing* [45]

of the V_j subspaces is the entire L2R space, and their intersection, the zero function:

$$\lim_{j\to\infty} V_j = \bigcup_{j=-\infty}^{\infty} V_j = \text{L2}R\,, \qquad \lim_{j\to-\infty} V_j = \bigcap_{j=-\infty}^{\infty} V_j = \{0\} \tag{20.1.2}$$

The spaces V_j have a special structure, being defined as the *linear spans* of the scaled and translated replicas of a single function $\phi(t)$, called the *scaling function*, or the *father wavelet*, which can be of compact support. The scaled/translated replicas of $\phi(t)$ are defined for any integers j, n by:

$$\phi_{jn}(t) = 2^{j/2}\phi(2^j t - n) \tag{20.1.3}$$

The functions $\phi_{jn}(t)$ are orthonormal for each fixed j, and form a basis of V_j. The orthonormality condition is defined with respect to the L2R inner product:[†]

$$(\phi_{jn}, \phi_{jm}) = \int_{-\infty}^{\infty} \phi_{jn}(t)\phi_{jm}(t)\,dt = \delta_{nm} \tag{20.1.4}$$

The factor $2^{j/2}$ in Eq. (20.1.3) serves to preserve the unit norm of ϕ_{jn} for each j. Conditions (20.1.1)–(20.1.4) are strong constraints and it is remarkable that such functions $\phi(t)$ exist other than the simple Haar function defined to be unity over $0 \le t \le 1$ and zero otherwise. Fig. 20.1.1 shows three examples, the Haar, and the Daubechies D_2 and D_3 cases, all of which have compact support (the support of D_2 is 3 time units and that of D_3, 5 units). The figure also shows a related function $\psi(t)$ derived from $\phi(t)$, called the *wavelet function*, or the *mother wavelet*.

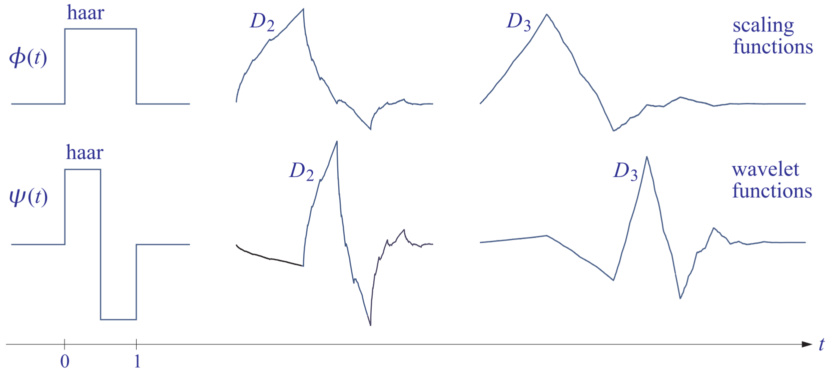


Fig. 20.1.1 Haar, Daubechies D_2 and D_3 scaling and wavelet functions $\phi(t), \psi(t)$.

Fig. 20.1.2 shows the scaled versions of the scaling and wavelet functions (for the D_2 case). Each successive copy $\phi(t), \phi(2t), \phi(2^2t), \phi(2^3t)$, etc., is compressed by a factor of two relative to the previous one, so that for higher and higher values of j, the basis function $\phi(2^j t)$ is capable of capturing smaller and smaller signal details.

[†] In this chapter, all time signals are assumed to be real-valued.

The *projection* of an arbitrary signal $f(t) \in$ L2R onto the subspace V_j is defined by the following expansion in the ϕ_{jn} basis:

$$f_j(t) = \sum_n c_{jn}\phi_{jn}(t) = \sum_n c_{jn}2^{j/2}\phi(2^j t - n) \tag{20.1.5}$$

with coefficients following from the orthonormality of $\phi_{jn}(t)$:

$$c_{jn} = (f_j, \phi_{jn}) = (f, \phi_{jn}) = \int_{-\infty}^{\infty} f(t)\phi_{jn}(t)\,dt = \int_{-\infty}^{\infty} f(t)2^{j/2}\phi(2^j t - n)\,dt \tag{20.1.6}$$

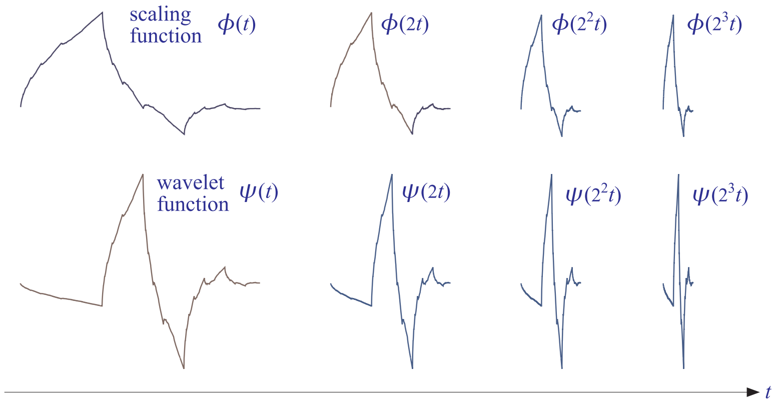


Fig. 20.1.2 Daubechies D_2 functions $\phi(t), \psi(t)$ and their compressed versions.

The projection $f_j(t)$ can be thought of as an approximation of $f(t)$ at scale j with time resolution of 2^{-j}. Because $V_i \subset V_j$ for $i \le j$, the signal $f_j(t)$ incorporates information about $f(t)$ from all coarser resolutions (cf. Eq. (20.2.8)).

The significance of the wavelet function $\psi(t)$ is that the orthogonal complement $V_j^\perp$ of V_j with respect to L2R is actually spanned by the scaled and translated versions of ψ, that is, $\psi_{in}(t) = 2^{i/2}\psi(2^i t - n)$ for $i \ge j$, which are orthogonal to $\phi_{jn}(t)$, and are also mutually orthonormal,

$$(\phi_{jn}, \psi_{im}) = 0\,, \quad i \ge j\,, \qquad (\psi_{in}, \psi_{i'n'}) = \delta_{ii'}\delta_{nn'} \tag{20.1.7}$$

Thus, we have the direct sum L2$R = V_j \oplus V_j^\perp$, resulting in the decomposition of $f(t)$ into two orthogonal parts:

$$f(t) = f_j(t) + w_j(t)\,, \quad f_j(t) \in V_j\,, \quad w_j(t) \in V_j^\perp\,, \quad f_j(t) \perp w_j(t) \tag{20.1.8}$$

The component $w_j(t)$ is referred to as the “detail,” and incorporates the details of $f(t)$ at all the higher resolution levels $i \ge j$, or finer time scales $2^{-i} \le 2^{-j}$. It admits the ψ-basis expansion:

$$w_j(t) = \sum_{i\ge j}\sum_n d_{in}\psi_{in}(t) = \sum_{i\ge j}\sum_n d_{in}2^{i/2}\psi(2^i t - n) \tag{20.1.9}$$

with detail coefficients $d_{in}=(w_j,\psi_{in})=(f,\psi_{in})$. In summary, one form of the multiresolution decomposition is,

$$\boxed{f(t)=f_j(t)+w_j(t)=\sum_n c_{jn}\phi_{jn}(t)+\sum_{i=j}^{\infty}\sum_n d_{in}\psi_{in}(t)} \qquad (20.1.10)$$

Another form is obtained in the limit $j\to-\infty$. Since $V_{-\infty}=\{0\}$, we have $f_{-\infty}(t)=0$, and we obtain the representation of $f(t)$ purely in terms of the wavelet basis $\psi_{in}(t)$:

$$\boxed{f(t)=\sum_{i=-\infty}^{\infty}\sum_n d_{in}\psi_{in}(t)}\,,\qquad d_{in}=\int_{-\infty}^{\infty}f(t)\psi_{in}(t)dt \qquad (20.1.11)$$

Yet another, and most practical, version of the multiresolution decomposition is obtained by noting that $V_\infty = \text{L2}R$. We may assume then that our working signal $f(t)$ belongs to some V_J for a sufficiently large value of J, representing the highest desired resolution, or finest scale. Since $f(t)\in V_J$, it follows from the decomposition $f(t)=f_J(t)+w_J(t)$ that $w_J(t)=0$, which implies that $d_{in}=0$ for $i\ge J$, and therefore,

$$f(t)=f_J(t)=\sum_n c_{Jn}\phi_{Jn}(t) \qquad (20.1.12)$$

Combining this with Eq. (20.1.10) applied at some lower resolution $j<J$, we obtain the two alternative forms (cf. Eq. (20.2.10)):

$$\boxed{f(t)=\sum_n c_{Jn}\phi_{Jn}(t)=\sum_n c_{jn}\phi_{jn}(t)+\sum_{i=j}^{J-1}\sum_n d_{in}\psi_{in}(t)=f_j(t)+w_j(t)} \qquad (20.1.13)$$

The mapping of the expansion coefficients from level J to levels j through $J-1$,

$$c_{Jn}\to\{c_{jn};\ d_{in},\, j\le i\le J-1\} \qquad (20.1.14)$$

is essentially the discrete wavelet transform (DWT). For sufficiently large J, the coefficients c_{Jn} can be taken to be the time samples of $f(t)$, sampled at the rate $f_s=2^J$, or sampling time interval $T_J=2^{-J}$. To see this, we note that the function $2^J\phi(2^Jt-n)$ tends to a Dirac delta function for large J (see [413] for a proof), so that,

$$2^J\phi(2^Jt-n)\approx 2^J\delta(2^Jt-n)=\delta(t-n2^{-J}) \qquad (20.1.15)$$

Therefore, $2^{J/2}\phi(2^Jt-n)\approx 2^{-J/2}\delta(t-n2^{-J})$, and Eq. (20.1.6) gives,

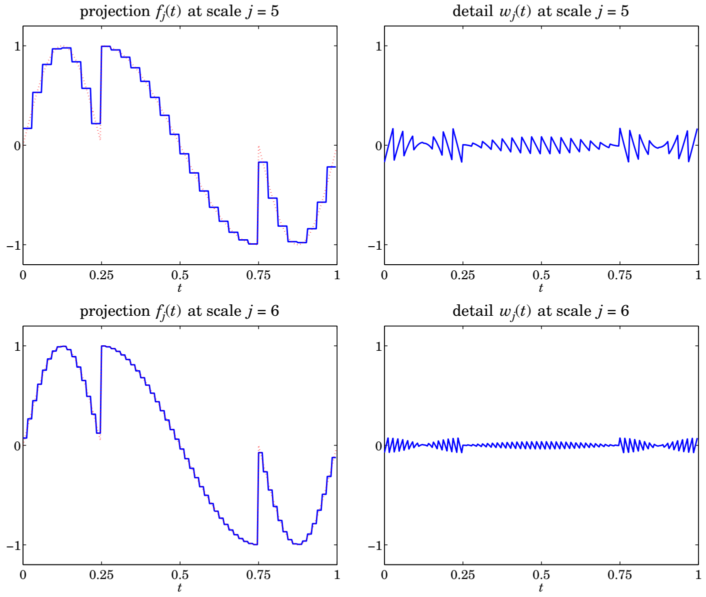
$$c_{Jn}\approx\int_{-\infty}^{\infty}f(t)2^{-J/2}\delta(t-n2^{-J})\Phi_0dt = 2^{-J/2}f(n2^{-J}) \qquad (20.1.16)$$

In practice, we may ignore the factor $2^{-J/2}$ and set simply $c_{Jn}=f(n2^{-J})=f(nT_J)$. The coefficients c_{Jn} serve as the input to the discrete wavelet transform. The approximation of c_{Jn} by the time samples is usually adequate, although there exist more precise ways to initialize the transform.

Example 20.1.1: An example of the decomposition (20.1.13) is shown in Fig. 20.1.3 using the Haar basis. The original signal (dotted line) is defined by sampling the following discontinuous function at $N=2^8=256$ equally spaced points over the interval $0\le t\le 1$,

$$f(t)=\begin{cases}\sin(4\pi t)\,, & 0\le t<0.25\\ \sin(2\pi t)\,, & 0.25\le t<0.75\\ \sin(4\pi t)\,, & 0.75\le t<1\end{cases} \qquad (20.1.17)$$

Thus, the highest resolution level is $J=\log_2 N=8$. The upper graphs show the components $f_j(t), w_j(t)$ for the lower resolution level of $j=5$. The bottom graphs correspond to $j=6$. As j increases, the step-function approximation becomes more accurate and captures better the two sharp breaks of the original signal. For each j, the sums of the left and right graphs make up the original signal.


projection $f_j(t)$ at scale j = 5

detail $w_j(t)$ at scale j = 5

projection $f_j(t)$ at scale j = 6

detail $w_j(t)$ at scale j = 6

Fig. 20.1.3 Haar-basis projections $f_j(t), w_j(t)$ from scale $J=8$ to scales $j=5,6$.

Fig. 20.1.4 shows the case of using the Daubechies D_3 wavelet basis for the same signal (20.1.17). The following MATLAB code generates the top graphs in the two figures:

```
J = 8; N = 2^J;
t1 = (0:N/4-1)'/N; t2 = (N/4:3*N/4-1)'/N; t3 = (3*N/4:N-1)'/N;
t = [t1; t2; t3];
y = [sin(4*pi*t1); sin(2*pi*t2); sin(4*pi*t3)];    % define signal

h = daub(1);                                       % use h=daub(3) for Fig. 20.1.4
j = 5; Y = dwtdec(y,h,j);                          % DWT decomposition to level j
fj = Y(:,1); wj = sum(Y(:,2:end),2);               % approximation f_j(t) and detail w_j(t)
```

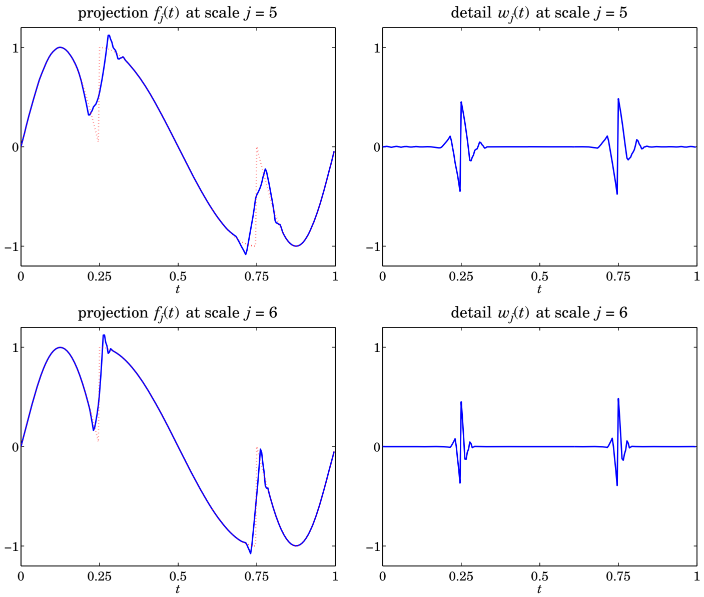


Fig. 20.1.4 Daubechies-D_3 projections $f_j(t), w_j(t)$ from scale $J=8$ to scales $j=5,6$.

```
figure; plot(t,fj, t,y,':'); figure; plot(t,wj);          % left, right graphs
```

The function `dwtdec` is explained in Sec. 20.5, but we mention here that its output Y is an $N\times(J-j+1)$ matrix whose first column holds the projection $f_j(t)$, and the sum of its other columns are the detail $w_j(t)$. □

20.2 Dilation Equations

The subspaces V_j have even more interesting structure than described so far. Since $V_0\subset V_1$, it follows that the scaling function $\phi(t)\in V_0$ can be expanded in the basis $\phi_{1n}(t)=2^{1/2}\phi(2t-n)$ that spans V_1, that is, there must exist coefficients h_n such that

$$\boxed{\phi(t)=\sum_n h_n\,2^{1/2}\phi(2t-n)}\qquad\text{(dilation equation)}\tag{20.2.1}$$

which is known as the dilation or refinement equation. The coefficients h_n are given by:

$$h_n=(\phi,\phi_{1n})=2^{1/2}\int_{-\infty}^{\infty}\phi(t)\phi(2t-n)\,dt\tag{20.2.2}$$

Moreover, the wavelet function $\psi(t)$ and its translates $\psi_{0n}=\psi(t-n)$ form an orthonormal basis for the orthogonal complement of V_0 relative to V_1, that is, the space

$W_0=V_1\setminus V_0$, so that we have the direct-sum decomposition:

$$V_0\oplus W_0=V_1\tag{20.2.3}$$

The space W_0 is referred to as the "detail" subspace. Because $\psi(t)\in W_0\subset V_1$, it also can be expanded in the $\phi_{1n}(t)$ basis, as in Eq. (20.2.1),

$$\boxed{\psi(t)=\sum_n g_n\,2^{1/2}\phi(2t-n)}\tag{20.2.4}$$

In a similar fashion, we have the decomposition $V_j\oplus W_j=V_{j+1}$, for all j, with W_j being spanned by the scaled/translated ψ-basis, $\psi_{jn}(t)=2^{j/2}\psi(2^jt-n)$. The dilation equations can also be written with respect to the ϕ_{jn},ψ_{jn} bases. For example,

$$\phi_{jk}(t)=2^{j/2}\phi(2^jt-k)=\sum_m h_m 2^{(j+1)/2}\phi(2^{j+1}t-2k-m)=\sum_m h_m\phi_{j+1,m+2k}(t)$$

and similarly for $\psi_{jk}(t)$. Thus, we obtain the alternative forms,

$$\begin{aligned}\phi_{jk}(t)&=\sum_m h_m\phi_{j+1,m+2k}(t)=\sum_n h_{n-2k}\phi_{j+1,n}(t)\\ \psi_{jk}(t)&=\sum_m g_m\phi_{j+1,m+2k}(t)=\sum_n g_{n-2k}\phi_{j+1,n}(t)\end{aligned}\tag{20.2.5}$$

Using the orthogonality property $(\phi_{j+1,n},\phi_{j+1,m})=\delta_{nm}$, we have the inner products,

$$\begin{aligned}h_{n-2k}&=(\phi_{jk},\phi_{j+1,n})\\ g_{n-2k}&=(\psi_{jk},\phi_{j+1,n})\end{aligned}\tag{20.2.6}$$

Also, because $\phi_{j+1,n}(t)$ is a basis for $V_{j+1}=V_j\oplus W_j$, it may be expanded into its two orthogonal parts belonging to the subspaces V_j and W_j, which are in turn spanned by ϕ_{jk} and ψ_{jk}, that is,

$$\phi_{j+1,n}=\sum_k(\phi_{j+1,n},\phi_{jk})\phi_{jk}+\sum_k(\phi_{j+1,n},\psi_{jk})\psi_{jk}$$

Using (20.2.6), we may rewrite this as,

$$\phi_{j+1,n}(t)=\sum_k h_{n-2k}\phi_{jk}(t)+\sum_k g_{n-2k}\psi_{jk}(t)\tag{20.2.7}$$

Eqs. (20.2.5)–(20.2.7) are the essential tools for deriving Mallat's pyramidal multiresolution algorithm for the discrete wavelet transform.

The various decompositions discussed in Sec. 20.1 can be understood in the geometric language of subspaces. For example, starting at level j and repeating the direct-sum decomposition, and using $V_{-\infty}=\{0\}$, we obtain the representation of the subspace V_j,

$$V_j=V_{j-1}\oplus W_{j-1}=V_{j-2}\oplus W_{j-2}\oplus W_{j-1}=\cdots=\bigoplus_{i=-\infty}^{j-1}W_i\tag{20.2.8}$$

which states that V_j incorporates the details of all coarser resolutions. Similarly, increasing j and using $V_\infty = \mathrm{L2}R$, we obtain the subspace interpretation of Eq. (20.1.10),

$$\begin{aligned}
V_{j+1} &= V_j \oplus W_j \\
V_{j+2} &= V_{j+1} \oplus W_{j+1} = V_j \oplus W_j \oplus W_{j+1} \\
V_{j+3} &= V_{j+2} \oplus W_{j+2} = V_j \oplus W_j \oplus W_{j+1} \oplus W_{j+2} \\
&\cdots \\
\mathrm{L2}R &= V_j \oplus \big(W_j \oplus W_{j+1} \oplus W_{j+2} \oplus \cdots\big) = V_j \oplus V_j^{\perp}
\end{aligned} \tag{20.2.9}$$

which explains the remark that the term $w_j(t)$ in (20.1.10) incorporates all the higher-level details. Finally, going from level $j < J$ to level $J-1$, we obtain the geometric interpretation of Eq. (20.1.13),

$$V_J = V_j \oplus \big(W_j \oplus W_{j+1} \oplus \cdots \oplus W_{J-1}\big)\,, \quad j < J \tag{20.2.10}$$

The coefficients h_n define a lowpass filter $H(z) = \sum_n h_n z^{-n}$ called the *scaling filter*. Similarly, g_n define a highpass filter $G(z)$, the *wavelet filter*. The coefficients h_n, g_n must satisfy certain orthogonality relations, discussed below, that follow from the dilation equations (20.2.5).

The filters h_n, g_n can be IIR or FIR, but the FIR ones are of more practical interest, and lead to functions $\phi(t), \psi(t)$ of compact support. Daubechies [413] has constructed several families of such FIR filters: the minimum-phase family or daublets, the least-asymmetric family or symmlets, and coiflets. The MATLAB function daub incorporates these three families:

```
h = daub(K,type);        % Daubechies scaling filters - daublets, symmlets, coiflets
```

```
h = daub(K,1) = Daublets K = 1,2,3,4,5,6,7,8,9,10, denoted as D_K (D_1 = Haar)
h = daub(K,2) = Symmlets K = 4,5,6,7,8,9,10, denoted as S_K
h = daub(K,3) = Coiflets K = 1,2,3,4,5
h = daub(K)   = equivalent to daub(K,1)

Daublets (minimum phase) have length = 2K and K vanishing moments for ψ(t).
Symmlets (least asymmetric) have length = 2K and K vanishing moments for ψ(t).
Coiflets have length = 6K and 2K vanishing moments for ψ(t), and 2K-1 for φ(t).
         for coiflets, h(n) is indexed over -2K <= n <= 4K-1

all filters have norm(h) = 1 and sum(h) = √2
```

For example, the scaling filters for the Haar, and daublet D_2 and D_3 cases, whose $\phi(t), \psi(t)$ functions were shown in Fig. 20.1.1, are obtained from the MATLAB calls:

```
h = daub(1) ⇒ h = [0.7071    0.7071]
h = daub(2) ⇒ h = [0.4830    0.8365    0.2241   -0.1294]
h = daub(3) ⇒ h = [0.3327    0.8069    0.4599   -0.1350   -0.0854    0.0352]
```

The filters h_n can be taken to be causal, i.e., h_n, $0 \le n \le M$, where M is the filter order, which is odd for the above three families, with $M = 2K-1$ for daublets and symmlets, and $M = 6K-1$ for coiflets (these are defined to be slightly anticausal, over $-2K \le n \le 4K-1$). The parameter K is related to certain flatness constraints or moment constraints for h_n at the Nyquist frequency $\omega = \pi$.

The filters g_n are defined to be the *conjugate* or *quadrature* mirror filters to h_n, that is, $g_n = (-1)^n h_n^R$, where $h_n^R = h_{M-n}$, $n = 0, 1, \dots, M$ is the reversed version of h_n.

In the z-domain, we have $H^R(z) = z^{-M}H(z^{-1})$, while multiplication by $(-1)^n$ is equivalent to the substitution $z \to -z$, therefore, $G(z) = H^R(-z) = (-z)^{-M}H(-z^{-1})$. In the frequency domain, this reads:

$$\boxed{G(\omega) = e^{-jM(\omega+\pi)}H^*(\omega+\pi) \quad \Leftrightarrow \quad g_n = (-1)^n h_{M-n}\,, \quad 0 \le n \le M} \tag{20.2.11}$$

with the frequency-responses defined by:

$$G(\omega) = \sum_{n=0}^{M} g_n e^{-j\omega n}\,, \quad H(\omega) = \sum_{n=0}^{M} h_n e^{-j\omega n} \tag{20.2.12}$$

The function cmf implements this definition:

```
g = cmf(h);        % conjugate mirror filter
```

For example, if $\mathbf{h} = [h_0, h_1, h_2, h_3]$, then, $\mathbf{g} = [h_3, -h_2, h_1, -h_0]$, e.g., we have for the daublet D_2:

```
h = daub(2) = [ 0.4830    0.8365    0.2241   -0.1294]
g = cmf(h)  = [-0.1294   -0.2241    0.8365   -0.4830]
```

Fig. 20.2.1 shows the magnitude responses of the Haar, Daubechies D_2, and Symmlet S_6 scaling and wavelet filters.

For all scaling filters, the DC gain of $H(\omega)$, and the Nyquist gain of $G(\omega)$, are equal to $\sqrt{2}$ because of the conditions (which are a consequence of the dilation equation):

$$\begin{aligned}
H(0) &= \sum_{n=0}^{M} h_n = \sqrt{2} \\
G(\pi) &= \sum_{n=0}^{M} (-1)^n g_n = \sum_{n=0}^{M} h_{M-n} = H(0) = \sqrt{2}
\end{aligned} \tag{20.2.13}$$

The dilation equations can be expressed in the frequency domain as follows:

$$\begin{aligned}
\Phi(\omega) &= 2^{-1/2}H(2^{-1}\omega)\Phi(2^{-1}\omega) \\
\Psi(\omega) &= 2^{-1/2}G(2^{-1}\omega)\Phi(2^{-1}\omega)
\end{aligned} \tag{20.2.14}$$

where $\Phi(\omega), \Psi(\omega)$ are the Fourier transforms:

$$\Phi(\omega) = \int_{-\infty}^{\infty} \phi(t)e^{-j\omega t}\,dt\,, \quad \Psi(\omega) = \int_{-\infty}^{\infty} \psi(t)e^{-j\omega t}\,dt \tag{20.2.15}$$

In fact, setting $\omega = 0$ in the first of (20.2.14) and assuming that $\Phi(0) \ne 0$, we immediately obtain the gain conditions (20.2.13). The iteration of Eqs. (20.2.14) leads to the

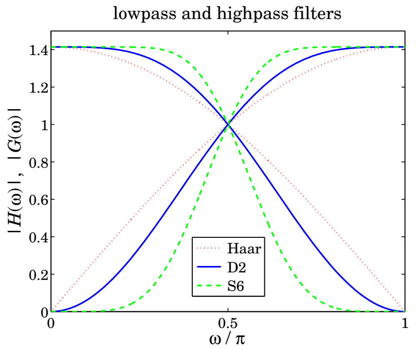


Fig. 20.2.1 Haar, Daubechies D_2, and Symmlet S_6 scaling and wavelet filters.

infinite product expressions:

$$
\begin{aligned}
\Phi(\omega) &= \Phi(0)\prod_{j=1}^{\infty}\left[2^{-1/2}H(2^{-j}\omega)\right] \\
\Psi(\omega) &= \Phi(0)\left[2^{-1/2}G(2^{-1}\omega)\right]\prod_{j=2}^{\infty}\left[2^{-1/2}H(2^{-j}\omega)\right]
\end{aligned}
\tag{20.2.16}
$$

We show later that $\Phi(0)$ can be chosen to be unity, $\Phi(0)=1$. As an example, Fig. 20.2.2 shows the normalized magnitude spectra $|\Phi(\omega)|$ and $|\Psi(\omega)|$, where the infinite products were replaced by a finite number of factors up to a maximum $j\le J$ chosen such that the next factor $J+1$ would add a negligible difference to the answer. For Fig. 20.2.2, an accuracy of 0.001 percent was achieved with the values of $J=7$ and $J=5$ for the left and right graphs, respectively. The following MATLAB code illustrates the generation of the left graph:

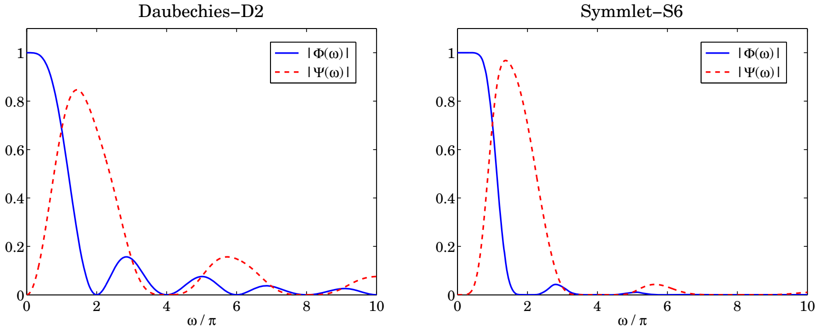


Fig. 20.2.2 Fourier transforms $\Phi(\omega),\Psi(\omega)$ of scaling and wavelet functions $\phi(t),\psi(t)$.

```
epsilon = 1e-5; Jmax = 30;                       % percent error = 100ε
f = linspace(0,10,513); w = pi*f;                % frequency range in units of π
h = daub(2)/sqrt(2); g = cmf(h);                 % normalize h_n such that H(0) = 1.
                                                 % use h=daub(6,2)/sqrt(2) for right graph
Phi0 = abs(freqz(h,1,w/2));                      % initialize recursions at |H(ω/2)|, |G(ω/2)|
Psi0 = abs(freqz(g,1,w/2));

for J = 2:Jmax,
   Phi = Phi0 .* abs(freqz(h,1,w/2^J));          % update by the factor |H(2^{-J}ω)|
   Psi = Psi0 .* abs(freqz(h,1,w/2^J));
   if norm(Phi-Phi0) < norm(Phi0) * epsilon, J, break; end      % stopping J
   Phi0 = Phi;
   Psi0 = Psi;
end

figure; plot(f,Phi, f,Psi,'--');
```

We observe from the graphs that $\Phi(\omega)$ has zeros at $\omega=2\pi m$ for non-zero integers m. This is justified in the next section. Similarly, $\Psi(\omega)$ vanishes at $\omega=2\pi m$ for even m, including zero.

Given the filters h_n,g_n, the dilation equations (20.2.1) and (20.2.4) can be solved iteratively for the functions $\phi(t),\psi(t)$ by the so-called *cascade algorithm*, which amounts to the iterations,

$$
\begin{aligned}
\phi^{(r+1)}(t) &= \sum_n h_n 2^{1/2}\phi^{(r)}(2t-n) \\
\psi^{(r+1)}(t) &= \sum_n g_n 2^{1/2}\phi^{(r)}(2t-n)
\end{aligned}
\tag{20.2.17}
$$

for $r=0,1,2,\dots$, starting with some simple initial choice, such as $\phi^{(0)}(t)=1$. The iteration converges quickly for all the scaling filters incorporated into the function daub. The algorithm can be cast as a convolutional operation with the so-called à trous[†] filters generated from the scaling filter. First, we note that if h_n,g_n have order M, and are defined over $0\le n\le M$, then the dilation equations imply that $\phi(t)$ and $\psi(t)$ will have compact support over $0\le t\le M$. Thus, we may evaluate the rth iterate $\phi^{(r)}(t)$ at the equally-spaced points, $t=2^{-r}n$, for $0\le n\le M2^r$, spanning the support interval. To this end, we define the discrete-time signals of the sampled $\phi^{(r)}(t)$:

$$
f^{(r)}(n)=2^{-r/2}\phi^{(r)}(2^{-r}n)
\tag{20.2.18}
$$

where $2^{-r/2}$ is a convenient normalization factor. It follows then from Eq. (20.2.17) that

$$
2^{-(r+1)/2}\phi^{(r+1)}(2^{-(r+1)}n)=\sum_m h_m 2^{-r/2}\phi^{(r)}(2^{-r}n-m)\,,\quad\text{or,}
$$

$$
f^{(r+1)}(n)=\sum_m h_m f^{(r)}(n-2^r m)=\sum_k h^{[r]}(k)f^{(r)}(n-k)
\tag{20.2.19}
$$

where we defined the à trous filter corresponding to the interpolation factor 2^r by

$$
h^{[r]}(k)=\sum_m h_m\delta(k-2^r m)
\tag{20.2.20}
$$

[†] "a trous" means "with holes" in French.

which is the original filter h_n with (2^r-1) zeros inserted between the h_n samples, so that its z-transform and frequency response are $H^{[r]}(z)=H(z^{2^r})$ and $H^{[r]}(\omega)=H(2^r\omega)$.

Thus, Eq. (20.2.19) can be interpreted as the convolution of the rth iterate with the rth à trous filter. The recursion can be iterated for $r=0,1,2,\dots,J$, for sufficiently large J (typically, $J=10$ works well.) The MATLAB function `casc` implements this algorithm:

```
[phi,psi,t] = casc(h,J,phi0);      % cascade algorithm
```

where t is the vector of final evaluation points $t=2^{-J}n$, $0\le n\le M2^J$. For example, the Daubechies D_2 functions $\phi(t),\psi(t)$ shown in Fig. 20.1.1 can be computed and plotted by the following code:

```
h = daub(2); J = 10; phi0 = 1;

[phi,psi,t] = casc(h,J,phi0);

figure;  plot(t,phi, t,psi,'--');
```

The scaling function output `phi` is normalized to unit L_2-norm, and the wavelet output `psi` is commensurately normalized. The following MATLAB code fragment from `casc` illustrates the construction method:

```
phi0=1;
for r=0:J-1,
   phi = conv(phi, upr(h,r));
end
```

where the function `upr` constructs the à trous filter $h^{[r]}(k)$ by upsampling h_n by a factor of 2^r. This function can also be implemented using the MATLAB's built-in function `upsample`. For example, if $\mathbf{h}=[h_0,h_1,h_2,h_3]$, then for $r=2$, the à trous filter will be,

$$\mathbf{h}^{[r]}=\mathrm{upr}(\mathbf{h},r)=\mathrm{upsample}(\mathbf{h},2^r)=[h_0,0,0,0,h_1,0,0,0,h_2,0,0,0,h_3,0,0,0]$$

20.3 Wavelet Filter Properties

The scaling and wavelet filters h_n,g_n must satisfy certain necessary constraints which are a consequence of the orthogonality of the scaling and wavelet basis functions. Using the property $(\phi_{j+1,n},\,\phi_{j+1,m})=\delta_{nm}$, it follows from Eq. (20.2.5) that,

$$(\phi_{j0},\phi_{jk})=\Big(\sum_n h_n\phi_{j+1,n},\,\sum_m h_{m-2k}\phi_{j+1,m}\Big)=\sum_{n,m}h_nh_{m-2k}(\phi_{j+1,n},\,\phi_{j+1,m})=\sum_n h_nh_{n-2k}$$

Similarly, we find,

$$\begin{aligned}(\psi_{j0},\psi_{jk})&=\Big(\sum_n g_n\phi_{j+1,n},\,\sum_m g_{m-2k}\phi_{j+1,m}\Big)=\sum_n g_ng_{n-2k}\\(\phi_{j0},\psi_{jk})&=\Big(\sum_n h_n\phi_{j+1,n},\,\sum_m g_{m-2k}\phi_{j+1,m}\Big)=\sum_n h_ng_{n-2k}\end{aligned}$$

But $(\phi_{j0},\phi_{jk})=(\psi_{j0},\psi_{jk})=\delta_k$ and $(\phi_{j0},\psi_{jk})=0$, therefore h_n,g_n must satisfy the orthogonality properties:

$$\boxed{\begin{aligned}\sum_n h_nh_{n-2k}&=\delta_k\\\sum_n g_ng_{n-2k}&=\delta_k\\\sum_n h_ng_{n-2k}&=0\end{aligned}}\tag{20.3.1}$$

These may also be expressed in the frequency domain. We will make use of the following cross-correlation identities that are valid for any two filters h_n,g_n and their frequency responses $H(\omega),G(\omega)$:

$$\begin{aligned}\sum_n h_ng_{n-k}\quad&\Leftrightarrow\quad H(\omega)G^*(\omega)\\(-1)^k\sum_n h_ng_{n-k}\quad&\Leftrightarrow\quad H(\omega+\pi)G^*(\omega+\pi)\\\Big[1+(-1)^k\Big]\sum_n h_ng_{n-k}\quad&\Leftrightarrow\quad H(\omega)G^*(\omega)+H(\omega+\pi)G^*(\omega+\pi)\end{aligned}\tag{20.3.2}$$

where the second follows from the "modulation" property of Fourier transforms, and the third, by adding the first two. We note next that Eqs. (20.3.1) can be written in the following equivalent manner obtained by replacing $2k$ by any k, even or odd:

$$\begin{aligned}[1+(-1)^k]\sum_n h_nh_{n-k}&=2\delta_k\\{}[1+(-1)^k]\sum_n g_ng_{n-k}&=2\delta_k\\{}[1+(-1)^k]\sum_n h_ng_{n-k}&=0\end{aligned}\tag{20.3.3}$$

Taking the Fourier transforms of both sides of (20.3.3) and using the transform properties (20.3.2), we obtain the frequency-domain equivalent conditions to Eqs. (20.3.1):

$$\boxed{\begin{aligned}&|H(\omega)|^2+|H(\omega+\pi)|^2=2\\&|G(\omega)|^2+|G(\omega+\pi)|^2=2\\&H(\omega)G^*(\omega)+H(\omega+\pi)G^*(\omega+\pi)=0\end{aligned}}\tag{20.3.4}$$

The conjugate mirror filter choice (20.2.11) for $G(\omega)$ automatically satisfies the third of Eqs. (20.3.4). Indeed, using the 2π-periodicity of $H(\omega)$, we have,

$$\begin{aligned}G^*(\omega)&=e^{jM(\omega+\pi)}H(\omega+\pi)\\G^*(\omega+\pi)&=e^{jM(\omega+2\pi)}H(\omega+2\pi)=e^{jM\omega}H(\omega)\end{aligned}$$

so that,

$$H(\omega)G^*(\omega)+H(\omega+\pi)G^*(\omega+\pi)=e^{jM\omega}H(\omega)H(\omega+\pi)\Big[e^{jM\pi}+1\Big]=0$$

where $e^{jM\pi} = -1$, because M was assumed to be odd. With this choice of $G(\omega)$, the first of (20.3.4) can be written in the following form, which will be used later on to derive the undecimated wavelet transform:

$$\boxed{\frac{1}{2}\big[H^*(\omega)H(\omega)+G^*(\omega)G(\omega)\big] = 1} \tag{20.3.5}$$

Setting $\omega = 0$ in the first of Eqs. (20.3.4), and using the DC gain constraint $H(0)= \sqrt{2}$, we find immediately that $H(\pi)= 0$, that is, the scaling filter must have a zero at the Nyquist frequency $\omega = \pi$. Since

$$H(\pi)= \sum_n (-1)^n h_n = \sum_{n=\text{even}} h_n - \sum_{n=\text{odd}} h_n\,,$$

it follows in conjunction with the DC condition that:

$$\sum_{n=\text{even}} h_n = \sum_{n=\text{odd}} h_n = \frac{1}{\sqrt{2}} \tag{20.3.6}$$

The correlation constraints and the DC gain condition,

$$\sum_n h_n h_{n-2k} = \delta_k\,, \quad \sum_n h_n = \sqrt{2}, \tag{20.3.7}$$

provide only $N/2+1$ equations, where N is the (even) length of the filter h_n. Therefore, one has $N/2 - 1$ additional degrees of freedom to specify the scaling filters uniquely. For example, Daubechies' minimum-phase D_K filters have length $N = 2K$ and K zeros at Nyquist. These zeros translate into K equivalent moment constraints, or derivative flatness constraints at Nyquist:

$$\sum_{n=0}^{N-1} (-1)^n n^i h_n = 0 \quad \Leftrightarrow \quad \frac{d^i H(\omega)}{d\omega^i}\bigg|_{\omega=\pi} = 0\,, \quad i = 0, 1, \dots, K-1 \tag{20.3.8}$$

The $i = 0$ case is already a consequence of the correlation constraint, therefore, this leaves $K - 1$ additional conditions, which together with the $K + 1$ equations (20.3.7), determines the $N = 2K$ coefficients h_n uniquely. The construction method may be found in [413]. As an example, we work out the three cases D_1, D_2, D_3 explicitly. The Haar D_1 case corresponds to $K = 1$ or $N = 2K = 2$, so that $\mathbf{h} = [h_0, h_1]$ must satisfy:

$$h_0^2 + h_1^2 = 1\,, \quad h_0 + h_1 = \sqrt{2} \tag{20.3.9}$$

with (lowpass) solution $h_0 = h_1 = 1/\sqrt{2}$. For the Daubechies D_2 case, we have $K = 2$ and $N = 2K = 4$, so that $\mathbf{h} = [h_0, h_1, h_2, h_3]$ must satisfy,

$$\begin{aligned} & h_0^2 + h_1^2 + h_2^2 + h_3^2 = 1 \\ & h_0 + h_2 = \frac{1}{\sqrt{2}}\,, \quad h_1 + h_3 = \frac{1}{\sqrt{2}} \\ & -h_1 + 2h_2 - 3h_3 = 0 \end{aligned} \tag{20.3.10}$$

where the third is the Nyquist moment constraint with $i = 1$, and the middle two are equivalent to the DC gain and the $h_0h_2 + h_1h_3 = 0$ correlation constraint; indeed, this follows from the identity:

$$\begin{aligned} & \Big(h_0 + h_2 - \frac{1}{\sqrt{2}}\Big)^2 + \Big(h_1 + h_3 - \frac{1}{\sqrt{2}}\Big)^2 = \\ & = 1 + (h_0^2 + h_1^2 + h_2^2 + h_3^2) - \sqrt{2}(h_0 + h_1 + h_2 + h_3) + 2(h_0h_2 + h_1h_3) \end{aligned}$$

Solving the three linear ones for h_1, h_2, h_3 in terms of h_0 and inserting them in the first one, we obtain the quadratic equation for h_0, with solutions:

$$4h_0^2 - \sqrt{2}\,h_0 - \frac{1}{4} = 0 \quad \Rightarrow \quad h_0 = \frac{1 \pm \sqrt{3}}{4\sqrt{2}}$$

The "+" choice leads to the following minimum-phase filter (the "−" choice leads to the reverse of that, which has maximum phase):

$$\begin{aligned} \mathbf{h} = [h_0, h_1, h_2, h_3] &= \frac{1}{4\sqrt{2}}\big[1 + \sqrt{3}\,,\ 3 + \sqrt{3}\,,\ 3 - \sqrt{3}\,,\ 1 - \sqrt{3}\big] \\ &= [0.4830\,,\ 0.8365\,,\ 0.2241\,,\ -0.1294] \end{aligned} \tag{20.3.11}$$

The corresponding transfer function $H(z)$ has a double zero at Nyquist $z = -1$ and one inside the unit circle at $z = 2 - \sqrt{3}$. In fact, $H(z)$ factors as follows:

$$H(z)= h_0(1 + z^{-1})^2(1 - (2 - \sqrt{3})z^{-1})$$

For the D_3 case corresponding to $K = 3$, we have the following two quadratic equations and four linear ones that must be satisfied by the filter $\mathbf{h} = [h_0, h_1, h_2, h_3, h_4, h_5]$:

$$\begin{aligned} & h_0^2 + h_1^2 + h_2^2 + h_3^2 + h_4^2 + h_5^2 = 1\,, \quad h_0h_4 + h_1h_5 = 0 \\ & h_0 + h_2 + h_4 = 1/\sqrt{2}\,, \quad h_1 + h_3 + h_5 = 1/\sqrt{2} \\ & -h_1 + 2h_2 - 3h_3 + 4h_4 - 5h_5 = 0 \\ & -h_1 + 2^2h_2 - 3^2h_3 + 4^2h_4 - 5^2h_5 = 0 \end{aligned} \tag{20.3.12}$$

where the last two correspond to the values $i = 1, 2$ in (20.3.8), and we have omitted the correlation constraint $h_0h_2 + h_1h_3 + h_2h_4 + h_3h_5 = 0$ as it is obtainable from Eqs. (20.3.12). Solving the linear ones for h_2, h_3, h_4, h_5 in terms of h_0, h_1, we find,

$$\begin{aligned} h_2 &= -4h_0 + 2h_1 + \sqrt{2}/8 \\ h_3 &= -2h_0 + 3\sqrt{2}/8 \\ h_4 &= 3h_0 - 2h_1 + 3\sqrt{2}/8 \\ h_5 &= 2h_0 - h_1 + \sqrt{2}/8 \end{aligned} \tag{20.3.13}$$

Inserting these into the first two of Eqs. (20.3.12), we obtain the quadratic system:

$$\begin{aligned} & 34h_0^2 - (32h_1 - \sqrt{2}/4)h_0 + 10h_1^2 - 5\sqrt{2}h_1/4 - 3/8 = 0 \\ & h_1^2 - \sqrt{2}h_1/8 - 3h_0^2 - 3\sqrt{2}h_0/8 = 0 \end{aligned} \tag{20.3.14}$$

Solving the second for h_1 in terms of h_0, we find:

$$h_1 = \frac{1}{16}\big[\sqrt{2} + \sqrt{768h_0^2 + 96\sqrt{2}h_0 + 2}\,\big] \tag{20.3.15}$$

and inserting this into the first of (20.3.14), we obtain:

$$64h_0^2 + 2\sqrt{2}h_0 - 2h_0\sqrt{768h_0^2 + 96\sqrt{2}h_0 + 2} - \frac{3}{8} = 0$$

or, the equivalent quartic equation:

$$1024h_0^4 - 128\sqrt{2}h_0^3 - 48h_0^2 - \frac{3\sqrt{2}}{2}h_0 + \frac{9}{64} = 0 \tag{20.3.16}$$

which has two real solutions and two complex-conjugate ones. Of the real solutions, the one that leads to a minimum-phase filter **h** is

$$h_0 = \frac{\sqrt{2}}{32}\left[1 + \sqrt{10} + \sqrt{5 + 2\sqrt{10}}\right] \tag{20.3.17}$$

With this solution for h_0, Eqs. (20.3.15) and (20.3.13) lead to the desired minimum-phase filter. Its transfer function $H(z)$ factors as:

$$H(z) = h_0 + h_1 z^{-1} + h_2 z^{-2} + h_3 z^{-3} + h_4 z^{-4} + h_5 z^{-5} = h_0(1+z^{-1})^3(1-z_1 z^{-1})(1-z_1^* z^{-1})$$

where z_1 is the following zero lying inside the unit circle:

$$z_1 = \frac{\sqrt{10} - 1 + j\sqrt{2\sqrt{10} - 5}}{1 + \sqrt{10} + \sqrt{5 + 2\sqrt{10}}} \quad \Rightarrow \quad |z_1| = \frac{\sqrt{6}}{1 + \sqrt{10} + \sqrt{5 + 2\sqrt{10}}} = 0.3254 \tag{20.3.18}$$

Finally, we mention that the K flatness constraints (20.3.8) at $\omega = \pi$ for $H(\omega)$ are equivalent to K flatness constraints for the wavelet filter $G(\omega)$ at DC, that is,

$$\left.\frac{d^i G(\omega)}{d\omega^i}\right|_{\omega=0} = 0\,, \quad i = 0, 1, \dots, K-1 \tag{20.3.19}$$

In turn, these are equivalent to the K vanishing moment constraints for the wavelet function $\psi(t)$, that is,

$$\int_{-\infty}^{\infty} t^i \psi(t)\,dt = 0\,, \quad i = 0, 1, \dots, K-1 \tag{20.3.20}$$

The equivalence between (20.3.19) and (20.3.20) is easily established by differentiating the dilation equation (20.2.14) for $\Psi(\omega)$ with respect to ω and setting $\omega = 0$.

Because the D_K filters have minimum phase by construction, their energy is concentrated at earlier times and their shape is very asymmetric. Daubechies' other two families of scaling and wavelet filters, the "least asymmetric" symmlets, and the coiflets, have a more symmetric shape. They are discussed in detail in [413].

Another consequence of the orthonormality of the ϕ and ψ bases can be stated in terms of the Fourier transforms $\Phi(\omega)$ and $\Psi(\omega)$ as identities in ω:

$$\begin{aligned} &\sum_{m=-\infty}^{\infty} \big|\Phi(\omega + 2\pi m)\big|^2 = \sum_{m=-\infty}^{\infty} \big|\Psi(\omega + 2\pi m)\big|^2 = 1 \\ &\sum_{m=-\infty}^{\infty} \Phi(\omega + 2\pi m)\Psi^*(\omega + 2\pi m) = 0 \end{aligned} \tag{20.3.21}$$

These follow by applying Parseval's identity to the cross-correlation inner products of the ϕ and ψ bases. For example, we have,

$$\begin{aligned} \delta_k &= (\phi_{j0}, \phi_{jk}) = (\phi_{00}, \phi_{0k}) = \int_{-\infty}^{\infty} \phi(t)\phi(t-k)\,dt = \frac{1}{2\pi}\int_{-\infty}^{\infty} \big|\Phi(\omega)\big|^2 e^{j\omega k}\,d\omega \\ &= \frac{1}{2\pi}\int_{-\pi}^{\pi}\left[\sum_{m=-\infty}^{\infty} \big|\Phi(\omega + 2\pi m)\big|^2\right] e^{j\omega k}\,d\omega \end{aligned}$$

where the last expression was obtained by noting that because k is an integer, the exponential $e^{j\omega k}$ is periodic in ω with period 2π, which allowed us to fold the infinite integration range into the $[-\pi, \pi]$ range. But this result is simply the inverse DTFT of the first of Eqs. (20.3.21). The other results are shown in a similar fashion using the inner products $(\psi_{j0}, \psi_{jk}) = \delta_k$ and $(\phi_{j0}, \psi_{jk}) = 0$.

It can be easily argued from Eqs. (20.2.16) that $\Phi(2\pi m) = 0$ for all non-zero integers m. Indeed, setting $m = 2^p(2q+1)$ for some integers $p \geq 0, q \geq 0$, it follows that after p iterations, an H-factor will appear such that $H\big((2q+1)\pi\big) = H(\pi) = 0$. Setting $\omega = 0$ in the first of Eqs. (20.3.21) and using this property, it follows that $|\Phi(0)|^2 = 1$. Thus, up to a sign, we may set:

$$\Phi(0) = \int_{-\infty}^{\infty} \phi(t)\,dt = 1 \tag{20.3.22}$$

20.4 Multiresolution and Filter Banks

We saw in Eq. (20.1.12) that a signal belonging to a higher-resolution subspace can be expanded in terms of its lower-resolution components. If J and J_0 are the highest and lowest resolutions of interest, then for a signal $f(t) \in V_J$, the multiresolution expansion will have the form:

$$f(t) = \sum_n c_{Jn}\phi_{Jn}(t) = \sum_k c_{J_0k}\phi_{J_0k}(t) + \sum_{j=J_0}^{J-1}\sum_k d_{jk}\psi_{jk}(t) \tag{20.4.1}$$

with the various terms corresponding to the direct-sum decomposition:

$$V_J = V_{J_0} \oplus \big(W_{J_0} \oplus W_{J_0+1} \oplus \cdots \oplus W_{J-1}\big) \tag{20.4.2}$$

The choice of J, J_0 is dictated by the application at hand. Typically, we start with a signal $f(t)$ sampled at $N = 2^J$ samples that are equally-spaced over the signal's duration. The duration interval can always be normalized to be $0 \leq t \leq 1$ so that the sample spacing is 2^{-J}. The lowest level is $J_0 = 0$ corresponding to sample spacing $2^{-J_0} = 1$, that is, one sample in the interval $0 \le t \le 1$. One does not need to choose $J_0 = 0$; any value $0 \le J_0 \le J - 1$ could be used.

The lower-level expansion coefficients $\{c_{J_0k}; d_{jk}, J_0 \le j \le J-1\}$ can be computed from those of the highest level c_{Jn} by *Mallat's multiresolution algorithm* [469], which establishes a connection between multiresolution analysis and filter banks.

The algorithm successively computes the coefficients at each level from those of the level just above. It is based on establishing the relationship between the expansion coefficients for the decomposition $V_{j+1} = V_j \oplus W_j$ and iterating it over $J_0 \le j \le J-1$. An arbitrary element $f(t)$ of V_{j+1} can be expanded in two ways:

$$f(t) = \underbrace{\sum_n c_{j+1,n}\phi_{j+1,n}(t)}_{V_{j+1}} = \underbrace{\sum_k c_{jk}\phi_{jk}(t)}_{V_j} + \underbrace{\sum_k d_{jk}\psi_{jk}(t)}_{W_j} \tag{20.4.3}$$

The right-hand side coefficients are:

$$\begin{aligned} c_{jk} &= (f, \phi_{jk}) = \Big(\sum_n c_{j+1,n}\phi_{j+1,n}, \phi_{jk}\Big) = \sum_n c_{j+1,n}(\phi_{j+1,n}, \phi_{jk}) \\ d_{jk} &= (f, \psi_{jk}) = \Big(\sum_n c_{j+1,n}\phi_{j+1,n}, \psi_{jk}\Big) = \sum_n c_{j+1,n}(\phi_{j+1,n}, \psi_{jk}) \end{aligned}$$

which become, using Eq. (20.2.6),

$$\boxed{\begin{aligned} c_{jk} &= \sum_n h_{n-2k}c_{j+1,n} \\ d_{jk} &= \sum_n g_{n-2k}c_{j+1,n} \end{aligned}} \qquad \text{(analysis)} \tag{20.4.4}$$

for $j = J-1, J-2, \dots, J_0$, initialized at $c_{Jn} = f(t_n)$, $n = 0, 1, \dots, 2^J-1$, with $t_n = n2^{-J}$, that is, the 2^J samples of $f(t)$ in the interval $0 \le t \le 1$. Conversely, the coefficients $c_{j+1,n}$ can be reconstructed from c_{jk}, d_{jk}:

$$\begin{aligned} c_{j+1,n} &= (f, \phi_{j+1,n}) = \Big(\sum_k c_{jk}\phi_{jk} + \sum_k d_{jk}\psi_{jk}\,,\ \phi_{j+1,n}\Big) \\ &= \sum_k c_{jk}(\phi_{jk}, \phi_{j+1,n}) + \sum_k d_{jk}(\psi_{jk}, \phi_{j+1,n}) \end{aligned}$$

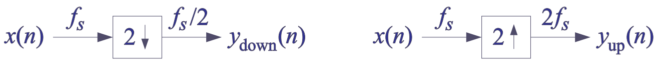
or,

$$\boxed{c_{j+1,n} = \sum_k h_{n-2k}c_{jk} + \sum_k g_{n-2k}d_{jk}} \qquad \text{(synthesis)} \tag{20.4.5}$$

for $j = J_0, J_0+1, \dots, J-1$. To see the filter bank interpretation of these results, let us define the *time-reversed* filters $\bar{h}_n = h_{-n}$ and $\bar{g}_n = g_{-n}$, and the downsampling and upsampling operations by a factor of two [418]:

$$\begin{aligned} y_{\text{down}}(n) &= x(2n) \\ y_{\text{up}}(n) &= \sum_k x(k)\delta(n-2k) = \begin{cases} x(k), & \text{if } n = 2k \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{20.4.6}$$

and pictorially,



The downsampling operation decreases the sampling rate by a factor of two by keeping only the even-index samples of the input. The upsampling operation increases the sampling rate by a factor of two by inserting a zero between successive input samples. It is the same as the "à trous" operation for filters that we encountered earlier.

With these definitions, the analysis algorithm (20.4.4) is seen to be equivalent to convolving with the time-reversed filters, followed by downsampling. Symbolically,

$$\begin{aligned} c_{jk} &= \sum_n \bar{h}_{2k-n}c_{j+1,n} = (\bar{h} * c_{j+1})(2k) \\ d_{jk} &= \sum_n \bar{g}_{2k-n}c_{j+1,n} = (\bar{g} * c_{j+1})(2k) \end{aligned} \quad \Rightarrow \quad \boxed{\begin{aligned} \mathbf{c}_j &= (\bar{\mathbf{h}} * \mathbf{c}_{j+1})_{\text{down}} \\ \mathbf{d}_j &= (\bar{\mathbf{g}} * \mathbf{c}_{j+1})_{\text{down}} \end{aligned}} \tag{20.4.7}$$

Similarly, the synthesis algorithm (20.4.5) is equivalent to upsampling the signals c_{jk} and d_{jk} by two and then filtering them through h_n, g_n,

$$c_{j+1,n} = \sum_k h_{n-2k}c_{jk} + \sum_k g_{n-2k}d_{jk} = \sum_m h_{n-m}c_{jm}^{\text{up}} + \sum_m g_{n-m}d_{jm}^{\text{up}} \tag{20.4.8}$$

where $c_{jm}^{\text{up}} = \sum_k c_{jk}\delta(m-2k)$. Symbolically,

$$\boxed{\mathbf{c}_{j+1} = \mathbf{h} * \mathbf{c}_j^{\text{up}} + \mathbf{g} * \mathbf{d}_j^{\text{up}}} \tag{20.4.9}$$

Fig. 20.4.1 shows a block diagram realization of the analysis and synthesis equations (20.4.7) and (20.4.9) in terms of a so-called *tree-structured iterated filter bank.*

In the figure, we used $J = 3$ and $J_0 = 0$. Each stage of the analysis bank produces the coefficients at the next coarser level. Similarly, the synthesis bank starts with the coarsest level and successively reconstructs the higher levels.

The time-reversed filters $\bar{h}_n, \bar{g}_n$ are still lowpass and highpass, indeed, their frequency responses are $\bar{H}(\omega) = H^*(\omega)$ and $\bar{G}(\omega) = G^*(\omega)$. Therefore, at the first analysis stage, the input signal c_3 is split into the low- and high-frequency parts c_2, d_2 representing, respectively, a smoother trend and a more irregular detail. At the second stage, the smooth trend c_2 is split again into a low and high frequency part, c_1, d_1, and so on. The subband frequency operation of the filter bank can be understood by looking at the spectra of the signals at the successive output stages.

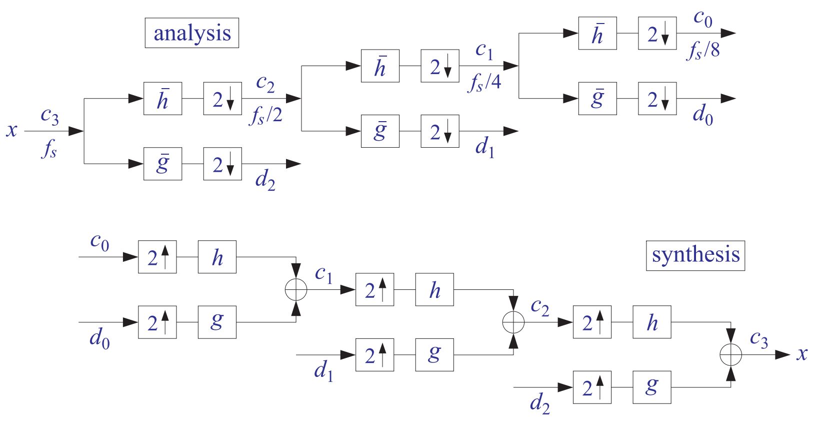


Fig. 20.4.1 Analysis and synthesis filter bank.

Because successive stages operate at different sampling rates, it is best to characterize the spectra using a common frequency axis, for example, the physical frequency f. The spectrum of a discrete-time signal $x(n)$ sampled at a rate f_s is defined by,

$$X(f)=\sum_n x(n)e^{-j\omega n}=\sum_n x(n)e^{-2\pi jfn/f_s} \qquad (20.4.10)$$

where $\omega=2\pi f/f_s$ is the digital frequency in radians/sample. We will use the notation $X(f,f_s)$ whenever it is necessary to indicate the dependence on f_s explicitly.

Just like the sampling of a continuous-time signal causes the periodic replication of its spectrum at multiples of the sampling rate, the operation of downsampling causes the periodic replication of the input spectrum at multiples of the downsampled rate. It is a general result that for a downsampling ratio by a factor L, and input and output rates of f_s and $f_s^{\text{down}}=f_s/L$, the downsampled signal $y_{\text{down}}(n)=x(nL)$ will have the following replicated spectrum at multiples of f_s^{down}:

$$Y_{\text{down}}(f)=\frac{1}{L}\sum_{m=0}^{L-1}X(f-mf_s^{\text{down}}) \qquad (20.4.11)$$

where according to (20.4.10),

$$Y_{\text{down}}(f)=\sum_n y_{\text{down}}(n)e^{-2\pi jfn/f_s^{\text{down}}}=\sum_n x(nL)e^{-2\pi jfnL/f_s} \qquad (20.4.12)$$

In particular, for downsampling by $L=2$, we have $f_s^{\text{down}}=f_s/2$ and

$$Y_{\text{down}}(f)=\frac{1}{2}\Big[X(f)+X(f-f_s^{\text{down}})\Big]=\sum_n x(2n)e^{-2\pi jf2n/f_s} \qquad (20.4.13)$$

If f_s is the sampling rate at the input stage for the signal c_3 of the analysis bank, then the rates for the signals c_2,c_1,c_0 will be $f_s/2,f_s/4,f_s/8$, respectively. Fig. 20.4.2 shows the corresponding spectra, including the effect of filtering and downsampling.

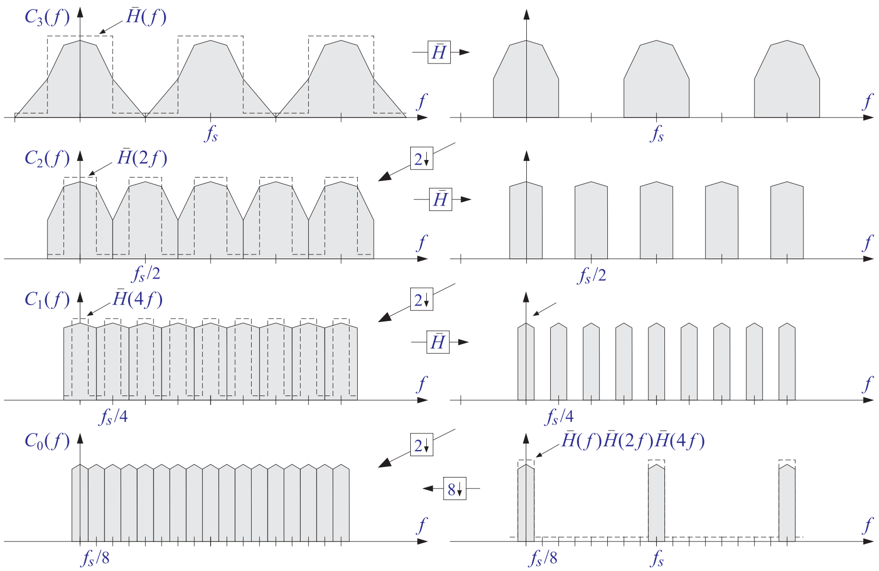


Fig. 20.4.2 Spectra of the signals c_3,c_2,c_1,c_0 at successive stages of the analysis bank of Fig. 20.4.1.

For clarity, we took $\bar H(f)$ to be an ideal lowpass filter with cutoff frequency equal to half the Nyquist frequency, that is, $f_s/4$. Starting at the top left with the input spectrum $C_3(f)$, which is replicated at multiples of f_s, the first lowpass filtering operation produces the spectrum at the upper right. According to Eq. (20.4.13), downsampling will replicate this spectrum at multiples of $f_s^{\text{down}}=f_s/2$, thereby filling the gaps created by the ideal filter, and resulting in the spectrum $C_2(f)$ shown on the left graph of the second row. The sampling rate at that stage is now $f_s/2$.

The second lowpass filtering operation of the signal c_2 indicated on Fig. 20.4.1 will be by the filter $\bar H(f,f_s/2)$ which is equal to $\bar H(2f,f_s)$ if referred to the original sampling rate f_s; indeed, we have,

$$\bar H(f,f_s/2)=\sum_n \bar h_n e^{-2\pi jfn/(f_s/2)}=\sum_n \bar h_n e^{-2\pi jf2n/f_s}=\bar H(2f,f_s) \qquad (20.4.14)$$

The filter $\bar H(2f,f_s)$ is the twice-compressed version of $\bar H(f,f_s)$, and still has an ideal shape but with cutoff frequency $f_s/8$. The result of the second filtering operation is shown on the right graph of the second row. The lowpass-filtered replicas are at multiples of $f_s/2$, and after the next downsampling operation, they will be replicated at multiples of $f_s/4$ resulting in the spectrum $C_1(f)$ of the signal c_1 shown on the left of the third row. At the new sampling rate $f_s/4$, the third-stage lowpass filter will be:

$$\bar H(f,f_s/4)=\bar H(2f,f_s/2)=\bar H(4f,f_s) \qquad (20.4.15)$$

which is the four-times compressed version of that at rate f_s, or twice-compressed of that of the previous stage. Its cutoff is now at $f_s/16$. The result of filtering by $\bar H(4f,f_s)$ is shown on the right of the third row, and its downsampled version replicated at multiples of $f_s/8$ is shown on the bottom left as the spectrum $C_0(f)$.

Thus, the output spectra $C_2(f), C_1(f), C_0(f)$ capture the frequency content of the original signal in the corresponding successive subbands, each subband having half the passband of the previous one (often referred to as an octave filter bank.)

The bottom-right graph shows an equivalent way of obtaining the same final output $C_0(f)$, namely, by first filtering by the combined filter,

$$\bar{H}(f, f_s)\,\bar{H}(2f, f_s)\,\bar{H}(4f, f_s) = \bar{H}(f, f_s)\,\bar{H}(f, f_s/2)\,\bar{H}(f, f_s/4)$$

running at the original rate f_s, and then dropping the rate all at once by a factor of $2^3 = 8$, which will cause a replication at multiples of $f_s/8$. This point of view is justified by applying the standard multirate identity depicted below [418]:

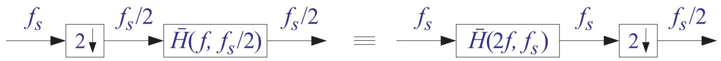


Fig. 20.4.3 shows the successive application of this identity to the three stages of Fig. 20.4.1 until all the downsamplers are pushed to the right-most end and all the filters to the left-most end. The corresponding sampling rates are indicated at the outputs of the downsamplers.

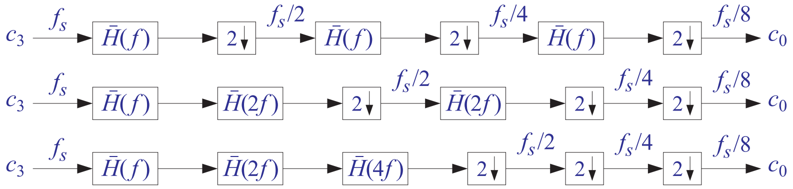


Fig. 20.4.3 Equivalent realizations of the lowpass portion of the analysis bank of Fig. 20.4.1.

For non-ideal filters $\bar{H}(f), \bar{G}(f)$, such as the scaling and wavelet filters, the downsampling replication property (20.4.13) will cause aliasing. However, because of Eq. (20.3.4), the filter bank satisfies the so-called *pefect reconstruction* property, which allows the aliasing to be canceled at the reconstruction, synthesis, stage.

20.5 Discrete Wavelet Transform

We summarize the analysis and synthesis algorithms:

$$\boxed{\begin{aligned} \mathbf{c}_{j-1} &= (\bar{\mathbf{h}} * \mathbf{c}_j)_{\text{down}} \\ \mathbf{d}_{j-1} &= (\bar{\mathbf{g}} * \mathbf{c}_j)_{\text{down}} \end{aligned}} \qquad j = J, J-1, \dots, J_0 + 1 \tag{20.5.1}$$

$$\boxed{\mathbf{c}_j = \mathbf{h} * \mathbf{c}_{j-1}^{\text{up}} + \mathbf{g} * \mathbf{d}_{j-1}^{\text{up}}} \qquad j = J_0{+}1, J_0{+}2, \dots, J \tag{20.5.2}$$

The discrete wavelet transform (DWT) consists of the coefficients generated by the analysis algorithm. The DWT can be defined for each resolution level. Starting with an input signal vector $\mathbf{x} = [x_0, x_1, \dots, x_{N-1}]^T$, where $N = 2^J$, the DWTs at successive stages are defined as the following sets of coefficients:

$$\begin{aligned} \mathbf{x} = \mathbf{c}_J &\to [\mathbf{c}_{J-1}, \mathbf{d}_{J-1}], && \text{(level } J-1) \\ &\to [\mathbf{c}_{J-2}, \mathbf{d}_{J-2}, \mathbf{d}_{J-1}], && \text{(level } J-2) \\ &\to [\mathbf{c}_{J-3}, \mathbf{d}_{J-3}, \mathbf{d}_{J-2}, \mathbf{d}_{J-1}], && \text{(level } J-3) \\ &\quad \vdots \\ &\to [\mathbf{c}_{J_0}, \mathbf{d}_{J_0}, \mathbf{d}_{J_0+1}, \dots, \mathbf{d}_{J-1}], && \text{(level } J_0) \end{aligned} \tag{20.5.3}$$

Starting with the coefficients at any level, the inverse discrete wavelet transform (IDWT) applies the synthesis algorithm to reconstruct the original signal $\mathbf{x}$.

In practice, there are as many variants of the DWT as there are ways to implement the filtering operations in (20.5.1)–(20.5.2), such as deciding on how to deal with the filter transients (the edge effects), realizing convolution in a matrix form, periodizing or symmetrizing the signals or not, and so on.

There exist several commercial implementations in MATLAB, Mathematica, Maple, and S+, incorporating the many variants, as well as several freely available packages in MATLAB, C++, and R [582–596].

In this section, we consider only the periodized version implemented both in matrix form and in filtering form using circular convolutions. Given a (possibly infinite) signal $x(n)$, we define its "modulo-N reduction" [2] as its periodic extension with period N:

$$\tilde{x}(n) = \sum_{p=-\infty}^{\infty} x(n + pN) \tag{20.5.4}$$

The signal $\tilde{x}(n)$ is periodic with period N, and therefore, we only need to know it over one period, $0 \le n \le N-1$. It is characterized by the property that it has the same N-point DFT as the signal $x(n)$, that is,

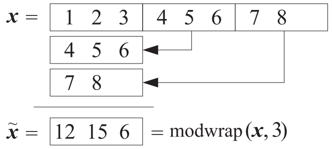
$$X(\omega_k) = \sum_{n=-\infty}^{\infty} x(n) e^{-j\omega_k n} = \sum_{n=0}^{N-1} \tilde{x}(n) e^{-j\omega_k n} \tag{20.5.5}$$

where ω_k are the DFT frequencies $\omega_k = 2\pi k/N$, $k = 0, 1, \dots, N-1$. The signal $\tilde{x}(n)$ can be visualized as dividing the original signal $x(n)$ into contiguous blocks of length N, then aligning them in time, and adding them up. This operation is referred to as "mod-N wrapping" and is depicted in Fig. 20.5.1 for a signal $x(n)$ of length $4N$.

The MATLAB function `modwrap` implements this operation. Its argument can be a row or column vector, or a matrix. For the matrix case, it wraps each column modulo N:

```
Y = modwrap(X,N);          % mod-N reduction of a matrix
```

For example, we have for the signal $\mathbf{x} = [1, 2, 3, 4, 5, 6, 7, 8]$ and $N = 3$,



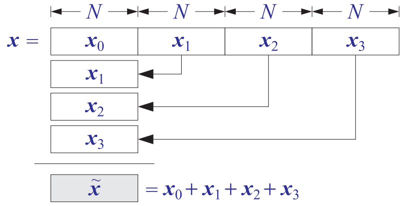


Fig. 20.5.1 Modulo-N reduction or wrapping.

Circular convolution is defined as the modulo-N reduction of ordinary linear convolution, that is,

$$\mathbf{y} = \mathbf{h} * \mathbf{x} \quad \Rightarrow \quad \mathbf{y}_{\text{circ}} = \tilde{\mathbf{y}} = \widetilde{\mathbf{h} * \mathbf{x}} \tag{20.5.6}$$

or more explicitly,

$$y(n) = \sum_m h(m)x(n-m) \quad \Rightarrow \quad \tilde{y}(n) = \sum_p y(n+pN)$$

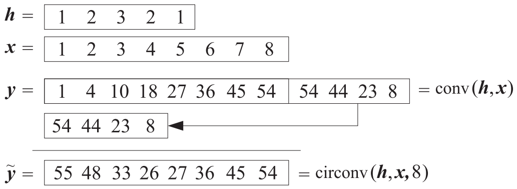
Its MATLAB implementation is straightforward with the help of the function `modwrap`, for example,

```
y = modwrap(conv(h,x), N);
```

This code has been incorporated into the function `circonv`, with usage:

```
y = circonv(h,x,N);          % mod-N circular convolution
```

For example, we have the outputs for $N = 8$:



h = 1 2 3 2 1

x = 1 2 3 4 5 6 7 8

y = 1 4 10 18 27 36 45 54 | 54 44 23 8 = conv(h,x)

54 44 23 8

$\tilde{y}$ = 55 48 33 26 27 36 45 54 = circonv(h,x,8)

Circular convolution can also be implemented in the frequency domain by computing the N-point DFTs of the signals $\mathbf{h}, \mathbf{x}$, multiplying them pointwise together, and performing an inverse N-point DFT. Symbolically,

$$\mathbf{y}_{\text{circ}} = \tilde{\mathbf{y}} = \widetilde{\mathbf{h} * \mathbf{x}} = \text{IDFT}\big[\text{DFT}(\mathbf{h})\cdot\text{DFT}(\mathbf{x})\big] \tag{20.5.7}$$

or, explicitly,

$$\tilde{y}(n) = \frac{1}{N}\sum_{k=0}^{N-1} H(\omega_k)X(\omega_k)e^{j\omega_k n} \tag{20.5.8}$$

where $H(\omega_k), X(\omega_k)$ are N-point DFTs as in Eq. (20.5.5). The following MATLAB code illustrates the implementation of the above example in the frequency and time domains:

```
h = [1 2 3 2 1];
x = [1 2 3 4 5 6 7 8];
H = fft(h,8); X = fft(x,8);       % calculate 8-point DFTs
Y = H.*X;                         % point-wise multiplication of the DFTs
ytilde = ifft(Y,8);               % inverse DFT generates ỹ = [55,48,33,26,27,36,45,54]
ytilde = circonv(h,x,8);          % time-domain calculation
```

The frequency method (20.5.7) becomes efficient if FFTs are used in the right-hand side. However, for our DWT functions, we have used the time-domain implementations, which are equally efficient because the typical wavelet filter lengths are fairly short. The convolutional operations in Eqs. (20.5.1) and (20.5.2) can now be replaced by their circular versions, denoted symbolically,

$$\boxed{\begin{aligned} \mathbf{c}_{j-1} &= \big(\text{circonv}(\bar{\mathbf{h}}, \mathbf{c}_j)\big)_{\text{down}} \\ \mathbf{d}_{j-1} &= \big(\text{circonv}(\bar{\mathbf{g}}, \mathbf{c}_j)\big)_{\text{down}} \end{aligned}} \quad j = J, J-1, \dots, J_0+1$$

$$\boxed{\mathbf{c}_j = \text{circonv}(\mathbf{h}, \mathbf{c}_{j-1}^{\text{up}}) + \text{circonv}(\mathbf{g}, \mathbf{c}_{j-1}^{\text{up}})} \quad j = J_0+1, J_0+2, \dots, J \tag{20.5.9}$$

DWT in Matrix Form

The convolutional operation $\mathbf{y} = \mathbf{h} * \mathbf{x}$ can be represented in matrix form:

$$\mathbf{y} = H\mathbf{x}$$

where H is the convolution matrix of the filter h_n, defined by its matrix elements:

$$H_{nm} = h_{n-m}$$

The convolution matrix corresponding to the time-reversed filter $\bar{h}_n = h_{-n}$ is given by the transposed matrix

$$\bar{H} = H^T$$

because $\bar{H}_{nm} = \bar{h}_{n-m} = h_{m-n} = H_{mn}$. Thus, in matrix notation, the typical convolutional and down- and up-sampling operations being performed at the analysis and synthesis stages have the forms:

$$\mathbf{y} = (H^T\mathbf{x})_{\text{down}}\,, \quad \mathbf{y} = H\mathbf{x}^{\text{up}} \tag{20.5.10}$$

Moreover, replacing the linear convolutions by circular ones amounts to replacing the convolutional matrices by their mod-N wrapped versions obtained by reducing their columns modulo-N, where N is the length of the input vector $\mathbf{x}$. Denoting $\tilde{H} = \text{modwrap}(H, N)$, then the circular version of (20.5.10) would read:

$$\tilde{\mathbf{y}} = (\tilde{H}^T\mathbf{x})_{\text{down}}\,, \quad \tilde{\mathbf{y}} = \tilde{H}\mathbf{x}^{\text{up}} \tag{20.5.11}$$

The reduced matrix $\tilde{H}$ will have size $N\times N$, and after downsampling, the output $\tilde{\mathbf{y}} = (\tilde{H}^T\mathbf{x})_{\text{down}}$ will have size $N/2$. Similarly, in the operation $\tilde{\mathbf{y}} = \tilde{H}\mathbf{x}^{\text{up}}$, the upsampled vector $\mathbf{x}^{\text{up}}$ will have length N, as will the output $\tilde{\mathbf{y}}$. Before upsampling, the input $\mathbf{x}$ had length $N/2$. Because every other entry of $\mathbf{x}^{\text{up}}$ is zero, the matrix operation $\tilde{H}\mathbf{x}^{\text{up}}$ can be simplified by replacing $\tilde{H}$ by its "downsampled" version $\tilde{H}_{\text{down}}$ obtained by keeping every other column, and acting on the original vector $\mathbf{x}$, that is, $\tilde{H}\mathbf{x}^{\text{up}} = \tilde{H}_{\text{down}}\mathbf{x}$. The matrix elements of H_{down}, before they are wrapped modulo-N, are $(H_{\text{down}})_{nk} = h_{n-2k}$.

To clarify these remarks, we look at some examples. Consider a length-6 filter, such as D_3 or a Coiflet-1 filter, $\mathbf{h} = [h_0, h_1, h_2, h_3, h_4, h_5]^T$ and take $N = 8$. If the length-4 signal vector $\mathbf{x} = [x_0, x_2, x_4, x_6]^T$ is upsampled by a factor of two, it will become the length-8 vector $\mathbf{x}^{\text{up}} = [x_0, 0, x_2, 0, x_4, 0, x_6, 0]^T$. Before wrapping them modulo-8, the convolution matrices H, H_{down} generate the following equivalent outputs:

$$\mathbf{y} = \begin{bmatrix} h_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ h_1 & h_0 & 0 & 0 & 0 & 0 & 0 & 0 \\ h_2 & h_1 & h_0 & 0 & 0 & 0 & 0 & 0 \\ h_3 & h_2 & h_1 & h_0 & 0 & 0 & 0 & 0 \\ h_4 & h_3 & h_2 & h_1 & h_0 & 0 & 0 & 0 \\ h_5 & h_4 & h_3 & h_2 & h_1 & h_0 & 0 & 0 \\ 0 & h_5 & h_4 & h_3 & h_2 & h_1 & h_0 & 0 \\ 0 & 0 & h_5 & h_4 & h_3 & h_2 & h_1 & h_0 \\ 0 & 0 & 0 & h_5 & h_4 & h_3 & h_2 & h_1 \\ 0 & 0 & 0 & 0 & h_5 & h_4 & h_3 & h_2 \\ 0 & 0 & 0 & 0 & 0 & h_5 & h_4 & h_3 \\ 0 & 0 & 0 & 0 & 0 & 0 & h_5 & h_4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & h_5 \end{bmatrix} \begin{bmatrix} x_0 \\ 0 \\ x_2 \\ 0 \\ x_4 \\ 0 \\ x_6 \\ 0 \end{bmatrix} = \begin{bmatrix} h_0 & 0 & 0 & 0 \\ h_1 & 0 & 0 & 0 \\ h_2 & h_0 & 0 & 0 \\ h_3 & h_1 & 0 & 0 \\ h_4 & h_2 & h_0 & 0 \\ h_5 & h_3 & h_1 & 0 \\ 0 & h_4 & h_2 & h_0 \\ 0 & h_5 & h_3 & h_1 \\ 0 & 0 & h_4 & h_2 \\ 0 & 0 & h_5 & h_3 \\ 0 & 0 & 0 & h_4 \\ 0 & 0 & 0 & h_5 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_2 \\ x_4 \\ x_6 \end{bmatrix}$$

or, $\mathbf{y} = H\mathbf{x}^{\text{up}} = H_{\text{down}}\mathbf{x}$. The circular convolution output can be obtained by either wrapping $\mathbf{y}$ modulo-8 or by wrapping H, H_{down} columnwise:

$$\tilde{\mathbf{y}} = \tilde{H}\mathbf{x}^{\text{up}} = \tilde{H}_{\text{down}}\mathbf{x} = \begin{bmatrix} h_0 & 0 & h_4 & h_2 \\ h_1 & 0 & h_5 & h_3 \\ h_2 & h_0 & 0 & h_4 \\ h_3 & h_1 & 0 & h_5 \\ h_4 & h_2 & h_0 & 0 \\ h_5 & h_3 & h_1 & 0 \\ 0 & h_4 & h_2 & h_0 \\ 0 & h_5 & h_3 & h_1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_2 \\ x_4 \\ x_6 \end{bmatrix} \tag{20.5.12}$$

Similarly, in the analysis operation $\tilde{\mathbf{y}} = (\tilde{H}^T\mathbf{x})_{\text{down}}$, downsampling amounts to keeping every other row of the matrix $\tilde{H}^T$, which is $\tilde{H}^T_{\text{down}}$. For example, for the length-8 signal $\mathbf{x} = [x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7]^T$, the corresponding operation will be:

$$\tilde{\mathbf{y}} = (\tilde{H}^T\mathbf{x})_{\text{down}} = \tilde{H}^T_{\text{down}}\mathbf{x} = \begin{bmatrix} h_0 & h_1 & h_2 & h_3 & h_4 & h_5 & 0 & 0 \\ 0 & 0 & h_0 & h_1 & h_2 & h_3 & h_4 & h_5 \\ h_4 & h_5 & 0 & 0 & h_0 & h_1 & h_2 & h_3 \\ h_2 & h_3 & h_4 & h_5 & 0 & 0 & h_0 & h_1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} \tag{20.5.13}$$

The wrapped/downsampled convolution matrix $\tilde{H}_{\text{down}}$ can be calculated very simply in MATLAB, using, for example, the built-in convolution matrix function `convmtx` and the function `modwrap`:

```
H = convmtx(h(:), N);      % ordinary convolution matrix with N columns, h entered as column
H = H(:, 1:2:N);           % downsampled convolution matrix
H = modwrap(H, N);         % wrapped column-wise modulo-N
```

Because $\mathbf{h}$ is fairly short and N typically large, the convolution matrix H can be defined as sparse. This can be accomplished by replacing `convmtx` by the function `convmat`, which we encountered before in Sec. 23.10. Similar convolution matrices $\tilde{G}_{\text{down}}$ can be constructed for the conjugate mirror filter g_n. The function `dwtmat` constructs both matrices for any scaling filter $\mathbf{h}$ and signal length N using `convmat`:

```
[H,G] = dwtmat(h,N);      % sparse DWT matrices
```

The output matrices H, G are defined as sparse and have dimension $N\times(N/2)$. They represent the matrices $\tilde{H}_{\text{down}}, \tilde{G}_{\text{down}}$.

We can now state the precise form of the matrix version of the periodized DWT algorithm. Given a signal (column) vector $\mathbf{x}$ of length $N = 2^J$, we define the DWT matrices H_j, G_j at level j with dimension $N_j\times(N_j/2)$, where $N_j = 2^j$, by

$$[H_j, G_j] = \text{dwtmat}(\mathbf{h}, N_j)\,, \qquad J_0 + 1 \le j \le J \tag{20.5.14}$$

Then, the analysis and synthesis algorithms are as follows, initialized with $\mathbf{c}_J = \mathbf{x}$,

$$\begin{aligned} &\text{(DWT)} \quad \boxed{\begin{aligned} \mathbf{c}_{j-1} &= H_j^T\mathbf{c}_j \\ \mathbf{d}_{j-1} &= G_j^T\mathbf{c}_j \end{aligned}} \quad j = J, J{-}1, \dots, J_0 + 1 \\ &\text{(IDWT)} \quad \boxed{\mathbf{c}_j = H_j\mathbf{c}_{j-1} + G_j\mathbf{d}_{j-1}} \quad j = J_0{+}1, J_0{+}2, \dots, J \end{aligned} \tag{20.5.15}$$

The column vector $\mathbf{c}_j$ has dimension $N_j = 2^j$, while the vectors $\mathbf{c}_{j-1}, \mathbf{d}_{j-1}$ have dimension half of that, $N_{j-1} = N_j/2 = 2^{j-1}$. The computations for the forward and inverse transforms are illustrated in Fig. 20.5.2. The *discrete wavelet transform* of $\mathbf{x}$ to level J_0 is the concatenation of the coefficient vectors:

$$\mathbf{w} = \begin{bmatrix} \mathbf{c}_{J_0} \\ \mathbf{d}_{J_0} \\ \mathbf{d}_{J_0+1} \\ \vdots \\ \mathbf{d}_{J-1} \end{bmatrix} \qquad \text{(DWT)} \tag{20.5.16}$$

Its total dimension is $N = 2^J$, as can be verified easily,

$$2^{J_0} + 2^{J_0} + \left(2^{J_0+1} + \cdots + 2^{J-1}\right) = 2^J$$

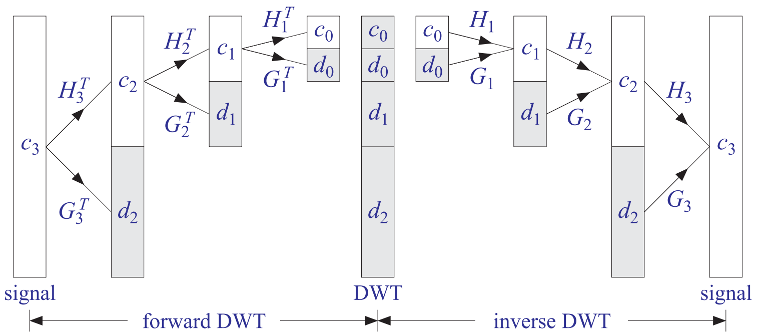


Fig. 20.5.2 Forward and inverse DWT in matrix form.

At each level j, the $N_j{\times}N_j$ matrix $U_j = [H_j, G_j]$ is an orthogonal matrix, as required by the consistency of the analysis and synthesis steps:

$$\begin{bmatrix}\mathbf{c}_{j-1}\\ \mathbf{d}_{j-1}\end{bmatrix} = \begin{bmatrix}H_j^T\\ G_j^T\end{bmatrix}\mathbf{c}_j \quad\Leftrightarrow\quad \mathbf{c}_j = H_j\mathbf{c}_{j-1} + G_j\mathbf{d}_{j-1} = [H_j, G_j]\begin{bmatrix}\mathbf{c}_{j-1}\\ \mathbf{d}_{j-1}\end{bmatrix}$$

implying the conditions $U_j^T U_j = U_j U_j^T = I_{N_j}$, or,

$$\begin{bmatrix}H_j^T\\ G_j^T\end{bmatrix}[H_j, G_j] = [H_j, G_j]\begin{bmatrix}H_j^T\\ G_j^T\end{bmatrix} = I_{N_j}$$

which are equivalent to the orthogonality conditions:

$$H_j^T H_j = G_j^T G_j = I_{N_j/2}\,, \quad H_j^T G_j = 0\,, \quad H_j H_j^T + G_j G_j^T = I_{N_j} \tag{20.5.17}$$

These follow from the scaling filter orthogonality properties (20.3.1). To see the mechanics by which this happens, consider again our length-6 filter h_n and the corresponding CMF filter g_n defined by $[g_0, g_1, g_2, g_3, g_4, g_5] = [h_5, -h_4, h_3, -h_2, h_1, -h_0]$. Let us also define the cross-correlation quantities:

$$R_k = \sum_n h_n h_{n-2k} \quad\Rightarrow\quad \begin{cases} R_0 = h_0^2 + h_1^2 + h_2^2 + h_3^2 + h_4^2 + h_5^2 \\ R_1 = h_5h_3 + h_4h_2 + h_3h_1 + h_2h_0 \\ R_2 = h_5h_1 + h_4h_0 \end{cases} \tag{20.5.18}$$

From Eq. (20.3.1), we have $R_k = \delta_k$, but let us not assume this just yet, but rather treat h_n as an arbitrary filter and g_n as the corresponding CMF filter. Then, starting with level $J = 3$, the wavelet matrices H_j at $j = 3, 2, 1$, will be:

$$H_3 = \begin{bmatrix} h_0 & 0 & h_4 & h_2 \\ h_1 & 0 & h_5 & h_3 \\ h_2 & h_0 & 0 & h_4 \\ h_3 & h_1 & 0 & h_5 \\ h_4 & h_2 & h_0 & 0 \\ h_5 & h_3 & h_1 & 0 \\ 0 & h_4 & h_2 & h_0 \\ 0 & h_5 & h_3 & h_1 \end{bmatrix}, \qquad \begin{aligned} H_2 &= \begin{bmatrix} h_0 + h_4 & h_2 \\ h_1 + h_5 & h_3 \\ h_2 & h_0 + h_4 \\ h_3 & h_1 + h_5 \end{bmatrix} \\ H_1 &= \begin{bmatrix} h_0 + h_2 + h_4 \\ h_1 + h_3 + h_5 \end{bmatrix} \end{aligned} \tag{20.5.19}$$

with similar definitions for G_j, $j = 3, 2, 1$. By explicit multiplication, we can verify:

$$H_3^T H_3 = G_3^T G_3 = \begin{bmatrix} R_0 & R_1 & 2R_2 & R_1 \\ R_1 & R_0 & R_1 & 2R_2 \\ 2R_2 & R_1 & R_0 & R_1 \\ R_1 & 2R_2 & R_1 & R_0 \end{bmatrix}, \quad H_3^T G_3 = 0$$

$$H_3 H_3^T + G_3 G_3^T = \begin{bmatrix} R_0 & 0 & R_1 & 0 & 2R_2 & 0 & R_1 & 0 \\ 0 & R_0 & 0 & R_1 & 0 & 2R_2 & 0 & R_1 \\ R_1 & 0 & R_0 & 0 & R_1 & 0 & 2R_2 & 0 \\ 0 & R_1 & 0 & R_0 & 0 & R_1 & 0 & 2R_2 \\ 2R_2 & 0 & R_1 & 0 & R_0 & 0 & R_1 & 0 \\ 0 & 2R_2 & 0 & R_1 & 0 & R_0 & 0 & R_1 \\ R_1 & 0 & 2R_2 & 0 & R_1 & 0 & R_0 & 0 \\ 0 & R_1 & 0 & 2R_2 & 0 & R_1 & 0 & R_0 \end{bmatrix}$$

Similarly, we have,

$$H_2^T H_2 = G_2^T G_2 = \begin{bmatrix} R_0 + 2R_2 & 2R_1 \\ 2R_1 & R_0 + 2R_2 \end{bmatrix}, \quad H_2^T G_2 = 0$$

$$H_2 H_2^T + G_2 G_2^T = \begin{bmatrix} R_0 + 2R_2 & 0 & 2R_1 & 0 \\ 0 & R_0 + 2R_2 & 0 & 2R_1 \\ 2R_1 & 0 & R_0 + 2R_2 & 0 \\ 0 & 2R_1 & 0 & R_0 + 2R_2 \end{bmatrix}$$

$$H_1^T H_1 = G_1^T G_1 = R_0 + 2R_1 + 2R_2\,, \quad H_1^T G_1 = 0$$

$$H_1 H_1^T + G_1 G_1^T = \begin{bmatrix} R_0 + 2R_1 + 2R_2 & 0 \\ 0 & R_0 + 2R_1 + 2R_2 \end{bmatrix}$$

Setting $R_0 = 1$ and $R_1 = R_2 = 0$ in all of the above, we verify the orthogonality properties (20.5.17) at all levels $j = 3, 2, 1$. We note that the matrix H_{j-1} can be derived very simply from H_j by keeping only the first $N_{j-1}/2 = N_j/4$ columns and wrapping them modulo-N_{j-1}, that is, in MATLAB notation:

$$H_{j-1} = \text{modwrap}\big(H_j(:, 1:N_{j-1}/2),\, N_{j-1}\big)\,, \qquad j = J, J{-}1, \dots, J_0 + 1 \tag{20.5.20}$$

and similarly for G_{j-1}. This simple operation has been incorporated into the function `dwtwrap`, with usage:

```
H_lower = dwtwrap(H);          % wrap a DWT matrix into a lower one
```

This is evident in Eq. (20.5.19), where H_2 is derivable from H_3, and H_1 from H_2. Because the successive DWT matrices H_j, G_j have different dimensions, $2^j{\times}2^{j-1}$, it is convenient to use a cell array to store them in MATLAB. The function `dwtcell` constructs and stores them in sparse form:

```
F = dwtcell(h,N);          % cell array of sparse DWT matrices
```

with the conventions $H_j = F\{1,j\}$ and $G_j = F\{2,j\}$, for $J_0 + 1 \le j \le J$, where N is the highest dimension. The function `fwtm` implements the analysis algorithm in (20.5.15). Its inputs are the signal vector $\mathbf{x}$, the cell array F, and the lowest desired level J_0,

```
w = fwtm(x,F,J0);          % fast wavelet transform in matrix form
```

The vector $\mathbf{w}$ is as in Eq. (20.5.16). If J_0 is omitted, it defaults to $J_0 = 0$. Once the cell array F is created, the function `fwtm` is extremely fast, even faster than the convolution-based function `fwt` discussed below. The function `ifwtm` implements the inverse DWT synthesis algorithm in (20.5.15),

```
x = ifwtm(w,F,J0);          % inverse fast wavelet transform in matrix form
```

An example is the following MATLAB code using the D_3 scaling filter:

```
x = [1 2 3 4 5 6 7 8];
h = daub(3);                  % h = [0.3327, 0.8069, 0.4599, −0.1350, −0.0854, 0.0352]
F = dwtcell(h,8);             % construct cell array of DWT matrices

w = fwtm(x,F,0);              % w = [12.7279, −1.4794, −4.4090, 2.2467, 0, 0, −3.7938, 0.9653]
x = ifwtm(w,F,0);             % returns x = [1, 2, 3, 4, 5, 6, 7, 8]
                              % similarly, for J0 = 1, 2, 3, we find,
w = fwtm(x,F,1);              % w = [7.9539, 10.0461, −4.409, 2.2467, 0, 0, −3.7938, 0.9653]
w = fwtm(x,F,2);              % w = [2.5702, 5.3986, 8.6288, 8.8583, 0, 0, −3.7938, 0.9653]
w = fwtm(x,F,3);              % w = [1, 2, 3, 4, 5, 6, 7, 8] = x, as expected since J = 3
```

These outputs can be understood by looking at the individual matrix operations. Defining, $\mathbf{c}_3 = \mathbf{x} = [1,2,3,4,5,6,7,8]^T$, and the level-3 matrices H_3, G_3, obtained from the call, $[H_3, G_3] = \text{dwtmat}(\mathbf{h}, 8)$,

$$H_3 = \begin{bmatrix} 0.3327 & 0 & -0.0854 & 0.4599 \\ 0.8069 & 0 & 0.0352 & -0.1350 \\ 0.4599 & 0.3327 & 0 & -0.0854 \\ -0.1350 & 0.8069 & 0 & 0.0352 \\ -0.0854 & 0.4599 & 0.3327 & 0 \\ 0.0352 & -0.1350 & 0.8069 & 0 \\ 0 & -0.0854 & 0.4599 & 0.3327 \\ 0 & 0.0352 & -0.1350 & 0.8069 \end{bmatrix}, \; G_3 = \begin{bmatrix} 0.0352 & 0 & 0.8069 & -0.1350 \\ 0.0854 & 0 & -0.3327 & -0.4599 \\ -0.1350 & 0.0352 & 0 & 0.8069 \\ -0.4599 & 0.0854 & 0 & -0.3327 \\ 0.8069 & -0.1350 & 0.0352 & 0 \\ -0.3327 & -0.4599 & 0.0854 & 0 \\ 0 & 0.8069 & -0.1350 & 0.0352 \\ 0 & -0.3327 & -0.4599 & 0.0854 \end{bmatrix}$$

we calculate the level-2 coefficient vectors $\mathbf{c}_2, \mathbf{d}_2$, and the level-2 DWT,

$$\mathbf{c}_2 = H_3^T\mathbf{c}_3 = \begin{bmatrix} 2.5702 \\ 5.3986 \\ 8.6288 \\ 8.8583 \end{bmatrix}, \quad \mathbf{d}_2 = G_3^T\mathbf{c}_3 = \begin{bmatrix} 0 \\ 0 \\ -3.7938 \\ 0.9653 \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} \mathbf{c}_2 \\ \mathbf{d}_2 \end{bmatrix}$$

which agrees with the above MATLAB output of `fwtm(x,F,2)`. Then, from the matrices H_2, G_2, obtained from $[H_2, G_2] = \text{dwtmat}(\mathbf{h}, 4)$, or, from $H_2 = \text{dwtwrap}(H_3)$,

$$H_2 = \begin{bmatrix} 0.2472 & 0.4599 \\ 0.8421 & -0.1350 \\ 0.4599 & 0.2472 \\ -0.1350 & 0.8421 \end{bmatrix}, \; G_2 = \begin{bmatrix} 0.8421 & -0.1350 \\ -0.2472 & -0.4599 \\ -0.1350 & 0.8421 \\ -0.4599 & -0.2472 \end{bmatrix}$$

we calculate the level-1 coefficient vectors $\mathbf{c}_1, \mathbf{d}_1$, and the level-1 DWT,

$$\mathbf{c}_1 = H_2^T\mathbf{c}_2 = \begin{bmatrix} 7.9539 \\ 10.0461 \end{bmatrix}, \quad \mathbf{d}_1 = G_2^T\mathbf{c}_2 = \begin{bmatrix} -4.4090 \\ 2.2467 \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{d}_1 \\ \mathbf{d}_2 \end{bmatrix}$$

which agrees with the above MATLAB output of `fwtm(x,F,1)`. Finally, from the matrices H_1, G_1, obtained from $[H_1, G_1] = \text{dwtmat}(\mathbf{h}, 2)$, or, from $H_1 = \text{dwtwrap}(H_2)$,

$$H_1 = \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix}, \; G_1 = \begin{bmatrix} 0.7071 \\ -0.7071 \end{bmatrix}$$

we find the level-0 coefficient vectors $\mathbf{c}_0, \mathbf{d}_0$, and the level-0 DWT,

$$\mathbf{c}_0 = H_1^T\mathbf{c}_1 = 12.7279, \quad \mathbf{d}_0 = G_1^T\mathbf{c}_1 = -1.4794, \quad \mathbf{w} = \begin{bmatrix} \mathbf{c}_0 \\ \mathbf{d}_0 \\ \mathbf{d}_1 \\ \mathbf{d}_2 \end{bmatrix}$$

Orthogonal DWT Transformation

The mapping of a length-N signal vector $\mathbf{x}$ to the length-N vector $\mathbf{w}$ of wavelet coefficients given in Eq. (20.5.16) is equivalent to an orthogonal matrix transformation, say, $\mathbf{w} = W^T\mathbf{x}$, with inverse $\mathbf{x} = W\mathbf{w}$, such that $W^TW = WW^T = I_N$. The overall $N{\times}N$ matrix W depends on the stopping level J_0 and can be constructed in terms of the matrices H_j, G_j of the successive stages of the analysis or synthesis algorithms. For example, we have for $N = 2^3$, and $J_0 = 2, 1, 0$,

$$W = [H_3,\, G_3]$$

$$W = [H_3[H_2,\, G_2],\, G_3] = [H_3H_2,\, H_3G_2,\, G_3]$$

$$W = [H_3H_2[H_1,\, G_1],\, H_3G_2,\, G_3] = [H_3H_2H_1,\, H_3H_2G_1,\, H_3G_2,\, G_3]$$

We verify the reconstruction of $\mathbf{x}$ from $\mathbf{w}$ starting at $J_0 = 0$,

$$[H_3H_2H_1,\, H_3H_2G_1,\, H_3G_2,\, G_3]\begin{bmatrix} \mathbf{c}_0 \\ \mathbf{d}_0 \\ \mathbf{d}_1 \\ \mathbf{d}_2 \end{bmatrix} = \begin{cases} H_3H_2\,(H_1\mathbf{c}_0 + G_1\mathbf{d}_0) + H_3G_2\mathbf{d}_1 + G_3\mathbf{d}_2 = \\ H_3\,(H_2\mathbf{c}_1 + G_2\mathbf{d}_1) + G_3\mathbf{d}_2 = \\ H_3\mathbf{c}_2 + G_3\mathbf{d}_2 = \mathbf{c}_3 = \mathbf{x} \end{cases}$$

The construction of W can be carried out with the following very simple recursive algorithm, stated in MATLAB notation,

$$\begin{aligned}
&W = I_N\,, \quad (N = 2^J)\\
&\text{for } j = J, J-1, \dots, J_0+1,\\
&\qquad W(:,\,1:2^j) = W(:,\,1:2^j)\,[H_j, G_j]
\end{aligned} \tag{20.5.21}$$

The algorithm updates the first 2^j columns of W at each level j. The MATLAB function `fwtmat` implements (20.5.21) and constructs W as a sparse matrix:

```
W = fwtmat(h,N,J0);        % overall DWT orthogonal matrix
```

As an example, for the D_3 scaling filter and $N = 8$ and lowest level $J_0 = 0$, we find:

$$W = \begin{bmatrix}
0.3536 & -0.3806 & 0.0802 & -0.2306 & 0.0352 & 0 & 0.8069 & -0.1350\\
0.3536 & -0.0227 & 0.7368 & -0.0459 & 0.0854 & 0 & -0.3327 & -0.4599\\
0.3536 & 0.2197 & 0.3443 & -0.1940 & -0.1350 & 0.0352 & 0 & 0.8069\\
0.3536 & 0.5535 & -0.3294 & -0.3616 & -0.4599 & 0.0854 & 0 & -0.3327\\
0.3536 & 0.3806 & -0.2306 & 0.0802 & 0.8069 & -0.1350 & 0.0352 & 0\\
0.3536 & 0.0227 & -0.0459 & 0.7368 & -0.3327 & -0.4599 & 0.0854 & 0\\
0.3536 & -0.2197 & -0.1940 & 0.3443 & 0 & 0.8069 & -0.1350 & 0.0352\\
0.3536 & -0.5535 & -0.3616 & -0.3294 & 0 & -0.3327 & -0.4599 & 0.0854
\end{bmatrix}$$

which generates the same DWT as the example above:

```
x = [1 2 3 4 5 6 7 8]'; h = daub(3); W = fwtmat(h,8,0);
w = W'*x;    % gives w = [12.7279, −1.4794, −4.4090, 2.2467, 0, 0, −3.7938, 0.9653]^T
```

The matrix W becomes more and more sparse as N increases. Its sparsity pattern is illustrated in Fig. 20.5.3, for the case of the D_3 scaling filter and dimensions $N = 64$ and $N = 512$. The graphs were generated by the MATLAB code:

```
h = daub(3); N = 64; W = fwtmat(h,N,0); spy(W); percent_nonzero = 100*nnz(W)/N^2
```

The percentages of nonzero entries were 30.5% for $N = 64$, and 6.7% for $N = 512$.

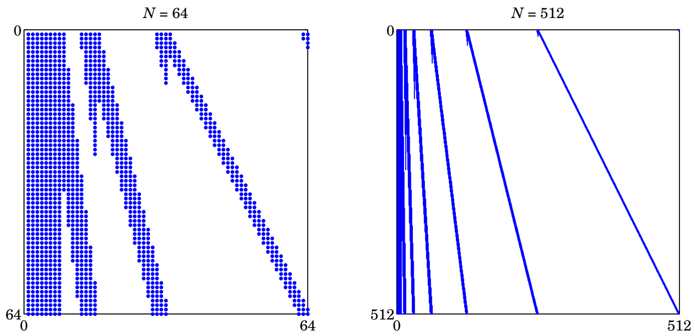


Fig. 20.5.3 Sparsity patterns of DWT matrices.

DWT in Convolutional Form

Next, we look at the detailed implementation of Eq. (20.5.9) using filtering by circular convolution. For practical implementation, we must replace the time-reversed filters $\bar{h}_n, \bar{g}_n$ of the analysis algorithm by their reversed versions, which are delayed by the filter order M to make them causal, that is, $h^R_n = \bar{h}_{n-M} = h_{M-n}$, or in the z-domain $H^R(z) = z^{-M}\bar{H}(z) = z^{-M}H(z^{-1})$.

In order to get the same output as the matrix implementation, we must compensate for such a delay by advancing the input by the same amount. In other words, filtering by $\bar{H}(z)$ is equivalent to advancing the input and then filtering by $H^R(z)$. In the z-domain,

$$Y(z) = \bar{H}(z)X(z) = z^M H^R(z)X(z) = H^R(z)\,[z^M X(z)]$$

With these changes, Eq. (20.5.9) now reads,

$$\boxed{\begin{aligned}
&\text{advance}(\mathbf{c}_j, M)\\
&\mathbf{c}_{j-1} = \big(\text{circonv}(\mathbf{h}^R, \mathbf{c}_j)\big)_{\text{down}}\\
&\mathbf{d}_{j-1} = \big(\text{circonv}(\mathbf{g}^R, \mathbf{c}_j)\big)_{\text{down}}
\end{aligned}} \quad j = J, J-1, \dots, J_0+1$$

$$\boxed{\mathbf{c}_j = \text{circonv}(\mathbf{h}, \mathbf{c}^{\text{up}}_{j-1}) + \text{circonv}(\mathbf{g}, \mathbf{c}^{\text{up}}_{j-1})} \quad j = J_0+1, J_0+2, \dots, J \tag{20.5.22}$$

The concrete MATLAB implementation for computing the forward DWT is:

```
g = cmf(h);                        % conjugate mirror of h
hR = flip(h);                      % reversed h
gR = flip(g);
M = length(h) - 1;                 % filter order

c = x(:);                          % initial smooth, x has length 2^J
w = [];                            % DWT coefficient vector

for j=J:-1:J0+1,                   % loop from finest down to coarsest level
   c = advance(c, M);              % length(c) = 2^j
   d = dn2(circonv(gR, c, 2^j));   % convolve circularly and downsample
   c = dn2(circonv(hR, c, 2^j));
   w = [d; w];                     % prepend detail d to previous details
end

w = [c; w];                        % prepend last smooth
```

The function `advance` actually performs a *circular* time-advance modulo the length of its argument vector. The function `dn2` performs downsampling by a factor of two. The results of each loop calculation are appended into the DWT vector $\mathbf{w}$. Similarly, the inverse DWT can be calculated by the loop:

```
w = w(:);                                                % work columnwise
c = wcoeff(w,J0);                                        % coarsest smooth at level J0
for j=J0+1:J,
  d = wcoeff(w,J0,j-1);                                  % get detail at level j−1
  c = circonv(h, up2(c), 2^j) + circonv(g, up2(d), 2^j); % output c is 2^j-dimensional
end
x = c;                                                   % reconstructed x
```

Here, the function `wcoeff(w,J0,j-1)` extracts the subvector $\mathbf{d}_{j-1}$ from the wavelet transform vector $\mathbf{w}$, and the function `up2` upsamples by a factor of two.

The MATLAB functions `fwt` and `ifwt` incorporate the above code segments to realize the convolutional forms of the DWT and IDWT:

```
w = fwt(x,h,J0);     % fast wavelet transform
x = ifwt(w,h,J0);    % inverse fast wavelet transform
```

Some examples are,

```
x = [1 2 3 4 5 6 7 8];
h = daub(3);
w = fwt(x,h,0);      % w = [12.7279, −1.4794, −4.4090, 2.2467, 0, 0, −3.7938, 0.9653]
x = ifwt(w,h,0);     % returns x = [1, 2, 3, 4, 5, 6, 7, 8]
w = fwt(x,h,1);      % w = [7.9539, 10.0461, −4.409, 2.2467, 0, 0, −3.7938, 0.9653]
w = fwt(x,h,2);      % w = [2.5702, 5.3986, 8.6288, 8.8583, 0, 0, −3.7938, 0.9653]
w = fwt(x,h,3);      % w = [1, 2, 3, 4, 5, 6, 7, 8] = x, as expected
```

A second optional output of `fwt` (and `fwtm`) is the $N\times(J-J_0+1)$ matrix V whose columns are the sub-blocks of $\mathbf{w}$ according to their resolution,

$$\mathbf{w}=\begin{bmatrix}\mathbf{c}_{J_0}\\ \mathbf{d}_{J_0}\\ \mathbf{d}_{J_0+1}\\ \vdots\\ \mathbf{d}_{J-1}\end{bmatrix}\quad\Rightarrow\quad V=\begin{bmatrix}\mathbf{c}_{J_0} & 0 & 0 & \cdots & 0\\ 0 & \mathbf{d}_{J_0} & 0 & \cdots & 0\\ 0 & 0 & \mathbf{d}_{J_0+1} & \cdots & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & 0 & \cdots & \mathbf{d}_{J-1}\end{bmatrix} \tag{20.5.23}$$

It is obtained by the calls,

```
[w,V] = fwt(x,h,J0);
[w,V] = fwtm(x,F,J0);
```

The computations in `fwt` are very efficient, resulting in an $O(N)$ algorithm, or more precisely, $O(MN)$, where M is the filter order. By contrast, the FFT is an $O(N\log_2 N)$ algorithm. However, because of their sparsity, the matrix versions are just as efficient if the sparse wavelet matrices are precomputed.

We mentioned earlier that there are several different implementations of the DWT. Different packages may produce different answers, sometimes only differing by a sign or a cyclic permutation within each level. For example, we obtained the following answers for the above example ($\mathbf{x}=[1,2,3,4,5,6,7,8]$ with D_3 and $J_0=0$) from the packages:

$$\begin{array}{l}
\mathbf{w}=[12.7279,-1.4794,-4.4090,\ \ 2.2467,\ \ 0.0000,\ \ 0.0000,-3.7938,\ \ 0.9653]=\text{fwt - ours}\\
\mathbf{w}=[12.7279,\ \ 1.4794,\ \ 4.4090,-2.2467,\ \ 3.7938,-0.9653,\ \ 0.0000,\ \ 0.0000]=\text{Wavelab850, Ref. [584]}\\
\mathbf{w}=[12.7279,-1.4794,-4.4090,\ \ 2.2467,-3.7938,\ \ 0.9653,\ \ 0.0000,\ \ 0.0000]=\text{Wavethresh, Ref. [588]}\\
\mathbf{w}=[12.7279,\ \ 1.4794,-2.2467,\ \ 4.4090,-0.9653,\ \ 3.7938,\ \ 0.0000,\ \ 0.0000]=\text{WMTSA, Ref. [592]}\\
\mathbf{w}=[12.7279,-1.4794,\ \ 2.2467,-4.4090,\ \ 0.9653,\ \ 0.0000,\ \ 0.0000,-3.7938]=\text{Uvi-Wave, Ref. [593]}\\
\mathbf{w}=[12.7279,\ \ 1.4794,\ \ 4.4090,-2.2467,\ \ 0.0000,\ \ 0.0000,\ \ 3.7938,-0.9653]=\text{Getz, Ref. [594]}\\
\mathbf{w}=[12.7279,-1.4794,-4.4090,\ \ 2.2467,\ \ 0.0000,\ \ 0.0000,-3.7938,\ \ 0.9653]=\text{Wavekit, Ref. [595]}
\end{array}$$

20.6 Multiresolution Decomposition

The multiresolution decomposition defined in Eq. (20.1.13), with coarsest level J_0, which was illustrated by Example 20.1.1, and implemented by the function `dwtdec`,

$$f(t)=\sum_n c_{Jn}\phi_{Jn}(t)=\sum_n c_{J_0n}\phi_{J_0n}(t)+\sum_{j=J_0}^{J-1}\sum_n d_{jn}\psi_{jn}(t)\,, \tag{20.6.1}$$

can be given a vectorial interpretation. Let $\mathbf{x}$ be the $N=2^J$ dimensional vector of time samples of the function $f(t)$ at the finest level J, and let W be the orthogonal DWT matrix down to level J_0, with corresponding DWT, $\mathbf{w}=W^T\mathbf{x}$, and inverse $\mathbf{x}=W\mathbf{w}$.

Writing the DWT $\mathbf{w}$ in the partitioned form of Eq. (20.5.23), we may write $\mathbf{x}$ as the sum of multiresolution components, corresponding to the terms of (20.6.1), with each term representing the part of $\mathbf{x}$ arising from a particular level j with all the other levels having zero coefficients:

$$\begin{aligned}
\mathbf{x}=W\begin{bmatrix}\mathbf{c}_{J_0}\\ \mathbf{d}_{J_0}\\ \mathbf{d}_{J_0+1}\\ \vdots\\ \mathbf{d}_{J-1}\end{bmatrix}
&=W\begin{bmatrix}\mathbf{c}_{J_0}\\ 0\\ 0\\ \vdots\\ 0\end{bmatrix}+W\begin{bmatrix}0\\ \mathbf{d}_{J_0}\\ 0\\ \vdots\\ 0\end{bmatrix}+W\begin{bmatrix}0\\ 0\\ \mathbf{d}_{J_0+1}\\ \vdots\\ 0\end{bmatrix}+\cdots+W\begin{bmatrix}0\\ 0\\ 0\\ \vdots\\ \mathbf{d}_{J-1}\end{bmatrix}\\
&=\mathbf{x}_{J_0}+\underbrace{\left(\bar{\mathbf{x}}_{J_0}+\bar{\mathbf{x}}_{J_0+1}+\cdots+\bar{\mathbf{x}}_{J-1}\right)}_{\mathbf{x}_{J_0}^{\perp}}\\
&=\mathbf{x}_{J_0}+\mathbf{x}_{J_0}^{\perp}
\end{aligned} \tag{20.6.2}$$

The terms $\mathbf{x}_{J_0},\mathbf{x}_{J_0}^{\perp}$ represent the two parts of $\mathbf{x}$ lying in the subspaces V_{J_0} and $V_{J_0}^{\perp}$. The individual terms of $\mathbf{x}_{J_0}^{\perp}=\bar{\mathbf{x}}_{J_0}+\bar{\mathbf{x}}_{J_0+1}+\cdots+\bar{\mathbf{x}}_{J-1}$ contain all the details for levels $J_0\le j\le J-1$. The various components are mutually orthogonal, as follows from the property $WW^T=I$, and the non-overlapping of the sub-blocks of $\mathbf{w}$,

$$\begin{aligned}
\mathbf{x}_{J_0}^T\bar{\mathbf{x}}_j&=0\,,\quad J_0\le j\le J-1\\
\bar{\mathbf{x}}_i^T\bar{\mathbf{x}}_j&=0\,,\quad J_0\le i,j\le J-1\,,\quad i\ne j
\end{aligned} \tag{20.6.3}$$

For the "diagonal" terms, we obtain the norms, again following from $WW^T=I$,

$$\|\mathbf{x}_{J_0}\|^2=\|\mathbf{c}_{J_0}\|^2\,,\quad \|\bar{\mathbf{x}}_j\|^2=\|\mathbf{d}_j\|^2\,,\quad J_0\le j\le J-1 \tag{20.6.4}$$

where $\|\mathbf{x}\|^2=\mathbf{x}^T\mathbf{x}$, which lead to the sum,

$$\|\mathbf{x}\|^2=\|\mathbf{x}_{J_0}\|^2+\sum_{j=J_0}^{J-1}\|\bar{\mathbf{x}}_j\|^2=\|\mathbf{c}_{J_0}\|^2+\sum_{j=J_0}^{J-1}\|\mathbf{d}_j\|^2=\|\mathbf{w}\|^2 \tag{20.6.5}$$

The $N\times(J-J_0+1)$ matrix $X=\left[\mathbf{x}_{J_0},\,\bar{\mathbf{x}}_{J_0},\,\bar{\mathbf{x}}_{J_0+1}\,\cdots\,\bar{\mathbf{x}}_{J-1}\right]$ incorporates the individual orthogonal columns and is produced as the output of the MATLAB function `dwtdec`,

```
X = dwtdec(x,h,J0);        % DWT decomposition into orthogonal multiresolution components
```

In fact, X is the product $X = WV$, where V is the DWT-component matrix given in (20.5.23). As an example of dwtdec, we have for $\mathbf{x} = [1,2,3,4,5,6,7,8]^T$ and D_3,

$$\mathbf{x} = \begin{bmatrix} 1\\2\\3\\4\\5\\6\\7\\8 \end{bmatrix} \quad\Rightarrow\quad \mathbf{w} = \begin{bmatrix} 12.7279\\ -1.4794\\ -4.4090\\ 2.2467\\ 0\\ 0\\ -3.7938\\ 0.9653 \end{bmatrix} \quad\Rightarrow\quad X = \begin{bmatrix} 4.5000 & 0.5631 & -0.8716 & -3.1915\\ 4.5000 & 0.0337 & -3.3518 & 0.8181\\ 4.5000 & -0.3251 & -1.9538 & 0.7789\\ 4.5000 & -0.8188 & 0.6399 & -0.3211\\ 4.5000 & -0.5631 & 1.1967 & -0.1336\\ 4.5000 & -0.0337 & 1.8578 & -0.3241\\ 4.5000 & 0.3251 & 1.6287 & 0.5462\\ 4.5000 & 0.8188 & 0.8541 & 1.8271 \end{bmatrix}$$

generated with the MATLAB code and test,

```
h = daub(3); x = [1 2 3 4 5 6 7 8]'; X = dwtdec(x,h,0);
[w,V] = fwt(x,h,0); W = fwtmat(h,8,0); norm(X-W*V)
```

20.7 Wavelet Denoising

Figure 20.7.1 shows some wavelet denoising examples consisting of the same four signals (bumps, blocks, heavisine, doppler) that we discussed in Sec. 23.19 under local polynomial modeling with adaptive variable bandwidth. These examples have served as benchmarks in the wavelet denoising literature [569–572].

Fig. 20.7.1 should be compared with Figs. 23.19.1–23.19.4. It should be evident that the results are comparable, with, perhaps, local polynomial modeling doing a bit better. The MATLAB codes generating the noisy signals were given in Sec. 23.19. The following code segment illustrates the generation of the upper row of graphs and demonstrates the use of the denoising function wdenoise:

```
F = inline('1./(1 + abs(x)).^4');                       % bumps function

N = 2048; t = (0:N-1)'/N; x = zeros(size(t));          % normalize time to 0 ≤ t ≤ 1

t0 = [10 13 15 23 25 40 44 65 76 78 81]/100;           % signal parameters
a = [40,50,30,40,50,42,21,43,31,51,42] * 1.0523;
w = [5,5,6,10,10,30,10,10,5,8,5]/1000;

for i=1:length(a),                                      % construct noise-free signal
    x = x + a(i) * F((t-t0(i))/w(i));
end

seed=2009; randn('state',seed); v = randn(size(t));    % generate noise

y = x + v;                                              % noisy signal with SNR, σx/σv = 7

h = daub(8,2); J0=5; type=1;                            % use Symmlet-8 and soft thresholding
xd = wdenoise(y,h,J0,type);                             % wavelet denoising

figure; plot(t,y,'-'); figure; plot(t,x,'-'); figure; plot(t,xd,'r-');      % top row
```

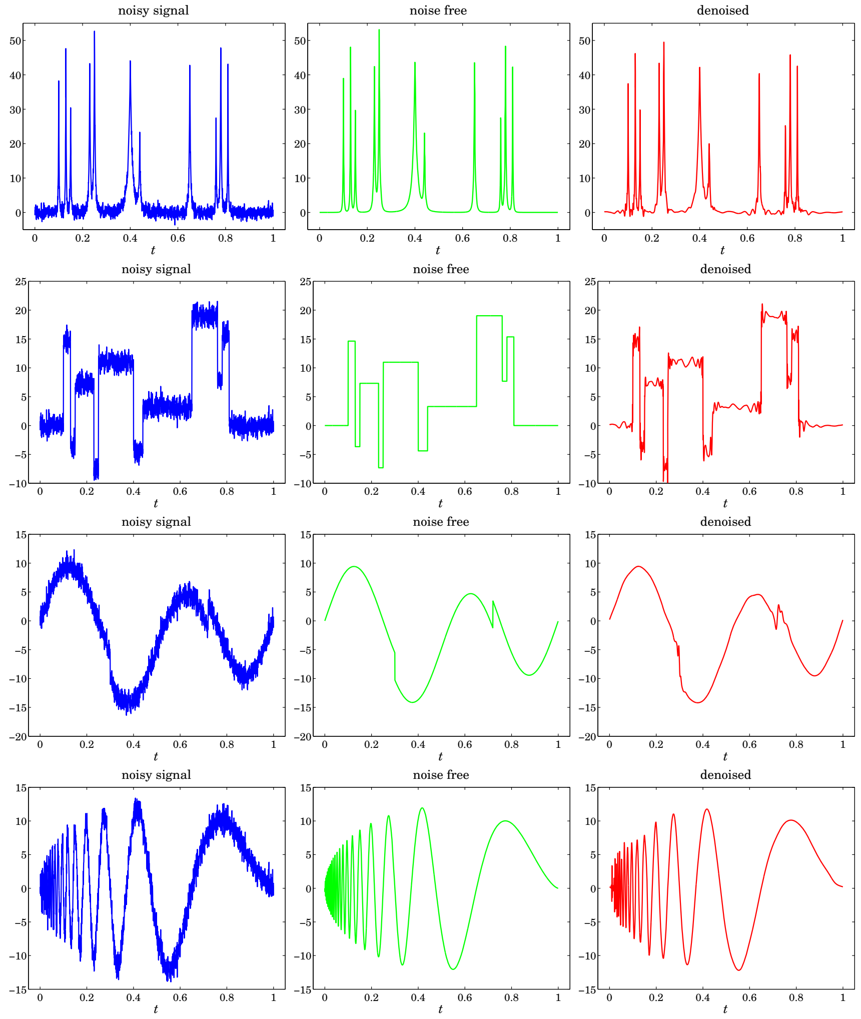


Fig. 20.7.1 Wavelet denoising.

The main idea in wavelet denoising is to (a) perform a DWT on the noisy signal down to some lower resolution level , (b) modify the wavelet detail coefficients by reducing them to zero if they fall below a certain threshold, and (c) perform an inverse DWT to obtain the denoised signal. The procedure is depicted below:

$$\mathbf{x} \xrightarrow{\text{DWT}} \mathbf{w} \xrightarrow{\text{thresh}} \mathbf{w}_{\text{thr}} \xrightarrow{\text{IDWT}} \mathbf{x}_{\text{thr}}$$

Given a wavelet coefficient d, we denote the thresholding operation by $d_{\text{thr}} = f(d,\lambda)$, where λ is threshold. There are various thresholding functions, but the two simplest ones are the so-called hard and soft thresholding, defined with the help of the unit-step function $u(x)$ as follows:

$$\begin{aligned} d_{\text{thr}} &= f(d,\lambda) = d\,u(|d|-\lambda) && \text{(hard)} \\ d_{\text{thr}} &= f(d,\lambda) = \operatorname{sign}(d)\,(|d|-\lambda)\,u(|d|-\lambda) && \text{(soft)} \end{aligned} \tag{20.7.1}$$

or, equivalently,

$$d_{\text{thr}}^{\text{hard}} = \begin{cases} d\,, & |d| \ge \lambda \\ 0\,, & |d| < \lambda \end{cases}\,, \qquad d_{\text{thr}}^{\text{soft}} = \begin{cases} d - \operatorname{sign}(d)\lambda\,, & |d| \ge \lambda \\ 0\,, & |d| < \lambda \end{cases}$$

If the wavelet transform starts at level J (input length $N = 2^J$) and proceeds down to level J_0, the wavelet transform coefficients will be $\mathbf{w} = \{c_{J_0 n}\,;\, d_{jn}\,,\ J_0 \le j \le J-1\}$. The thresholding operation is applied only to the detail coefficients d_{jn}, replacing them by their thresholded values, with a possibly level-dependent threshold λ_j, that is,

$$d_{jn}^{\text{thr}} = f(d_{jn}, \lambda_j) \tag{20.7.2}$$

The simplest possibility is to use the same threshold for all levels. Donoho & Johnstone [569] suggest the following "universal" threshold,

$$\lambda = \sigma\sqrt{2\log_2 N} \qquad \text{(universal threshold)} \tag{20.7.3}$$

where σ^2 is the variance of the additive noise in the data. Since σ is not known, it can be estimated from the wavelet detail coefficients $\mathbf{d}_{J-1}$ at level $J-1$, which for a smooth desired signal are presumably dominated mostly by the noise component. The vector $\mathbf{d} \equiv \mathbf{d}_{J-1}$ has length $N/2 = 2^{J-1}$ and one may estimate σ by using either the standard deviation of $\mathbf{d}$, or its mean-absolute-deviation (MAD), that is,

$$\hat{\sigma} = \operatorname{std}(\mathbf{d})\,, \qquad \hat{\sigma} = \frac{\operatorname{mad}(\mathbf{d})}{0.6745} = \frac{\operatorname{median}(\,|\mathbf{d} - \operatorname{median}(\mathbf{d})\,|\,)}{0.6745} \tag{20.7.4}$$

where the factor 0.6745 arises from the implicit assumption that $\mathbf{d}$ is a vector of zero-mean independent normally-distributed components (for a zero-mean, unit-variance, gaussian random variable x, one has the relationship, $\operatorname{median}(|x|) = 0.6745$).

Donoho & Johnstone's [569] so-called VisuShrink method uses the universal threshold with the MAD estimate of σ and soft thresholding. The MATLAB function `wdenoise` implements the VisuShrink procedure, but also allows the use of hard thresholding:

```
y = wdenoise(x,h,J0,type);        % wavelet denoising
```

It is possible to derive the soft thresholding rule, as well as some of the other rules, from a regularized optimization point of view. Let $\mathbf{y}$ and $\mathbf{w} = W^T\mathbf{y}$ be the noisy data vector and its DWT, and let $\hat{\mathbf{x}}$ and $\hat{\mathbf{w}} = W^T\hat{\mathbf{y}}$ be the sought estimate and its DWT of the desired signal component $\mathbf{x}$ in the noisy signal model $\mathbf{y} = \mathbf{x} + \mathbf{v}$. An estimation criterion similar to the smoothing spline and reproducing kernel criteria that we considered earlier is the following performance index,

$$\mathcal{J} = \|\mathbf{y} - \hat{\mathbf{x}}\|^2 + P(\hat{\mathbf{x}}) = \min$$

where the first term is the L_2-norm and the second, a positive penalty term. Since the DWT matrix W is orthogonal the first term can be written in terms of the DWTs $\|\mathbf{y} - \hat{\mathbf{x}}\|^2 = \|\mathbf{w} - \hat{\mathbf{w}}\|^2$. Therefore, with a redefinition of P, we may replace the above criterion with one that is formulated in the wavelet domain:

$$\begin{aligned} \mathcal{J} &= \|\mathbf{w} - \hat{\mathbf{w}}\|^2 + P(\hat{\mathbf{w}}) \\ &= \|\mathbf{c}_{J_0} - \hat{\mathbf{c}}_{J_0}\|^2 + \sum_{j=J_0}^{J-1} \|\mathbf{d}_j - \hat{\mathbf{d}}_j\|^2 + P(\hat{\mathbf{d}}_{J_0}, \dots, \hat{\mathbf{d}}_{J-1}) = \min \end{aligned} \tag{20.7.5}$$

where in the second expression, we used the component representation (20.5.23), and we assumed that P depends only on the wavelet detail coefficients. The following particular choice of P using the L_1 norm leads to the soft thresholding rule:

$$\mathcal{J} = \|\mathbf{c}_{J_0} - \hat{\mathbf{c}}_{J_0}\|^2 + \sum_{j=J_0}^{J-1}\sum_{n=0}^{N_j-1} \|d_{jn} - \hat{d}_{jn}\|^2 + 2\lambda \sum_{j=J_0}^{J-1}\sum_{n=0}^{N_j-1} |\hat{d}_{jn}| = \min \tag{20.7.6}$$

where $N_j = 2^j$ is the dimension of the vector $\mathbf{d}_j$. The minimization with respect to $\mathbf{c}_{J_0}$ gives $\hat{\mathbf{c}}_{J_0} = \mathbf{c}_{J_0}$. Since the d_{jn} terms are decoupled, their minimization can be carried on a typical such term, that is, with the simple scalar criterion:

$$\mathcal{J} = |d - \hat{d}|^2 + 2\lambda|\hat{d}| = \min \tag{20.7.7}$$

whose solution is the soft-thresholding rule,

$$\hat{d} = \begin{cases} d - \operatorname{sign}(d)\lambda\,, & |d| \ge \lambda \\ 0\,, & |d| < \lambda \end{cases} \tag{20.7.8}$$

Other variants of wavelet thresholding and other applications and uses of wavelets in statistics can be found in Refs. [567–581].

20.8 Undecimated Wavelet Transform

In this section, we discuss the undecimated wavelet transform (UWT), also known as the stationary, redundant, maximum-overlap, translation- or shift-invariant wavelet transform [496–509]. It has certain advantages over the conventional DWT exhibiting, for example, better performance in denoising applications. Its minor disadvantage is that it generates $N\log_2 N$ wavelet coefficients instead of N, and its computational cost is $O(N\log_2 N)$ instead of $O(N)$.

The essential feature of the wavelet transform is the property that successive stages of the analysis filter bank in Fig. 20.4.1 probe the frequency content of the input signal at successively lower frequency bands.

This property was depicted in Fig. 20.4.2 in which the output spectrum after three stages, shown at the bottom two graphs, was the result of filtering by the cascaded filter $\bar{H}(\omega)\bar{H}(2\omega)\bar{H}(4\omega)$, where $\omega = 2\pi f/f_s$, with f_s being the sampling rate at the finest scale. This frequency property is preserved whether the output is undecimated, as in the bottom right graph of Fig. 20.4.2, or decimated as in the left bottom graph. The reason for downsampling the outputs after each splitting stage is to keep constant the total number of samples produced by the two filters.

Fig. 20.8.1 shows the analysis bank redrawn to emphasize this frequency property. In the middle graph, all downsamplers are pushed to the overall outputs, and in the bottom graph, the downsamplers have been removed altogether.

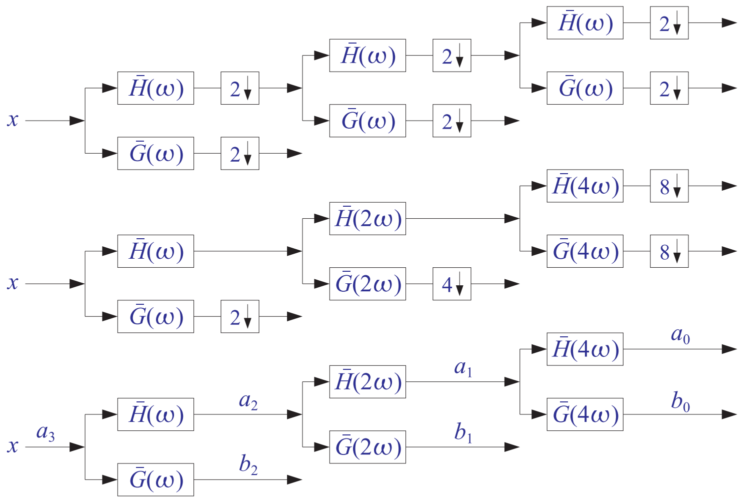


Fig. 20.8.1 Decimated and undecimated filter banks.

The bottom graph effectively implements the undecimated wavelet transform. The individual stages no longer have the orthogonality properties of the usual DWT, such as Eqs. (20.5.17). However, perfect reconstruction can still be achieved by using the property (20.3.5) for the scaling and wavelet filters:

$$\frac{1}{2}\left[\bar{H}(\omega)H(\omega)+\bar{G}(\omega)G(\omega)\right]=1 \tag{20.8.1}$$

where $\bar{H}(\omega)=H^*(\omega)$ denotes the frequency response of the time-reversed filter $\bar{h}_n$. This relationship admits a block diagram realization as shown in Fig. 20.8.2.

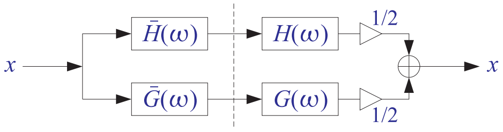


Fig. 20.8.2 Analysis and synthesis of single stage.

Because it is an identity in ω, the same relationship and block diagram will still be valid for the filter pairs $H(2\omega), G(2\omega)$ and $H(4\omega), G(4\omega)$ leading to an overall analysis and synthesis filter bank with perfect reconstruction as shown in Fig. 20.8.3.

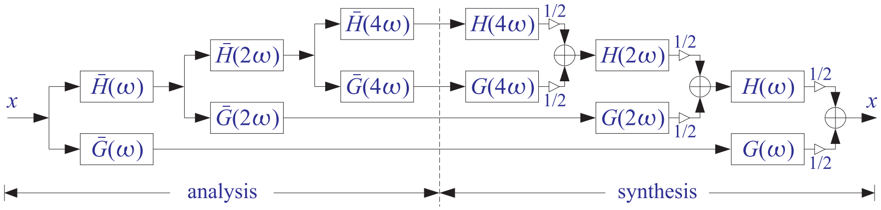


Fig. 20.8.3 Analysis and synthesis filter banks for the UWT.

Thus, it is possible with undecimated filtering operations to achieve (a) the desirable subband filter characteristics of the DWT, and (b) perfect reconstruction. To make the algorithm more concrete, first we recall that the filter with frequency response $H(2^r\omega)$ is the à trous filter defined in Eq. (20.2.20), that is,

$$h^{[r]}(k)=\sum_n h(n)\delta(k-2^r n) \quad\Leftrightarrow\quad H^{[r]}(\omega)=H(2^r\omega) \tag{20.8.2}$$

Then, denoting the successive analysis bank output signals by $a_j(n), b_j(n)$, we obtain the following analysis and synthesis algorithm written in convolutional form:

$$\begin{aligned}
&\boxed{\begin{aligned}\mathbf{a}_{j-1}&=\bar{\mathbf{h}}^{[J-j]}*\mathbf{a}_j\\ \mathbf{b}_{j-1}&=\bar{\mathbf{g}}^{[J-j]}*\mathbf{a}_j\end{aligned}} \quad J\geq j\geq J_0+1, \qquad \text{(analysis)}\\
&\boxed{\mathbf{a}_j=\frac{1}{2}\Big[\mathbf{h}^{[J-j]}*\mathbf{a}_{j-1}+\mathbf{g}^{[J-j]}*\mathbf{b}_{j-1}\Big]} \quad J_0+1\leq j\leq J, \qquad \text{(synthesis)}
\end{aligned} \tag{20.8.3}$$

where J, J_0 are the finest and coarsest desired resolution levels, and we must initialize the analysis algorithm by the overall input $a_J(n)=x(n)$. Different à trous filters are used in each stage, unlike the DWT that uses the same filters $\mathbf{h}, \mathbf{g}$. The correctness of the algorithm can be verified by writing Eqs. (20.8.3) in the frequency domain and applying the identity (20.8.1).

To make the algorithm practical we may use mod-N circular convolutions, where $N=2^J$ is the length of the input signal block $\mathbf{x}$. The à trous filters $\mathbf{h}^{[r]}, \mathbf{g}^{[r]}$ can be represented by $N\times N$ matrices H_r, G_r, which are the ordinary convolution matrices of $\mathbf{h}^{[r]}, \mathbf{g}^{[r]}$ reduced modulo-N column-wise. Similarly, the time-reversed filters $\bar{\mathbf{h}}^{[r]}, \bar{\mathbf{g}}^{[r]}$ will be represented by the transposed matrices H_r^T, G_r^T. The construction of these matrices is straightforward, for example,

```
h = h(:);   g = cmf(h);                          % h, g filters

hr = upr(h,r);        gr = upr(g,r);             % upsample by 2^r
Hr = convmtx(hr, N); Gr = convmtx(gr, N);        % ordinary convolution matrices
Hr = modwrap(Hr, N); Gr = modwrap(Gr, N);        % wrapped mod-N column-wise
```

These steps have been incorporated into the function `uwtmat`, except the function `convmat` is used in place of `convmtx` to make the matrices sparse:

```
[Hr,Gr] = uwtmat(h,N,r);        % undecimated wavelet transform matrices
```

The matrices H_r, G_r satisfy the matrix version of Eq. (20.8.1):

$$\frac{1}{2}\Big[H_r H_r^T + G_r G_r^T\Big] = I_N \tag{20.8.4}$$

The concrete matrix realization of the UWT can be stated then as follows:

$$\boxed{\begin{aligned}\mathbf{a}_{j-1} &= H_{J-j}^T\,\mathbf{a}_j\\ \mathbf{b}_{j-1} &= G_{J-j}^T\,\mathbf{a}_j\end{aligned}}\quad J \ge j \ge J_0+1, \qquad \text{(analysis)}$$

$$\boxed{\mathbf{a}_j = \frac{1}{2}\Big[H_{J-j}\,\mathbf{a}_{j-1} + G_{J-j}\,\mathbf{b}_{j-1}\Big]}\quad J_0+1 \le j \le J, \qquad \text{(synthesis)} \tag{20.8.5}$$

where all the vectors are N-dimensional, initialized at $\mathbf{a}_J = \mathbf{x}$. The algorithm is illustrated in Fig. 20.8.4. The UWT is the $N\times(J-J_0+1)$ matrix U defined column-wise by:

$$U = [\mathbf{a}_{J_0}, \mathbf{b}_{J_0}, \mathbf{b}_{J_0+1}, \dots, \mathbf{b}_{J-1}] \quad \text{(UWT)} \tag{20.8.6}$$

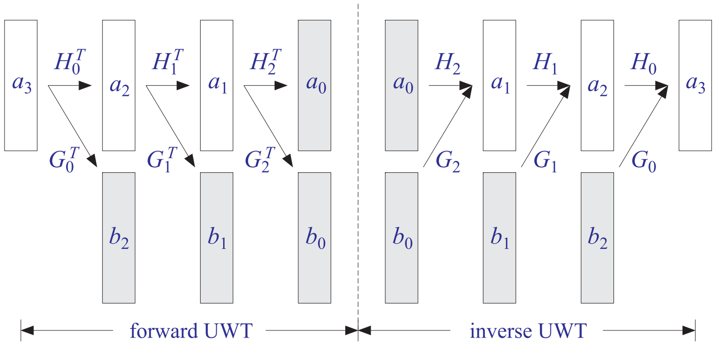


Fig. 20.8.4 Undecimated wavelet transform.

The MATLAB functions uwtm and iuwtm implement the algorithms in Eq. (20.8.5):

```
U = uwtm(x,h,J0);        % UWT in matrix form
x = iuwtm(U,h);          % inverse UWT in matrix form
```

An example is as follows:

```
h = daub(3); x = [1 2 3 4 5 6 7 8]';
J0=0; U = uwtm(x,h,J0);                  % or, set J0 = 1 and J0 = 2
xinv = iuwtm(U,h); norm(x-xinv)
```

which generates, for $J_0 = 2, 1, 0$,

$$\texttt{U = uwtm(x,h,2)} = \begin{bmatrix} 2.5702 & 0\\ 3.9844 & 0\\ 5.3986 & 0\\ 6.5310 & 2.6614\\ 8.6288 & -3.7938\\ 11.1231 & -0.1147\\ 8.8583 & 0.9653\\ 3.8173 & 0.2818 \end{bmatrix} = [\mathbf{a}_2, \mathbf{b}_2]$$

$$\texttt{U = uwtm(x,h,1)} = \begin{bmatrix} 7.9539 & -4.4090 & 0\\ 11.0848 & -1.5166 & 0\\ 12.3278 & 0.0351 & 0\\ 12.1992 & 0.4022 & 2.6614\\ 10.0461 & 2.2467 & -3.7938\\ 6.9152 & 4.8818 & -0.1147\\ 5.6722 & 2.1272 & 0.9653\\ 5.8008 & -3.7674 & 0.2818 \end{bmatrix} = [\mathbf{a}_1, \mathbf{b}_1, \mathbf{b}_2] \tag{20.8.7}$$

$$\texttt{U = uwtm(x,h,0)} = \begin{bmatrix} 12.7279 & -1.4794 & -4.4090 & 0\\ 12.7279 & 2.9484 & -1.5166 & 0\\ 12.7279 & 4.7063 & 0.0351 & 0\\ 12.7279 & 4.5243 & 0.4022 & 2.6614\\ 12.7279 & 1.4794 & 2.2467 & -3.7938\\ 12.7279 & -2.9484 & 4.8818 & -0.1147\\ 12.7279 & -4.7063 & 2.1272 & 0.9653\\ 12.7279 & -4.5243 & -3.7674 & 0.2818 \end{bmatrix} = [\mathbf{a}_0, \mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2]$$

The functions uwtm and iuwtm are somewhat slow because they generate the required matrices H_r, G_r on the fly at each stage. Of course, it would be possible to precompute the matrices and save them in a cell or a three-dimensional array as was done in the function fwtm. The functions uwt and iuwt are much faster versions that produce the same results and are implemented using circular convolutions:

```
U = uwt(x,h,J0);        % UWT in convolutional form
x = iuwt(U,h);          % inverse UWT
```

The following MATLAB code shows a possible implementation of the analysis part:

```
M=length(h)-1;
g=cmf(h); hR=flip(h); gR=flip(g);       % construct reversed filters
a = x;                                   % x is N = 2^J dimensional column vector
for r=0:J-J0-1,                          % à trous interpolation factor is 2^r, level j = J - r
   a = advance(a, 2^r*M);                % establishes equivalence with matrix form
   hRr = upr(hR,r);                      % reversed filters upsampled by 2^r
   gRr = upr(gR,r);
   b = circonv(gRr,a,N);                 % modulo-N circular convolution
   a = circonv(hRr,a,N);
   U = [b,U];                            % accumulate the columns of U
end
U = [a,U];
```

The time-advancing operation is necessary to compensate for the use of the reversed filters rather than the time-reversed ones. Although this algorithm works, it is wasteful because the à trous filters $\mathbf{h}^{[r]}$ have length $2^r(M+1)$ consisting mostly of zeros and only $(M+1)$ nonzero coefficients, where M is the filter order of $\mathbf{h}$. The computational cost of the indicated circular convolution operations is of the order of $2^r(M+1)N$. It is possible to restructure these operations so that only the nonzero filter coefficients are used, thereby reducing the computational cost to $(M+1)N$. Ordinary and mod-N circular convolution by the à trous filter (20.8.2) can be written as follows:

$$\begin{aligned} y(n) &= \sum_k h^{[r]}(k)x(n-k) = \sum_m h_m x(n-2^r m) \\ \tilde{y}(n)_{\text{mod-}N} &= \sum_p y(n+pN) = \sum_{p,m} h_m x(n+pN-2^r m) \end{aligned}$$

We assume that $N = 2^J$ and that the à trous factor is such that $r \le J$, so that we may write $N = 2^r L$, where $L = 2^{J-r}$. Thus, a length-N block can be divided into 2^r sub-blocks of length L. We show below that the mod-N circular convolution can be replaced by 2^r mod-L circular convolutions. The total computational cost reduces then to $2^r L(M+1) = N(M+1)$. Setting $n = 2^r i + k$, with $0 \le k \le 2^r - 1$, we may define the k-th sub-block input and output signals:

$$x_k(i) = x(2^r i + k)\,, \quad y_k(i) = y(2^r i + k)\,, \quad 0 \le k \le 2^r - 1$$

It follows then that $y_k(i)$ is the convolution of $x_k(i)$ with the original filter h_m, and that the mod-N circular convolution output can be obtained by mod-L reduction:

$$\begin{aligned} y_k(i) = y(2^r i + k) &= \sum_m h_m x(2^r i + k - 2^r m) = \sum_m h_m x_k(i-m) \\ \tilde{y}(2^r i + k)_{\text{mod-}N} &= \sum_{p,m} h_m x(2^r i + k + 2^r pL - 2^r m) = \sum_{p,m} h_m x_k(i + pL - m) \\ &= \sum_p y_k(i + pL) = \tilde{y}_k(i)_{\text{mod-}L} \end{aligned}$$

These operations have been incorporated into the MATLAB function `convat`,

```
y = convat(h,x,r);          % convolution à trous
```

which is equivalent to the mod-N operation, where N is the length of `x`:

```
y = circonv(upr(h,r),x,N);
```

The essential part of the function `uwt` is then,

```
M=length(h)-1;
g=cmf(h); hR=flip(h); gR=flip(g);        % construct reversed filters
a = x;                                   % x is N = 2^J dimensional column vector
for r=0:J-J0-1,                          % à trous interpolation factor is 2^r, level j = J − r
   a = advance(a, 2^r*M);                % establishes equivalence with matrix form
   b = convat(gR, a, r);                 % convolution à trous
   a = convat(hR, a, r);
   U = [b,U];                            % accumulate the columns of U
end
U = [a,U];
```

Because all stages of the analysis and synthesis filter banks in Fig. 20.8.3 operate at the same sampling rate, the UWT satisfies a time-invariance property, in the sense that a time-delay in the input will cause the same delay in the outputs from all stages. Hence, the alternative name “stationary” or “translation-invariant” wavelet transform. Such time-invariance property is not shared by the ordinary DWT.

As an example, the DWTs and UWTs of a signal and its circularly-delayed version by three time units are as follows, using the D_3 scaling filter and coarsest level $J_0 = 0$:

$$\mathbf{x} = \begin{bmatrix} 1\\2\\3\\4\\5\\6\\7\\8 \end{bmatrix} \;\Rightarrow\; \mathbf{w} = \begin{bmatrix} 12.7279\\-1.4794\\-4.4090\\2.2467\\0\\0\\-3.7938\\0.9653 \end{bmatrix} \;\Rightarrow\; U = \begin{bmatrix} 12.7279 & -1.4794 & -4.4090 & 0\\ 12.7279 & 2.9484 & -1.5166 & 0\\ 12.7279 & 4.7063 & 0.0351 & 0\\ 12.7279 & 4.5243 & 0.4022 & 2.6614\\ 12.7279 & 1.4794 & 2.2467 & -3.7938\\ 12.7279 & -2.9484 & 4.8818 & -0.1147\\ 12.7279 & -4.7063 & 2.1272 & 0.9653\\ 12.7279 & -4.5243 & -3.7674 & 0.2818 \end{bmatrix}$$

$$\mathbf{x} = \begin{bmatrix} 6\\7\\8\\1\\2\\3\\4\\5 \end{bmatrix} \;\Rightarrow\; \mathbf{w} = \begin{bmatrix} 12.7279\\-2.9484\\4.8818\\-1.5166\\-0.1147\\0.2818\\0\\2.6614 \end{bmatrix} \;\Rightarrow\; U = \begin{bmatrix} 12.7279 & -2.9484 & 4.8818 & -0.1147\\ 12.7279 & -4.7063 & 2.1272 & 0.9653\\ 12.7279 & -4.5243 & -3.7674 & 0.2818\\ 12.7279 & -1.4794 & -4.4090 & 0\\ 12.7279 & 2.9484 & -1.5166 & 0\\ 12.7279 & 4.7063 & 0.0351 & 0\\ 12.7279 & 4.5243 & 0.4022 & 2.6614\\ 12.7279 & 1.4794 & 2.2467 & -3.7938 \end{bmatrix}$$

We note that every column of U gets delayed circularly by three time units.

Multiresolution Decomposition with the UWT

The synthesis filter bank or the synthesis algorithm for the UWT can be viewed as a system with $J-J_0+1$ inputs, i.e., the columns of $U = [\mathbf{a}_{J_0}, \mathbf{b}_{J_0}, \mathbf{b}_{J_0+1}, \dots, \mathbf{b}_{J-1}]$, and one output, the signal $\mathbf{x}$. The UWT multiresolution decomposition resolves $\mathbf{x}$ into components arising from the individual inputs when the other inputs are zero:

$$\begin{aligned} U = [\mathbf{a}_{J_0}, \mathbf{b}_{J_0}, \mathbf{b}_{J_0+1}, \dots, \mathbf{b}_{J-1}] &\xrightarrow{\text{IUWT}} \mathbf{x} \\ = [\mathbf{a}_{J_0},\; 0,\; 0,\; \dots,\; 0\;] &\xrightarrow{\text{IUWT}} \mathbf{x}_{J_0} \\ +\,[\;0,\; \mathbf{b}_{J_0},\; 0,\; \dots,\; 0\;] &\xrightarrow{\text{IUWT}} \bar{\mathbf{x}}_{J_0} \\ +\,[\;0,\; 0,\; \mathbf{b}_{J_0+1},\; \dots,\; 0\;] &\xrightarrow{\text{IUWT}} \bar{\mathbf{x}}_{J_0+1} \\ \cdots\qquad\qquad & \qquad\;\; \cdots \\ +\,[\;0,\; 0,\; 0,\; \dots,\; \mathbf{b}_{J-1}] &\xrightarrow{\text{IUWT}} \bar{\mathbf{x}}_{J-1} \end{aligned}$$

so that we have the sum,

$$\boxed{\mathbf{x} = \mathbf{x}_{J_0} + \bar{\mathbf{x}}_{J_0} + \bar{\mathbf{x}}_{J_0+1} + \cdots + \bar{\mathbf{x}}_{J-1}} \tag{20.8.8}$$

This is similar to the DWT decomposition (20.6.2), with each term reflecting a different resolution level, except that the terms are not mutually orthogonal. The MATLAB function `uwtdec` implements this decomposition:

```
X = uwtdec(x,h,J0);        % UWT multiresolution decomposition
```

where X consists of the columns, $X = [\mathbf{x}_{J_0}, \tilde{\mathbf{x}}_{J_0}, \tilde{\mathbf{x}}_{J_0+1}, \dots, \tilde{\mathbf{x}}_{J-1}]$.

Fig. 20.8.5 shows an application to the monthly housing starts from January 1988 to April 2009 (i.e., 256 months), using the symmlet S_8 scaling filter and going down to resolution level $J_0 = 5$. This a subset of the dataset that we used repeatedly in Ch.9 of [45].

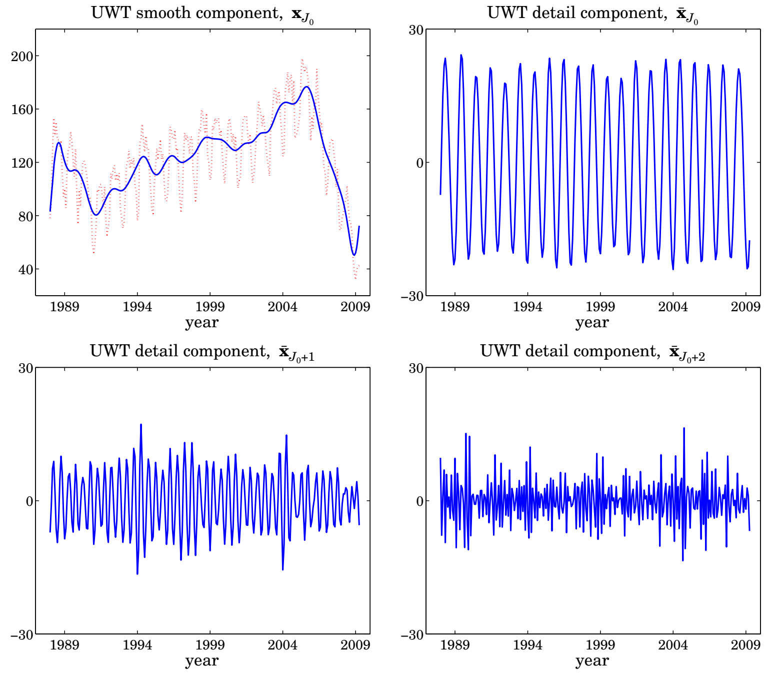


Fig. 20.8.5 UWT decomposition of monthly housing data, using S_8 with $J = 8$ and $J_0 = 5$.

The upper left graph shows the smooth component arising from the UWT coefficients $\mathbf{a}_{J_0}$. The remaining graphs arise from the detail coefficients. $\mathbf{b}_j$, $J_0 \le j \le J-1$. The sum of the four components is equal to the original data (dotted line in the upper-left graph.) The following MATLAB code generates the four graphs:

```
Y = loadfile('newhouse.dat');             % data file in OSP toolbox

y = Y(349:end,1);                         % selects Jan.88 - Apr.09 = 256 months
t = taxis(y,12,1988)';                    % adjust time axis
```

```
h=daub(8,2); J0=5;                  % symmlet S_8, note N = 256 = 2^8  ⇒ J = 8
X = uwtdec(y,h,J0);                 % UWT decomposition, try also X = dwtdec(y,h,J0)

figure; plot(t,y,':', t,X(:,1), '-');               % upper left graph
figure; plot(t,X(:,2));                             % upper right
figure; plot(t,X(:,3)); figure; plot(t,X(:,4));     % lower graphs
```

See Fig. 20.9.1 for an alternative way of plotting the UWT (or DWT) decomposition and the UWT (or DWT) wavelet coefficients using the function `plotdec`.

Wavelet Denoising with the UWT

The application of the UWT to denoising applications follows the same approach as the DWT. The detail columns $\mathbf{b}_j$ of U get thresholded by a possibly level-dependent threshold and the inverse UWT is constructed. The procedure is depicted below:

$$\mathbf{x} \xrightarrow{\text{UWT}} U = [\mathbf{a}_{J_0}, \mathbf{b}_{J_0}, \dots, \mathbf{b}_{J-1}] \xrightarrow{\text{thresh}} U^{\text{thr}} = [\mathbf{a}_{J_0}, \mathbf{b}^{\text{thr}}_{J_0}, \dots, \mathbf{b}^{\text{thr}}_{J-1}] \xrightarrow{\text{IUWT}} \mathbf{x}_{\text{thr}}$$

The MATLAB function `wduwt` implements this denoising procedure using the universal threshold (20.7.3) and soft or hard thresholding:

```
y = wduwt(x,h,J0,type);        % wavelet denoising with UWT
```

Fig. 20.8.6 shows the same denoising example as that in Fig. 20.7.1, but denoised using the UWT. The following MATLAB code generates the top-row graphs:

```
F = inline('1./(1 + abs(x)).^4');                 % bumps function

N = 2048; t = (0:N-1)'/N; x = zeros(size(t));     % normalize time to 0 ≤ t ≤ 1

t0 = [10 13 15 23 25 40 44 65 76 78 81]/100;      % signal parameters
a = [40,50,30,40,50,42,21,43,31,51,42] * 1.0523;
w = [5,5,6,10,10,30,10,10,5,8,5]/1000;

for i=1:length(a),                                % construct noise-free signal
   x = x + a(i) * F((t-t0(i))/w(i));
end

seed=2009; randn('state',seed); v = randn(size(t));   % generate noise

y = x + v;                                        % noisy signal with SNR, σ_x/σ_v = 7


h = daub(8,2); J0=5; type=1;                      % use Symmlet-8 and soft thresholding
xd = wduwt(y,h,J0,type);                          % wavelet denoising using the UWT

figure; plot(t,y,'-'); figure; plot(t,x,'-'); figure; plot(t,xd,'r-');   % top row
```

Comparing Figs. 20.7.1 and 20.8.6, we note that the UWT outperforms the DWT, an observation that has been made repeatedly in the denoising literature.

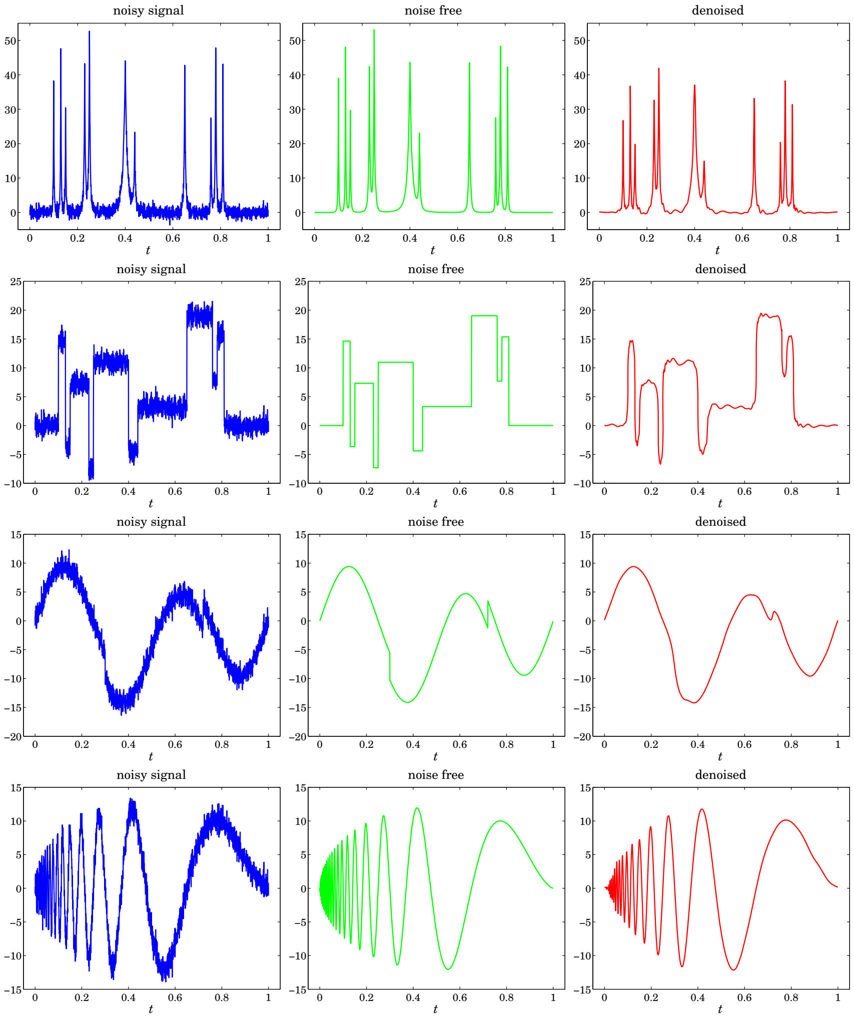


Fig. 20.8.6 Wavelet denoising with the UWT.

20.9 *MATLAB Functions*

We summarize the MATLAB functions that we discussed in this chapter, and give few more details about `plotdec` that can be used to plot wavelet decompositions and wavelet coefficients.

```
Wavelet functions
------------------------------------------------
```

```
advance   - circular time-advance (left-shift) of a vector
casc      - cascade algorithm for φ and ψ wavelet functions
circonv   - circular convolution
cmf       - conjugate mirror of a filter
convat    - convolution à trous
convmat   - sparse convolution matrix
daub      - Daubechies scaling filters (daublets, symmlets, coiflets)
dn2       - downsample by a factor of 2
dwtcell   - create cell array of sparse discrete wavelet transform matrices
dwtdec    - DWT decomposition into orthogonal multiresolution components
dwtmat    - discrete wavelet transform matrices - sparse
dwtmat2   - discrete wavelet transform matrices - nonsparse
dwtwrap   - wrap a DWT matrix into a lower DWT matrix
flip      - flip a column, a row, or both
fwt       - fast wavelet transform using convolution and downsampling
fwtm      - fast wavelet transform in matrix form
fwtmat    - overall DWT orthogonal matrix
ifwt      - inverse fast wavelet transform - convolutional form
ifwtm     - inverse fast wavelet transform - matrix form
iuwt      - inverse undecimated wavelet transform - convolutional form
iuwtm     - inverse undecimated wavelet transform - matrix form
modwrap   - wrap matrix column-wise mod-N
phinit    - eigenvector initialization of scaling function φ
plotdec   - plot DWT/UWT decomposition or DWT/UWT coefficients
up2       - upsample a vector by factor of 2
upr       - upsample a vector by factor of 2^r
uwt       - undecimated wavelet transform - convolutional form
uwtdec    - UWT multiresolution decomposition
uwtm      - undecimated wavelet transform - matrix form
uwtmat    - undecimated wavelet transform matrices - sparse
uwtmat2   - undecimated wavelet transform matrices - nonsparse
w2V       - wavelet vector to wavelet matrix
wcoeff    - extract wavelet coefficients from DWT at given level
wdenoise  - Donoho & Johnstone's VisuShrink denoising procedure
wduwt     - wavelet denoising with undecimated wavelet transform
wthr      - soft/hard level-dependent wavelet thresholding
------------------------------------------------
```

The function `plotdec` allows a compact display of a DWT or UWT decomposition, or the display of the DWT/UWT wavelet smooth and detail coefficients:

```
plotdec(X,type,lin,Jmax);      % plot DWT/UWT decomposition or DWT/UWT coefficients
```

with inputs:

`X` = $N\times(J-J_0+1)$ matrix of DWT/UWT decomposition signals or DWT/UWT coefficients, $N = 2^J$
`type` = `'xs','xd','ws','wd'` (x=decomposition, w=wavelet coeffs, s=include smooth, d=details only)
`Jmax` = highest resolution level to plot, $J_{\max} \le J-1$, minimum is determined from $J_0 = J+1-\text{size}(X,2)$
`lin` = `'l','s'` for line or stem plot

See the help for this function for several usage examples. Fig. 20.9.1 shows an alternative plot of the UWT decomposition of Fig. 20.8.5, showing only the detail components, including a plot of the UWT wavelet coefficients. The MATLAB code used to generate the four graphs was as follows:

```
h=daub(8,2); J0=5;
Y = loadfile('newhouse.dat');          % load housing data - file in OSP toolbox
y = Y(349:end,1);                      % selects Jan.88 - Apr.09 = 256 months

Xdwt = dwtdec(y,h,J0);                 % DWT decomposition
[w,V] = fwt(y,h,J0);                   % DWT coefficients
Xuwt = uwtdec(y,h,J0);                 % UWT decomposition
U = uwt(y,h,J0);                       % UWT coefficients

figure; plotdec(Xdwt,'xd');  figure; plotdec(V,'wd');      % upper graphs
figure; plotdec(Xuwt,'xd');  figure; plotdec(U,'wd');      % lower graphs
```

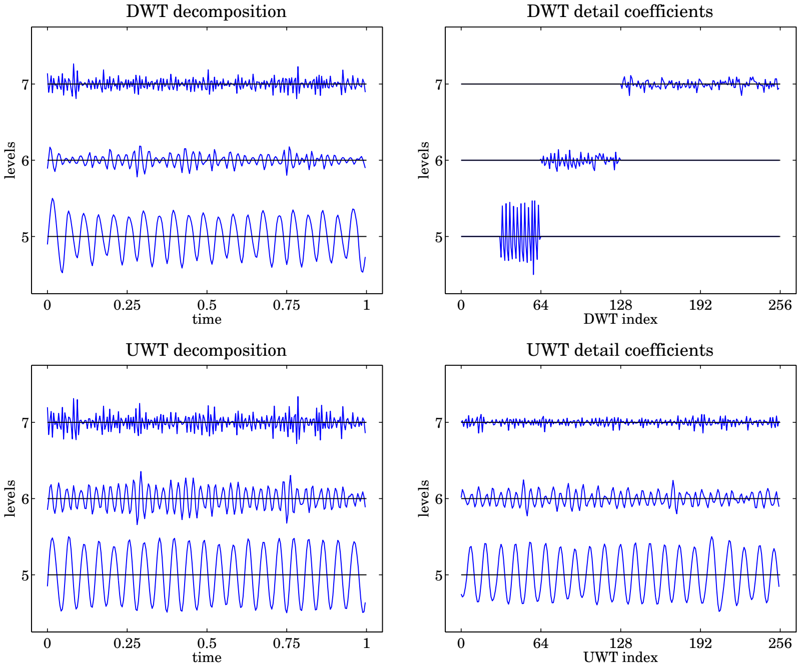


Fig. 20.9.1 UWT/DWT decompositions and wavelet coefficients of housing data.

20.10 Problems

20.1 An alternative way of determining the Daubechies D_2 scaling filter is to assume that its transfer function has the form (with $K = 2$ zeros at Nyquist):

$$H(z)= h_0(1+z^{-1})^2(1-z_1z^{-1})$$

Show that z_1 must the be a solution of the quadratic equation $z^2 - 4z + 1 = 0$. Pick that solution that has $|z_1| < 1$ and verify that the resulting filter $H(z)$ meets all the design constraints (20.3.10).

20.2 To determining the Daubechies D_3 scaling filter assume that its its transfer function has the following form with $K = 3$ zeros at Nyquist:

$$H(z)= h_0(1+z^{-1})^3\big(1-(a+jb)z^{-1}\big)\big(1-(a-jb)z^{-1}\big)$$

including a complex zero $z_1 = a+jb$, constrained such that $|z_1| < 1$. Show that the design constraints (20.3.12) imply that b is given by

$$b=\sqrt{\frac{a+a^3-3a^2}{3-a}}$$

and that a is obtained as the following solution of the quartic equation:

$$12a^4-72a^3+152a^2-132a+27=0 \quad\Rightarrow\quad a=\frac{3}{2}-\frac{1}{6}\sqrt{15+12\sqrt{10}}$$

Verify that the zero $z_1 = a + jb$ coincides with that in Eq. (20.3.18). By expanding $H(z)$, express the filter coefficients h_n in terms of a, b, and normalize them to add up to $\sqrt{2}$.

20.3 Prove the downsampling replication property (20.4.11) by working backwards, that is, start from the Fourier transform expression and show that

$$\frac{1}{L}\sum_{m=0}^{L-1}X(f-mf_s^{\text{down}})=\sum_k s(k)x(k)e^{-2\pi jfk/f_s}=\sum_n x(nL)e^{-2\pi jfnL/f_s}=Y_{\text{down}}(f)$$

where $s(k)$ is the periodic "sampling function" with the following representations:

$$s(k)=\frac{1}{L}\sum_{m=0}^{L-1}e^{-2\pi jkm/L}=\frac{1}{L}\frac{1-e^{-2\pi jk}}{1-e^{-2\pi jk/L}}=\sum_n\delta(k-nL)$$

Moreover, show that the above representations are nothing but the inverse L-point DFT of the DFT of one period of the periodic pulse train:

$$s(k)=[\dots,1,\underbrace{0,0,\dots,0}_{L-1\text{ zeros}},1,\underbrace{0,0,\dots,0}_{L-1\text{ zeros}},1,\underbrace{0,0,\dots,0}_{L-1\text{ zeros}},\dots]=\sum_n\delta(k-nL)$$

20.4 Show that the solution to the optimization problem (20.7.7) is the soft-thresholding rule of Eq. (20.7.8).

20.5 Study the "Tikhonov regularizer" wavelet thresholding function:

$$d_{\text{thr}}=f(d,\lambda,a)=d\,\frac{|d|^a}{|d|^a+\lambda^a}\,,\qquad a>0,\ \lambda>0$$

21

Discretization Methods

In this chapter, we consider the numerical solution of the differential equations describing linear time-invariant continuous-time systems, by converting their analog transfer functions into digital ones and implementing them in MATLAB as difference equations, and comparing their output with that of the built-in function **lsim**, as well as showing how to take into account the specified initial conditions of the continuous-time system. The following discretization schemes are considered:

(i) forward Euler
(ii) backward Euler
(iii) trapezoidal, also known as bilinear or Tustin transformation
(iv) zero-order hold
(v) first-order hold

We begin with a short review of continuous-time systems, then, we summarize the discretization procedures for first-order and second-order systems. The built-in functions **c2dm** and **c2d** can also be used to convert a continuous-time system to an equivalent discrete-time one, however, they do not include the forward and backward Euler methods. But they do include the trapezoidal (Tustin) and zero-order hold (the default) and first-order hold methods.

The function **lsim** is used for simulating the behavior of continuous-time systems, but it cannot be used to actually replace the continuous-time system by an equivalent discrete-time one that can then be implemented digitally, for example, on a digital signal processor. Moreover, **lsim**, and MATLAB in general, process signals on a block basis and are not so well-suited for real-time processing. However, once the equivalent discrete-time transfer function is available, it can easily be implemented in real-time.

21.1 Continuous-Time Systems

As in the discrete-time case, linear time-invariant (LTI) continuous-time systems [597] are characterized by an *impulse response* function, $h(t)$, such that the input/output equation is given by continuous-time convolution,

$$y(t)=\int_{-\infty}^{\infty}h(t')x(t-t')dt'\,,\qquad x(t)\longrightarrow\boxed{H}\longrightarrow y(t) \tag{21.1.1}$$

We will assume that both the impulse response $h(t)$ and the input signal $x(t)$ are causal (i.e., right-sided) functions of time, so that the operation (21.1.1) is restricted to non-negative times, where we also implicitly assumed zero initial conditions (ICs) for the output (non-zero ICs are discussed below),

$$y(t)=\int_{0}^{\infty}h(t-t')x(t')dt'\,,\qquad t\ge 0 \tag{21.1.2}$$

Essentially all practical continuous-time LTI systems are a subset of the above, and are described by *linear constant-coefficient differential equations* (LCCDEs) derived from a physical description of the system. This implies that the impulse response $h(t)$ will also satisfy an LCCDE, but with an impulsive input, $x(t)=\delta(t)$.

As an example, consider a second-order LTI system described by a second-order differential equation of the following form, and its corresponding transfer function $H(s)$,

$$x(t)\longrightarrow\boxed{H}\longrightarrow y(t)\qquad \begin{array}{l}\boxed{\ddot{y}(t)+a_1\dot{y}(t)+a_2y(t)=b_0\ddot{x}(t)+b_1\dot{x}(t)+b_2x(t)}\\[1ex] \boxed{H(s)=\dfrac{b_0s^2+b_1s+b_2}{s^2+a_1s+a_2}}\end{array} \tag{21.1.3}$$

where we took Laplace transforms of both sides of the differential equation using the formal mapping, $s\leftrightarrow d/dt$, assuming zero ICs, and defined $H(s)$ as the ratio of the Laplace transform of the output to the Laplace transform of the input,

$$(s^2+a_1s+a_2)Y(s)=(b_0s^2+b_1s+b_2)X(s)\quad\Rightarrow\quad H(s)=\frac{Y(s)}{X(s)}=\frac{b_0s^2+b_1s+b_2}{s^2+a_1s+a_2}$$

and used the simplified dot-notation for the time derivatives,

$$\dot{f}(t)=\frac{df(t)}{dt}\,,\quad \ddot{f}(t)=\frac{d^2f(t)}{dt^2}\,,\quad \text{etc.}$$

Similarly, a first-order LTI system is characterized by the following first-order differential equation and transfer function,

$$\boxed{\begin{array}{l}\dot{y}(t)+a_1y(t)=b_0\dot{x}(t)+b_1x(t)\\[1ex] H(s)=\dfrac{Y(s)}{X(s)}=\dfrac{b_0s+b_1}{s+a_1}\end{array}} \tag{21.1.4}$$

The *stability and causality* of the continuous-time system requires that the poles of its transfer function, $H(s)$, lie in the left-hand s-plane. We will assume this is the cases in our examples in this chapter.

The differential equation (21.1.3) is to be solved for $y(t)$ with a given input $x(t)$ and given initial conditions, $y(0^-)$, $\dot{y}(0^-)$, specified at $t=0^-$. Alternatively, as is done in the classical method of solving differential equations, one can specify the initial conditions, $y(0^+)$, $\dot{y}(0^+)$, at time $t=0^+$.

For the first-order case of Eq. (21.1.4), the initial condition at $t=0^-$ is the single number $y(0^-)$, or equivalently at $t=0^+$, the number, $y(0^+)$.

For simple types of inputs, such as unit-steps, ramps, finite pulses, exponentials, or sinusoids, the solutions of the differential equations can be obtained analytically by a variety of methods, for example,

(a) using Laplace transforms with the given input $x(t)$ and given ICs at $t=0^-$

(b) using Laplace transforms assuming zero input but with the given ICs at $t=0^-$, and adding to that the convolutional solution of Eq. (21.1.2)

(c) using the classical method with initial conditions at $t=0^+$, implemented for example using the **dsolve** function of MATLAB's symbolic toolbox, or, alternatively, by adding a homogeneous part to an appropriate forced-response part and fixing the homogeneous terms using the $t=0^+$ conditions

For more complicated and arbitrary inputs, the LCCDEs can be solved numerically by,

(a) using MATLAB's built-in **lsim** function

(b) converting the continuous-time *differential equation* into a discrete-time *difference equation* using some sort of discretization scheme, such as zero-order-hold, forward or backward Euler, or bilinear/trapezoidal transformation, and then iterating it numerically (**lsim** uses either a zero- or a first-order hold by default).

21.2 Mapping of Initial Conditions

The mapping between the two sets of conditions at $t=0^-$ to those at $t=0^+$ is as follows, up to order 3,

$$\boxed{\begin{aligned}
y(0^+) &= y(0^-)+b_0x(0^+)\\
\dot{y}(0^+) &= \dot{y}(0^-)+b_0\dot{x}(0^+)+(b_1-a_1b_0)x(0^+)\\
\ddot{y}(0^+) &= \ddot{y}(0^-)+b_0\ddot{x}(0^+)+(b_1-a_1b_0)\dot{x}(0^+)+\big(b_2-a_2b_0-a_1(b_1-a_1b_0)\big)x(0^+)
\end{aligned}} \tag{21.2.1}$$

These assume that the input signal $x(t)$ is *causal* and that it *does not have* any impulsive delta-function terms, like $\delta(t)$, but it may be discontinuous at $t=0$. For first-order systems only the first equation in (21.2.1) is needed, for second-order systems, only the first two are needed, and for third-order, all three are needed. For a proof, see Appendix Sec. 21.18.

The general solution of the differential equations for a given causal input $x(t)$ and subject to the given initial conditions can be decomposed as a sum of two types of terms, depending on which set of ICs one uses,

$$\boxed{y(t) = \underbrace{y_{zi}(t)}_{\substack{\text{zero-input}\\ \text{IC}(0^-)\neq 0}} + \underbrace{y_{zs}(t)}_{\substack{\text{zero-state}\\ \text{IC}(0^-)=0\\ h(t)*x(t)}} = \underbrace{y_{\text{homog}}(t)}_{\substack{\text{homogeneous}\\ \text{IC}(0^+)\neq 0}} + \underbrace{y_f(t)}_{\substack{\text{forced response}\\ \text{easy to guess}}}} \tag{21.2.2}$$

The *zero-input/zero-state decomposition* uses the initial conditions at $t=0^-$, and each term separately, or both simultaneously, can be determined by Laplace transform methods. The relevant MATLAB functions are, **laplace**, **solve**, **partfrac**, **ilaplace**.

The zero-state term, $y_{zs}(t)$, can also be obtained by convolving the system's impulse response $h(t)$ with the input signal $x(t)$, that is, by the operation, $y_{zs}(t)=h(t)*x(t)$, but that is generally less convenient than the Laplace method.

The *homogeneous/forced-response decomposition* uses the ICs at $t=0^+$. It is equivalent to the classical method of solution and can be implemented in MATLAB with the function **dsolve**, which requires the conditions at $t=0^+$. Both decompositions are special cases of the more general, but not unique, decomposition into a homogeneous part and a particular solution,

$$y(t)=\underbrace{y_{\text{homog}}(t)}_{\text{homogeneous}}+\underbrace{y_{\text{part}}(t)}_{\text{particular}} \tag{21.2.3}$$

The homogeneous part in Eq. (21.2.2) or (21.2.3) is expressible as a linear combination of the so-called *characteristic modes* of the system (corresponding to the system poles on the s-plane), with coefficients fixed by the initial conditions at $t=0^+$. This approach is illustrated by examples below.

Since convolution in the time-domain becomes multiplication in the s-domain, the Laplace transform of the zero-state component will be,

$$y_{zs}(t)=h(t)*x(t) \quad\Leftrightarrow\quad Y_{zs}(s)=H(s)X(s) \tag{21.2.4}$$

Thus, the Laplace transform of the total solution will be as follows, with the part $Y_{zi}(s)$ incorporating the initial conditions at $t=0^-$,

$$y(t)=y_{zi}(t)+y_{zs}(t) \quad\Leftrightarrow\quad Y(s)=Y_{zi}(s)+Y_{zs}(s)=Y_{zi}(s)+H(s)X(s) \tag{21.2.5}$$

Eq. (21.2.1) can also be applied separately to the zero-input and zero-state parts. By definition, the zero-input component, $y_{zi}(t)$, is the solution of Eq. (21.1.3) when the input is zero, $x(t)=0$. Also by definition, the zero-state component, $y_{zs}(t)$, is the solution of (21.1.3) with the given input $x(t)$, but subject to *zero* initial conditions at $t=0^-$. It follows that the initial conditions of $y_{zi}(t)$ are the same as those of the total solution $y(t)$, indeed,

$$y(0^-)=y_{zi}(0^-)+y_{zs}(0^-)=y_{zi}(0^-)+0=y_{zi}(0^-)$$

and similarly for the other conditions. Thus, the conditions (21.2.1) for $y_{zi}(t)$ will be, after setting $x(t)=0$,

$$\boxed{\begin{aligned}
y_{zi}(0^+) &= y_{zi}(0^-)=y(0^-)\\
\dot{y}_{zi}(0^+) &= \dot{y}_{zi}(0^-)=\dot{y}(0^-)\\
\ddot{y}_{zi}(0^+) &= \ddot{y}_{zi}(0^-)=\ddot{y}(0^-)
\end{aligned}} \tag{21.2.6}$$

We note that here there is no distinction between $t=0^-$ and $t=0^+$, and moreover, the conditions for $y_{zi}(t)$ match those of the total solution $y(t)$. The $y_{zi}(t)$ component can be determined either by Laplace or by the **dsolve** method both using the same initial conditions at $t = 0^-$. For the zero-state component, Eq. (21.2.1) becomes,

$$\boxed{\begin{aligned} y_{zs}(0^-) &= 0\,, \quad y_{zs}(0^+) = b_0 x(0^+)\\ \dot{y}_{zs}(0^-) &= 0\,, \quad \dot{y}_{zs}(0^+) = b_0\dot{x}(0^+) + (b_1 - a_1 b_0)x(0^+)\\ \ddot{y}_{zs}(0^-) &= 0\,, \quad \ddot{y}_{zs}(0^+) = b_0\ddot{x}(0^+) + (b_1 - a_1 b_0)\dot{x}(0^+) + \big(b_2 - a_2 b_0 - a_1(b_1 - a_1 b_0)\big)x(0^+)\end{aligned}} \tag{21.2.7}$$

It follows that the zero-state component $y_{zs}(t)$ can be determined by three methods:

(i) convolution, $y_{zs}(t) = h(t) * x(t)$

(ii) Laplace method applied to the zero initial conditions at $t = 0^-$ of Eq. (21.2.7)

(iii) **dsolve** method applied to the $t = 0^+$ conditions of Eq. (21.2.7).

21.3 Forced Response

In the decomposition of Eq. (21.2.2), the form of the forced response term, $y_f(t)$, can be guessed easily in simple cases of the driving input.[†] Consider a 1st order, or 2nd order, filter with real-valued coefficients and with poles p_1, or p_1, p_2 with $p_1 \neq p_2$, that lie in the left-hand s-plane,

$$H(s) = \frac{b_0 s + b_1}{s + a_1} \equiv \frac{b_0 s + b_1}{s - p_1}\,, \quad H(s) = \frac{b_0 s^2 + b_1 s + b_2}{s^2 + a_1 s + a_2} \equiv \frac{b_0 s^2 + b_1 s + b_2}{(s - p_1)(s - p_2)}$$

Then, the forced response will have the following forms for exponential, sinusoidal, step, ramp, or quadratic input signals,

$x(t)$	$y_f(t)$	input type
e^{pt}	$B\,e^{pt}$	exponential, $p \neq p_i\,,\ B = H(p)$
$e^{p_i t}$	$B_i\,t\,e^{p_i t}$	exponential, $p = p_i,\ B_i = \lim\limits_{s=p_i}\big[(s - p_i)H(s)\big]$
$e^{j\Omega_0 t}$	$H(\Omega_0)\,e^{j\Omega_0 t}$	sinusoidal, $H(\Omega_0) = H(s)\big\vert_{s=j\Omega_0}$
$\cos(\Omega_0 t)$	$\mathrm{Re}\big[H(\Omega_0)\,e^{j\Omega_0 t}\big]$	sinusoidal
$\sin(\Omega_0 t)$	$\mathrm{Im}\big[H(\Omega_0)\,e^{j\Omega_0 t}\big]$	sinusoidal
A_0	B_0	constant, $B_0 = A_0 H(0)$
$A_0 + A_1 t$	$B_0 + B_1 t$	linear
$A_0 + A_1 t + A_2 t^2$	$B_0 + B_1 t + B_2 t^2$	quadratic

In the last three polynomial cases, one inserts the polynomial form of $y_f(t)$ into the differential equation and matches like powers of t, for example, for the 2nd order system and 2nd order polynomial input, we have,

$$\begin{aligned} y_f(t) &= B_0 + B_1 t + B_2 t^2\\ x(t) &= A_0 + A_1 t + A_2 t^2 \end{aligned} \quad \Rightarrow$$

[†] B. P. Lathi, *Linear Systems & Signals*, 2nd ed., Oxford University Press, 2005.

$$\begin{aligned} \ddot{y}_f + a_1\dot{y}_f + a_2 y_f &= 2B_2 + a_1(B_1 + 2B_2 t) + a_2(B_0 + B_1 t + B_2 t^2)\\ b_0\ddot{x} + b_1\dot{x} + b_2 x &= b_0 2A_2 + b_1(A_1 + 2A_2 t) + b_2(A_0 + A_1 t + A_2 t^2) \end{aligned}$$

and equating the two right-hand sides and matching like powers of t gives three equations in the three unknowns B_0, B_1, B_2 to be solved in terms of A_0, A_1, A_2.

21.4 Solution Procedures

For simplicity, we consider a 1st order, or a 2nd order filter with real-valued coefficients, with poles at p_1, or at p_1, p_2 with $p_1 \neq p_2$, that lie in the left-hand s-plane,

$$\begin{aligned} &\text{(1st order):} \qquad H(s) = \frac{b_0 s + b_1}{s + a_1} \equiv \frac{b_0 s + b_1}{s - p_1}\\ &\text{(2nd order):} \qquad H(s) = \frac{b_0 s^2 + b_1 s + b_2}{s^2 + a_1 s + a_2} \equiv \frac{b_0 s^2 + b_1 s + b_2}{(s - p_1)(s - p_2)} \end{aligned} \tag{21.4.1}$$

and corresponding differential equations,

$$\begin{aligned} &\text{(1st order):} \qquad \dot{y} + a_1 y = b_0\dot{x} + b_1 x\\ &\text{(2nd order):} \qquad \ddot{y} + a_1\dot{y} + a_2 y = b_0\ddot{x} + b_1\dot{x} + b_2 x \end{aligned} \tag{21.4.2}$$

The problem is to solve Eqs. (21.4.2) for a given causal input signal $x(t)$ and specified initial conditions, $y(0^-)$ in the 1st order case, or, $y(0^-), \dot{y}(0^-)$ in the 2nd order one.

In the *Laplace method*, we take Laplace transforms of both sides of Eqs. (21.4.2) incorporating the initial conditions at $t = 0^-$,

(1st order):

$$\big[sY(s) - y(0^-)\big] + a_1 Y(s) = b_0 sX(s) + b_1 X(s)$$

(2nd order):

$$\big[s^2 Y(s) - sy(0^-) - \dot{y}(0^-)\big] + a_1\big[sY(s) - y(0^-)\big] + a_2 Y(s) = b_0 s^2 X(s) + b_1 sX(s) + b_2 X(s)$$

and solve for the Laplace transform of the output $Y(s)$,

$$\begin{aligned} &\text{(1st order):} \quad Y(s) = \underbrace{\frac{y(0^-)}{s + a_1}}_{\text{zero-input}} + \underbrace{\left(\frac{b_0 s + b_1}{s + a_1}\right)X(s)}_{\text{zero-state}}\\ &\text{(2nd order):} \quad Y(s) = \underbrace{\frac{(s + a_1)y(0^-) + \dot{y}(0^-)}{s^2 + a_1 s + a_2}}_{\text{zero-input}} + \underbrace{\left(\frac{b_0 s^2 + b_1 s + b_2}{s^2 + a_1 s + a_2}\right)X(s)}_{\text{zero-state}} \end{aligned} \tag{21.4.3}$$

with the first term representing the zero-input part, and the second, the zero-state part. The time-domain solution $y(t)$ is then obtained by inverting the Laplace transforms in Eq. (21.4.3) using partial fraction techniques.

In the *classical method*, we first transform the ICs at $t = 0^-$ to the ICs at $t = 0^+$ using the mappings of Eq. (21.2.7). Then pick a particular solution of the differential equations (21.4.2), typically, a forced solution $y_f(t)$ using the table on the previous page as a guide. The general solution is then obtained by adding to $y_f(t)$ a homogeneous solution consisting of a sum of characteristic modes,

$$\begin{aligned} &\text{(1st order):} \qquad y(t)=c_1e^{p_1t}+y_f(t) \\ &\text{(2nd order):} \qquad y(t)=c_1e^{p_1t}+c_2e^{p_2t}+y_f(t) \end{aligned} \tag{21.4.4}$$

and fix the coefficients c_1, c_2 by imposing the $t=0^+$ initial conditions, i.e.,

$$\text{(1st order):} \qquad y(0^+)=c_1+y_f(0^+)$$

$$\text{(2nd order):} \qquad \begin{cases} y(0^+)=c_1+c_2+y_f(0^+) \\ \dot{y}(0^+)=c_1p_1+c_2p_2+\dot{y}_f(0^+) \end{cases}$$

21.5 Steady-State Sinusoidal Response

For the special case of a causal sinusoidal input, we can find the complete (zero-state) solution as the sum of the forced term and the transients arising from the filter poles. Consider a 2nd order filter with distinct poles p_1, p_2, and a complex sinusoidal input,

$$H(s)=\frac{b_0s^2+b_1s+b_2}{(s-p_1)(s-p_2)}\,, \qquad x(t)=e^{j\Omega_0t}u(t) \quad\Rightarrow\quad X(s)=\frac{1}{s-j\Omega_0}$$

then, the *zero-state* output will have Laplace transform,

$$Y(s)=H(s)X(s)=\frac{b_0s^2+b_1s+b_2}{(s-p_1)(s-p_2)(s-j\Omega_0)}$$

which can be expanded in partial fractions in the form,

$$Y(s)=\frac{b_0s^2+b_1s+b_2}{(s-p_1)(s-p_2)(s-j\Omega_0)} = \underbrace{\frac{H(\Omega_0)}{s-j\Omega_0}}_{\text{forced response}} + \underbrace{\frac{R_1}{s-p_1}+\frac{R_2}{s-p_2}}_{\text{transients}} \tag{21.5.1}$$

with, $H(\Omega_0)=H(s)\big|_{s=j\Omega_0}$, and inverted into the time domain,

$$y(t)=\underbrace{H(\Omega_0)\,e^{j\Omega_0t}}_{\text{steady-state}}+\underbrace{R_1e^{p_1t}+R_2e^{p_2t}}_{\text{transients}}, \quad t\ge 0 \tag{21.5.2}$$

The first term represents the steady-state sinusoidal response, and the last two terms, the transients which decay exponentially.

To find R_1, R_2, one must carry out the above partial fraction expansion on $Y(s)$, which can be facilitated by the use of the function **residue** applied on $Y(s)$, or by the use of the symbolic toolbox functions **laplace** and **partfrac**.

If the input were real-valued, that is, either, $\cos(\Omega_0t)u(t)$, or, $\sin(\Omega_0t)u(t)$, then (assuming that the filter coefficients are real), the solution is obtained by extracting the real or the imaginary parts of the solution in Eq. (21.5.2).

21.6 Continuous-Time Example

Consider the following linear system,

$$\ddot{y}(t)+3\dot{y}(t)+2y(t)=\dot{x}(t) \tag{21.6.1}$$

driven by the causal input,

$$x(t)=10e^{-3t}u(t)$$

and subject to the initial conditions at $t=0^-$,

$$y(0^-)=0\,, \quad \dot{y}(0^-)=-5$$

a. Determine the transfer function $H(s)$ of this system, and determine analytically (i.e., by hand) its partial fraction expansion (PFE). Then, determine the PFE again using MATLAB's **residue** function, and alternatively, using the **partfrac** function of the symbolic toolbox.

b. Using inverse Laplace transforms, determine analytically the impulse response $h(t)$ of this system. Then, determine it again using MATLAB's symbolic toolbox.

c. Using Laplace transforms, determine analytically the *zero-input response*, y_{zi}, subject to the given initial conditions. Then, determine it again by working exclusively in the time domain and expressing it as a linear combination of characteristic modes, and fixing the expansion coefficients from the initial conditions. Finally, determine the solution again with MATLAB's symbolic toolbox, using the **ilaplace** function and, alternatively, the **dsolve** function.

d. For the given input $x(t)$, determine the *zero-state response* by analytically performing the convolution operation, $y(t)=h(t)*x(t)$.

e. Determine the above zero-state response analytically using Laplace transforms. Then, determine it again with MATLAB's symbolic toolbox, using the **ilaplace** function and, alternatively, the **dsolve** function.

f. For the given input and initial conditions, determine the *full* solution of Eq. (21.6.1) consisting of the sum of the zero-input and zero-state responses found above. Then, determine it again analytically using Laplace transforms and carrying out the partial fraction expansions by hand. Then, determine the full solution again using the function **ilaplace** of the symbolic toolbox.

g. Given the above initial conditions, $y(0^-)=0$, $\dot{y}(0^-)=-5$, what are the corresponding initial conditions at $t=0^+$, that is, $y(0^+), \dot{y}(0^+)$? Using the conditions at $t=0^+$, re-derive the full solution of part (g), using the "classical method" described in Section 2.5 of the text. Then, derive it again with the symbolic toolbox and the function **dsolve**.

h. Using the built-in function **lsim**, compute the output $y(t)$ that corresponds to the given input $x(t)$ and initial conditions, $y(0^-)=0$, $\dot{y}(0^-)=-5$, and plot it over the time interval $0\le t\le 6$. This is a bit tricky since the initial conditions are non-zero.

Solution

a. Taking Laplace transforms of both sides of Eq. (21.6.1) with no initial conditions, we have,

$$s^2Y(s)+3sY(s)+2Y(s)=sX(s) \quad\Rightarrow\quad \boxed{H(s)=\frac{Y(s)}{X(s)}=\frac{s}{s^2+3s+2}}$$

For the PFE, we have,

$$H(s)=\frac{s}{s^2+3s+2}=\frac{s}{(s+2)(s+1)}=\frac{A}{s+2}+\frac{B}{s+1}$$

where

$$A=\frac{s}{s+1}\bigg|_{s=-2}=\frac{-2}{-2+1}=2\,,\qquad B=\frac{s}{s+2}\bigg|_{s=-1}=\frac{-1}{-1+2}=-1$$

so that

$$\boxed{H(s)=\frac{s}{s^2+3s+2}=\frac{s}{(s+2)(s+1)}=\frac{2}{s+2}-\frac{1}{s+1}} \tag{21.6.2}$$

Using the **residue** function we find,

```
num = [1,0]; den = [1,3,2];
[r,p] = residue(num,den)

% r =
%      2
%     -1
% p =
%     -2
%     -1
```

where the residues r_1, r_2 are the same as A, B. Using the symbolic toolbox and the function **partfrac**, we obtain the same PFE result,

```
syms s
H = s/(s^2+3*s+2);
H = partfrac(H);     % H = 2/(s + 2) - 1/(s + 1)
```

b. Inverting the PFE in Eq. (21.6.2), we find,

$$\boxed{h(t)=2e^{-2t}u(t)-e^{-t}u(t)}$$

where we used the basic transform pair,

$$e^{-at}u(t) \longleftrightarrow \frac{1}{s+a}$$

Using the symbolic toolbox, we obtain the same,

```
syms s
H = s/(s^2+3*s+2);
h = ilaplace(H)       % h = 2*exp(-2*t) - exp(-t)
```

c. Let us solve this for arbitrary initial conditions, $y(0^-)=y_0$ and $\dot{y}(0^-)=\dot{y}_0$, and at the end set $y_0=0$ and $\dot{y}_0=-5$. The differential equation (21.6.1) with $x(t)=0$ transforms in the s-domain into,

$$\ddot{y}(t)+3\dot{y}(t)+2y(t)=0 \quad\Rightarrow\quad s^2Y(s)-sy_0-\dot{y}_0+3\big(sY(s)-y_0\big)+2Y(s)=0$$

Solving for $Y(s)$ and performing its partial fraction expansion, we have,

$$Y(s)=\frac{sy_0+\dot{y}_0+3y_0}{s^2+3s+2}=\frac{sy_0+\dot{y}_0+3y_0}{(s+1)(s+2)}=\frac{\dot{y}_0+2y_0}{s+1}-\frac{\dot{y}_0+y_0}{s+2}$$

which gives the zero-input response in the time domain,

$$\boxed{y_{zi}(t)=(\dot{y}_0+2y_0)e^{-t}-(\dot{y}_0+y_0)e^{-2t}}\,,\quad t\geq 0 \tag{21.6.3}$$

and in the specific case of $y_0=0$ and $\dot{y}_0=-5$,

$$\boxed{y_{zi}(t)=-5e^{-t}+5e^{-2t}}\,,\quad t\geq 0 \tag{21.6.4}$$

An alternative approach is to work in the time-domain and express $y(t)$ and its derivative as a linear combination of characteristic modes, and fix the expansion coefficients from the initial conditions, that is, set

$$\begin{aligned} y(t)&=c_1e^{-t}+c_2e^{-2t}\\ \dot{y}(t)&=-c_1e^{-t}-2c_2e^{-2t}\end{aligned}$$

and at $t=0^-$, impose the conditions,

$$\begin{aligned} y(0^-)&=c_1+c_2=y_0\\ \dot{y}(0^-)&=-c_1-2c_2=\dot{y}_0\end{aligned} \quad\Rightarrow\quad \begin{aligned} c_1&=\dot{y}_0+2y_0\\ c_2&=-\dot{y}_0-y_0\end{aligned}$$

which results in the same answer as in Eq. (21.6.3). The same expression is obtained using the **ilaplace** function of the symbolic toolbox, where `y0,dy0` stand for the constants $y_0, \dot{y}_0$,

```
syms s y0 dy0 Y
Y = solve(s^2*Y-s*y0-dy0 + 3*(s*Y-y0)+2*Y==0,Y)
Y = partfrac(Y,s)         % Y = (dy0 + 2*y0)/(s + 1) - (dy0 + y0)/(s + 2)
yzi = ilaplace(Y)         % yzi = exp(-t)*(dy0 + 2*y0) - exp(-2*t)*(dy0 + y0)
```

Alternatively, we can use the **dsolve** function,

```
syms y0 dy0
yzi = dsolve('D2y + 3*Dy+ 2*y = 0', 'y(0)=y0', 'Dy(0)=dy0')
```

d. The convolutional expression for the zero-state output is,

$$y_{zs}(t) = \int_{-\infty}^{\infty} h(t')x(t-t')\,dt' \tag{21.6.5}$$

Because the input $x(t) = 10e^{-3t}u(t)$ is causal, the range of its argument in Eq. (21.6.5) must be restricted to, $t - t' \ge 0$. Similarly, because $h(t')$ is causal, its argument must be $t' \ge 0$. Combining the two inequalities, we have,

$$\begin{array}{c} t - t' \ge 0 \\ t' \ge 0 \end{array} \quad \Rightarrow \quad \begin{array}{c} t \ge 0 \\ 0 \le t' \le t \end{array}$$

Thus, $y_{zs}(t)$ must also be causal, and for $t \ge 0$, the integral in (21.6.5) simplifies into,

$$\begin{aligned} y_{zs}(t) &= \int_0^t h(t')x(t-t')\,dt' = \int_0^t (2e^{-2t'} - e^{-t'})10e^{-3(t-t')}\,dt' \\ &= 10e^{-3t}\int_0^t (2e^{-2t'} - e^{-t'})e^{3t'}\,dt' = 10e^{-3t}\int_0^t (2e^{t'} - e^{2t'})\,dt' \\ &= 10e^{-3t}\left[2(e^t - 1) - \frac{1}{2}(e^{2t} - 1)\right] = -5e^{-t} + 20e^{-2t} - 15e^{-3t} \end{aligned}$$

thus,

$$\boxed{y_{zs}(t) = \left[-5e^{-t} + 20e^{-2t} - 15e^{-3t}\right]u(t)} \tag{21.6.6}$$

The integration can also be performed with the **int** function of the symbolic toolbox,

```
syms t tau
x = 10*exp(-3*(t-tau));
h =  2*exp(-2*tau) - exp(-tau);
yzs = int(h*x, tau, 0, t)       % yzs = 20*exp(-2*t) - 5*exp(-t) - 15*exp(-
3*t)
```

e. The Laplace transform of the input $x(t) = 10e^{-3t}u(t)$ is, $X(s) = 10/(s+3)$. It follows that the transform of the zero-state output will be,

$$Y(s) = H(s)X(s) = \frac{s}{s^2 + 3s + 2}\cdot\frac{10}{s+3} = \frac{10s}{(s+1)(s+2)(s+3)}$$

with PFE,

$$Y(s) = \frac{10s}{(s+1)(s+2)(s+3)} = \frac{A}{s+1} + \frac{B}{s+2} + \frac{C}{s+3}$$

where,

$$\begin{aligned} A &= (s+1)Y(s)\Big|_{s=-1} = \frac{10s}{(s+2)(s+3)}\Big|_{s=-1} = \frac{10(-1)}{(-1+2)(-1+3)} = -5 \\ B &= (s+2)Y(s)\Big|_{s=-2} = \frac{10s}{(s+1)(s+3)}\Big|_{s=-2} = \frac{10(-2)}{(-2+1)(-2+3)} = 20 \\ C &= (s+3)Y(s)\Big|_{s=-3} = \frac{10s}{(s+1)(s+2)}\Big|_{s=-3} = \frac{10(-3)}{(-3+1)(-3+2)} = -15 \end{aligned}$$

Inverting the Laplace transform $Y(s)$, we obtain the time-domain zero-state response, which agrees with that of Eq. (21.6.6),

$$y_{zs}(t) = \left[-5e^{-t} + 20e^{-2t} - 15e^{-3t}\right]u(t)$$

The PFE residues can also be obtained by the function **residue**, where the outputs r_1, r_2, r_3 correspond to C, B, A, respectively,

```
[r,p] = residue([10,0], conv([1 3 2],[1 3]))

% r =
%   -15.0000
%    20.0000
%    -5.0000
% p =
%    -3.0000
%    -2.0000
%    -1.0000
```

The indicated convolution operation, `conv([1 3 2],[1 3])`, results in the coefficients, [1, 6, 11, 6], and effectively multiplies the polynomials,

$$(s^2 + 3s + 2)(s + 3) = s^3 + 6s^2 + 11s + 6$$

The PFE and the Laplace inversions can also be accomplished with the symbolic toolbox,

```
syms s
H = s/(s^2+3*s+2);
X = 10/(s+3);
Y = H*X;                % Y = 10*s/((s + 3)*(s^2 + 3*s + 2))
Y = partfrac(Y)         % Y = 20/(s + 2) - 5/(s + 1) - 15/(s + 3)
yzs = ilaplace(Y)       % yzs = 20*exp(-2*t) - 5*exp(-t) - 15*exp(-3*t)
```

For $t \ge 0$, we obtain from the above solution,

$$\begin{array}{l} y_{zs}(t) = -5e^{-t} + 20e^{-2t} - 15e^{-3t} \\ \dot{y}_{zs}(t) = 5e^{-t} - 40e^{-2t} + 45e^{-3t} \end{array} \quad \Rightarrow \quad \begin{array}{l} y_{zs}(0^+) = -5 + 20 - 15 = 0 \\ \dot{y}_{zs}(0^+) = 5 - 40 + 45 = 10 \end{array}$$

The term, “zero-state” solution refers to zero initial conditions at time $t = 0^-$. As we see above, at $t = 0^+$ the initial conditions are not zero. See part (h) for more discussion on this issue, and on how to predict the conditions at $t = 0^+$ from those at $t = 0^-$. The symbolic toolbox solution using the function **ilaplace** requires the $t = 0^-$ conditions, whereas the solution using **dsolve**, requires the $t = 0^+$ conditions.

In the present case, since we just found $y_{zs}(0^+) = 0$, $\dot{y}_{zs}(0^+) = 10$, we can apply the **dsolve** function, noting that $\dot{x}(t) = -3\cdot 10e^{-3t}$ for $t \ge 0^+$,

```
syms t yzs(t)
yzs = dsolve('D2y+3*Dy+2*y = 10*(-3)*exp(-3*t)', 'y(0)=0', 'Dy(0)=10')
```

which results in the same solution as that of Eq. (21.6.6).

f. Adding up the zero-input and zero-state solutions of Eqs. (21.6.3) and (21.6.6), and combining like exponential terms, we obtain the total solution of Eq. (21.6.1), which meets the arbitrary initial conditions, $y(0^-)=y_0,\ \dot{y}(0^-)=\dot{y}_0$,

$$\boxed{y(t)=(\dot{y}_0+2y_0-5)e^{-t}+(20-y_0-\dot{y}_0)e^{-2t}-15e^{-3t}}\,,\quad t\ge 0^+ \tag{21.6.7}$$

and for the particular values, $y(0^-)=0,\ \dot{y}(0^-)=-5$,

$$\boxed{y(t)=-10e^{-t}+25e^{-2t}-15e^{-3t}}\,,\quad t\ge 0^+ \tag{21.6.8}$$

The first two terms depend only on the characteristic modes e^{-t}, e^{-2t}, and are referred to as the "natural response" or "homogeneous solution", whereas the last term depends only on the input $x(t)=10e^{-3t}$ and is referred to as the "particular solution" or "forced response",

$$y(t)=\underbrace{(\dot{y}_0+2y_0-5)e^{-t}+(20-y_0-\dot{y}_0)e^{-2t}}_{\text{homogeneous}}-\underbrace{15e^{-3t}}_{\text{forced}}$$

The factor -15 in the forced response can be predicted in advance using the following result: Given a system with transfer function $H(s)$ and an exponential causal input $x(t)=Ae^{-at}$, then the forced response output is simply, $y_{\text{forced}}(t)=AH(-a)e^{-at}$, where $H(-a)$ is the transfer function $H(s)$ evaluated at $s=-a$ $\big($assuming that $s=-a$ is not a pole of $H(s)\big)$. Thus, in our example,

$$y_{\text{forced}}(t)=10H(-3)e^{-3t}=10\cdot\frac{s}{s^2+3s+2}\bigg|_{s=-3}e^{-3t}=-15e^{-3t}$$

Next, we derive the total solution using Laplace transforms and partial fraction expansions. The approach is similar to that of part (d), except here the right-hand sides are not zero. For the case of arbitrary initial conditions, $y(0^-)=y_0$ and $\dot{y}(0^-)=\dot{y}_0$, the transform of the differential equation (21.6.1) is,

$$\ddot{y}(t)+3\dot{y}(t)+2y(t)=\dot{x}(t)\quad\Rightarrow\quad s^2Y(s)-sy_0-\dot{y}_0+3\big(sY(s)-y_0\big)+2Y(s)=sX(s)$$

where the transform of $\dot{x}(t)$ was, $sX(s)-x(0^-)=sX(s)$, since $x(0^-)=0$ because $x(t)$ is causal. Solving for $Y(s)$, and replacing $X(s)=10/(s+3)$, we obtain,

$$\begin{aligned}
Y(s)&=\frac{sy_0+\dot{y}_0+3y_0+sX(s)}{s^2+3s+2}=\frac{(sy_0+\dot{y}_0+3y_0)(s+3)+10s}{(s^2+3s+2)(s+3)}\\
&=\frac{(sy_0+\dot{y}_0+3y_0)(s+3)+10s}{(s+1)(s+2)(s+3)}=\frac{A}{s+1}+\frac{B}{s+2}+\frac{C}{s+3}\\
&=\frac{\dot{y}_0+2y_0-5}{s+1}+\frac{20-y_0-\dot{y}_0}{s+2}-\frac{15}{s+3}
\end{aligned}\tag{21.6.9}$$

where we may verify easily,

$$\begin{aligned}
A&=\frac{(sy_0+\dot{y}_0+3y_0)(s+3)+10s}{(s+2)(s+3)}\bigg|_{s=-1}=\dot{y}_0+2y_0-5\\
B&=\frac{(sy_0+\dot{y}_0+3y_0)(s+3)+10s}{(s+1)(s+3)}\bigg|_{s=-2}=20-y_0-\dot{y}_0\\
C&=\frac{(sy_0+\dot{y}_0+3y_0)(s+3)+10s}{(s+1)(s+2)}\bigg|_{s=-3}=-15
\end{aligned}$$

It follows that the inverse Laplace transform of Eq. (21.6.9) is as in Eq. (21.6.7). The same partial fraction expansion and inverse transform can be obtained easily by the symbolic toolbox,

```
syms s y0 dy0 Y
X = 10/(s+3);
Y = solve(s^2*Y-s*y0-dy0 + 3*(s*Y-y0) + 2*Y == s*X, Y)
Y = partfrac(Y,s)    % Y = (dy0+2*y0-5)/(s + 1) + (20-dy0-y0)/(s + 2) - 15/(s + 3)
y = ilaplace(Y)      % y = exp(-t)*(dy0+2*y0-5) + exp(-2*t)*(20-dy0-y0) - 15*exp(-
3*t)
```

g. We recall that for a second-order system of the form,

$$\ddot{y}(t)+a_1\dot{y}(t)+a_2y(t)=b_0\ddot{x}(t)+b_1\dot{x}(t)+b_2x(t)\quad\Rightarrow\quad\boxed{H(s)=\frac{b_0s^2+b_1s+b_2}{s^2+a_1s+a_2}}$$

and for a causal input $x(t)$ that does not have any $\delta(t)$ terms at $t=0$, the mapping between the initial conditions at $t=0^-$ and those at $t=0^+$ is given by,

$$\boxed{\begin{aligned}
y(0^+)&=y(0^-)+b_0x(0^+)\\
\dot{y}(0^+)&=\dot{y}(0^-)+b_0\dot{x}(0^+)+(b_1-b_0a_1)x(0^+)
\end{aligned}}\tag{21.6.10}$$

For our particular system, we have, $[b_0,b_1,b_2]=[0,1,0]$, so that Eqs. (21.6.10) become,

$$\begin{aligned}
y(0^+)&=y(0^-)\\
\dot{y}(0^+)&=\dot{y}(0^-)+x(0^+)
\end{aligned}\tag{21.6.11}$$

Thus, for the input $x(t)=10e^{-3t}u(t)$, and initial conditions $y_0,\dot{y}_0$ at $t=0^-$, we have,

$$\begin{aligned}
y(0^+)&=y_0\\
\dot{y}(0^+)&=\dot{y}_0+10
\end{aligned}\tag{21.6.12}$$

These are the conditions that must be used in applying the classical method, or the **dsolve** function. In the classical method, we construct the solution as the sum of a particular solution and a general homogeneous solution. For the particular solution, we may take the forced response, which in our example is, $y_{\text{forced}}(t)=$

$-15e^{-3t}$. For the homogeneous solution we form a linear combination of the characteristic modes e^{-t}, e^{-2t}. Thus,

$$\begin{aligned} y(t) &= c_1e^{-t} + c_2e^{-2t} - 15e^{-3t} \\ \dot{y}(t) &= -c_1e^{-t} - 2c_2e^{-2t} + 45e^{-3t} \end{aligned} \qquad \text{(classical method)}$$

for $t \geq 0$. Imposing the $t = 0^+$ conditions (21.6.12), we have,

$$\begin{aligned} y(0^+) &= c_1 + c_2 - 15 = y_0 \\ \dot{y}(0^+) &= -c_1 - 2c_2 + 45 = \dot{y}_0 + 10 \end{aligned} \quad\Rightarrow\quad \begin{aligned} c_1 &= \dot{y}_0 + 2y_0 - 5 \\ c_2 &= 20 - \dot{y}_0 - y_0 \end{aligned}$$

Thus, we obtain the same solution as that in Eq. (21.6.7), for $t \geq 0^+$,

$$y(t) = c_1e^{-t} + c_2e^{-2t} - 15e^{-3t} = (\dot{y}_0 + 2y_0 - 5)e^{-t} + (20 - y_0 - \dot{y}_0)e^{-2t} - 15e^{-3t}$$

Finally, the same solution can be obtained with the **dsolve** function applied with the initial conditions at $t = 0^+$ of Eq. (21.6.12),

```
syms t y0 dy0 y(t)
dy = diff(y,t); ddy = diff(dy,t);
x = 10*exp(-3*t); dx = diff(x,t);
y = dsolve(ddy + 3*dy + 2*y == dx, y(0) == y0, dy(0) == dy0+10)
```

h. Suppose that one naively tries to use the function **lsim** to compute the system output for the given input. This can be done simply by the MATLAB code,

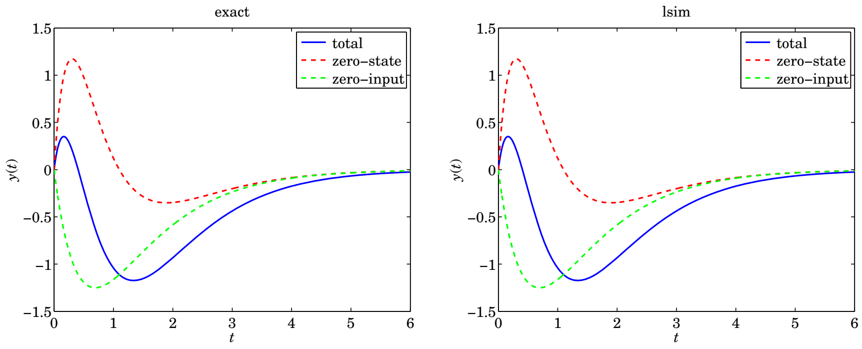
```
t = linspace(0,6,601);          % desired range of t's
x = 10*exp(-3*t);               % input signal
s = tf('s');                    % transfer function variable
H = s/(s^2+3*s+2);              % transfer function object - class(H) is tf
y = lsim(H,x,t);                % assumes zero initial conditions
```

This code, however, will generate only the zero-state part, $y_{zs}(t)$, of the correct answer. The function **lsim** can handle initial conditions, but those are for state-space realizations only. If the initial conditions are specified in terms of the output $y(t)$ and its derivatives, then one must map these initial conditions to the proper state-vector initial conditions to be used in **lsim**.

Such mapping can be accomplished by the so-called observability matrix (we'll discuss it at a later date). The built-in function **obsv** allows one to perform such mapping and thus, use **lsim** with any desired initial conditions at $t = 0^-$. The following MATLAB code illustrates the procedure.

```
y0 = 0; dy0 = -5;          % given initial conditions at t=0-
t = linspace(0,6,601);     % desired time range
x = 10*exp(-3*t);          % input signal
s = tf('s');               % transfer function variable
H = s/(s^2+3*s+2);         % transfer function object - class(H) is tf
S = ss(H);                 % S is state-space model of H - class(S) is ss
yi = [y0; dy0];            % vector of initial conditions with respect to y
si = obsv(S) \ yi;         % map yi to initial state-vector si
y = lsim(S,x,t,si);        % run model S with initial state si
yzs = lsim(S,x,t);         % run model S with zero initial state si=0
plot(t,y, t,yzs,'--');     % compare the nonzero-state and zero-state outputs
```

The computed outputs are shown on the right graph below. Those on the left graph are the exact responses derived in Eqs. (21.6.6) and (21.6.7) . They are virtually indistinguishable from the numerically computed ones using **lsim**.



21.7 Discretization Schemes – Summary

Here, we consider the numerical solution of the differential equations describing linear time-invariant continuous-time (CT) systems, by replacing their analog transfer function by an *approximately equivalent* discrete-time (DT) transfer function, which can be implemented by difference equations.

We summarize the steps of obtaining the DT transfer function for the five most used discretization methods: forward-Euler, backward-Euler, trapezoidal, zero-order hold, and first-order hold. We also discuss how to take into account the initial conditions of the CT system that are typically specified at time, $t = 0^-$.

For a second-order CT system with *differential equation* and analog s-domain transfer function, $H_a(s)$,

$$\boxed{\begin{aligned} &\ddot{y}(t) + A_1\dot{y}(t) + A_2y(t) = B_0\ddot{x}(t) + B_1\dot{x}(t) + B_2x(t) \\ &H_a(s) = \frac{B_0s^2 + B_1s + B_2}{s^2 + A_1s + A_2} \end{aligned}} \qquad \text{(CT system)} \qquad (21.7.1)$$

the approximately equivalent DT system will be described by a second-order *difference equation* and a discrete-time z-domain transfer function, $H_d(z)$,

$$\boxed{\begin{aligned} &y_n + a_1y_{n-1} + a_2y_{n-2} = b_0x_n + b_1x_{n-1} + b_2x_{n-2} \\ &H_d(z) = \frac{b_0 + b_1z^{-1} + b_2z^{-2}}{1 + a_1z^{-1} + a_2z^{-2}} \end{aligned}} \qquad \text{(DT system)} \qquad (21.7.2)$$

where y_n is an approximation to the value of $y(t)$ at the sampling instant, $t_n = nT$, where T is a small time increment, and x_n is the measured value of the input at, $t = t_n$, that is, for, $n = 0, 1, 2, \ldots,$

$$y_n \approx y(t_n) = y(nT)$$
$$x_n = x(t_n) = x(nT)$$

The relationship between the DT coefficients, $\{b_0, b_1, b_2, a_1, a_2\}$, and the CT coefficients, $\{B_0, B_1, B_2, A_1, A_2\}$, depends on the value of T and the chosen discretization scheme. The difference equation (21.7.2) can be iterated by writing it in the form,

$$\boxed{\begin{aligned}&\text{for } n = 0, 1, 2, \dots,\\ &\quad y_n = -a_1 y_{n-1} - a_2 y_{n-2} + b_0 x_n + b_1 x_{n-1} + b_2 x_{n-2}\end{aligned}} \tag{21.7.3}$$

where the two previously computed outputs, y_{n-2}, y_{n-1}, are used compute the current one, y_n. For example, we have explicitly for $n = 0, 1, 2, 3$,

$$\begin{aligned}y_0 &= -a_1 y_{-1} - a_2 y_{-2} + b_0 x_0 + b_1 x_{-1} + b_2 x_{-2}\\ y_1 &= -a_1 y_0 - a_2 y_{-1} + b_0 x_1 + b_1 x_0 + b_2 x_{-1}\\ y_2 &= -a_1 y_1 - a_2 y_0 + b_0 x_2 + b_1 x_1 + b_2 x_0\\ y_3 &= -a_1 y_2 - a_2 y_1 + b_0 x_3 + b_1 x_2 + b_2 x_1\,, \quad \text{and so on,}\end{aligned}$$

To get the iteration going, we need to know the two initial values y_{-1}, y_{-2}. The values of x_{-1}, x_{-2} can be taken to be zero since we always assume a causal input. Because the given initial conditions $y(0^-), \dot{y}(0^-)$ of the differential equation are specified at $t = 0^-$, and T is small, we may choose these starting values as follows,

$$\boxed{\begin{aligned}y_{-1} &\approx y(0^-)\\ y_{-2} &\approx y(0^-) - T\dot{y}(0^-)\end{aligned}} \tag{21.7.4}$$

The second one may be justified by the following approximation of the derivative,

$$\dot{y}(0^-) \approx \frac{y(-T) - y(-2T)}{T} \approx \frac{y_{-1} - y_{-2}}{T} \quad \Rightarrow \quad y_{-2} \approx y_{-1} - T\dot{y}(0^-)$$

For a third-order CT system with initial conditions, $\{y(0^-), \dot{y}(0^-), \ddot{y}(0^-)\}$, we would need three initial values, $\{y_{-1}, y_{-2}, y_{-3}\}$, for the equivalent third-order difference equation. The first two, $\{y_{-1}, y_{-2}\}$, are obtained from Eq. (21.7.4), while the third one can be solved from the approximation of the second derivative,

$$\ddot{y}(0^-) \approx \frac{y_{-1} - 2y_{-2} + y_{-3}}{T^2} = \frac{y(0^-) - 2\big(y(0^-) - T\dot{y}(0^-)\big) + y_{-3}}{T^2}\,, \quad \text{or,}$$

$$\boxed{y_{-3} = T^2\ddot{y}(0^-) - 2T\dot{y}(0^-) + y(0^-)} \tag{21.7.5}$$

Similarly, for a first-order system we have the differential and corresponding difference equations,

$$\begin{aligned}&\dot{y}(t) + A_1 y(t) = B_0\dot{x}(t) + B_1 x(t) \quad \Rightarrow \quad H_a(s) = \frac{B_0 s + B_1}{s + A_1}\\ &y_n + a_1 y_{n-1} = b_0 x_n + b_1 x_{n-1} \quad \Rightarrow \quad H_d(z) = \frac{b_0 + b_1 z^{-1}}{1 + a_1 z^{-1}}\end{aligned} \tag{21.7.6}$$

and iterated in the following form,

$$\boxed{\begin{aligned}&\text{for } n = 0, 1, 2, \dots,\\ &\quad y_n = -a_1 y_{n-1} + b_0 x_n + b_1 x_{n-1}\end{aligned}} \tag{21.7.7}$$

where now only the starting value y_{-1} is needed, and we may choose it as in Eq. (21.7.4),

$$\boxed{y_{-1} \approx y(0^-)} \tag{21.7.8}$$

21.8 Forward/Backward Euler, and Trapezoidal Rules

These methods can be implemented simply by replacing the s variable in the CT transfer function, $H_a(s)$, by a function of the form, $s = f(z)$, mapping the s to the z plane, resulting in the DT transfer function,

$$H_d(z) = H_a(z)\Big|_{s=f(z)} \tag{21.8.1}$$

The three cases of the forward-Euler, backward-Euler, and trapezoidal rules, can be handled in a unified way by the following choice of mapping function, $s = f(z)$,

$$\boxed{s = f(z) = \frac{1 - z^{-1}}{p + qz^{-1}}} \quad \Rightarrow \quad H_d(z) = H_a(s)\Big|_{s=\frac{1-z^{-1}}{p+qz^{-1}}} = H_a\left(\frac{1 - z^{-1}}{p + qz^{-1}}\right) \tag{21.8.2}$$

where the parameters p, q are defined as follows in the three cases, in terms of the discretization time step T,

$$\boxed{\begin{array}{rlll}\text{forward Euler:} & p = 0\,, \quad q = T & \Rightarrow & s = \dfrac{1}{T}(z - 1)\\ \text{backward Euler:} & p = T\,, \quad q = 0 & \Rightarrow & s = \dfrac{1}{T}(1 - z^{-1})\\ \text{trapezoidal/bilinear/Tustin:} & p = q = \dfrac{1}{2}T & \Rightarrow & s = \dfrac{2}{T}\dfrac{1 - z^{-1}}{1 + z^{-1}}\end{array}} \tag{21.8.3}$$

their justification of Eqs. (21.8.3) is discussed below in Sec. 21.12. The mapping between the coefficients $\{b_0, b_1, b_2, a_1, a_2\}$ and $\{B_0, B_1, B_2, A_1, A_2\}$ is obtained from the algebraic relationship,

$$\begin{aligned}\frac{B_0 s^2 + B_1 s + B_2}{s^2 + A_1 s + A_2}\Bigg|_{s=\frac{1-z^{-1}}{p+qz^{-1}}} &= \frac{B_0\left(\dfrac{1 - z^{-1}}{p + qz^{-1}}\right)^2 + B_1\left(\dfrac{1 - z^{-1}}{p + qz^{-1}}\right) + B_2}{\left(\dfrac{1 - z^{-1}}{p + qz^{-1}}\right)^2 + A_1\left(\dfrac{1 - z^{-1}}{p + qz^{-1}}\right) + A_2}\\ &= \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}\end{aligned}$$

The algebra can be carried out quickly with MATLAB's symbolic toolbox,

```
syms B0 B1 B2 A1 A2 s z p q
H = (B0*s^2 + B1*s + B2)/(s^2 + A1*s + A2);
Hd = collect(subs(H,s,(1-z)/(p+q*z)))
```

and we obtain the following relationships that depend on the choices of p, q,

$$\boxed{\begin{aligned} b_0 &= \frac{B_0 + B_1 p + B_2 p^2}{1 + A_1 p + A_2 p^2} \\ b_1 &= \frac{B_1(q-p) - 2B_0 + 2B_2 pq}{1 + A_1 p + A_2 p^2}\,, \quad b_2 = \frac{B_0 - B_1 q + B_2 q^2}{1 + A_1 p + A_2 p^2} \\ a_1 &= \frac{A_1(q-p) - 2 + 2A_2 pq}{1 + A_1 p + A_2 p^2}\,, \quad a_2 = \frac{1 - A_1 q + A_2 q^2}{1 + A_1 p + A_2 p^2} \end{aligned}} \tag{21.8.4}$$

For the first-order case, we define similarly,

$$\frac{b_0 + b_1 z^{-1}}{1 + a_1 z^{-1}} = \frac{B_0 s + B_1}{s + A_1}\bigg|_{s = \frac{1 - z^{-1}}{p + q z^{-1}}}$$

which leads to,

$$\boxed{b_0 = \frac{B_0 + B_1 p}{1 + A_1 p}\,, \quad b_1 = \frac{B_1 q - B_0}{1 + A_1 p}\,, \quad a_1 = \frac{A_1 q - 1}{1 + A_1 p}} \tag{21.8.5}$$

21.9 Zero-Order and First-Order Holds

The justification of the zero-order hold (ZOH) procedure will be discussed below in Sec. 21.14. The corresponding discrete-time transfer function is defined by,

$$H_d(z) = (1 - z^{-1})\cdot \mathcal{Z}\left[\frac{H_a(s)}{s}\right] \qquad \text{(zero-order hold)} \tag{21.9.1}$$

where $\mathcal{Z}[G(s)]$ denotes the z-transform of $G(s)$, a notation and operation to be clarified below. This formula leads to the following computational steps.

Step 1: Start with the analog transfer function $H_a(s)$, then form $H_a(s)/s$, and expand it in partial fractions. For example, for a first-order transfer function we have,

$$H_a(s) = \frac{B_0 s + B_1}{s + p_1} \quad \Rightarrow \quad \frac{H_a(s)}{s} = \frac{B_0 s + B_1}{s(s + p_1)} = \frac{R_0}{s} + \frac{R_1}{s + p_1} \tag{21.9.2}$$

with residues,

$$R_0 = \frac{B_1}{p_1}\,, \quad R_1 = \frac{B_0 - B_1 p_1}{p_1} \tag{21.9.3}$$

Similarly, for a second-order transfer function with distinct poles $(p_1 \neq p_2)$, we obtain,

$$\begin{aligned} H_a(s) &= \frac{B_0 s^2 + B_1 s + B_2}{(s + p_1)(s + p_2)} \quad \Rightarrow \\ \frac{H_a(s)}{s} &= \frac{B_0 s^2 + B_1 s + B_2}{s(s + p_1)(s + p_2)} = \frac{R_0}{s} + \frac{R_1}{s + p_1} + \frac{R_2}{s + p_2} \end{aligned} \tag{21.9.4}$$

where the residues are given by,

$$R_0 = \frac{B_2}{p_1 p_2}\,, \quad R_1 = \frac{B_0 p_1^2 - B_1 p_1 + B_2}{p_1(p_1 - p_2)}\,, \quad R_2 = \frac{B_0 p_2^2 - B_1 p_2 + B_2}{p_2(p_2 - p_1)} \tag{21.9.5}$$

while for the case of a double-pole, $(p_1 = p_2)$, we have,

$$\begin{aligned} H_a(s) &= \frac{B_0 s^2 + B_1 s + B_2}{(s + p_1)^2} \quad \Rightarrow \\ \frac{H_a(s)}{s} &= \frac{B_0 s^2 + B_1 s + B_2}{s(s + p_1)^2} = \frac{R_0}{s} + \frac{R_1}{s + p_1} + \frac{R_2}{(s + p_2)^2} \end{aligned} \tag{21.9.6}$$

with residues,

$$R_0 = \frac{B_2}{p_1^2}\,, \quad R_1 = \frac{B_0 p_1^2 - B_2}{p_1^2}\,, \quad R_2 = -\frac{B_0 p_1^2 - B_1 p_1 + B_2}{p_1} \tag{21.9.7}$$

Step 2: Next, replace single- and double-pole terms as follows in terms of z (applicable also when $p_1 = 0$),

$$\begin{aligned} \frac{1}{s + p_1} \quad &\Rightarrow \quad \frac{1}{1 - e^{-p_1 T} z^{-1}} \\ \frac{1}{(s + p_1)^2} \quad &\Rightarrow \quad \frac{T z^{-1}}{(1 - e^{-p_1 T} z^{-1})^2} \end{aligned} \tag{21.9.8}$$

Step 3: After making these replacements, multiply by an overall factor of $(1 - z^{-1})$ to obtain the final DT transfer function. Thus, for the first-order case, we have,

$$\begin{aligned} H_d(z) &= (1 - z^{-1})\left[\frac{R_0}{1 - z^{-1}} + \frac{R_1}{1 - e^{-p_1 T} z^{-1}}\right] \\ &= R_0 + \frac{R_1(1 - z^{-1})}{1 - e^{-p_1 T} z^{-1}} \equiv \frac{b_0 + b_1 z^{-1}}{1 + a_1 z^{-1}} \end{aligned} \tag{21.9.9}$$

where, with R_0, R_1 given by Eq. (21.9.3),

$$b_0 = R_0 + R_1\,, \quad b_1 = -(R_1 + e^{-p_1 T} R_0)\,, \quad a_1 = -e^{-p_1 T} \tag{21.9.10}$$

Similarly, the second-order case with distinct poles gives,

$$\begin{aligned} H_d(z) &= (1 - z^{-1})\left[\frac{R_0}{1 - z^{-1}} + \frac{R_1}{1 - e^{-p_1 T} z^{-1}} + \frac{R_2}{1 - e^{-p_2 T} z^{-1}}\right] \\ &\equiv \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \end{aligned} \tag{21.9.11}$$

where, with R_0, R_1, R_2 given by Eq. (21.9.5),

$$\begin{aligned} b_0 &= R_0 + R_1 + R_2 = B_0 \\ b_1 &= -R_1 - R_2 - R_1 e^{-p_2 T} - R_2 e^{-p_1 T} - R_0(e^{-p_1 T} + e^{-p_2 T}) \\ b_2 &= R_1 e^{-p_2 T} + R_2 e^{-p_1 T} + R_0 e^{-p_1 T} e^{-p_2 T} \\ a_1 &= -e^{-p_1 T} - e^{-p_2 T}\,, \quad a_2 = e^{-p_1 T} e^{-p_2 T} \end{aligned} \tag{21.9.12}$$

Lastly, for the case of a double-pole, we have,

$$\begin{aligned} H_d(z) &= (1-z^{-1})\left[\frac{R_0}{1-z^{-1}} + \frac{R_1}{1-e^{-p_1T}z^{-1}} + \frac{R_2\,Tz^{-1}}{(1-e^{-p_2T}z^{-1})^2}\right] \\ &\equiv \frac{b_0+b_1z^{-1}+b_2z^{-2}}{1+a_1z^{-1}+a_2z^{-2}} \end{aligned} \tag{21.9.13}$$

where, with R_0, R_1, R_2 given by Eq. (21.9.7),

$$\begin{aligned} b_0 &= R_0 + R_1 = B_0 \\ b_1 &= (R_2T - 2R_0 - R_1)\,e^{-p_1T} - R_1 \\ b_2 &= (R_1 - R_2T + R_0e^{-p_1T})\,e^{-p_1T} \\ a_1 &= -2e^{-p_1T}, \quad a_2 = e^{-2p_1T} \end{aligned} \tag{21.9.14}$$

First-Order Hold

The first-order hold (FOH) is given as follows, to which the same substitutions given in step-2 are to be applied,

$$H_d(z) = \frac{(1-z^{-1})^2}{Tz^{-1}}\cdot \mathcal{Z}\left[\frac{H_a(s)}{s^2}\right] \qquad \text{(first-order hold)} \tag{21.9.15}$$

It is justified in Sec. 21.16.

21.10 Sample-by-Sample Processing

Real-time digital processing means the processing of a sampled input signal on a sample-by-sample basis. Each arriving input sample is subjected to a series of computational steps (referred to as the *sample processing algorithm*) that calculate the current output sample. These computations must be completed within the sampling time interval T that separates incoming time samples.

Modern DSPs are extremely fast and can easily perform hundreds or even thousands of such operations between samples. For example, for a typical hi-fi audio signal sampled at a rate of 40 kHz (40,000 samples/sec), the time interval between samples is $T = 1/40000$ sec $= 25\ \mu$sec. A modern DSP has an instruction time of about 1 nsec for performing a typical multiplication or addition. Therefore, during the interval of $T = 25\ \mu$sec $= 25{,}000$ nsec, it can perform, $25{,}000$ basic instructions, which are more than enough for typical audio processing.

Discrete-time transfer functions of the type of Eq. (21.7.2), as well as higher order ones, can be implemented in real time using different, but mathematically equivalent, block diagram realizations—each block diagram representing the *computational steps* of a particular sample processing algorithm. We discussed realizations in Chap. 7. Here, we review briefly three standard realizations: the *direct, canonical, and transposed*, and their state-space versions, illustrating them with a second-order transfer function,

$$H_d(z) = \frac{b_0+b_1z^{-1}+b_2z^{-2}}{1+a_1z^{-1}+a_2z^{-2}} \tag{21.10.1}$$

The *direct-form realization* (also known as direct-form-1, or, DF-1) attempts to directly realize the right-hand side terms of the corresponding difference equation (21.7.3) relating $x(n)$ and $y(n)$,

$$y(n) = -a_1y(n-1) - a_2y(n-2) + b_0x(n) + b_1x(n-1) + b_2x(n-2)$$

We do not wish to use any arrays because, for real-time processing, the input and output signals can have infinite length. But we do need to keep track of the two previously computed output samples, $y(n-1), y(n-2)$, and the two previously available input samples, $x(n-1), x(n-2)$. To this end, let us use the following notation for these delayed signals,

$$\begin{aligned} v_1(n) &= x(n-1) \\ v_2(n) &= x(n-2) = v_1(n-1) \\ w_1(n) &= y(n-1) \\ w_2(n) &= y(n-2) = w_1(n-1) \end{aligned}$$

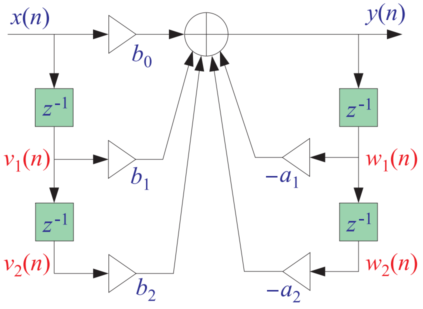
They may be referred to as the internal “states” of the filter. Then, the difference equation can be written as a sum of terms, all occurring at the same instant n,

$$y(n) = -a_1w_1(n) - a_2w_2(n) + b_0x(n) + b_1v_1(n) + b_2v_2(n)$$

Once the current output $y(n)$ is calculated, the states can be updated to the values that they must have at the next time instant, $n+1$. From their definition, we see that their next values are,

$$\begin{aligned} v_1(n+1) &= x(n) \\ v_2(n+1) &= x(n-1) = v_1(n) \\ w_1(n+1) &= y(n) \\ w_2(n+1) &= y(n-1) = w_1(n) \end{aligned}$$

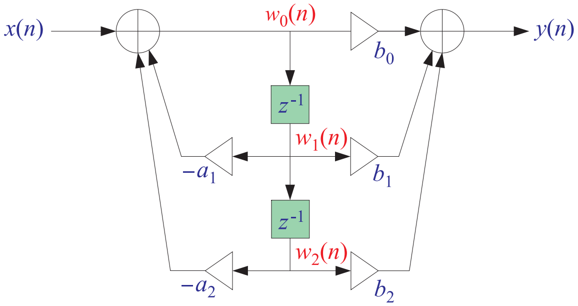
The figure below shows this realization and its sample processing algorithm, where the input delays hold the signals $v_1(n), v_2(n)$ and the output delays hold, $w_1(n), w_2(n)$,



```
initialize $w_1, w_2, v_1, v_2$
for each input sample $x$, do,
    $y = -a_1 w_1 - a_2 w_2 + b_0 x + b_1 v_1 + b_2 v_2$
    $w_2 = w_1$
    $w_1 = y$
    $v_2 = v_1$
    $v_1 = x$
```
(21.10.2)

The operations of updating the contents of the w_2, w_1 delays must be done in the indicated order, and similarly for v_2, v_1. Although the direct form is a straightforward realization (also having fairly robust numerical properties in terms of overflows and coefficient quantization), it requires twice as many delays as they may be necessary. The canonical and transposed realizations use only two delays (for a 2nd order filter), but at the expense of introducing some auxiliary signals—however, the total number of multiplication operations remain the same for all three realizations.

The *canonical realization* (also known as direct-form-2, DF-2, or, controller-canonical-form) is shown below together with its sample processing algorithm.



```
initialize $w_1, w_2$
for each input sample $x$, do,
    $w_0 = -a_1 w_1 - a_2 w_2 + x$
    $y = b_0 w_0 + b_1 w_1 + b_2 w_2$
    $w_2 = w_1$
    $w_1 = w_0$
```
(21.10.3)

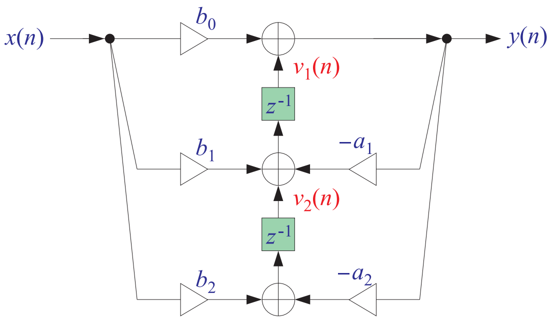
It uses the auxiliary signal $w_0(n)$ that runs between the input and output adders and is neither $x(n)$ nor $y(n)$. It must be computed first at the left adder, and its value then passed to the right adder to compute $y(n)$. The signals $w_1(n)$ and $w_2(n)$ are simply delayed versions of $w_0(n)$ and therefore, must be updated to the next time instant as follows,

$$\begin{aligned} w_2(n) &= w_0(n-2) \\ w_1(n) &= w_0(n-1) \end{aligned} \quad\Rightarrow\quad \begin{aligned} w_2(n) &= w_1(n-1) \\ w_1(n) &= w_0(n-1) \end{aligned} \quad\Rightarrow\quad \begin{aligned} w_2(n+1) &= w_1(n) \\ w_1(n+1) &= w_0(n) \end{aligned}$$

Thus, the realization is described by the following system of *first-order* difference equations from which the above sample processing algorithm is derived, where the indicated computational order (i.e., updating w_2 first, and w_1, second) matters only when stating the sample processing algorithm because the values of w_1, w_2 are overwritten from one sampling instant to the next,

$$\boxed{\begin{aligned} w_0(n) &= -a_1 w_1(n) - a_2 w_2(n) + x(n) \\ y(n) &= b_0 w_0(n) + b_1 w_1(n) + b_2 w_2(n) \\ w_2(n+1) &= w_1(n) \\ w_1(n+1) &= w_0(n) \end{aligned}} \qquad \text{(canonical realization)} \qquad (21.10.4)$$

The *transposed realization* (also known as the observer-canonical form) is shown below together with its sample processing algorithm.



```
initialize $v_1, v_2$
for each input sample $x$, do,
    $y = b_0 x + v_1$
    $v_1 = b_1 x - a_1 y + v_2$
    $v_2 = b_2 x - a_2 y$
```
(21.10.5)

The contents of the two delay registers, $v_1(n), v_2(n)$, are the internal states . Since the corresponding inputs to the delays must be the next values, $v_1(n+1), v_2(n+2)$, it follows that this realization is described by the following system of *first-order* difference equations,

$$\boxed{\begin{aligned} y(n) &= b_0 x(n) + v_1(n) \\ v_1(n+1) &= b_1 x(n) - a_1 y(n) + v_2(n) \\ v_2(n+1) &= b_2 x(n) - a_2 y(n) \end{aligned}} \qquad \text{(transposed realization)} \qquad (21.10.6)$$

Every realization has a *transposed version* obtained by the following transposition rules:

- replace adders by nodes
- replace nodes by adders

- reverse all flows
- exchange input with output

In this sense, the above transposed realization is recognized to be the transposed version of the canonical form. The canonical realization is perhaps the most widely used realization, however, it can often suffer from overflows and coefficient quantization effects. It has the advantage that it can be implemented in DSP hardware using circular delay-line buffers which reduce the number of operations per time update. The transposed realization is fairly robust in terms of overflows and coefficient quantization, and is used by MATLAB's built-in function **filter**.

State-Space Realizations

Block diagram realizations can also be cast in *state-space form* with the contents of the delays that appear in the block diagram chosen to represent the internal states of the realization.

A so-called ABCD state-space realization has the following standard form, written as a system of *first-order difference* equations,

$$\boxed{\begin{aligned} \mathbf{s}(n+1) &= A\mathbf{s}(n)+Bx(n) \\ y(n) &= C\mathbf{s}(n)+Dx(n) \end{aligned}} \qquad \text{(ABCD state-space realization)} \tag{21.10.7}$$

where the state vector $\mathbf{s}(n)$ and the matrices A, B, C, D have appropriate dimensions. The corresponding sample processing algorithm for computing the output sample and updating the state vector can be stated as follows, where the operations must be done in the indicated order,

```
initialize $\mathbf{s}$, then,
for each input sample $x$, do,
    $y = C\mathbf{s} + Dx$,    output
    $\mathbf{s} = A\mathbf{s} + Bx$,    next state
```

(ABCD sample processing algorithm) (21.10.8)

For example, the state vectors for the canonical and transposed realizations of our 2nd order example are the two-dimensional vectors chosen as the contents of the two delay registers that appear in their respective block diagrams, that is,

$$\mathbf{s}(n)=\begin{bmatrix} w_1(n) \\ w_2(n) \end{bmatrix} = \text{canonical}, \qquad \mathbf{s}(n)=\begin{bmatrix} v_1(n) \\ v_2(n) \end{bmatrix} = \text{transposed} \tag{21.10.9}$$

The corresponding A, B, C, D matrices have dimensions, 2×2, 2×1, 1×2, and 1×1, respectively, and are given as follows in the two cases,

$$\begin{aligned} &\text{(canonical):} && A = \begin{bmatrix} -a_1 & -a_2 \\ 1 & 0 \end{bmatrix}, && B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, && C = [c_1, c_2], && D = b_0 \\ &\text{(transposed):} && A = \begin{bmatrix} -a_1 & 1 \\ -a_2 & 0 \end{bmatrix}, && B = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}, && C = [1, 0], && D = b_0 \end{aligned} \tag{21.10.10}$$

where we defined the parameters,

$$\begin{aligned} c_1 &= b_1 - b_0 a_1 \\ c_2 &= b_2 - b_0 a_2 \end{aligned}$$

Using the state-vector definition in Eq. (21.10.9), we may derive the state-space form of the transposed realization by rewriting Eq. (21.10.6) in the following way,

$$\begin{aligned} y(n) &= b_0 x(n) + v_1(n) \\ v_1(n+1) &= b_1 x(n) - a_1 y(n) + v_2(n) = b_1 x(n) - a_1\big[b_0 x(n) + v_1(n)\big] + v_2(n) \\ v_2(n+1) &= b_2 x(n) - a_2 y(n) = b_2 x(n) - a_2\big[b_0 x(n) + v_1(n)\big] \end{aligned}$$

or,

$$\begin{aligned} v_1(n+1) &= -a_1 v_1(n) + v_2(n) + (b_1 - b_0 a_1)\,x(n) = -a_1 v_1(n) + v_2(n) + c_1 x(n) \\ v_2(n+1) &= -a_2 v_1(n) + (b_2 - b_0 a_2)\,x(n) = -a_2 v_1(n) + c_2 x(n) \\ y(n) &= v_1(n) + b_0 x(n) \end{aligned}$$

or, reassembled in ABCD form,

$$\begin{bmatrix} v_1(n+1) \\ v_2(n+1) \end{bmatrix} = \begin{bmatrix} -a_1 & 1 \\ -a_2 & 0 \end{bmatrix} \begin{bmatrix} v_1(n) \\ v_2(n) \end{bmatrix} + \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} x(n)$$

$$y(n) = [1\,,\ 0] \begin{bmatrix} v_1(n) \\ v_2(n) \end{bmatrix} + b_0 x(n)$$

Similarly, we have for the canonical form,

$$\begin{aligned} w_1(n+1) &= w_0(n) = -a_1 w_1(n) - a_2 w_2(n) + x(n) \\ w_2(n+1) &= w_1(n) \\ y(n) &= b_0 w_0(n) + b_1 w_1(n) + b_2 w_2(n) \\ &= b_0\big[-a_1 w_1(n) - a_2 w_2(n) + x(n)\big] + b_1 w_1(n) + b_2 w_2(n) \\ &= (b_1 - b_0 a_1) w_1(n) + (b_2 - b_0 a_2) w_2(n) + b_0 x(n) \\ &= c_1 w_1(n) + c_2 w_2(n) + b_0 x(n) \end{aligned}$$

or, in ABCD form,

$$\begin{bmatrix} w_1(n+1) \\ w_2(n+1) \end{bmatrix} = \begin{bmatrix} -a_1 & -a_2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} w_1(n) \\ w_2(n) \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} x(n)$$

$$y(n) = [c_1, c_2] \begin{bmatrix} w_1(n) \\ w_2(n) \end{bmatrix} + b_0 x(n)$$

Note that the ABCD parameters of the canonical and transposed cases are related to each other by the following mappings, which actually apply more generally to all transposed realizations and are effectively equivalent to the four transposition rules mentioned above,

$$\boxed{\begin{array}{l} A \longrightarrow A^T \\ B \longrightarrow C^T \\ C \longrightarrow B^T \\ D \longrightarrow D \end{array}} \qquad \text{(transposition mapping)} \tag{21.10.11}$$

In terms of the ABCD state-space parameters, the transfer function of the discrete-time system can be obtained by taking z-transforms of both sides of Eqs. (21.10.7) and eliminating the state variable,

$$\begin{aligned} z\,\mathbf{S}(z) &= A\,\mathbf{S}(z)+B\,X(z) \\ Y(z) &= C\,\mathbf{S}(z)+D\,X(z) \end{aligned} \quad \Rightarrow \quad \begin{aligned} \mathbf{S}(z) &= (zI-A)^{-1}B\,X(z) \\ Y(z) &= C\,(zI-A)^{-1}B\,X(z)+D\,X(z)\,, \quad \text{or,} \end{aligned}$$

$$\boxed{H_d(z)=\frac{Y(z)}{X(z)}=C\,(zI-A)^{-1}B+D} \tag{21.10.12}$$

where I denotes the identity matrix. We note that the mapping (21.10.11) leaves (21.10.12) invariant. The corresponding impulse response is obtained by inverting Eq. (21.10.12) causally,

$$h_d(n)=C\,A^{n-1}B\,u(n-1)+D\,\delta(n) \tag{21.10.13}$$

We demonstrate Eq. (21.10.12) explicitly for the canonical realization with parameters given by Eq. (21.10.10),

$$\begin{aligned} zI-A &= \begin{bmatrix} z & 0 \\ 0 & z \end{bmatrix} - \begin{bmatrix} -a_1 & -a_2 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} z+a_1 & -a_2 \\ -1 & z \end{bmatrix} \\ \det(zI-A) &= z^2+a_1z+a_2 \\ (zI-A)^{-1} &= \frac{1}{\det(zI-A)}\begin{bmatrix} z & a_2 \\ 1 & z+a_1 \end{bmatrix} = \frac{1}{z^2+a_1z+a_2}\begin{bmatrix} z & a_2 \\ 1 & z+a_1 \end{bmatrix} \\ C\,(zI-A)^{-1}B &= \frac{1}{z^2+a_1z+a_2}[c_1,c_2]\begin{bmatrix} z & a_2 \\ 1 & z+a_1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{c_1z+c_2}{z^2+a_1z+a_2} \\ H_d(z) &= C\,(zI-A)^{-1}B+D = \frac{c_1z+c_2}{z^2+a_1z+a_2}+b_0 \\ &= \frac{(b_1-b_0a_1)\,z+(b_2-b_0a_2)}{z^2+a_1z+a_2}+b_0 = \frac{b_0z^2+b_1z+b_2}{z^2+a_1z+a_2} \\ &= \frac{b_0+b_1z^{-1}+b_2z^{-2}}{1+a_1z^{-1}+a_2z^{-2}} \end{aligned}$$

MATLAB’s built-in function, **tf2ss**, maps a transfer function defined by numerator and denominator coefficients, `num,den`, to an ABCD state space form that is by default the canonical realization,

```
[A,B,C,D] = tf2ss(num,den);    % canonical state-space form
```

For example, for our 2nd order $H_d(z)$, it generates the parameters of the canonical form in Eq. (21.10.10),

```
num = [b0,b1,b2];
den = [1,a1,a2];
[A,B,C,D] = tf2ss(num,den);    % canonical state-space form
```

21.11 Initialization Procedures

The sample processing algorithms (21.10.2)–(21.10.5) as well as (21.10.8) require that the internal states (i.e., the contents of the delay registers) be properly initialized. Normally, they are initialized to zero, a choice that corresponds to the so-called “‘zero-state” output. However, since the discrete-time systems discussed in this chapter are meant to represent, and numerically solve, continuous-time systems with arbitrary initial conditions specified at $t=0^-$ or $t=0^+$, we need to be able to incorporate such conditions into the discrete case.

We discuss here only the 2nd order case, but the methods can easily be extended to any order. For the case of the direct-form realization, we already discussed in Eqs. (21.7.3) and (21.7.4) how to approximate the two initial output samples, y_{-2}, y_{-1}, at sampled times, $n=-2$, and, $n=-1$, in terms of given initial values, $y(0^-), \dot{y}(0^-)$, specified at, $t=0^-$, for the differential equation, that is,

$$\boxed{\begin{aligned} y_{-1} &\approx y(0^-) \\ y_{-2} &\approx y(0^-)-T\dot{y}(0^-) \end{aligned}} \tag{21.11.1}$$

Recalling that, $w_1(n)=y(n-1)$, $w_2(n)=y(n-2)$, we have, $w_1(0)=y(-1)$, $w_2(0)=y(-2)$. Thus, the following initial values must be used in the direct-form algorithm (21.10.2), with the iteration starting at $n=0$, and assuming a causal input x_n,

$$\begin{aligned} w_1(0) &= y_{-1} = y(0^-) \\ w_2(0) &= y_{-2} = y(0^-)-T\dot{y}(0^-) \\ v_1(0) &= x_{-1} = 0\,, \quad \text{causal input} \\ v_2(0) &= x_{-2} = 0 \end{aligned} \tag{21.11.2}$$

In the other realizations, however, the internal states, $w_i(n)$ or $v_i(n)$, are not directly related to $y(n)$ and therefore, Eq. (21.11.1) cannot be used directly. Since any realization can be mapped into a state-space form, the following procedure can be used (for 2nd order systems) to map the values, y_{-2}, y_{-1}, to the initial value, $\mathbf{s}(0)$, of the two-dimensional state vector $\mathbf{s}(n)$. Given an ABCD realization (21.10.7), we apply Eqs. (21.10.7) at $n=-2$ and $n=-1$, assuming a causal input, i.e., $x_{-2}=x_{-1}=0$,

$$\begin{aligned} y_{-2} &= C\mathbf{s}(-2)+Dx_{-2} = C\mathbf{s}(-2) \\ \mathbf{s}(-1) &= A\mathbf{s}(-2)+Bx_{-2} = A\mathbf{s}(-2) \\ y_{-1} &= C\mathbf{s}(-1)+Dx_{-1} = C\mathbf{s}(-1) = CA\mathbf{s}(-2) \\ \mathbf{s}(0) &= A\mathbf{s}(-1)+Bx_{-1} = A\mathbf{s}(-1) = A^2\mathbf{s}(-2) \end{aligned}$$

or, arranging, y_{-2}, y_{-1}, into a column,

$$\begin{aligned}
\begin{bmatrix} y_{-2} \\ y_{-1} \end{bmatrix} &= \begin{bmatrix} C \\ CA \end{bmatrix} \mathbf{s}(-2) \equiv F\mathbf{s}(-2) \\
\mathbf{s}(0) &= A^2 \mathbf{s}(-2)
\end{aligned} \tag{21.11.3}$$

where we defined the so-called "observability" matrix, which is a 2 × 2 matrix in the 2nd order case,[†]

$$F = \begin{bmatrix} C \\ CA \end{bmatrix} \qquad \text{(observability matrix)} \tag{21.11.4}$$

The types of 2nd order systems that are of interest in practice are so-called "observable" systems[‡] and are characterized by the property that their observability matrix F is *invertible*, i.e., the inverse F^{-1} exists. This allows Eq. (21.11.3) to be solved for $\mathbf{s}(-2)$ which is then used to calculate $\mathbf{s}(0)$,

$$\mathbf{s}(-2) = F^{-1} \begin{bmatrix} y_{-2} \\ y_{-1} \end{bmatrix} \quad \Rightarrow \quad \mathbf{s}(0) = A^2 \mathbf{s}(-2) = A^2 F^{-1} \begin{bmatrix} y_{-2} \\ y_{-1} \end{bmatrix}$$

Thus, given the approximate initial output values (21.11.1), we calculate the initial state vector $\mathbf{s}(0)$ by,

$$\mathbf{s}(0) = A^2 F^{-1} \begin{bmatrix} y_{-2} \\ y_{-1} \end{bmatrix} = A^2 F^{-1} \begin{bmatrix} y(0^-) - T\dot{y}(0^-) \\ y(0^-) \end{bmatrix} \tag{21.11.5}$$

The sample processing algorithm (21.10.8) is then iterated starting at $n = 0$. The examples below clarify these operations. The observability matrix can be computed with MATLAB's built-in function, **obsv**, or with the more specialized function, **obsmat**, placed on the ISP2e toolbox, that allows its calculation for either the canonical or the transposed realizations. The two functions have usage,

```
F = obsv(A,C);                % based on a given ABCD state-space form

F = obsmat(num,den,type);     % type = 'c', 't', for canonical or transposed
```

For our 2nd order example, we may derive the overall transformation matrix $A^2 F^{-1}$ in analytical form. For the canonical realization, we have,

$$C = [c_1, c_2], \quad CA = [c_1, c_2] \begin{bmatrix} -a_1 & -a_2 \\ 1 & 0 \end{bmatrix} = [-a_1 c_1 + c_2\,,\ -a_2 c_1]$$

$$F = \begin{bmatrix} C \\ CA \end{bmatrix} = \begin{bmatrix} c_1 & c_2 \\ -a_1 c_1 + c_2 & -a_2 c_1 \end{bmatrix}$$

$$F^{-1} = \frac{1}{a_1 c_1 c_2 - a_2 c_1^2 - c_2^2} \begin{bmatrix} -a_2 c_1 & -c_2 \\ a_1 c_1 - c_2 & c_1 \end{bmatrix}$$

$$A^2 = \begin{bmatrix} -a_1 & -a_2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} -a_1 & -a_2 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} a_1^2 - a_2 & a_1 a_2 \\ -a_1 & -a_2 \end{bmatrix}$$

$$A^2 F^{-1} = \begin{bmatrix} a_1^2 - a_2 & a_1 a_2 \\ -a_1 & -a_2 \end{bmatrix} \frac{1}{a_1 c_1 c_2 - a_2 c_1^2 - c_2^2} \begin{bmatrix} -a_2 c_1 & -c_2 \\ a_1 c_1 - c_2 & c_1 \end{bmatrix}, \quad \text{or,}$$

$$A^2 F^{-1} = \frac{1}{a_1 c_1 c_2 - a_2 c_1^2 - c_2^2} \begin{bmatrix} c_1 a_2^2 - a_1 a_2 c_2 & a_1 a_2 c_1 + a_2 c_2 - a_1^2 c_2 \\ a_2 c_2 & a_1 c_2 - a_2 c_1 \end{bmatrix} \quad \text{(canonical)} \tag{21.11.6}$$

The transposed case is simpler,

$$C = [1\,,\ 0], \quad CA = [1\,,\ 0] \begin{bmatrix} -a_1 & 1 \\ -a_2 & 0 \end{bmatrix} = [-a_1\,,\ 1]$$

$$F = \begin{bmatrix} C \\ CA \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -a_1 & 1 \end{bmatrix} \quad \Rightarrow \quad F^{-1} = \begin{bmatrix} 1 & 0 \\ a_1 & 1 \end{bmatrix}$$

$$A^2 = \begin{bmatrix} -a_1 & 1 \\ -a_2 & 0 \end{bmatrix} \begin{bmatrix} -a_1 & 1 \\ -a_2 & 0 \end{bmatrix} = \begin{bmatrix} a_1^2 - a_2 & -a_1 \\ a_1 a_2 & -a_2 \end{bmatrix}$$

$$A^2 F^{-1} = \begin{bmatrix} a_1^2 - a_2 & -a_1 \\ a_1 a_2 & -a_2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ a_1 & 1 \end{bmatrix}, \quad \text{or,}$$

$$A^2 F^{-1} = \begin{bmatrix} -a_2 & -a_1 \\ 0 & -a_2 \end{bmatrix} \qquad \text{(transposed)} \tag{21.11.7}$$

There are similar initialization issues in the continuous-time case. The MATLAB function **lsim** for simulating a CT system assumes by default zero initial values. For non-zero values one must use the observability matrix to map these values to those for the internal state required by **lsim**, e.g., see part-(h) of the example of Sec. 21.6.

Example

In the example of Sec. 21.6, we considered the following linear system,

$$\ddot{y}(t) + 3\dot{y}(t) + 2y(t) = \dot{x}(t) \quad \Rightarrow \quad H_a(s) = \frac{s}{s^2 + 3s + 2} \tag{21.11.8}$$

[†] for an Mth order case, F is an $M \times M$ matrix defined as $F = [C;\ CA;\ CA^2;\ \cdots\ CA^{M-1}]$

[‡] `https://en.wikipedia.org/wiki/Observability`

driven by the input, $x(t) = 10e^{-3t}u(t)$, and subject to the initial conditions at $t = 0^-$,

$$y(0^-) = 0\,, \quad \dot{y}(0^-) = -5$$

Using Laplace and **dsolve** methods, we found the following exact solutions for the zero-input, zero-state, and total responses, for $t \ge 0$,

$$\begin{aligned} y_{zi}(t) &= -5e^{-t} + 5e^{-2t} \\ y_{zs}(t) &= -5e^{-t} + 20e^{-2t} - 15e^{-3t} \\ y(t) &= y_{zi}(t) + y_{zs}(t) = -10e^{-t} + 25e^{-2t} - 15e^{-3t} \end{aligned} \tag{21.11.9}$$

Here, we wish to solve the above system numerically by converting it to a discrete-time system and compare the numerical zero-input, zero-state, and total response solutions to those obtained using **lsim** and to the exact ones of Eq. (21.11.9).

a. Determine explicit expressions for the discrete-time coefficients $[b_0, b_1, b_2]$ and $[a_1, a_2]$ of the approximating difference equation using both the pq and the zero-older hold discretization schemes of Eqs. (21.8.4) and (21.9.12).

b. Using a sampling time $T = 0.01$, evaluate the coefficient expressions of part (a) for the trapezoidal case. Then, using the sample processing algorithm of the canonical form Eq. (21.10.3), compute the output signals, $y_{zi}(t), y_{zs}(t), y(t)$, by applying the appropriate input and initial conditions, and compare these outputs with the exact and **lsim** outputs.

c. Repeat part (b), using the zero-order hold discretization method and implement the discrete-time filter using the transposed sample processing algorithm (21.10.5). In addition, compare the outputs computed by (21.10.5) with those computed using the built-in function **filter**, which also uses the transposed form.

Solution

a. The analog transfer function coefficients are $[B_0, B_1, B_2] = [0, 1, 0]$, and $[A_1, A_2] = [3, 2]$. It follows from Eq. (21.8.4),

$$\begin{aligned} b_0 &= \frac{p}{1+3p+2p^2}\,, \quad b_1 = \frac{q-p}{1+3p+2p^2}\,, \quad b_2 = -\frac{q}{1+3p+2p^2} \\ a_1 &= \frac{3(q-p)-2+4pq}{1+3p+2p^2}\,, \quad a_2 = \frac{1-3q+2q^2}{1+3p+2p^2} \end{aligned} \tag{21.11.10}$$

and in particular, for the trapezoidal case with, $p = q = T/2$,

$$\begin{aligned} b_0 &= \frac{T}{T^2+3T+2}\,, \quad b_1 = 0\,, \quad b_2 = -\frac{T}{T^2+3T+2} \\ a_1 &= \frac{2T^2-4}{T^2+3T+2}\,, \quad a_2 = \frac{T^2-3T+2}{T^2+3T+2} \end{aligned} \tag{21.11.11}$$

Similarly for the ZOH case, we find,

$$\begin{aligned} b_0 &= 0\,, \quad b_1 = (e^{-T} - e^{-2T})\,, \quad b_2 = -(e^{-T} - e^{-2T}) \\ a_1 &= -(e^{-T} + e^{-2T})\,, \quad a_2 = e^{-3T} \end{aligned} \tag{21.11.12}$$

b. Evaluating Eq. (21.11.11) for $T = 0.01$, we find the coefficients and corresponding discrete-time transfer function,

$$[b_0, b_1, b_2] = [0.0049,\ 0,\ -0.0049]\,, \quad [a_1, a_2] = [-1.9702,\ 0.9704]$$

$$H_d(z) = \frac{0.0049 - 0.0049z^{-2}}{1 - 1.9702z^{-1} + 0.9704z^{-2}}$$

c. Similarly, Eq. (21.11.12) gives,

$$[b_0, b_1, b_2] = [0,\ 0.0099,\ -0.0099]\,, \quad [a_1, a_2] = [-1.9702,\ 0.9704]$$

$$H_d(z) = \frac{0.0099z^{-1} - 0.0099z^{-2}}{1 - 1.9702z^{-1} + 0.9704z^{-2}}$$

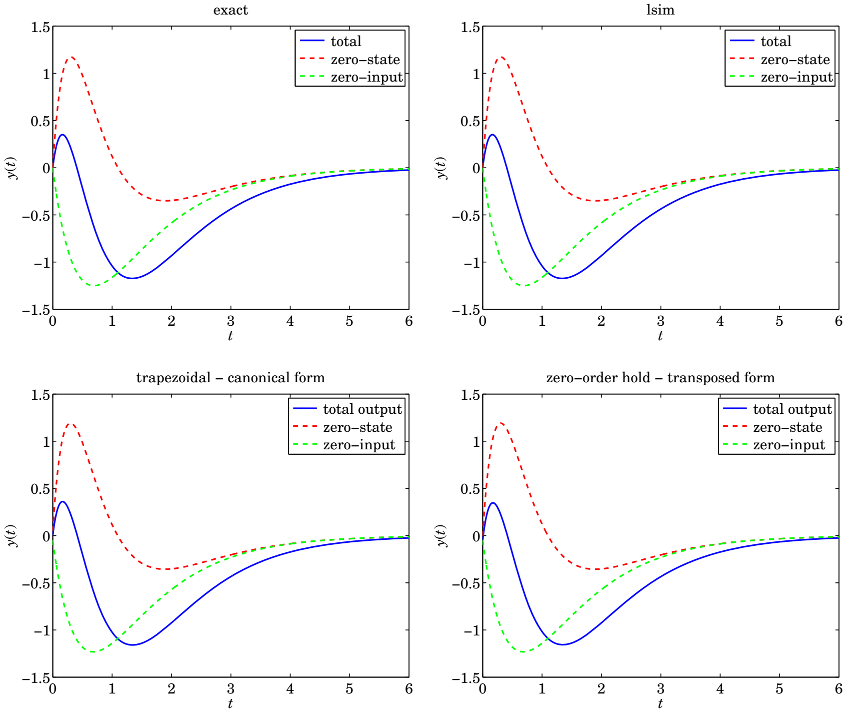
The following MATLAB code illustrates all the numerical computations — the four graphs at the end are visually indistinguishable.

```
y0 = 0; doty0 = -5;                                  % initial conditions at t=0-

x = @(t) 10*exp(-3*t).*(t>=0);                       % input signal

T = 0.01;                                            % sampling time interval
t = 0:T:6;                                           % sampled times, t=n*T

yzi = -5*exp(-t) +  5*exp(-2*t);                     % exact zero-input,
yzs = -5*exp(-t) + 20*exp(-2*t) - 15*exp(-3*t);      % exact zero-state
ye  = -10*exp(-t) + 25*exp(-2*t) - 15*exp(-3*t);     % exact total

figure; plot(t,ye,'b-', t,yzs,'r--', t,yzi,'g--')
title('exact');                                      % plot exact outputs
xlabel('\itt'); ylabel('{\ity}({\itt})');
legend(' total', ' zero-state', ' zero-input', 'location','ne')
xaxis(0,6,0:6); yaxis(-1.5,1.5, -1.5:0.5:1.5);

xt = x(t);                    % input signal samples
s = tf('s');                  % transfer function variable, class tf
H = s/(s^2+3*s+2);            % transfer function object, class tf
S = ss(H);                    % S is canonical state-space model of H, class ss
F = obsv(S);                  % observability matrix for CT canonical

% F = obsmat([0 1 0], [1 3 2], 'c');   % alternative calculation of F

yi = [y0; doty0];             % initial conditions with respect to y
si = F \ yi;                  % initial state-vector, here, si = [0; 2.5]

                              % run LSIM on state model S
ya   = lsim(S,xt,t,si);       % approximate total output with non-zero ICs, si
yazs = lsim(S,xt,t);          % approximate zero-state output with zero ICs, si=0
```

```
yazi = lsim(S,0*xt,t,si);     % approximate zero-input output with non-zero ICs, si

figure; plot(t,ya,’b-’, t,yazs,’r--’, t,yazi,’g--’)
title(’lsim’);                                       % plot LSIM outputs
xlabel(’\itt’); ylabel(’{\ity}({\itt})’);
legend(’ total’, ’ zero-state’, ’ zero-input’, ’location’,’ne’)
xaxis(0,6,0:6); yaxis(-1.5,1.5, -1.5:0.5:1.5);

Err = [norm(ya-ye’)/norm(ye)*100, ...       2% percent errors, exact vs. LSIM
       norm(yazs-yzs’)/norm(yzs)*100, ...
       norm(yazi-yzi’)/norm(yzi)*100]       % Err = [0.0047, 0.0075, 0]


% part (b) - trapezoidal case ------------------------------------------------------

b0 = T/(T^2 + 3*T + 2);                 % trapezoidal coefficients
b1 = 0;
b2 = -T/(T^2 + 3*T + 2);
a1 = 2*(T^2 - 2)/(T^2 + 3*T + 2);
a2 = (T^2 - 3*T + 2)/(T^2 + 3*T + 2);

b = [b0,b1,b2];
a = [1, a1,a2];

num2str([b;a], ’    %1.8f’)             % print with more decimals

%    0.00492587     0.00000000    -0.00492587
%    1.00000000    -1.97024777     0.97044481

% canonical-form -- zero-state response yc_zs(n)

w1 = 0; w2 = 0;                         % initial values
for n=0:length(t)-1                     % iterate canonical form
    w0 = -a1*w1 - a2*w2 + x(n*T);
    yczs(n+1) = b0*w0 + b1*w1 + b2*w2;
    w2 = w1;
    w1 = w0;
end

% canonical-form -- total response yc(n)

A = tf2ss(b,a);                         % state matrix for canonical form
F = obsmat(b,a,’c’);                    % observability matrix for canonical
si = A*A*inv(F)*[y0-T*doty0; y0];       % initial state vector
w1 = si(1); w2 = si(2);                 % initial values
for n=0:length(t)-1                     % iterate canonical form
    w0 = -a1*w1 - a2*w2 + x(n*T);
    yc(n+1) = b0*w0 + b1*w1 + b2*w2;
    w2 = w1;
    w1 = w0;
end

% canonical-form -- zero-input response yc_zi(n)

A = tf2ss(b,a);
F = obsmat(b,a,’c’);
si = A*A*inv(F)*[y0-T*doty0; y0];
w1 = si(1); w2 = si(2);                 % initial values
```

```
for n=0:length(t)-1                     % iterate canonical form
    w0 = -a1*w1 - a2*w2 + 0*x(n*T);     % note, input has been zeroed
    yczi(n+1) = b0*w0 + b1*w1 + b2*w2;
    w2 = w1;
    w1 = w0;
end

figure; plot(t,yc,’b-’, t,yczs,’r--’, t,yczi,’g--’);  % plot trapezoidal outputs
title(’trapezoidal - canonical form’);
xlabel(’\itt’); ylabel(’{\ity}({\itt})’);
legend(’ total output’, ’ zero-state’, ’ zero-input’, ’location’,’ne’)
xaxis(0,6,0:6); yaxis(-1.5,1.5, -1.5:0.5:1.5);

Ec = [norm(yc-ye)/norm(ye)*100, ...      % percent errors, exact vs. trapezoidal
      norm(yczs-yzs)/norm(yzs)*100, ...
      norm(yczi-yzi)/norm(yzi)*100]      % Ec = [1.4988  2.2483  2.0495]

% part (c) - zero-order hold case ----------------------------------------------

b0 = 0;                                 % ZOH coefficients
b1 = exp(-T)-exp(-2*T);
b2 = -exp(-T)+exp(-2*T);
a1 = -exp(-T)-exp(-2*T);
a2 = exp(-3*T);

b = [b0,b1,b2];
a = [1, a1,a2];

num2str([b;a], ’    %1.8f’)        % print with more decimals

%    0.00000000     0.00985116    -0.00985116
%    1.00000000    -1.97024851     0.97044553

% transposed-form -- zero-state response ytzs(n)

v1 = 0; v2 = 0;                         % initial values
for n=0:length(t)-1                     % iterate transposed form
    ytzs(n+1) = b0*x(n*T) + v1;
    v1 = b1*x(n*T) - a1*ytzs(n+1) + v2;
    v2 = b2*x(n*T) - a2*ytzs(n+1);
end

yfzs = filter(b,a,x(t));          % output using FILTER with zero initial conditions

% transposed-form -- total response yt(n)

A = tf2ss(b,a).’;                       % state matrix for transposed form
F = obsmat(b,a,’t’);                    % observability matrix for transposed
si = A*A*inv(F)*[y0-T*doty0; y0];       % initial state vector for transposed form
v1 = si(1); v2 = si(2);                 % initial values
for n=0:length(t)-1                     % iterate transposed form
    yt(n+1) = b0*x(n*T) + v1;
    v1 = b1*x(n*T) - a1*yt(n+1) + v2;
    v2 = b2*x(n*T) - a2*yt(n+1);
end

yf = filter(b,a,x(t),si);         % output using FILTER with initial conditions
```

```
% transposed-form -- zero-input response ytzi(n)

A = tf2ss(b,a).';                     % A = transposed of canonical case
F = obsmat(b,a,'t');
si = A*A*inv(F)*[y0-T*doty0; y0];     % initial state vector
v1 = si(1); v2 = si(2);               % initial values
for n=0:length(t)-1                   % iterate transposed form
    ytzi(n+1) = b0*x(n*T)*0 + v1;        % zero input
    v1 = b1*x(n*T)*0 - a1*ytzi(n+1) + v2;
    v2 = b2*x(n*T)*0 - a2*ytzi(n+1);
end

yfzi = filter(b,a,x(t)*0,si);     % output using FILTER with initial conditions

figure; plot(t,yt,'b-', t,ytzs,'r--', t,ytzi,'g--')     % plot ZOH outputs
title('zero-order hold - transposed form')
xlabel('\itt'); ylabel('{\ity}({\itt})');
legend(' total output', ' zero-state', ' zero-input', 'location','ne')
xaxis(0,6,0:6); yaxis(-1.5,1.5, -1.5:0.5:1.5);

Et = [norm(yt-ye)/norm(ye)*100, ...      % percent errors, exact vs. ZOH
      norm(ytzs-yzs)/norm(yzs)*100, ...
      norm(ytzi-yzi)/norm(yzi)*100]      % Et = [2.0645  1.5151  2.0464]

Ef = [norm(yt-yf)/norm(yf)*100, ...      % percent errors, transposed vs. FILTER
      norm(ytzs-yfzs)/norm(yfzs)*100,...
      norm(ytzi-yfzi)/norm(yfzi)*100]    % Ef = [8.12e-12, 3.34e-12, 0]


% for completeness, we also include the calculation using the direct form
% ----------------------------------------------------------------------------
% direct-form -- total response yd(n)
% w1 = y0;                               % initialize w1,w2
% w2 = y0-T*doty0;
% v1 = 0;                                % initialize v,v2, where x(t) is causal
% v2 = 0;
% for n=0:length(t)-1
%   yd(n+1) = -a1*w1 - a2*w2 + b0*x(n*T) + b1*v1 + b2*v2;   % difference equation
%   w2 = w1;                                                % time updates
%   w1 = yd(n+1);
%   v2 = v1;
%   v1 = x(n*T);
% end
%
% norm(yd-yf)*100/norm(yf)   % percentage error with respect to FILTER = 4.4434e-12
```



21.12 Forward / Backward Euler, and Trapezoidal Rules

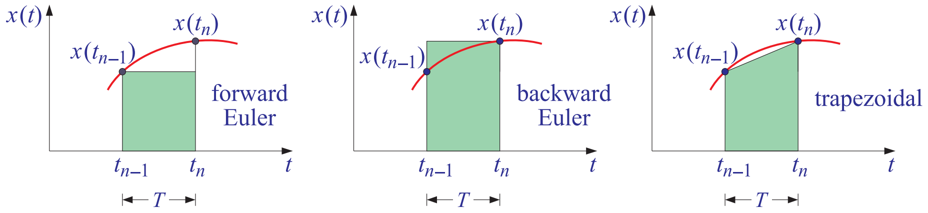
The forward/backward Euler and trapezoidal discretization rules given in Eqs. (21.8.2) and (21.8.3) can be understood intuitively by considering the simple case of an integrator system, that is, one whose input/output differential equation and solution are,

$$\boxed{\dot{y}(t)=x(t)} \quad\Rightarrow\quad \boxed{y(t)=y(0^-)+\int_{0^-}^{t}x(t')\,dt'}\,,\qquad t\ge 0^- \tag{21.12.1}$$

Given a discretization time-step T, then by subtracting the values of $y(t)$ at the two successive time instants, $t_n = nT$, and, $t_{n-1} = (n-1)T$, we obtain from Eq. (21.12.1),

$$y(t_n)-y(t_{n-1})=\int_{t_{n-1}}^{t_n}x(t)\,dt \tag{21.12.2}$$

which represents the area under the curve $x(t)$ over the subinterval $[t_{n-1},t_n]$. The three discretization rules arise by approximating this area in three slightly different ways, as shown below.



In the forward Euler case, the area is approximated by the rectangle of base T and height equal to the left sample $x(t_{n-1})$, extrapolated forward. In the backward Euler case, the right sample $x(t_n)$ is extrapolated backward, defining a rectangular area of base T. In the trapezoidal case, the two points $x(t_{n-1}), x(t_n)$ are connected by a straight line forming a trapezoid of base T (its area is the average of the heights times the base). Thus, the three approximations lead to the difference equations,

$$\begin{aligned}
\text{forward Euler:}\quad & y(t_n)-y(t_{n-1}) = T\cdot x(t_{n-1}) \\
\text{backward Euler:}\quad & y(t_n)-y(t_{n-1}) = T\cdot x(t_n) \\
\text{trapezoidal:}\quad & y(t_n)-y(t_{n-1}) = T\cdot \frac{x(t_{n-1})+x(t_n)}{2}
\end{aligned}$$

And, introducing the p, q definitions of Eq. (21.8.3), the above may be written in a unified compact way,

$$y_n - y_{n-1} = p x_n + q x_{n-1} \tag{21.12.3}$$

where we denoted $x(t_n)$ by x_n and similarly for y_n. In the z-domain this leads to the discrete-time transfer function,

$$H_d(z) = \frac{Y(z)}{X(z)} = \frac{p + q z^{-1}}{1 - z^{-1}} \tag{21.12.4}$$

and if we compare it with the original continuous-time transfer function of the integrator, that is,

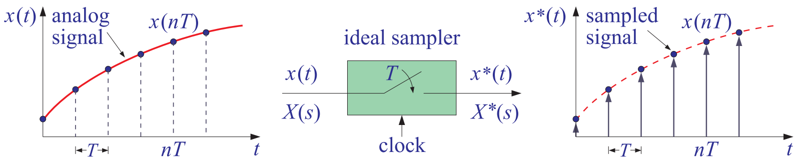
$$H_a(s) = \frac{Y(s)}{X(s)} = \frac{1}{s} \tag{21.12.5}$$

we obtain the identification of the s variable in terms of z as given by Eq. (21.8.2), in the sense that Eq. (21.12.4) acts as if it were the integrator (21.12.5),

$$\frac{1}{s} = \frac{p + q z^{-1}}{1 - z^{-1}} \quad\Rightarrow\quad \boxed{s = \frac{1 - z^{-1}}{p + q z^{-1}}} \tag{21.12.6}$$

21.13 Ideal Sampling, Starred Laplace Transform, z-Transform

An ideal sampler, depicted below, represents the periodic measurement of a continuous-time signal whereby a switch closes periodically , say every T seconds driven by a sampling clock, and capturing the time samples, $x(nT)$, of the analog signal.



Because such ideal switch closes and opens instantaneously, the duration of each sample will be zero, thus, the resulting sampled signal, denoted here by $x^*(t)$,[†] can be viewed as a continuous-time signal consisting of a sum of delta-function pulses, each weighted by the corresponding sample values, and with zero values between samples,

$$\boxed{x^*(t) = \sum_n x(nT)\,\delta(t - nT) = \text{ideally-sampled signal}} \tag{21.13.1}$$

Assuming a causal signal $x(t)$, the above summation can be restricted to $n \ge 0$. The Laplace transform of the ideally sampled signal $x^*(t)$ is referred to as the *starred Laplace transform*,[‡]

$$X^*(s) = \int_{0^-}^{\infty} x^*(t)e^{-st}dt = \int_{0^-}^{\infty} \sum_{n\ge 0} x(nT)\delta(t-nT)e^{-st}dt = \sum_{n\ge 0} x(nT)\int_{0^-}^{\infty}\delta(t-nT)e^{-st}dt$$

or,

$$\boxed{X^*(s) = \sum_{n\ge 0} x(nT)e^{-nsT} = \text{starred Laplace transform}} \tag{21.13.2}$$

With the replacement, $z = e^{sT}$, Eq. (21.13.2) is recognized as the z-transform of the sequence $x(nT)$,

$$\boxed{X(z) = X^*(s)\Big|_{z=e^{sT}} = \sum_{n\ge 0} x(nT)\,z^{-n} = z\text{-transform}} \tag{21.13.3}$$

Often, the following abused notation is used for this z-transform, $X(z) = \mathcal{Z}[X(s)]$, referred to as the *z-transform of a Laplace transform*, that is,

$$\boxed{X(z) = \mathcal{Z}[X(s)] = X^*(s)\Big|_{z=e^{sT}} = \sum_{n\ge 0} x(nT)\,z^{-n}} \tag{21.13.4}$$

which actually consists of the following series of steps going from $X(s)$ to $X(z)$, first, perform an inverse Laplace transform on $X(s)$ to get the analog time signal $x(t)$, then,

[†] A notation not to be confused with complex conjugation
[‡] A notation used primarily in the control systems literature

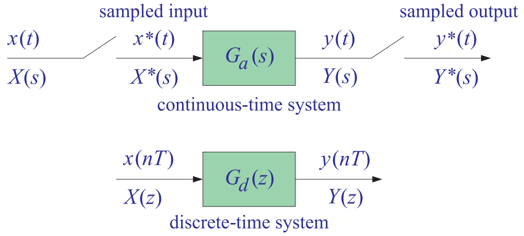
sample $x(t)$ at the sampling instants $t_n = nT$ to obtain the sampled signal $x(nT)$, and finally, perform a z-transform,

$$\boxed{X(s) \xrightarrow{\mathcal{L}^{-1}} x(t) \xrightarrow{\text{sample}} x(nT) \xrightarrow{\mathcal{Z}} X(z) = \sum_{n\geq 0} x(nT)z^{-n}} \tag{21.13.5}$$

and these can be combined into the more accurate but awkward notation,

$$X(z) = \mathcal{Z}\left[\mathcal{L}^{-1}[X(s)]\Big|_{\text{sampled}}\right] \tag{21.13.6}$$

We note also that if the sampled signal $x^*(t)$ is further filtered by an analog system with transfer function $G_a(s)$ and impulse response $g_a(t)$, then, if the analog output is (synchronously) sampled at the same rate, the overall system can be thought of a discrete-time system with transfer function, $G_d(z) = \mathcal{Z}[G_a(s)]$, and impulse response, $g_d(n) = g_a(nT)$, as depicted below,



Indeed, we have the following relationships, assuming causal system and input,

$$\begin{aligned}
&Y(s) = G_a(s)X^*(s)\\
&y(t) = \int_{0^-}^{\infty} g_a(t-t')\,x^*(t')\,dt' = \sum_{m\geq 0} g_a(t-mT)\,x(mT)\\
&y(nT) = \sum_{m\geq 0} g_a(nT-mT)\,x(mT) \quad\Rightarrow\quad g_d(n) = g_a(nT)\\
&Y^*(s) = \big[G_a(s)X^*(s)\big]^* = G_a^*(s)X^*(s) \quad\Rightarrow\quad G_d(z) = \mathcal{Z}[G_a(s)]\\
&Y(z) = G_d(z)X(z)
\end{aligned} \tag{21.13.7}$$

where the indicated factorization, $[G_a(s)X^*(s)]^* = G_a^*(s)X^*(s)$, is valid because one of the factors is already starred.[†] We will use the results of Eq. (21.13.7) in another set discussing digital control systems.

Finally, we note that another way to justify the discretization mappings of Eq. (21.12.6) is to replace the exact relationship, $z = e^{sT}$, with an approximate one based on the following small-x Taylor series expansions of the exponential, that is,

[†] we note the properties that, in general, $[G_1 G_2]^* \neq G_1^* G_2^*$, but, $[G_1 G_2^*]^* = G_1^* G_2^*$

$$\begin{aligned}
e^x &\approx 1 + x\\
e^x &= \frac{1}{e^{-x}} \approx \frac{1}{1-x}\\
e^x &= \frac{e^{x/2}}{e^{-x/2}} \approx \frac{1+x/2}{1-x/2}
\end{aligned}$$

It follows that since T is small, we may make the same approximations,

$$\begin{aligned}
&z = e^{sT} \approx 1 + sT && \Rightarrow \quad s = \frac{1}{T}(z-1) && \text{forward Euler}\\
&z = e^{sT} = \frac{1}{e^{-sT}} \approx \frac{1}{1-sT} && \Rightarrow \quad s = \frac{1}{T}(1-z^{-1}) && \text{backward Euler}\\
&z = e^{sT} = \frac{e^{sT/2}}{e^{-sT/2}} \approx \frac{1+sT/2}{1-sT/2} && \Rightarrow \quad s = \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}} && \text{trapezoidal}
\end{aligned}$$

21.14 Zero-Order Hold

Here, we discuss briefly the three-step design procedure of the zero-order hold discretization scheme resulting in Eq. (21.9.1),

$$H_d(z) = (1-z^{-1})\cdot\mathcal{Z}\left[\frac{H_a(s)}{s}\right] \qquad \text{(zero-order hold)} \tag{21.14.1}$$

where the notation, $G_d(z) = \mathcal{Z}[G(s)]$, was defined above, that is,

$$G_d(z) = \mathcal{Z}[G(s)] = G^*(s)\Big|_{z=e^{sT}} = \sum_{n\geq 0} g(nT)\,z^{-n} \tag{21.14.2}$$

$$G(s) \xrightarrow{\mathcal{L}^{-1}} g(t) \xrightarrow{\text{sample}} g(nT) \xrightarrow{\mathcal{Z}} G_d(z) = \sum_{n\geq 0} g(nT)z^{-n} \tag{21.14.3}$$

$$G_d(z) = \mathcal{Z}\left[\mathcal{L}^{-1}[G(s)]\Big|_{\text{sampled}}\right] \tag{21.14.4}$$

Let us assume that the given (stable and causal) continuous-time system has a proper[†] transfer function $H_a(s)$ with M distinct poles lying in the left-hand s-plane, and assume that its partial-fraction expansion (PFE) has already been made in the form,

$$H_a(s) = R_0 + \sum_{i=1}^{M} \frac{R_i}{s+p_i} \tag{21.14.5}$$

with $\mathrm{Re}(p_i) > 0$, so that the corresponding causal impulse response is,

$$h_a(t) = R_0\,\delta(t) + \sum_{i=1}^{M} R_i e^{-p_i t} u(t) \tag{21.14.6}$$

[†] i.e., the degree of its numerator is at most equal to that of its denominator

Then, the zero-state output due to a causal input $x(t)$ will be, for $t \ge 0$,

$$y(t) = \int_{0^-}^{t} h_a(t-t')x(t')\,dt' = \int_{0^-}^{t}\Big[R_0\,\delta(t-t') + \sum_{i=1}^{M} R_i\, e^{-p_i(t-t')}u(t-t')\Big]x(t')\,dt'\,, \quad \text{or,}$$

$$y(t) = R_0 x(t) + \sum_{i=1}^{M} R_i\,\underbrace{e^{-p_i t}\int_{0}^{t} e^{p_i t'}x(t')\,dt'}_{y_i(t)} = R_0 x(t) + \sum_{i=1}^{M} R_i y_i(t) \tag{21.14.7}$$

It follows from the definition of the partial output $y_i(t)$ after evaluating it at the two successive time instants $t_n = nT$ and $t_{n-1} = (n-1)T$, that it will satisfy the exact relationship,

$$y_i(t_n) - e^{-p_i T} y_i(t_{n-1}) = e^{-p_i t_n}\int_{t_{n-1}}^{t_n} e^{p_i t'}x(t')\,dt' \tag{21.14.8}$$

The *zero-order hold approximation* consists of holding the value of $x(t')$ constant at $x(t_{n-1})$ over the small time interval $[t_{n-1}, t_n]$, that is, replacing $x(t') \approx x(t_{n-1})$ within the integral (21.14.8). The t' integration then can be done explicitly,

$$e^{-p_i t_n}\int_{t_{n-1}}^{t_n} e^{p_i t'}\,dt' = e^{-p_i t_n}\,\frac{e^{p_i t_n} - e^{p_i t_{n-1}}}{p_i} = \frac{1 - e^{-p_i(t_n - t_{n-1})}}{p_i} = \frac{1 - e^{-p_i T}}{p_i}$$

where we used $t_n - t_{n-1} = T$. We obtain then the zero-order hold approximation of the exact equation (21.14.8),

$$y_i(t_n) - e^{-p_i T} y_i(t_{n-1}) = \frac{1 - e^{-p_i T}}{p_i}\, x(t_{n-1}) \tag{21.14.9}$$

and taking z-transforms of both sides, we find,

$$Y_i(z) - e^{-p_i T} z^{-1} Y_i(z) = \frac{1 - e^{-p_i T}}{p_i}\, z^{-1} X(z)\,, \quad \text{or,}$$

$$Y_i(z) = \frac{1 - e^{-p_i T}}{p_i}\,\frac{z^{-1}}{1 - e^{-p_i T} z^{-1}}\, X(z)$$

which can be written as an identity in z in the form,

$$Y_i(z) = \frac{1 - e^{-p_i T}}{p_i}\,\frac{z^{-1}}{1 - e^{-p_i T} z^{-1}}\, X(z) = \frac{1}{p_i}\left[1 - \frac{1 - z^{-1}}{1 - e^{-p_i T} z^{-1}}\right] X(z) \tag{21.14.10}$$

Sampling Eq. (21.14.7) at $t = t_n$, then taking z-transforms, and using Eq. (21.14.10), we find the overall discrete-time transfer function that incorporates the zero-order hold approximation,

$$y(t_n) = R_0 x(t_n) + \sum_{i=1}^{M} R_i y_i(t_n)$$

$$Y(z) = R_0 X(z) + \sum_{i=1}^{M} R_i Y_i(z) = R_0 X(z) + \sum_{i=1}^{M}\frac{R_i}{p_i}\left[1 - \frac{1 - z^{-1}}{1 - e^{-p_i T} z^{-1}}\right] X(z)$$

$$H_d(z) = \frac{Y(z)}{X(z)} = R_0 + \sum_{i=1}^{M}\frac{R_i}{p_i}\,\frac{(1 - e^{-p_i T})z^{-1}}{1 - e^{-p_i T} z^{-1}}$$

$$H_d(z) = R_0 + \sum_{i=1}^{M}\frac{R_i}{p_i}\left[1 - \frac{1 - z^{-1}}{1 - e^{-p_i T} z^{-1}}\right]$$

$$H_d(z) = \left(R_0 + \sum_{i=1}^{M}\frac{R_i}{p_i}\right) - \sum_{i=1}^{M}\frac{R_i}{p_i}\,\frac{1 - z^{-1}}{1 - e^{-p_i T} z^{-1}}\,, \quad \text{or,}$$

$$\boxed{H_d(z) = R_0 + \sum_{i=1}^{M}\frac{R_i}{p_i}\,\frac{(1 - e^{-p_i T})z^{-1}}{1 - e^{-p_i T} z^{-1}} = \left(R_0 + \sum_{i=1}^{M}\frac{R_i}{p_i}\right) - \sum_{i=1}^{M}\frac{R_i}{p_i}\,\frac{1 - z^{-1}}{1 - e^{-p_i T} z^{-1}}} \tag{21.14.11}$$

with discrete-time impulse response, for $n \ge 0$,

$$\boxed{h_d(n) = R_0\,\delta[n] + \sum_{i=1}^{M}\frac{R_i}{p_i}\,(1 - e^{-p_i T})\,e^{-p_i T(n-1)}\,u[n-1]} \tag{21.14.12}$$

Next, we demonstrate that Eq. (21.14.11) is identical to Eq. (21.14.1) and that steps 1–3 can be used to obtain it. To this end, we form, $G(s) = H_a(s)/s$, and perform its PFE, and follow the progression of steps shown in Eq. (21.14.3),

$$G(s) = \frac{H_a(s)}{s} = \frac{R_0}{s} + \sum_{i=1}^{M}\frac{R_i}{s(s+p_i)} = \frac{R_0}{s} + \sum_{i=1}^{M}\frac{R_i}{p_i}\left[\frac{1}{s} - \frac{1}{s+p_i}\right]$$

$$g(t) = R_0 u(t) + \sum_{i=1}^{M}\frac{R_i}{p_i}\left[u(t) - e^{-p_i t}u(t)\right]$$

$$g(t_n) = R_0 u(t_n) + \sum_{i=1}^{M}\frac{R_i}{p_i}\left[u(t_n) - e^{-p_i T n}u(t_n)\right]$$

$$G_d(z) = \sum_{n=0}^{\infty} g(t_n) z^{-n} = \frac{R_0}{1 - z^{-1}} + \sum_{i=1}^{M}\frac{R_i}{p_i}\left[\frac{1}{1 - z^{-1}} - \frac{1}{1 - e^{-p_i T} z^{-1}}\right], \quad \text{or}$$

$$G_d(z) = \left(R_0 + \sum_{i=1}^{M}\frac{R_i}{p_i}\right)\frac{1}{1 - z^{-1}} - \sum_{i=1}^{M}\frac{R_i}{p_i}\,\frac{1}{1 - e^{-p_i T} z^{-1}} \tag{21.14.13}$$

where to be more precise, by $t_n = nT$ we shall mean $t_n^+ = nT+0 = \lim_{\epsilon\to 0^+}(nT+\epsilon)$, that is, evaluating the time samples $g(t_n)$ from the right side (the causal side), so that $u(t_n)$ becomes equal to the discrete-time unit-step $u[n]$.

Next, by multiplying $G_d(z)$ by the factor $(1-z^{-1})$, we obtain the final discrete-time transfer function, which agrees with (21.14.11),

$$(1-z^{-1})G_d(z) = \left(R_0 + \sum_{i=1}^{M}\frac{R_i}{p_i}\right) - \sum_{i=1}^{M}\frac{R_i}{p_i}\,\frac{1-z^{-1}}{1-e^{-p_iT}z^{-1}} = H_d(z)$$

Comparing $G(s)$ and $G_d(z)$ we observe that we are effectively making the substitutions of Eq. (21.9.8), in step-3 of the construction procedure,

$$\begin{aligned} G(s) &= \frac{R_0}{s} + \sum_{i=1}^{M}\frac{R_i}{p_i}\left[\frac{1}{s} - \frac{1}{s+p_i}\right] \\ &\Downarrow \\ G_d(z) &= \frac{R_0}{1-z^{-1}} + \sum_{i=1}^{M}\frac{R_i}{p_i}\left[\frac{1}{1-z^{-1}} - \frac{1}{1-e^{-p_iT}z^{-1}}\right] \end{aligned}$$

An alternative and faster way of showing Eqs. (21.14.11) and (21.14.12) is to work directly with Eq. (21.14.1) and rewrite it in the form,

$$H_d(z) = (1-z^{-1})\cdot\mathcal{Z}\left[\frac{H_a(s)}{s}\right] = \mathcal{Z}\left[F(s)\right]\,, \quad F(s) = \frac{1-e^{-sT}}{s}H_a(s)$$

which, in the time-domain, means that the discrete-time samples $h_d(n)$ will be,

$$h_d(n) = f(t_n^+) \tag{21.14.14}$$

After using the differentiation and delay properties of Laplace transforms, we find from the definition of $F(s)$,

$$sF(s) = H_a(s) - e^{-sT}H_a(s) \quad\Rightarrow\quad \dot f(t) = h_a(t) - h_a(t-T)$$

where a term $f(0^-)$ was dropped because of the assumed causality of $h_a(t)$. Integrating this relationship and dropping another constant of integration for the same reason, we find,

$$f(t) = \int_{t-T}^{t} h_a(t')\,dt' \tag{21.14.15}$$

The same result follows by noting that, $f(t) = g(t) - g(t-T)$, where $g(t)$ is the inverse Laplace transform of, $G(s) = H_a(s)/s$, which by the integration property of Laplace transforms is,

$$g(t) = \int_{0^-}^{t} h_a(t')dt' \quad\Rightarrow\quad f(t) = \int_{0^-}^{t} h_a(t')dt' - \int_{0^-}^{t-T} h_a(t')dt' = \int_{t-T}^{t} h_a(t')\,dt'$$

Thus, evaluating Eq. (21.14.14) at $t = nT$, or rather, at t_n^+, we have, for $n \geq 0$,

$$\boxed{h_d(n) = f(t_n^+) = \int_{t_n^+-T}^{t_n^+} h_a(t')\,dt'} \quad \text{(zero-order hold)} \tag{21.14.16}$$

Next, we apply Eqs. (21.14.15)–(21.14.16) to the given analog system of Eq. (21.14.5) whose impulse response is given by Eq. (21.14.6), assuming $t \geq 0^+$,

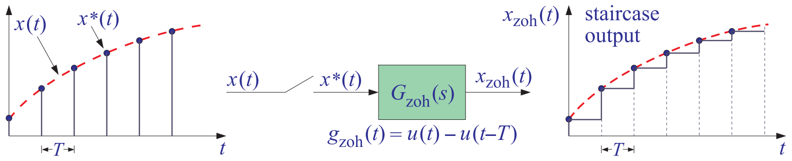
$$\begin{aligned} f(t) &= \int_{t-T}^{t} h_a(t')\,dt' = \int_{t-T}^{t}\left[R_0\,\delta(t') + \sum_{i=1}^{M} R_i e^{-p_i t'}u(t')\right]dt' \\ &= R_0\int_{t-T}^{t}\delta(t')\,dt' + \sum_{i=1}^{M} R_i\,\frac{e^{-p_i\max(0,t-T)} - e^{-p_i t}}{p_i} \end{aligned}$$

where because of the unit-step $u(t')$, the lower limit of integration in the summation terms was constrained to be both, $t - T < t'$ and $0 < t'$, or, $\max(0, t-T) < t'$. The R_0 term vanishes if $t \geq T^+$, or $n \geq 1$, and is equal to R_0 if $t = 0^+$. Similarly, the summation terms vanish at $t = 0$. Thus, based on these properties, we eventually obtain the following expression, which is precisely Eq. (21.14.12),

$$h_d(n) = f(t_n^+) = R_0\,\delta[n] + \sum_{i=1}^{M}\frac{R_i}{p_i}(1-e^{-p_iT})e^{-p_iT(n-1)}\,u[n-1] \tag{21.14.17}$$

Staircase Reconstructor and Zero-Order Hold

Another useful way to understand the zero-order hold operation is to view it as a *staircase reconstruction* filter, or, as a *sample & hold* operation or as a D/A conversion operation, as shown below. It takes as input an ideally sampled signal $x^*(t)$ and reconstructs it back into analog form by filling the time gaps between samples by holding each sample constant for a duration of T seconds.



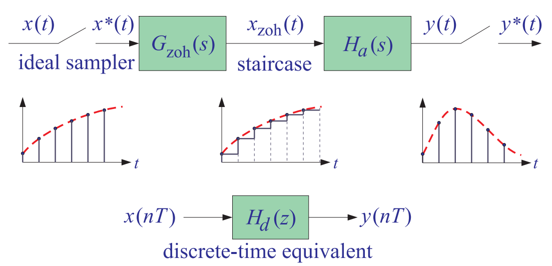
It may be thought of as the filtering the sampled signal $x^*(t)$ by an analog filter whose impulse response is a (causal) rectangular pulse of duration of T seconds,

$$g_{\text{zoh}}(t) = u(t) - u(t-T) \quad\Rightarrow\quad G_{\text{zoh}}(s) = \frac{1-e^{-sT}}{s} \tag{21.14.18}$$

so that the staircase output becomes, for a causal input,

$$\begin{aligned} x_{\text{zoh}}(t) &= \int_{0^-}^{\infty} x^*(t')\,g_{\text{zoh}}(t-t')\,dt' = \int_{0^-}^{\infty}\left[\sum_{n\geq 0} x(nT)\,\delta(t'-nT)\right]g_{\text{zoh}}(t-t')\,dt' \\ &= \sum_{n\geq 0} x(nT)\,g_{\text{zoh}}(t-nT) = \sum_{n\geq 0} x(nT)\left[u(t-nT) - u(t-nT-T)\right] \end{aligned} \tag{21.14.19}$$

The zero-order hold discretization formula (21.14.1) follows if the staircase-reconstructed signal $x_{\text{zoh}}(t)$ is filtered further by the analog filter $H_a(s)$, as shown below, followed by the sampling of the resulting analog output.



The indicated filtering operations can be expressed using Laplace transforms as follows, followed by extracting starred Laplace transforms, and replacing those by z-transforms. Noting that the quantity, $(1-e^{-sT})$, is already starred (at samples $t=0$ and $t=T$ only), we have,

$$\begin{aligned}
Y(s) &= H_a(s)X_{\text{zoh}}(s) = H_a(s)G_{\text{zoh}}(s)X^*(s) = (1-e^{-sT})X^*(s)\,\frac{H_a(s)}{s}\\
Y^*(s) &= \left[(1-e^{-sT})X^*(s)\,\frac{H_a(s)}{s}\right]^* = (1-e^{-sT})X^*(s)\left[\frac{H_a(s)}{s}\right]^* \qquad (21.14.20)\\
Y(z) &= (1-z^{-1})\cdot\mathcal{Z}\left[\frac{H_a(s)}{s}\right]X(z) = H_d(z)X(z)
\end{aligned}$$

The last equation following from the second one by setting $z = e^{sT}$ and using Eq. (21.13.4). Such filtering viewpoint is very useful in considering the discretization of feedback control systems and will be discussed further in Chap. 22.

21.15 Step Invariance and Impulse Invariance

There is yet another viewpoint of the zero-order hold that is intuitive. Writing Eq. (21.14.1) in the more accurate form,

$$H_d(z) = (1-z^{-1})\cdot\mathcal{Z}\left[\mathcal{L}^{-1}\left[\frac{H_a(s)}{s}\right]\bigg|_{\text{sampled}}\right] \qquad (21.15.1)$$

then, dividing by the factor, $(1-z^{-1})$, and taking inverse z-transforms, we may re-write Eq. (21.15.1) in the equivalent form,

$$\boxed{\mathcal{Z}^{-1}\left[\frac{H_d(z)}{1-z^{-1}}\right] = \mathcal{L}^{-1}\left[\frac{H_a(s)}{s}\right]\bigg|_{\text{sampled}}} \qquad \text{(step invariance)} \qquad (21.15.2)$$

Since the Laplace transform of a unit-step $u(t)$ is $1/s$, it follows that the quantity $H_a(s)/s$ will be the Laplace transform of the output of $H_a(s)$ when the input is $u(t)$, i.e., the step response of the filter $H_a(s)$. Similarly, $H_d(z)/(1-z^{-1})$ is the z-transform of the step-response of the discrete-time system $H_d(z)$.

Therefore, Eq. (21.15.2) states that, in the time domain, the step response of the discrete-time filter $H_d(z)$ must *match* the sampled version of the step response of the continuous-time filter $H_a(s)$, a property referred to as *step invariance*.

Impulse invariance is another simple method of discretization by requiring that the impulse response of the discrete-time system $H_d(z)$ match the sampled version of the impulse response of the continuous-time system.

It can be derived as a further approximation of the zero-order hold when the sampling interval T is small. A simple derivation, is to take the small-T limit of the quantity,

$$\frac{1-e^{-sT}}{s} \approx \frac{1-(1-sT)}{s} = T$$

which leads to the approximation of the zero-order hold,

$$H_d(z) = \mathcal{Z}\left[(1-e^{-sT})\,\frac{H_a(s)}{s}\right] \approx \mathcal{Z}\big[TH_a(s)\big]$$

or, in the time domain,

$$h_d(n) \approx T h_a(nT) \qquad \text{(impulse invariance)} \qquad (21.15.3)$$

The same result can also be obtained by approximating the integral in Eq. (21.14.16) as follows,

$$h_d(n) = \int_{nT-T}^{nT} h_a(t')\,dt' \approx T h_a(nT)$$

A slightly better approximation results if we use the trapezoidal approximation to the integral,

$$h_d(n) = \int_{nT-T}^{nT} h_a(t')\,dt' \approx T\,\frac{h_a(nT)+h_a(nT-T)}{2} \qquad (21.15.4)$$

This expression also fixes a small issue with the conventional impulse invariance method (21.15.3) at $n=0$, which gives, assuming a causal $h_a(t)$,

$$h_d(0) = \frac{T}{2}\,h_a(0^+)$$

This leads to a corrected version[†] of the impulse invariance method, for $n \ge 0$,

$$\boxed{h_d(n) \approx T h_a(nT) - \delta[n]\,\frac{T}{2}\,h_a(0^+)} \qquad \text{(impulse invariance)} \qquad (21.15.5)$$

We will not have any further use of the impulse invariance method, since the bilinear and zero-order hold discretization methods are adequate for our purposes in this book.

[†] R. A. Gabel and R. A. Roberts, *Signals and Linear Systems*, 3/e, Wiley, New York, 1987; L. B. Jackson, “A Correction to Impulse Invariance,” *IEEE Signal Processing Letters*, **7**, 273 (2000); W. F. G. Mecklenbräuker, “Remarks on and Correction to the Impulse Invariant Method for the Design of IIR Digital Filters,” *Signal Processing*, **80**, 1687 (2000).

21.16 First-Order Hold

The first-order hold has a discrete-time transfer function similar to Eq. (21.14.1),

$$H_d(z) = \frac{(1-z^{-1})^2}{Tz^{-1}}\cdot \mathcal{Z}\left[\frac{H_a(s)}{s^2}\right] \qquad \text{(first-order hold)} \qquad (21.16.1)$$

It is actually a modified version of the standard first-order hold, called a *triangular hold*, and also used by MATLAB's built-in function **c2d**. Under the same assumptions of distinct poles as for the zero-order hold, the exact equations Eq. (21.14.5)–(21.14.8) are still valid, that is, for $t \ge 0$,

$$\begin{aligned} H_a(s) &= R_0 + \sum_{i=1}^{M}\frac{R_i}{s+p_i}\,, \quad \text{Re}(p_i) > 0 \\ y(t) &= R_0 x(t) + \sum_{i=1}^{M} R_i y_i(t) \end{aligned} \qquad (21.16.2)$$

with the partial output $y_i(t)$ satisfying,

$$y_i(t_n) - e^{-p_iT}y_i(t_{n-1}) = e^{-p_i t_n}\int_{t_{n-1}}^{t_n} e^{p_i t'}x(t')\,dt' \qquad (21.16.3)$$

where in the zero-order hold case, $x(t')$ was approximated by $x(t_{n-1})$ within the interval, $[t_{n-1}, t_n]$. By contrast, the *first-order hold approximation* replaces $x(t')$ by the more accurate approximation of a straight line connecting the points $x(t_{n-1})$ and $x(t_n)$,

$$x(t') \approx x(t_{n-1}) + \frac{x(t_n) - x(t_{n-1})}{T}\,(t' - t_{n-1})\,, \quad t_{n-1} \le t' \le t_n$$

The integral (21.16.3) can then be done exactly, resulting in the following difference equation, and its z-transform,

$$y_i(t_n) - e^{-p_iT}y_i(t_{n-1}) = \frac{1}{Tp_i^2}\Big[(e^{-p_iT} + p_iT - 1)x(t_n) + (1 - e^{-p_iT} - p_iTe^{-p_iT})x(t_{n-1})\Big]$$

$$Y_i(z) = \frac{1}{Tp_i^2}\,\frac{(e^{-p_iT} + p_iT - 1) + (1 - e^{-p_iT} - p_iTe^{-p_iT})z^{-1}}{1 - e^{-p_iT}z^{-1}}\,X(z) \qquad (21.16.4)$$

Thus, the overall output and discrete-time transfer function will be,

$$\begin{aligned} Y(z) &= R_0 X(z) + \sum_{i=1}^{M} R_i Y_i(z) \\ &= R_0 X(z) + \sum_{i=1}^{M}\frac{R_i}{Tp_i^2}\,\frac{(e^{-p_iT} + p_iT - 1) + (1 - e^{-p_iT} - p_iTe^{-p_iT})z^{-1}}{1 - e^{-p_iT}z^{-1}}\,X(z) \\ H_d(z) &= \frac{Y(z)}{X(z)} = R_0 + \sum_{i=1}^{M}\frac{R_i}{Tp_i^2}\,\frac{(e^{-p_iT} + p_iT - 1) + (1 - e^{-p_iT} - p_iTe^{-p_iT})z^{-1}}{1 - e^{-p_iT}z^{-1}}\,, \quad \text{or,} \end{aligned}$$

$$\boxed{H_d(z) = R_0 + \sum_{i=1}^{M}\frac{R_i}{Tp_i^2}\,\frac{(e^{-p_iT} + p_iT - 1) + (1 - e^{-p_iT} - p_iTe^{-p_iT})z^{-1}}{1 - e^{-p_iT}z^{-1}}} \qquad (21.16.5)$$

Next, we show the Eq. (21.16.5) is identical to (21.16.1) under the substitutions of Eq. (21.9.8),

$$\begin{aligned} \frac{1}{s+p_1} &\Rightarrow \frac{1}{1 - e^{-p_1T}z^{-1}} & & & \frac{1}{s} &\Rightarrow \frac{1}{1-z^{-1}} \\ & & &\text{and} & & \\ \frac{1}{(s+p_1)^2} &\Rightarrow \frac{Tz^{-1}}{(1 - e^{-p_1T}z^{-1})^2} & & & \frac{1}{s^2} &\Rightarrow \frac{Tz^{-1}}{(1-z^{-1})^2} \end{aligned} \qquad (21.16.6)$$

For the given $H_a(s)$ in Eq. (21.16.2), we have,

$$\frac{H_a(s)}{s^2} = \frac{R_0}{s^2} + \sum_{i=1}^{M}\frac{R_i}{s^2(s+p_i)} = \frac{R_0}{s^2} + \sum_{i=1}^{M}\frac{R_i}{p_i^2}\left[\frac{1}{s+p_i} - \frac{1}{s} + \frac{p_i}{s^2}\right]$$

and making the substitutions (21.16.6), we find the corresponding z-transform,

$$\mathcal{Z}\left[\frac{H_a(s)}{s^2}\right] = \frac{R_0\,Tz^{-1}}{(1-z^{-1})^2} + \sum_{i=1}^{M}\frac{R_i}{p_i^2}\left[\frac{1}{1 - e^{-p_1T}z^{-1}} - \frac{1}{1-z^{-1}} + \frac{p_iTz^{-1}}{(1-z^{-1})^2}\right]$$

Thus, according to Eq. (21.14.1), we must have,

$$H_d(z) = \frac{(1-z^{-1})^2}{Tz^{-1}}\cdot\mathcal{Z}\left[\frac{H_a(s)}{s^2}\right] = R_0 + \sum_{i=1}^{M}\frac{R_i}{Tp_i^2}\cdot z\cdot\left[\frac{(1-z^{-1})^2}{1 - e^{-p_1T}z^{-1}} - (1-z^{-1}) + p_iTz^{-1}\right]$$

which is easily shown to be identically equal to Eq. (21.16.5).

Zero-Order and First-Order Hold of an Integrator

We note also that the zero-order hold approximation of a simple integrator is equivalent to the forward Euler rule, whereas its first-order hold approximation is equivalent to the trapezoidal rule. Indeed, for the zero-order hold, we substitute, $H_a(s) = 1/s$, for the transfer function of the integrator,

$$H_d(z) = (1-z^{-1})\cdot\mathcal{Z}\left[\frac{H_a(s)}{s}\right] = (1-z^{-1})\cdot\mathcal{Z}\left[\frac{1}{s^2}\right] = (1-z^{-1})\cdot\frac{Tz^{-1}}{(1-z^{-1})^2} = \frac{T}{z-1}$$

while for the first-order hold we have,

$$\begin{aligned} H_d(z) &= \frac{(1-z^{-1})^2}{Tz^{-1}}\cdot\mathcal{Z}\left[\frac{H_a(s)}{s^2}\right] = \frac{(1-z^{-1})^2}{Tz^{-1}}\cdot\mathcal{Z}\left[\frac{1}{s^3}\right] \\ &= \frac{(1-z^{-1})^2}{Tz^{-1}}\cdot\frac{T^2z^{-1}(1+z^{-1})}{2(1-z^{-1})^3} = \frac{T}{2}\,\frac{1+z^{-1}}{1-z^{-1}} \end{aligned}$$

where we used the transformations,

$$\begin{aligned} \frac{1}{s^2} &\xrightarrow{L^{-1}} t\,u(t) \xrightarrow{\text{sample}} nT\,u(nT) \xrightarrow{Z} \mathcal{Z}\left[\frac{1}{s^2}\right] = \frac{Tz^{-1}}{(1-z^{-1})^2} \\ \frac{1}{s^3} &\xrightarrow{L^{-1}} \frac{1}{2}t^2u(t) \xrightarrow{\text{sample}} \frac{1}{2}(nT)^2u(nT) \xrightarrow{Z} \mathcal{Z}\left[\frac{1}{s^3}\right] = \frac{T^2z^{-1}(1+z^{-1})}{2(1-z^{-1})^3} \end{aligned}$$

21.17 Ramp Invariance

We note finally, that Eq. (21.16.1) can be rearranged as follows to obtain the so-called *ramp invariance* property, which states that the time-domain ramp response of the discrete-time system $H_d(z)$ must match the sampled version of the ramp response of the continuous-time system,

$$\boxed{\mathcal{Z}^{-1}\left[\frac{Tz^{-1}}{(1-z^{-1})^2}\,H_d(z)\right] = \mathcal{L}^{-1}\left[\frac{H_a(s)}{s^2}\right]\Bigg|_{\text{sampled}}} \qquad \text{(ramp invariance)} \qquad (21.17.1)$$

This follows by realizing that $1/s^2$ is the Laplace transform of the ramp input, $t\,u(t)$, and that $Tz^{-1}/(1-z^{-1})^2$ is the z-transform of the discrete-time ramp input, $nT\,u(nT)$.

21.18 Appendix

Proof of Eq. (21.2.1) Using Laplace Transforms

We will prove only the second-order case, the other cases being similar. The initial value theorem of Laplace transforms states that for the Laplace transform pair, $y(t) \longleftrightarrow Y(s)$, and initial conditions $y(0^-), \dot{y}(0^-)$, we have the limits,

$$\begin{aligned} y(0^+) &= \lim_{s\to\infty}\big[sY(s)\big] \\ \dot{y}(0^+) &= \lim_{s\to\infty}\Big[s\big[sY(s)-y(0^-)\big]\Big] \end{aligned}$$

where the second one follows from the first by noting that $\big[sY(s)-y(0^-)\big]$ is the Laplace transform of $\dot{y}(t)$. The solution of the differential equation (21.1.3) for an arbitrary causal input $x(t)$, subject to the initial conditions, $y(0^-), \dot{y}(0^-)$, can be expressed in the s-domain in the following form,

$$\ddot{y}(t)+a_1\dot{y}(t)+a_2y(t) = b_0\ddot{x}(t)+b_1\dot{x}(t)+b_2x(t) \quad \Rightarrow$$

$$\big[s^2Y(s)-sy(0^-)-\dot{y}(0^-)\big] + a_1\big[sY(s)-y(0^-)\big] + a_2Y(s) = (b_0s^2+b_1s+b_2)X(s) \quad \Rightarrow$$

$$\begin{aligned} Y(s) &= \frac{(s+a_1)y(0^-)+\dot{y}(0^-)}{s^2+a_1s+a_2} + \left[\frac{b_0s^2+b_1s+b_2}{s^2+a_1s+a_2}\right]X(s) \\ &= \frac{(s+a_1)y(0^-)+\dot{y}(0^-)}{s^2+a_1s+a_2} + \left[b_0 + \frac{(b_1-a_1b_0)s+(b_2-a_2b_0)}{s^2+a_1s+a_2}\right]X(s) \end{aligned}$$

after a long-division step. Thus,

$$Y(s) = \frac{(s+a_1)y(0^-)+\dot{y}(0^-)}{s^2+a_1s+a_2} + b_0X(s) + \left[\frac{(b_1-a_1b_0)s+(b_2-a_2b_0)}{s^2+a_1s+a_2}\right]X(s) \qquad (21.18.1)$$

multiplying by s, we have,

$$sY(s) = \frac{s(s+a_1)y(0^-)+s\dot{y}(0^-)}{s^2+a_1s+a_2} + b_0sX(s) + \left[\frac{(b_1-a_1b_0)s+(b_2-a_2b_0)}{s^2+a_1s+a_2}\right]sX(s)$$

Taking the limit as $s\to\infty$, we have,

$$\begin{aligned} &\lim_{s\to\infty}\big[sY(s)\big] = \\ &= \lim_{s\to\infty}\left[\frac{s(s+a_1)y(0^-)+s\dot{y}(0^-)}{s^2+a_1s+a_2} + b_0sX(s) + \left[\frac{(b_1-a_1b_0)s+(b_2-a_2b_0)}{s^2+a_1s+a_2}\right]sX(s)\right] \\ &= y(0^-)+b_0x(0^+) \end{aligned}$$

where we assumed that the limit $x(0^+) = \lim_{s\to\infty}\big[sX(s)\big]$ exists, and for the same reason, we also dropped the limit of the last term. This verifies the $y(0^+)$ part of Eq. (21.2.1). For the $\dot{y}(0^+)$ part, it follows from Eq. (21.18.1) that,

$$\begin{aligned} &s\big[sY(s)-y(0^-)\big] = \\ &= \frac{s^2\dot{y}(0^-)-sa_2y(0^-)}{s^2+a_1s+a_2} + b_0s^2X(s) + \left[\frac{(b_1-a_1b_0)s^2+(b_2-a_2b_0)s}{s^2+a_1s+a_2}\right]\big[sX(s)\big] \end{aligned}$$

Assuming that the limits, $x(0^+) = \lim_{s\to\infty}\big[sX(s)\big]$, and, $\dot{x}(0^+) = \lim_{s\to\infty}\big[s^2X(s)\big]$, both exist, it follows that the limit of the first term will be $\dot{y}(0^-)$, the limit of the second term, $b_0\dot{x}(0^+)$, and the limit of the third term, $(b_1-a_1b_0)x(0^+)$. Thus, we have,

$$\dot{y}(0^+) = \lim_{s\to\infty}\Big[s\big[sY(s)-y(0^-)\big]\Big] = \dot{y}(0^-)+b_0\dot{x}(0^+)+(b_1-a_1b_0)x(0^+)$$

Proof of Eq. (21.2.1) Using State-Space Realizations

Perhaps the most straightforward proof of Eq. (21.2.1) is by using state-space realizations. For example, for a 3d order filter, the so-called controller/canonical state space form is described in terms of a 3-dimensional state vector, $\mathbf{v}(t) = \big[v_1(t), v_2(t), v_3(t)\big]^T$ that satisfies the matrix equations,

$$\begin{aligned} \frac{d\mathbf{v}(t)}{dt} &= A\mathbf{v}(t)+Bx(t) \\ y(t) &= C\mathbf{v}(t)+Dx(t) \end{aligned} \qquad (21.18.2)$$

where A, B, C, D are defined by,

$$\begin{aligned} &A = \begin{bmatrix} -a_1 & -a_2 & -a_3 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \\ &C = \big[b_1-b_0a_1,\ b_2-b_0a_2,\ b_3-b_0a_3\big], \quad D = b_0 \end{aligned} \qquad (21.18.3)$$

in terms of the transfer function coefficients,

$$H(s) = \frac{b_0s^3+b_1s^2+b_2s+b_3}{s^3+a_1s^2+a_2s+a_3}$$

By integrating Eq. (21.18.2) over the interval $[0^-, 0^+]$ and assuming that $x(t)$ has no delta-function terms, we find that $\mathbf{v}(t)$ is continuous at $t=0$,

$$\mathbf{v}(0^+) = \mathbf{v}(0^-) \qquad (21.18.4)$$

Differentiating $y(t)$ twice, and substituting, $\dot{\mathbf{v}} = A\mathbf{v} + Bx$, we find,

$$
\begin{aligned}
y(t) &= C\mathbf{v}(t) + Dx(t) \\
\dot{y}(t) &= C\dot{\mathbf{v}}(t) + D\dot{x}(t) = CA\mathbf{v}(t) + CBx(t) + D\dot{x}(t) \\
\ddot{y}(t) &= CA\dot{\mathbf{v}}(t) + CB\dot{x}(t) + D\ddot{x} = CA^2\mathbf{v}(t) + CABx(t) + CB\dot{x}(t) + D\ddot{x}(t)
\end{aligned}
$$

Evaluating these across the interval $[0^-, 0^+]$ and using Eq. (21.18.4), and the causality conditions, $x(0^-) = \dot{x}(0^-) = \ddot{x}(0^-) = 0$, we find,

$$
\begin{aligned}
y(0^+) - y(0^-) &= Dx(0^+) \\
\dot{y}(0^+) - \dot{y}(0^-) &= CBx(0^+) + D\dot{x}(0^+) \\
\ddot{y}(0^+) - \ddot{y}(0^-) &= CABx(0^+) + CB\dot{x}(0^+) + D\ddot{x}(0^+)
\end{aligned}
\tag{21.18.5}
$$

Using Eq. (21.18.3), we recognize that these are exactly equivalent to Eq. (21.2.1), indeed,

$$
\begin{aligned}
CAB &= [b_1 - b_0 a_1,\; b_2 - b_0 a_2,\; b_3 - b_0 a_3]\begin{bmatrix} -a_1 & -a_2 & -a_3 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \\
&= b_2 - b_0 a_2 - a_1(b_1 - b_0 a_1) \\
CB &= [b_1 - b_0 a_1,\; b_2 - b_0 a_2,\; b_3 - b_0 a_3]\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = b_1 - b_0 a_1 \\
D &= b_0
\end{aligned}
$$

21.19 MATLAB function – c2d2

The supplied MATLAB function **c2d2** in the ISP2e toolbox is a simple alternative to the built-in function **c2d** for converting a first-order or second-order CT system to a DT system. Its inputs are the analog transfer function coefficients, $\mathbf{B} = [B_0, B_1, B_2]$, $\mathbf{A} = [1, A_1, A_2]$, the time-step T, and the discretization method, and its outputs are the numerator and denominator coefficients of the discrete system, $\mathbf{b} = [b_0, b_1, b_2]$, and, $\mathbf{a} = [1, a_1, a_2]$, for example,

```
B = [2,1,1];
A = [1,4,3];
T = 0.01;

[b,a] = c2d2(B,A,T,'tr');       % trapezoidal (default)
[b,a] = c2d2(B,A,T,'fe');       % forward Euler
[b,a] = c2d2(B,A,T,'be');       % backward Euler
[b,a] = c2d2(B,A,T,'zoh');      % zero-order hold
```

22 Control Systems

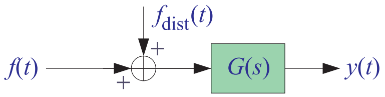
In this chapter, we present a short introduction to feedback control systems [597–609], focusing mainly on PID controllers, which are widely used in industrial applications. We also discuss digital control systems and present several application examples, including a nonliner one.

22.1 Feedback Control Systems

Control systems are used to force the output of a physical system—referred to as the "plant"—to follow a particular prescribed reference input.

For example, in the cruise control of a car, airplane, or ship, the controller forces the car to move at a preset speed, or in the tracking of a target by a rotating radar antenna, the controller generates the appropriate torques to the antenna forcing it to track the moving target, or in the control of a thermostat, the controller turns the furnace on or off for periods of time, so that the temperature follows a prescribed setting.

A physical plant to be controlled can usually be modeled as a continuous-time system with a particular input, $f(t)$, that can be controlled, plus a possible *disturbance* input, $f_{\text{dist}}(t)$, due to various types of disturbances, such as, for example, wind gusts in the cruise control example or in the antenna tracking case. A typical plant is shown below, assuming that it is represented as an LTI system with a transfer function $G(s)$,



(22.1.1)

Control systems are usually implemented in feedback form, feeding back from the plant output $y(t)$ to the plant input $f(t)$ through a controller, winch can be implemented either as an analog or a digital system.

A typical feedback control system is shown in Fig. 22.1.1, in which the output $y(t)$ is fed back and subtracted from a desired *reference* input $r(t)$ that the plant is supposed to follow, and the resulting *error* signal, $e(t) = r(t) - y(t)$, is fed into a controller with transfer function $G_c(s)$, designed to generate the appropriate input $f(t)$ to the plant.

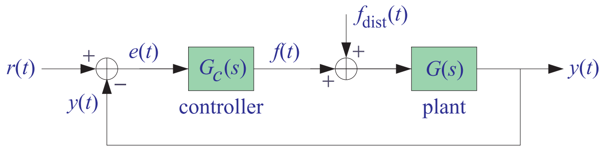


Fig. 22.1.1 Typical feedback control system.

An alternative feedback arrangement, shown in Fig. 22.1.2, is to place the controller in the feedback path, instead of the feed-forward path. Below we will look at examples of both arrangements.

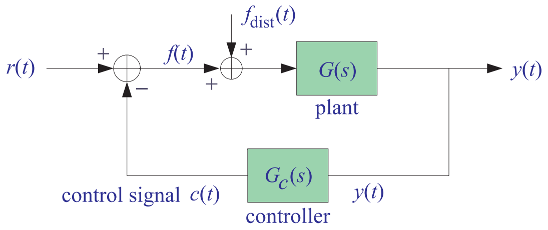


Fig. 22.1.2 Alternative feedback arrangement.

In most of the examples presented in this chapter, we will assume that both the plant $G(s)$ and the controller $G_c(s)$ are LTI systems. For the feedback system shown in Fig. 22.1.1, the so-called *closed-loop transfer function* is the transfer function $H(s)$ from the reference input $r(t)$ to the plant output $y(t)$. There is also a *disturbance transfer function* $H_{\rm dist}(s)$ from the disturbance input, $f_{\rm dist}(t)$, to the output $y(t)$, therefore, because there are two inputs and one output, the overall transfer relationships from the two inputs $R(s), F_{\rm dist}(s)$ to the two outputs $Y(s)$ and error $E(s)$ are as follows:

$$\boxed{\begin{aligned} Y(s) &= H(s)R(s) + H_{\rm dist}(s)F_{\rm dist}(s) \\ E(s) &= H_{\rm err}(s)R(s) - H_{\rm dist}(s)F_{\rm dist}(s) \end{aligned}} \qquad (22.1.2)$$

where, since $E(s) = R(s) - Y(s)$, it is straightforward to show that,

$$\begin{aligned} H(s) &= \frac{G_c(s)G(s)}{1+G_c(s)G(s)} = \text{closed-loop transfer function} \\ H_{\rm err}(s) &= 1 - H(s) = \frac{1}{1+G_c(s)G(s)} = \text{error transfer function} \\ H_{\rm dist}(s) &= \frac{G(s)}{1+G_c(s)G(s)} = \text{disturbance transfer function} \end{aligned} \qquad (22.1.3)$$

22.2 PID Control

There are many choices for the controller transfer function $G_c(s)$. However, for our purposes in this chapter, we will consider the so-called *proportional-integral-derivative* (PID) controller, which is an effective and very widely used industrial controller. Its transfer function has the form.

$$\boxed{G_c(s) = k_p + \frac{k_i}{s} + k_d\, s} \qquad \text{(PID controller)} \qquad (22.2.1)$$

The parameters, k_p, k_i, k_d, are to be chosen to optimize the performance of the control system in terms of its speed of response and its ability to track various types of reference inputs, such as step functions or ramps. Generally, we have following guidance regarding their choices, to be illustrated by examples later on,

- Increasing k_p will decrease the rise time but increase the overshoot.
- Increasing k_i will increase the overshoot and the settling time and decrease the rise time.
- k_i must be nonzero in order to guarantee zero steady-state error, i.e., $e(t)\to 0$, for both step and ramp inputs.
- Increasing k_d will decrease the overshoot and the settling time.

The *steady-state tracking error* due to a particular reference input $r(t)$ can be calculated with the help of the final-value theorem of Laplace transforms, that is,

$$\lim_{t\to\infty} e(t) = \lim_{s\to 0}\big[sE(s)\big] = \lim_{s\to 0}\big[sH_{\rm err}(s)R(s)\big] = \lim_{s\to 0}\left[\frac{sR(s)}{1+G_c(s)G(s)}\right] \qquad (22.2.2)$$

For a step input $r(t) = u(t)$, or a ramp input $r(t) = tu(t)$, we have $R(s) = 1/s$, or $R(s) = 1/s^2$, respectively, and for these Eq. (22.2.2) reads,

$$\text{step input:} \qquad \lim_{t\to\infty} e(t) = \lim_{s\to 0}\left[\frac{sR(s)}{1+G_c(s)G(s)}\right] = \lim_{s\to 0}\left[\frac{s}{s+G(s)\,(k_p s + k_i + k_d s^2)}\right]$$

$$\text{ramp input:} \qquad \lim_{t\to\infty} e(t) = \lim_{s\to 0}\left[\frac{sR(s)}{1+G_c(s)G(s)}\right] = \lim_{s\to 0}\left[\frac{1}{s+G(s)\,(k_p s + k_i + k_d s^2)}\right]$$

Whether the steady-state tracking error $e(t)$ tends to zero depends, of course, on the particular choice of $G(s)$. However, these expressions suggest that at least k_i must be non-zero. This will be observed in some examples below.

22.3 Digital Control Systems

In a digital control system, the controller portion of the feedback system of Fig. 22.1.1 is replaced by a digital transfer function $G_c(z)$, as shown below in Fig. 22.3.1.

The digital controller is operating at a sampling time interval T, or sampling rate $f_s = 1/T$. The sampled output $f(nT)$ of the controller must be passed through an A/D

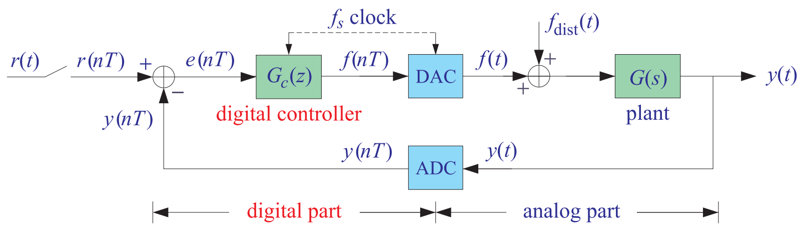


Fig. 22.3.1 Typical digital control system.

converter resulting into a continuous-time signal $f(t)$ that drives the physical plant. In turn, the continuous-time output $y(t)$ of the plant is fed back through a D/A converter operating at the same sampling time interval T, and the resulting sampled signal $y(nT)$ is compared with the sampled version $r(nT)$ of he reference input, generating the sampled input $e(nT)$ to the digital controller.

The A/D converter can be modeled as a zero-order hold transfer function $G_{\rm zoh}(s)$, while the D/A can be replaced by a sampler. By replacing all signals by their sampled versions, including the disturbance signal, one may think of the entire system as discrete-time system. In the Laplace domain, the sampled signals are represented by their corresponding starred Laplace transforms.

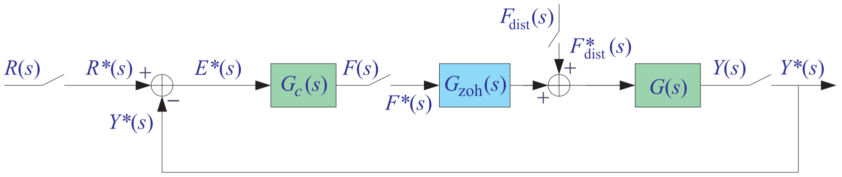


Fig. 22.3.2 Equivalent discrete-time control system.

The overall equivalent discrete-time system is shown in Fig. 22.3.2, in which the digital transfer function has been replaced by a analog one, $G_c(s)$, whose starred Laplace transform is such that, $G_c(z)= G_c^*(s)\big|_{z=e^{sT}}$. The corresponding z-domain transfer functions of the feedback system are obtained as follows:

$$G_{\rm zoh}(s)=\frac{1-e^{-sT}}{s}$$

$$Y(s)=G(s)\big[G_{\rm zoh}(s)F^*(s)+F^*_{\rm dist}(s)\big]$$

$$F(s)=G_c(s)E^*(s)=G_c(s)\big[R^*(s)-Y^*(s)\big]$$

and taking the starred-Laplace transforms,†

$$\begin{aligned}Y^*(s)&=\big[G(s)G_{\rm zoh}(s)\big]^*F^*(s)+G^*(s)F^*_{\rm dist}(s)\\ F^*(s)&=G_c^*(s)\big[R^*(s)-Y^*(s)\big]\end{aligned}$$

† using the properties that, in general, $[G_1G_2]^*\neq G_1^*G_2^*$, but, $[G_1G_2^*]^*=G_1^*G_2^*$

Denoting, $G_d(z)=\big[G(s)G_{\rm zoh}(s)\big]^*$, and, $G(z)=G^*(s)$, $Y(z)=Y^*(s)$, etc., we may rewrite the above as,

$$\begin{aligned}Y(z)&=G_d(z)F(z)+G(z)F_{\rm dist}(z)\\ F(z)&=G_c(z)\big[R(z)-Y(z)\big]\end{aligned}$$

which leads to the feedback discrete-time transfer functions:

$$\boxed{\begin{aligned}Y(z)&=H_d(z)R(z)+H_{\rm dist}(z)F_{\rm dist}(z)\\ E(z)&=H_{\rm err}(z)R(z)-H_{\rm dist}(z)F_{\rm dist}(z)\end{aligned}} \tag{22.3.1}$$

with,

$$\begin{aligned}H_d(z)&=\frac{G_c(z)G_d(z)}{1+G_c(z)G_d(z)}=\text{closed-loop}\\ H_{\rm err}(z)&=\frac{G_c(z)G_d(z)}{1+G_c(z)G_d(z)}=\text{error}\\ H_{\rm dist}(z)&=\frac{G(z)}{1+G_c(z)G_d(z)}=\text{disturbance}\end{aligned} \tag{22.3.2}$$

where the discrete-time transfer functions $G_d(z), G(z)$ are defined by,

$$\begin{aligned}G_d(z)&=\mathcal{Z}\big[G_{\rm zoh}(s)G(s)\big]=\big[G_{\rm zoh}(s)G(s)\big]^*\Big|_{z=e^{sT}}\\ G(z)&=\mathcal{Z}\big[G(s)\big]=G^*(s)\Big|_{z=e^{sT}}\end{aligned} \tag{22.3.3}$$

Eqs. (22.3.2) are the discrete-time versions of Eqs. (22.1.3). The overall equivalent digital control system, incorporating these transfer functions, and operating on the sampled signals, is shown in Fig. 22.3.3.

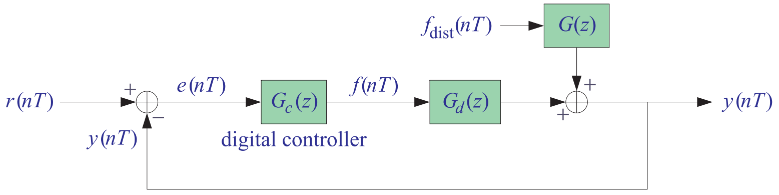


Fig. 22.3.3 Digital control system.

Finally, we note that for *digital PID* control, we may start with a standard analog PID controller $G_c(s)$ and replace the integrator part by the trapezoidal rule, and the differentiation part by the backward Euler rule, that is, we define $G_c(z)$ in this case,

$$\begin{aligned}G_c(s)&=k_p+\frac{k_i}{s}+k_d\,s=\text{analog PID controller}\\ &\Downarrow\\ G_c(z)&=k_p+\frac{k_iT}{2}\left(\frac{1+z^{-1}}{1-z^{-1}}\right)+\frac{k_d}{T}(1-z^{-1})=\text{digital PID controller}\end{aligned} \tag{22.3.4}$$

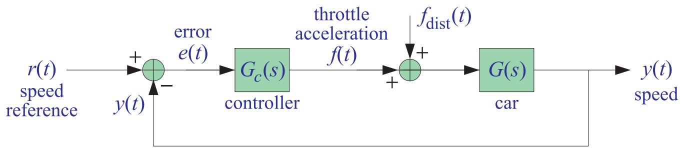
22.4 Examples

22.4.1 Cruise Control

A car's motion in the presence of linear drag and under the influence of the engine's accelerating force $F(t)$, or acceleration, $f(t) = F(t)/m$, is described by the first-order linear system for the velocity $v(t)$, and the corresponding transfer function $G(s)$, where the term, $\alpha v(t)$, represents a drag force taken to be proportional to the velocity, and α is a drag coefficient,

$$\boxed{\frac{dv(t)}{dt} = f(t) - \alpha v(t)} \quad \Rightarrow \quad \boxed{G(s) = \frac{V(s)}{F(s)} = \frac{1}{s+\alpha}} \tag{22.4.1}$$

In a cruise control system, depicted below, a controlling system generates the appropriate throttle/acceleration input signal $f(t)$ to the car's dynamics that causes the car to reach a prescribed reference speed, as set by the driver. The actual speed, denoted in the figure by $y(t)$, is fed back and subtracted from the desired reference speed $r(t)$, and the resulting error signal, $e(t) = r(t) - y(t)$, is used by the controller to generate the appropriate acceleration $f(t)$.



We saw that the *closed-loop transfer function* for such system, from the overall reference input $r(t)$ to the final output $y(t)$, and ignoring the disturbance input, is given in terms of the car's and controller's transfer functions $G(s)$ and $G_c(s)$ by,

$$H(s) = \frac{Y(s)}{R(s)} = \frac{G_c(s)G(s)}{1 + G_c(s)G(s)} \tag{22.4.2}$$

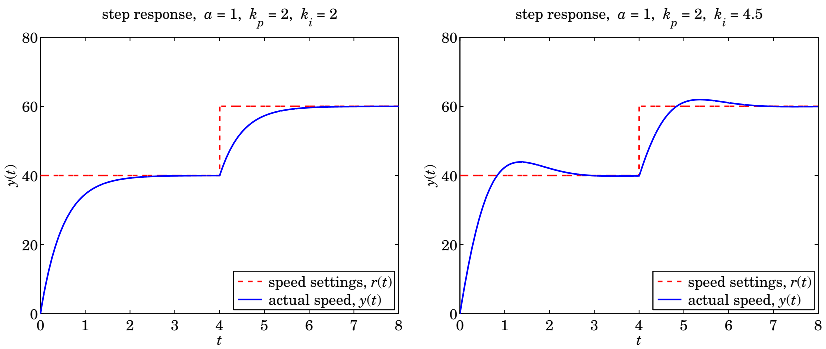
Instead of using a full PID controller of the form,

$$G_c(s) = k_p + \frac{k_i}{s} + k_d s \qquad \text{(PID controller)} \tag{22.4.3}$$

we will use a PI controller defined by,

$$G_c(s) = k_p + \frac{k_i}{s} \qquad \text{(PI controller)} \tag{22.4.4}$$

The figure below shows the closed-loop responses when the reference speed is set to 40 mph for a period $0 \le t \le 4$, and then is reset to a new value of 60 mph when $t > 4$, for the two sets of PI parameters, $k_p = 2\,, k_i = 2$, and, $k_p = 2\;\; k_i = 4.5$.



In both cases, the output reaches the desired reference values after the transients caused by the sudden changes in the reference settings die out. In the right graph, the speed of response is shorter but at the expense of an overshoot.

Typically, increasing the PI parameter k_i decreases the rise time but increases the overshoot. Inserting Eqs. (22.4.1) and (22.4.4) into (22.4.2), we find the closed-loop transfer function, as well as the error transfer function,

$$\begin{aligned} H(s) &= \frac{G_c(s)G(s)}{1 + G_c(s)G(s)} = \frac{k_p s + k_i}{s^2 + (k_p + \alpha)s + k_i} \\ H_{\text{err}}(s) &= 1 - H(s) = \frac{1}{1 + G_c(s)G(s)} = \frac{s(s+\alpha)}{s^2 + (k_p + \alpha)s + k_i} \end{aligned} \tag{22.4.5}$$

The unit-step response is obtained by setting $r(t) = u(t)$, or, $R(s) = 1/s$, and computing an inverse Laplace transform with the help of partial fraction expansions,

$$Y(s) = H(s)R(s) = \frac{H(s)}{s} \quad \Rightarrow \quad y(t) = \text{ilaplace}\big(Y(s)\big)$$

For the two cases in the above figure, we find for, $k_p = 2\,,\; k_i = 2$,

$$\begin{aligned} H(s) &= \frac{2s+2}{s^2+3s+2} = \frac{2(s+1)}{(s+1)(s+2)} = \frac{2}{s+2} \\ Y(s) &= \frac{H(s)}{s} = \frac{2}{s(s+2} = \frac{1}{s} - \frac{1}{s+2} \\ y(t) &= 1 - e^{-2t}\,, \quad t \ge 0 \end{aligned}$$

while for, $k_p = 2\,,\; k_i = 4.5$, we have,

$$\begin{aligned} H(s) &= \frac{2s+4.5}{s^2+3s+4.5} \\ Y(s) &= \frac{H(s)}{s} = \frac{1}{s} - \frac{s+1}{s^2+3s+4.5} = \frac{1}{s} - \left[\frac{\frac{1}{6}(3-j)}{s+1.5+1.5j} + \frac{\frac{1}{6}(3+j)}{s+1.5-1.5j}\right] \\ y(t) &= 1 - 2\,\text{Re}\left[\frac{1}{6}(3-j)\,e^{-1.5t}\,e^{-1.5jt}\right] = 1 - \frac{1}{3}e^{-1.5t}\big[3\cos(1.5t) - \sin(1.5t)\big] \end{aligned}$$

The MATLAB code used to generate the above graphs was as follows,

```
a = 1; kp = 2; ki = 4.5;      % for left graph, use ki = 2

t = linspace(0,8,2401);       % time range

s = tf('s');                  % tf class
G = 1/(s+a);                  % car's transfer function
Gc = kp + ki/s;               % PI controller

H = minreal(Gc*G/(1+Gc*G));   % closed-loop, minreal() removes common factors
% H = feedback(Gc*G,1);       % alternative construction of H

T1 = 4;                       % switch time
v1 = 40; v2 = 60;             % reference speeds

r = v1*(t<=T1) + v2*(t>T1);   % reference input r(t)

y = lsim(H, r, t);            % computed output y(t)

figure; plot(t,r,'r--', t,y,'b-');
legend(' speed settings, r(t)', ' actual speed, y(t)', 'location','se')
```

We note also that the final steady-state error for the step response is zero for both choices of k_i, indeed, we have from Eq. (22.2.2), with $R(s) = 1/s$,

$$\lim_{t\to\infty} e(t) = \lim_{s\to 0}\big[sH_{\rm err}(s)R(s)\big] = \lim_{s\to 0}\left[\frac{s(s+\alpha)}{s^2+(k_p+\alpha)s+k_i}\right] =$$

22.4.2 Radar Tracking Antenna

Consider a dish antenna sitting on a rotating base that can be rotated azimuthally by a drive motor to track a flying aircraft. The dynamics of the rotating structure is described by the equations:

$$J\ddot{\theta}(t) = -\beta\dot{\theta}(t) + N(t) + N_{\rm dist}(t)$$

where $\theta(t)$ is the azimuthal angle, $N(t)$ is the torque applied by the drive motor, $N_{\rm dist}(t)$ is a torque due to disturbances such as wind gusts or steady wind noise, J is the moment of inertia of the structure, and β is a frictional constant that quantifies an opposing frictional torque that is proportional to the angular velocity $\dot{\theta}$.

It is desired to design a control system that generates an appropriate torque $N(t)$ such that the angle $\theta(t)$ will follow a desired reference angle $\theta_{\rm ref}(t)$, that is,

$$\theta(t) \longrightarrow \theta_{\rm ref}(t)$$

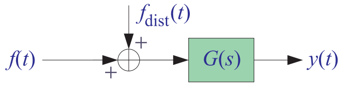
For example, if one wishes to point the antenna towards a given fixed angle θ_1, then, $\theta_{\rm ref}(t) = \theta_1 u(t)$. To point initially towards θ_1 and t_0 seconds later to point towards θ_2, one would choose, $\theta_{\rm ref}(t) = \theta_1 u(t) + (\theta_2 - \theta_1)u(t-t_0)$.

Similarly, to track a uniformly moving aircraft, one would choose the ramp function $\theta_{\rm ref}(t) = \omega_0 t u(t)$, or, more correctly, $\theta_{\rm ref}(t) = \arctan(\omega_0 t)u(t)$.

By some redefinitions, the above system can be replaced by the following standardized form where the output $y(t)$ represents $\theta(t)$, and $f(t), f_{\rm dist}(t)$ represent the torque inputs $N(t), N_{\rm dist}(t)$,

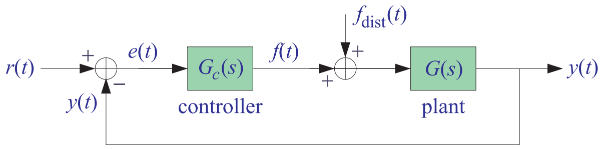
$$\ddot{y}(t) = -a\dot{y}(t) + f(t) + f_{\rm dist}(t) \quad\Leftrightarrow\quad Y(s) = G(s)\big[F(s) + F_{\rm dist}(s)\big] \qquad (22.4.6)$$

where the system transfer function is,



$$\boxed{G(s) = \frac{1}{s(s+a)}} \qquad (22.4.7)$$

The control system is implemented as the feedback system shown below,



where the overall reference input $r(t)$ represents the desired reference angle $\theta_{\rm ref}(t)$, and the controller $G_c(s)$ is designed to generate the appropriate torque input $f(t)$ to make the system follow the reference input, i.e., $y(t)\to r(t)$, or for the tracking error signal, $e(t) = r(t) - y(t) \to 0$.

In this example, we design a PID controller and experiment with its settings, and also investigate its tracking ability and its robustness in the presence of disturbance inputs. The PID controller has the transfer function:

$$\boxed{G_c(s) = k_p + \frac{k_i}{s} + k_d s} \quad \text{(PID controller)} \qquad (22.4.8)$$

Following the discussion of Sec. 22.1, we can show that the overall transfer relationships from the two inputs $R(s)$, $F_{\rm dist}(s)$ to the two outputs $Y(s)$ and $E(s)$ are as follows:

$$\boxed{\begin{aligned} Y(s) &= H(s)R(s) + H_{\rm dist}(s)F_{\rm dist}(s) \\ E(s) &= H_{\rm err}(s)R(s) - H_{\rm dist}(s)F_{\rm dist}(s) \end{aligned}} \qquad (22.4.9)$$

where for the PID case in particular,

$$\boxed{\begin{aligned} H(s) &= \frac{G_c(s)G(s)}{1+G_c(s)G(s)} = \frac{k_d s^2 + k_p s + k_i}{s^3 + (a+k_d)s^2 + k_p s + k_i} = \text{closed-loop} \\ H_{\rm err}(s) &= \frac{1}{1+G_c(s)G(s)} = \frac{s^2(s+a)}{s^3 + (a+k_d)s^2 + k_p s + k_i} = \text{error} \\ H_{\rm dist}(s) &= \frac{G(s)}{1+G_c(s)G(s)} = \frac{s}{s^3 + (a+k_d)s^2 + k_p s + k_i} = \text{disturbance} \end{aligned}} \qquad (22.4.10)$$

The *steady-state tracking error* due to a particular reference input $r(t)$ is found to be,

$$\lim_{t\to\infty} e(t) = \lim_{s\to 0}\big[sH_{\rm err}(s)R(s)\big] = \lim_{s\to 0}\left[\frac{s^3(s+a)R(s)}{s^3 + (a+k_d)s^2 + k_p s + k_i}\right] \qquad (22.4.11)$$

For a step input, $r(t) = u(t)$, or for a ramp input, $r(t) = tu(t)$, we have, $R(s) = 1/s$, or, $R(s) = 1/s^2$, respectively, and for these choices, Eq. (22.4.11) implies that the tracking error will be zero provided $k_i \neq 0$.

The corresponding digital control system, depicted in Figs. 22.3.1–22.3.3, is characterized by the discrete-time transfer functions, $G_d(z), G(z)$, defined in Eq. (22.3.3), which turn out to be in the present example,

$$\begin{aligned} G_d(z) &= \mathcal{Z}\big[G_{\text{zoh}}(s)G(s)\big] = \frac{(aT + e^{-aT} - 1)z^{-1} + (1 - e^{-aT} - aTe^{-aT})z^{-2}}{a^2(1 - z^{-1})(1 - e^{-aT}z^{-1})} \\ G(z) &= \mathcal{Z}\big[G(s)\big] = \frac{(1 - e^{-aT})z^{-1}}{a(1 - z^{-1})(1 - e^{-aT}z^{-1})} \end{aligned} \tag{22.4.12}$$

and for $G_c(z)$, we may use the digital PID approximation given by Eq. (22.3.4),

$$G_c(z) = k_p + \frac{k_i T}{2}\left(\frac{1 + z^{-1}}{1 - z^{-1}}\right) + \frac{k_d}{T}(1 - z^{-1}) = \text{discrete PID controller} \tag{22.4.13}$$

Computer Experiments

With the above background information, please carry out the following experiments.

(a) Starting with the parameter values,

$$a = 2, \quad k_p = 10, \quad k_i = 5, \quad k_d = 3 \tag{22.4.14}$$

construct the *transfer function objects* for the system $G(s)$, controller $G_c(s)$, closed-loop feedback system $H(s)$, tracking error $H_{\text{err}}(s)$, and disturbance transfer function $H_{\text{dist}}(s)$, using, for example, the MATLAB code:

```
a = 2; kp = 10; ki = 5; kd = 3;
s = tf(’s’);
G = 1/(s*(s+a));
Gc = kp + ki/s + kd*s;
H = minreal(Gc*G/(1+Gc*G));
Herr = minreal(1/(1+Gc*G));
Hdist = minreal(G/(1+Gc*G));
```

where the **minreal** function removes any possible common factors from the numerator and denominator transfer functions, resulting in a minimal realization—this happens for example in the case $k_i = 0$ in which some s factors can be canceled.

First, determine the *poles* of the closed-loop transfer function $H(s)$ and from the pole lying closest to the imaginary axis on the s-plane, calculate the 40-dB *time constant* of $H(s)$. Note that the poles can be determined by using the function **roots** or **pzmap**, e.g.,

```
p = roots(H.den{1});  % H.den{1} is the vector of denominator coefficients of H(s)
p = pzmap(H);
```

Next, define t as a vector of 1001 equally-spaced time samples spanning the interval $0 \le t \le 20$. Using the **lsim** function, *calculate and plot* the unit-step response of $H(s)$ over this time range. Is the observed transient time consistent with the 40-dB time constant? You may find it useful to define the unit-step function as,

```
u = @(t) double(t>=0);
```

Then, increase the PID parameters by doubling their values one at a time, and *plot* the corresponding step responses, and comment on the effect of such changes.

(b) For the parameter values defined in Eq. (22.4.14), and for the same time range as in part (a), generate the following four reference input signals describing the typical reference angle situations mentioned in the introduction.

$$\begin{aligned} r(t) &= u(t) + u(t-10) && \text{(switches from } r = 1 \text{ to } r = 2 \text{ at } t = 10\text{)} \\ r(t) &= 0.1\,t\,u(t) && \text{(uniformly moving aircraft)} \\ r(t) &= \arctan(0.1\,t)\,u(t) && \text{(uniformly moving with correct angle)} \\ r(t) &= \begin{cases} 0.04\,t\,, & 0 \le t \le 10 \\ -2 + 0.69\,t - 0.07\,t^2 + 0.0025\,t^3\,, & 10 \le t \le 14 \\ 0.8 + 0.2\,(t - 14)\,, & 14 \le t \le 20 \end{cases} && \text{(accelerating)} \end{aligned} \tag{22.4.15}$$

The fourth case, emulates a situation where the aircraft is moving at constant speed until $t = 10$ and then between $t = 10$ and $t = 14$, it accelerates to a new speed. The expression between $10 \le t \le 14$ is the cubic Hermite interpolation polynomial that interpolates smoothly between the two speeds.

For each of the four $r(t)$ inputs, compute the corresponding output $y(t)$ of the closed-loop system H, using the function **lsim**, as follows

```
y = lsim(H,r,t);
```

On the same graph, plot both $y(t)$ and $r(t)$ with different linestyles, observing whether the controlled system is capable of following the desired input reference setting. On a separate graph, plot the tracking error signal $e(t)$ versus t.

For the particular case of the ramp input, $r(t) = 0.1\,t\,u(t)$, set temporarily $k_i = 0$, and recompute and plot the system output $y(t)$ and error $e(t)$, noting that the steady-state error $e(t)$ is no longer zero, although the slope of the output does follow the slope of the reference input. After this part, set k_i back to its non-zero value.

(c) Because of the difficulty in implementing the derivative term $k_d s$ of the PID controller, the following modified variant is often used:

$$G_c(s) = k_p + \frac{k_i}{s} + \frac{k_d s}{\tau s + 1} \tag{22.4.16}$$

where τ is a very small quantity. In this case, show that the closed-loop transfer function becomes,

$$H(s) = \frac{(k_d + \tau k_p)s^2 + (k_p + \tau k_i)s + k_i}{\tau s^4 + (\tau a + 1)s^3 + (a + k_d + \tau k_p)s^2 + (k_p + \tau k_i)s + k_i} \tag{22.4.17}$$

Set $\tau = 0.05$ and use the PID parameters of Eq. (22.4.14).

Determine the transfer function $H_f(s)$ from the overall input $r(t)$ to the controller's torque output $f(t)$ and for all four choices of $r(t)$ of Eq. (22.4.15), compute the torque $f(t)$ and plot it versus t.[†] This will give you a sense of the actual input being applied to the controlled system $G(s)$ that causes it to follow the reference input $r(t)$. Set $\tau = 0$ after this part is complete.

(d) Here, you will investigate how the controlled system responds to a disturbance. Consider two types of disturbances, one imitating a wind gust lasting for a brief period of time, say, $4 \le t \le 6$, and the other imitating steady wind noise. They can be generated by the following MATLAB code, for the same length-1001 vector of t's that you defined in part (a),

```
fdist = 2*(u(t-4)-u(t-6));         % wind gust

                                   % wind noise
seed=2016; rng(seed);              % initialize random number generator
fdist = randn(size(t));            % zero-mean, unit-variance noise
```

For each type of disturbance, compute the corresponding system output using the disturbance transfer function H_{dist}, and add it to the previously obtained output from each of the four reference signals $r(t)$ to get the total system output:

```
ydist = lsim(Hdist,fdist,t);
y = lsim(H,r,t);
ytot = y + ydist;
```

For each of the resulting eight cases (2 disturbances × 4 reference signals), plot the signals $y_{\text{tot}}(t)$ and $r(t)$ on the same graph, observing how the system recovers (or not) from the disturbance.

(e) Next, you will study the behavior of the discrete PID control system, described by Eqs. (22.4.10) – (22.4.13). For this part, you may ignore the disturbance input. A reasonable initial choice for the discretization sampling time interval T_s is to choose it to be a small fraction of the effective time constant of the closed-loop system H. The time constant is the inverse of the smallest damping constant and can be obtained with the help of the function **pzmap**:

```
p = pzmap(H);                    % poles of H
teff = 1/abs(max(real(p)));      % effective time constant
Ts = teff/20;                    % initial choice of Ts
```

The zero-order hold discretization of the system $G(s)$ is given by Eq. (22.4.12), but it can also be obtained using the **c2d** function:

```
T = Ts;            % to be changed later to T = Ts/2, 2*Ts, 3*Ts
Gd = c2d(G,T);     % ZOH discretization by default
```

[†] MATLAB will complain if you tried to do this part with $\tau = 0$.

The discrete PID controller $G_c(z)$ of Eq. (22.4.13) and the discrete feedback transfer function $H_d(z)$ can be constructed by the code:

```
z = tf('z');
Gc = kp + ki*T*(z+1)/(z-1)/2 + kd*(z-1)/z/T;
Hd = feedback(Gc*Gd,1);
```

The time vector t must now be resampled at multiples of the chosen interval T, that is, $t_n = nT$, and in order for it to span the interval $0 \le t \le 20$, we must redefine:

```
tn = 0:T:20;
```

Using this new vector of t's, construct the discrete-time reference inputs:

$$r_n = u(t_n) + 2u(t_n - 10)$$

$$r_n = 0.1\, t_n$$

$$r_n = \arctan(0.1\, t_n)$$

and compute the output y_n of the discrete closed-loop system:

```
yn = lsim(Hd,rn);
% yn = filter(Hd.num{1}, Hd.den{1}, rn);   % alternative evaluation of yn
```

On the same graph, plot y_n versus the sampled time t_n, together with the output $y(t)$ of the continuous-time system computed in the previous parts using the original length-1001 time vector t, that is,

```
figure; plot(tn,yn,'r-', t,y,'b-');
```

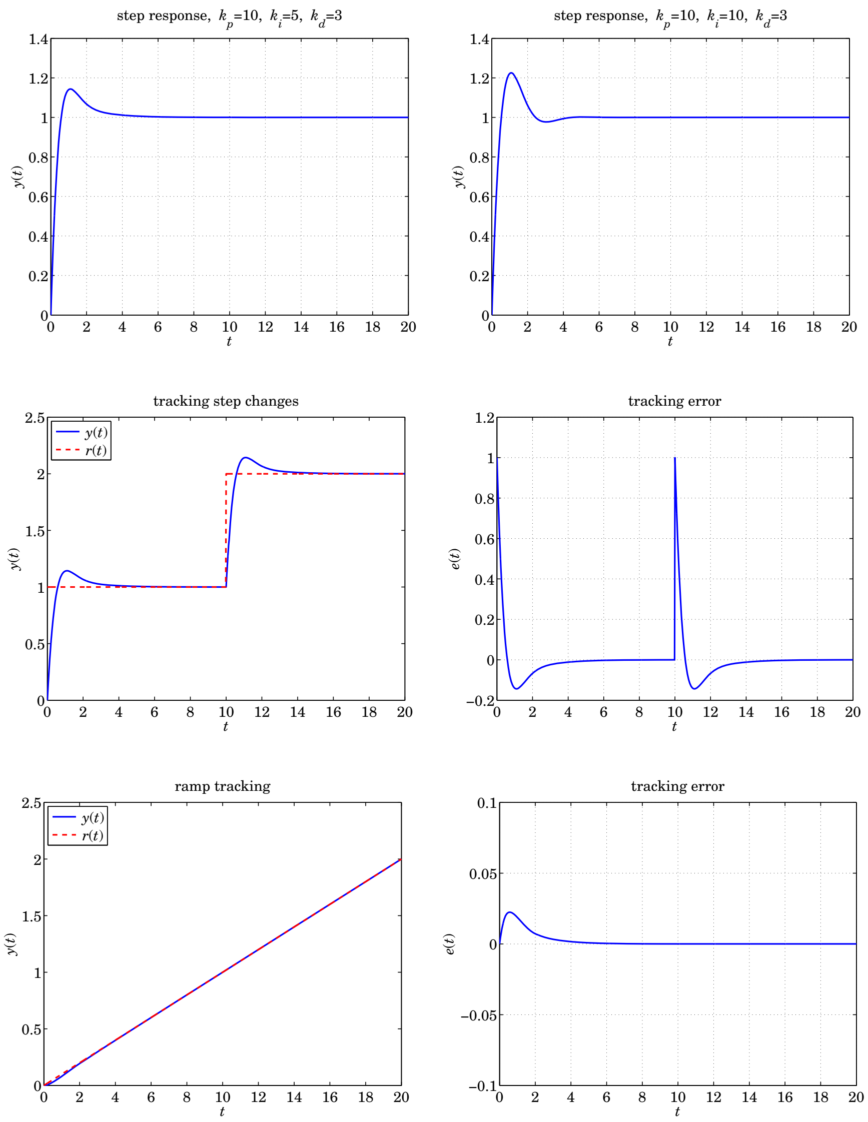
Compare the outputs of the discrete and continuous time systems.

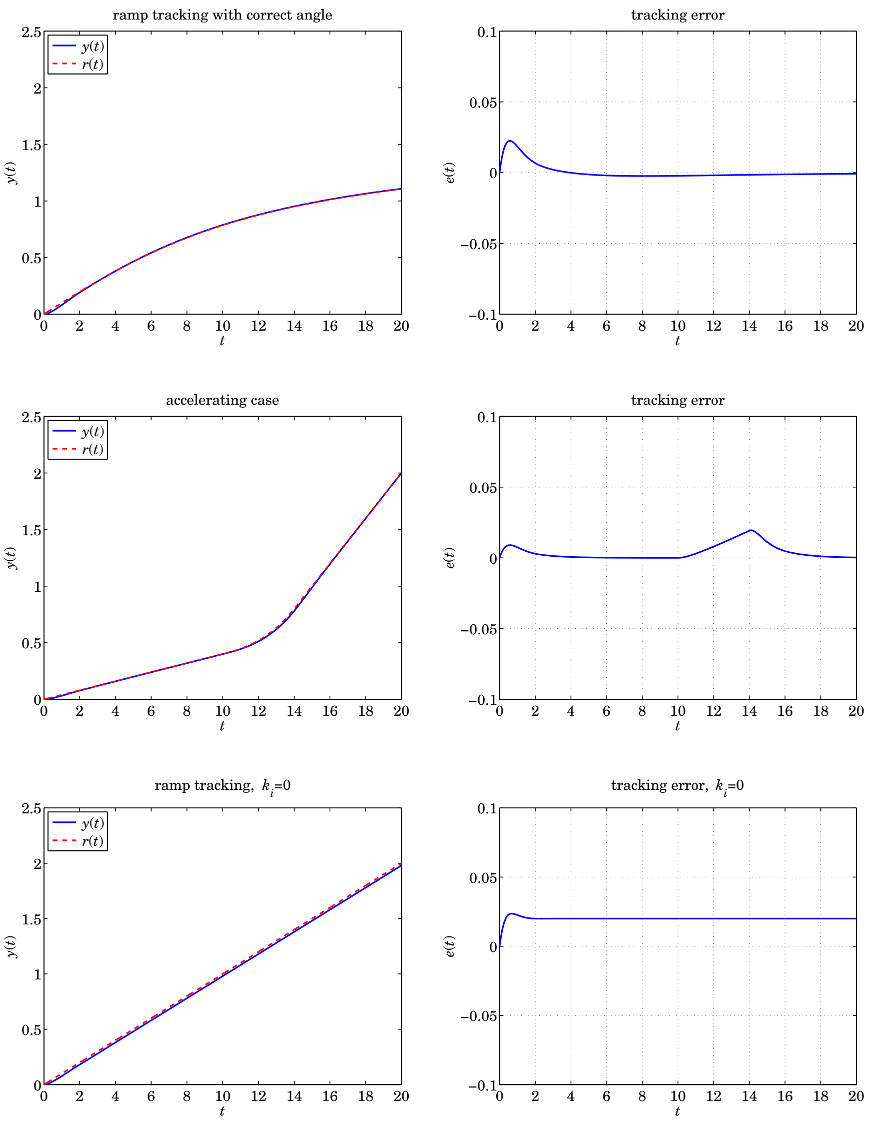
(f) Repeat part (e) using the alternative choices of the interval T:

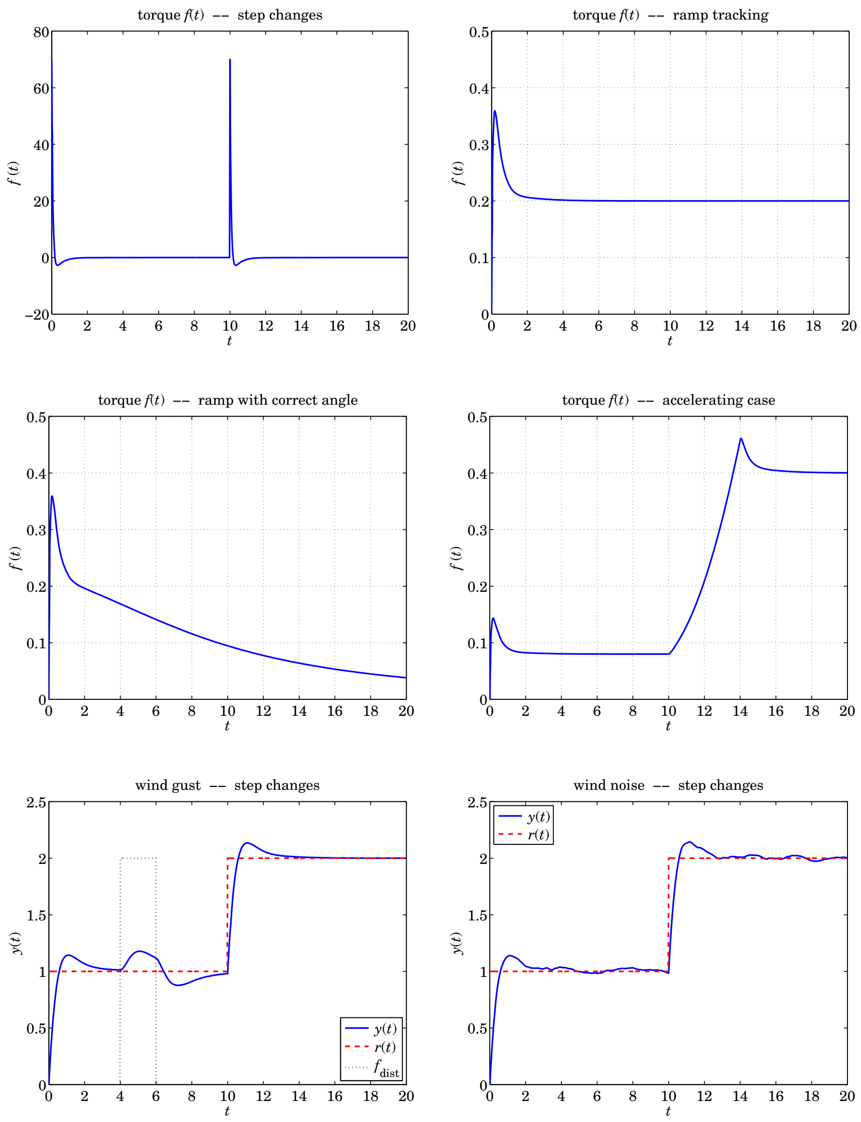
$$T = \frac{1}{2}T_s\,, \quad T = 2T_s\,, \quad T = 3T_s$$

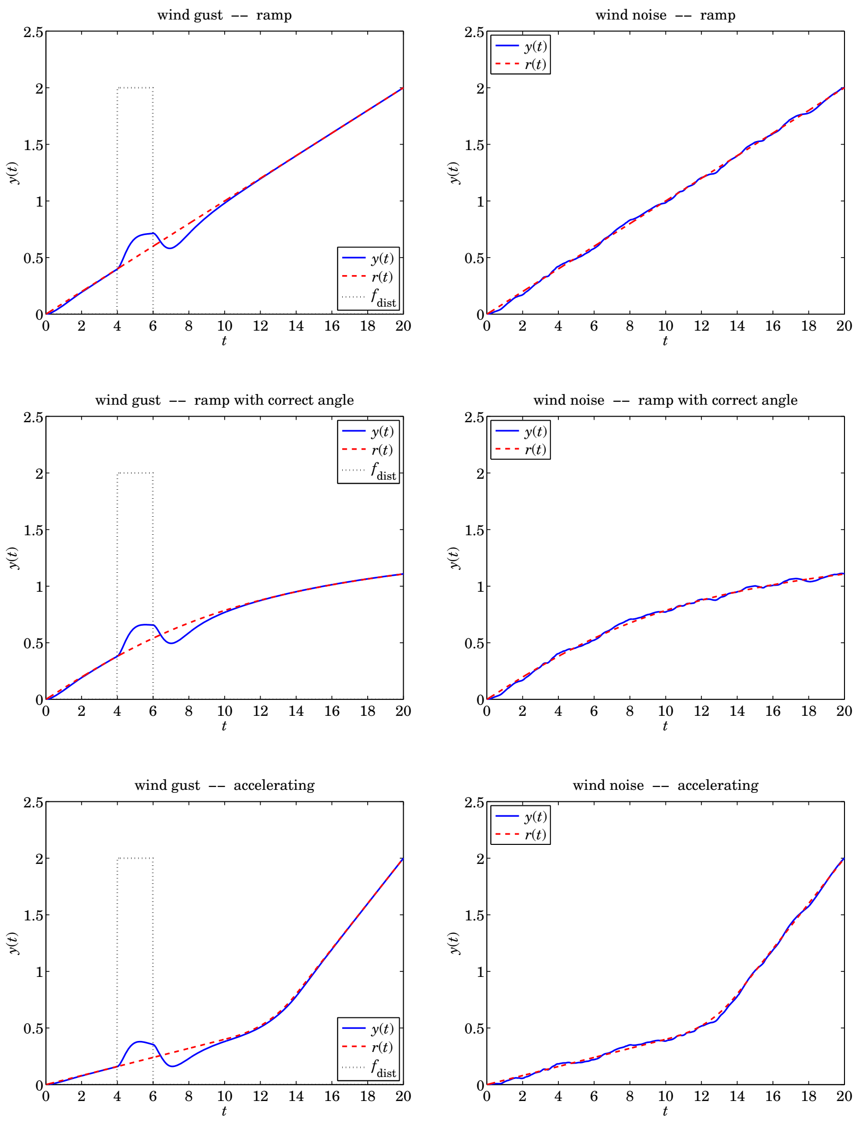
And discuss the improvement or deterioration of the expected response.

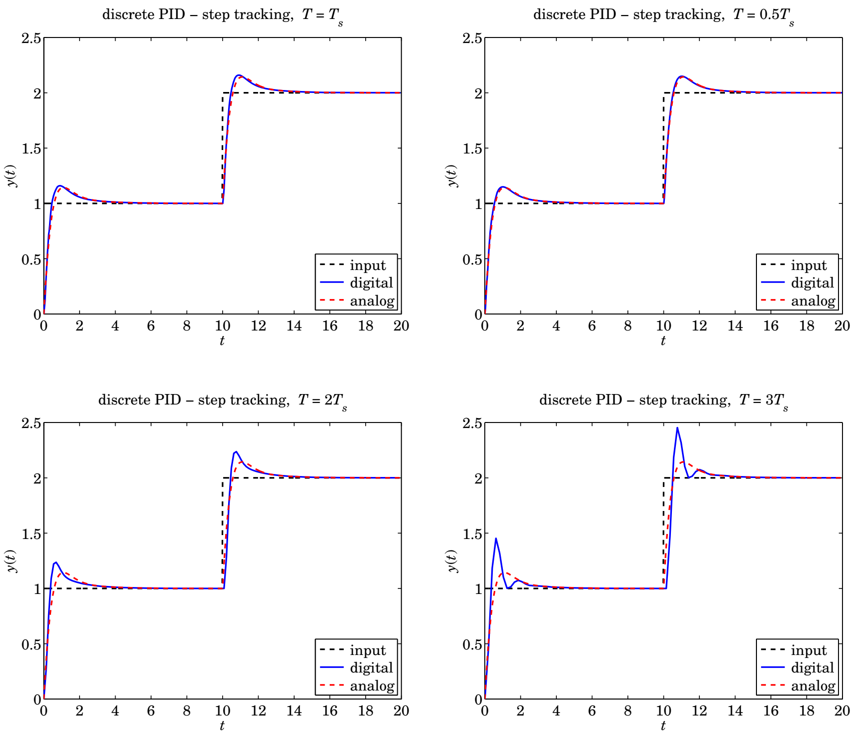
Typical Graphs






torque $f(t)$ -- step changes
torque $f(t)$ -- ramp tracking
torque $f(t)$ -- ramp with correct angle
torque $f(t)$ -- accelerating case
wind gust -- step changes
wind noise -- step changes
$f(t)$
$y(t)$
t
$y(t)$
$r(t)$
f_{dist}
80
60
40
20
0
−20
0.5
0.4
0.3
0.2
0.1
2.5
2
1.5
1
0
2
4
6
8
10
12
14
16
18
20

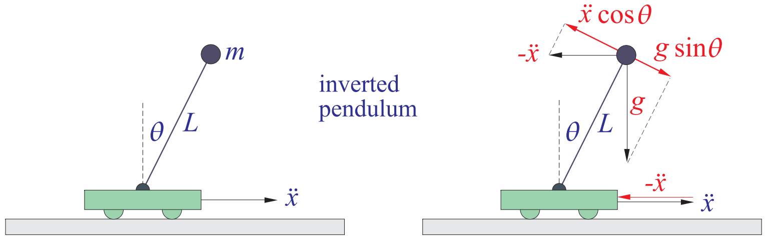

wind gust -- ramp
wind noise -- ramp
wind gust -- ramp with correct angle
wind noise -- ramp with correct angle
wind gust -- accelerating
wind noise -- accelerating
$y(t)$
t
$y(t)$
$r(t)$
f_{dist}
2.5
2
1.5
1
0.5
0
0
2
4
6
8
10
12
14
16
18
20



22.4.3 Inverted Pendulum

The inverted pendulum provides a simple model for many applications, such as walking and standing of humans and upright animals, walking robots, rockets during liftoff, stabilizing tall buildings, transporting large vertical objects, and the Segway self-balancing two-wheeled vehicle [610–619].

The inverted pendulum, depicted below, consists of a mass m concentrated at the end of a (weightless) rod of length L whose other end is connected to a frictionless pivot hinge on a cart moving with acceleration $\ddot{x}(t)$. By controlling $\ddot{x}(t)$, the pendulum can remain stable in its upside-down position.



The picture on the right shows the pendulum in a coordinate frame in which the cart is at rest and the mass is subjected to the negative acceleration, $-\ddot{x}(t)$. That acceleration together with gravity g, projected along a direction perpendicular to the rod, induce a net torque, $\tau = mL(g\sin\theta - \ddot{x}\cos\theta)$, that tends to rotate the pendulum. The resulting equation of motion is,

$$mL^2\ddot{\theta} = \tau = mL(g\sin\theta - \ddot{x}\cos\theta) \quad\Rightarrow\quad \ddot{\theta} = \frac{g}{L}\sin\theta - \frac{\ddot{x}}{L}\cos\theta \qquad (22.4.18)$$

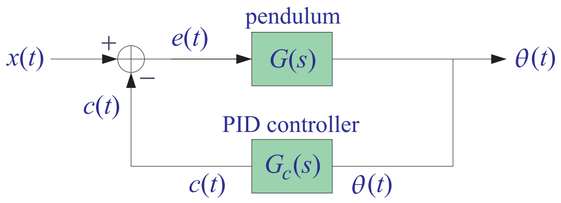
For small deviations θ from the vertical, Eq. (22.4.18) can be linearized by making the small-angle approximations, $\sin\theta \approx \theta$, and, $\cos\theta \approx 1 - \frac{1}{2}\theta^2 \approx 1$, resulting in,

$$\ddot{\theta} = \omega_0^2\,\theta - \frac{\ddot{x}}{L}\,, \qquad \omega_0 = \sqrt{\frac{g}{L}} \qquad (22.4.19)$$

where ω_0 is the natural frequency of oscillation of a pendulum in its stable hanging-down position. In the absence of $\ddot{x}$, Eq. (22.4.19) is unstable since its solution is a linear combination of the normal modes, $e^{\omega_0 t}, e^{-\omega_0 t}$, the first of which is unstable for $t \to +\infty$. The transfer function of the LTI system (22.4.19) from the input $x(t)$ to the output $\theta(t)$, is obtained using Laplace transforms,

$$x(t) \longrightarrow \boxed{G(s)} \longrightarrow \theta(t) \qquad \boxed{G(s) = -\frac{s^2/L}{s^2 - \omega_0^2}} \qquad (22.4.20)$$

The instability of the system is also evident from Eq. (22.4.20) by noting that $G(s)$ has poles at $s = \omega_0$ and $s = -\omega_0$, the first of which lies in the right-half s-plane. The system can be stabilized by feeding back the output signal $\theta(t)$ through a properly chosen controller $G_c(s)$, as shown below.



In this example, we will use a PID controller of the following PI form [597], where L is a common factor introduced for convenience that multiplies both k_p, k_i,

$$\boxed{G_c(s) = k_p L + \frac{k_i L}{s}} \qquad (22.4.21)$$

The closed-loop transfer function from $x(t)$ to $\theta(t)$, as well as the transfer function from $x(t)$ to the pendulum's effective input $e(t)$, are given by,

$$\begin{aligned} H(s) &= \frac{\Theta(s)}{X(s)} = \frac{G(s)}{1 + G_c(s)G(s)} = \frac{s^2/L}{(k_p - 1)s^2 + k_i s + \omega_0^2} \\ H_e(s) &= \frac{E(s)}{X(s)} = \frac{1}{1 + G_c(s)G(s)} = -\frac{s^2 - \omega_0^2}{(k_p - 1)s^2 + k_i s + \omega_0^2} \end{aligned} \qquad (22.4.22)$$

By proper selection of the PI gains k_p, k_i, the above transfer functions can stabilized with their poles lying strictly in the left-hand s-plane. For example, in order to place the closed-loop poles at the negative-real locations, $s = -\alpha_1$ and $s = -\alpha_2$, with α_1, α_2 arbitrary positive numbers, one can determine the gains k_p, k_i by requiring the identity in s,

$$(k_p - 1)s^2 + k_i s + \omega_0^2 \equiv (k_p - 1)(s + \alpha_1)(s + \alpha_2)$$

which leads to the following solution for k_p, k_i in terms of the given α_1, α_2,

$$\boxed{\begin{aligned} k_p &= 1 + \frac{\omega_0^2}{\alpha_1 \alpha_2} \\ k_i &= \omega_0^2 \left(\frac{1}{\alpha_1} + \frac{1}{\alpha_2}\right) \end{aligned}} \qquad (22.4.23)$$

With these choices for k_p, k_i, the transfer functions (22.4.22) read,

$$\boxed{\begin{aligned} H(s) &= \frac{\alpha_1 \alpha_2}{L\omega_0^2} \, \frac{s^2}{(s + \alpha_1)(s + \alpha_2)} \\ H_e(s) &= -\frac{\alpha_1 \alpha_2}{\omega_0^2} \, \frac{s^2 - \omega_0^2}{(s + \alpha_1)(s + \alpha_2)} \end{aligned}} \qquad (22.4.24)$$

Computer Experiments

(a) Assuming $\alpha_1 \neq \alpha_2$, show that the angle response $\theta(t)$, and the effective input $e(t)$, due to a sudden unit-step shift in position, $x(t) = x_0 u(t)$, are given as follows, for $t \geq 0$,

$$\begin{aligned} \theta(t) &= \frac{x_0 \alpha_1 \alpha_2}{L\omega_0^2(\alpha_1 - \alpha_2)} \left[\alpha_1 e^{-\alpha_1 t} - \alpha_2 e^{-\alpha_2 t}\right] \\ e(t) &= x_0 - \frac{x_0}{\omega_0^2(\alpha_1 - \alpha_2)} \left[\alpha_2(\alpha_1^2 - \omega_0^2)e^{-\alpha_1 t} - \alpha_1(\alpha_2^2 - \omega_0^2)e^{-\alpha_2 t}\right] \end{aligned} \qquad (22.4.25)$$

so that the pendulum angle $\theta(t)$ asymptotically tends to the vertical position, $\theta = 0$, while the effective input $e(t)$ becomes a unit-step, like the $x(t)$ input.

(b) For $\alpha_1 \neq \alpha_2$, show that the angle response $\theta(t)$, and the effective input $e(t)$, due to a uniformly moving cart, $x(t) = v_0 t u(t)$, are given as follows, for $t \geq 0$,

$$\begin{aligned} \theta(t) &= \frac{v_0 \alpha_1 \alpha_2}{L\omega_0^2(\alpha_1 - \alpha_2)} \left[e^{-\alpha_2 t} - e^{-\alpha_1 t}\right] \\ e(t) &= v_0 t - \frac{v_0(\alpha_1 + \alpha_2)}{\alpha_1 \alpha_2} + \frac{v_0 \left[\alpha_2^2(\alpha_1^2 - \omega_0^2)e^{-\alpha_1 t} - \alpha_1^2(\alpha_2^2 - \omega_0^2)e^{-\alpha_2 t}\right]}{\alpha_1 \alpha_2 \omega_0^2(\alpha_1 - \alpha_2)} \end{aligned} \qquad (22.4.26)$$

so that, again, $\theta(t)$ stabilizes at the vertical $\theta = 0$ position, while $e(t)$ follows the uniformly moving input $x(t)$ up to a delay.

(c) For $\alpha_1 \neq \alpha_2$, and a uniformly accelerating cart, $x(t) = \frac{1}{2}a_0 t^2 u(t)$, show that the angle response $\theta(t)$, and the effective input $e(t)$, are given as follows, for $t \geq 0$,

$$\begin{aligned} \theta(t) &= \theta_0 + \frac{\theta_0}{\alpha_1 - \alpha_2} \left[\alpha_2 e^{-\alpha_1 t} - \alpha_1 e^{-\alpha_2 t}\right] \\ e(t) &= \frac{1}{2} a_0 t^2 - v_1 t + x_1 - \frac{a_0 \left[\alpha_2^3(\alpha_1^2 - \omega_0^2)e^{-\alpha_1 t} - \alpha_1^3(\alpha_2^2 - \omega_0^2)e^{-\alpha_2 t}\right]}{\alpha_1^2 \alpha_2^2 \omega_0^2(\alpha_1 - \alpha_2)} \\ \theta_0 &= \frac{a_0}{g}, \quad v_1 = \frac{a_0(\alpha_1 + \alpha_2)}{\alpha_1 \alpha_2}, \quad x_1 = a_0 \left(\frac{1}{\alpha_1^2} + \frac{1}{\alpha_2^2} + \frac{1}{\alpha_1 \alpha_2} - \frac{1}{\omega_0^2}\right) \end{aligned} \qquad (22.4.27)$$

We note that asymptotically, $\theta(t) \to \theta_0$, so that the equilibrium angle is slightly off the vertical. This can be understood from the acceleration diagram above in which the torque becomes zero when the acceleration due to gravity and that due to $\ddot{x} = a_0$ cancel each other, which happens at an angle θ_0 such that,

$$g \sin\theta_0 - \ddot{x} \cos\theta_0 = 0 \quad \Rightarrow \quad \tan\theta_0 = \frac{\ddot{x}}{g} = \frac{a_0}{g}$$

or, using the small-angle approximation, $\tan\theta_0 \approx \theta_0$, we have, $\theta_0 = a_0/g$.

(d) With $\alpha_1 \neq \alpha_2$, suppose that the cart is moving back and forth sinusoidally with a frequency ω, that is, $x(t) = A\sin(\omega t)u(t)$, then we expect that in the steady state, the inverted pendulum will also be oscillating with the same frequency about the vertical position $\theta = 0$. Noting that, $\sin(\omega t) = \text{Im}[e^{j\omega t}]$, show that in this case the angle $\theta(t)$ is given by the following expression, where the last two terms of $\theta(t)$ represent the transients and the first term, the steady state,

$$\begin{aligned}
x(t) &= A\sin(\omega t) = A\,\text{Im}[e^{j\omega t}] \\
\theta(t) &= \frac{A\alpha_1\alpha_2}{L\omega_0^2}\,\text{Im}\Big[R\,e^{j\omega t} + R_1 e^{-\alpha_1 t} + R_2 e^{-\alpha_2 t}\Big] \\
R &= \frac{(i\omega)^2}{(\alpha_1 + j\omega)(\alpha_2 + j\omega)} \\
R_1 &= \frac{\alpha_1^2}{(\alpha_1 - \alpha_2)(\alpha_1 + j\omega)} \\
R_2 &= \frac{\alpha_2^2}{(\alpha_2 - \alpha_1)(\alpha_2 + j\omega)}
\end{aligned} \tag{22.4.28}$$

(e) Repeat parts (a–d) when α_1, α_2 are equal, say, $\alpha_1 = \alpha_2 \equiv \alpha > 0$.

(f) Consider the following numerical values,

$$\omega_0 = 1\,,\quad L = 1\,,\quad \alpha_1 = 3\,,\quad \alpha_2 = 2$$
$$x_0 = 0.5\,,\quad v_0 = 1\,,\quad a_0 = 1\,,\quad A = 1\,,\quad \omega = 2$$

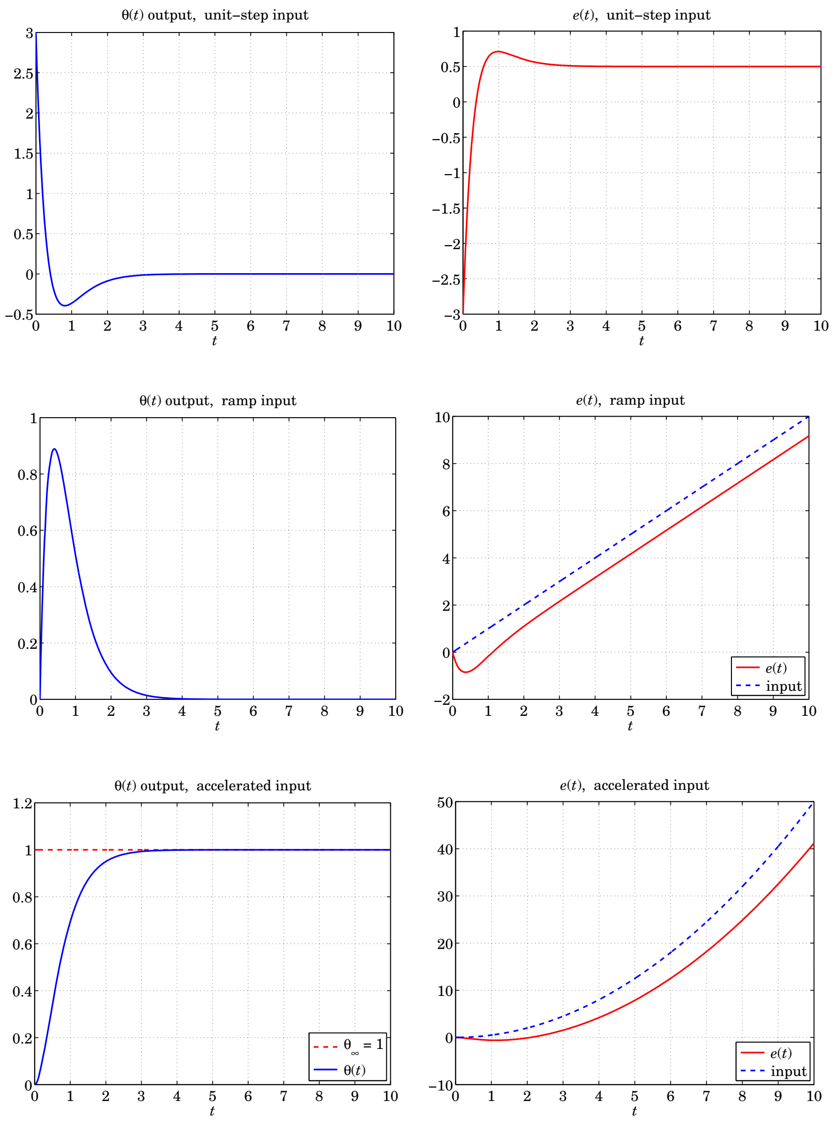
Calculate the values of the PI gains k_p, k_i.

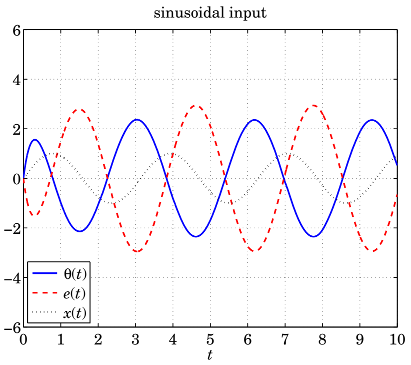
Calculate and plot the signals $\theta(t), e(t)$ for the four cases (a–d) using the exact formulas Eq. (22.4.25)–(22.4.28), over the time interval, $0 \le t \le 10$, with a time step of $T = 0.01$.

Moreover, using the transfer functions of Eq. (22.4.24), calculate the same signals $\theta(t), e(t)$ using the built-in function, **lsim**, and verify that they are essentially the same as the exact ones.

In the sinusoidal case, observe how the effective input $e(t)$ to the pendulum oscillates with a phase difference of 180° with respect to $\theta(t)$, that is, when the cart swings to the left, the pendulum swings to the right, and conversely — see last graph.

Typical Graphs





The typical MATLAB code for part (e), using LSIM, is as follows.

```
w0 = 1; L = 1; a1 = 3; a2 = 2;
x0 = 1/2;  v0 = 1;  a0 = 1;  A = 1; w = 2;
g = L*w0^2;

kp = 1 + w0^2/a1/a2;
ki = w0^2*(1/a1+1/a2);

s = tf(’s’);                % class tf
G = -s^2/L/(s^2-w0^2);      % open-loop transfer function
Gc = kp*L + ki*L/s;         % PI controller
H = G/(1+Gc*G);             % closed-loop, can also do, H = feedback(G,Gc);
He = 1/(1+Gc*G);            % equivalently, He = feedback(1,Gc*G);

t = linspace(0,10,1001);    % time-step, T = 10/1000 = 0.01

x = x0*ones(size(t));       % unit-step input
% x = v0*t;                 % ramp input, uncomment as necessary
% x = 1/2*a0*t.^2;          % uniform acceleration
% x = A*sin(w*t);           % sinusoidal input

y = lsim(H,x,t);            % closed-loop output, theta(t)
e = lsim(He,x,t);           % effective input, e(t)

figure; plot(t,y,’b-’);
figure; plot(t,e,’r-’);
```

The above graphs don’t display the LSIM outputs because they are virtually identical to the exact ones.

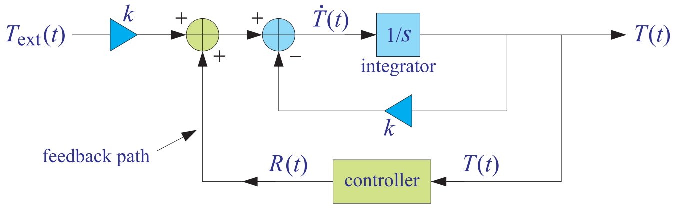
22.4.4 Thermostat Model

A typical home furnace supplies an amount of heat that increases the air temperature of a room by $R_0 = 20$ °F per hour. The time rate of change of the room temperature is

governed by Newton’s law of cooling:[†]

$$\frac{dT(t)}{dt} = -k\big[T(t)-T_{\text{ext}}(t)\big] + R(t) \tag{22.4.29}$$

where $T(t)$ is the room temperature at time t, $T_{\text{ext}}(t)$ is the external temperature, k is a measure of the loss of heat through the walls, and $R(t)$ is the rate of temperature increase per hour supplied by the furnace (like the R_0 above.) A block diagram realization of Eq. (22.4.29) is shown below. It is similar to that of the previous example, but with an extra feedback loop for calculating the control signal $R(t)$.



A typical home thermostat can be programmed to several temperature settings during the day. Here, we will assume two settings, a higher temperature setting T_H for the first 12 hours of a day, and a lower setting T_L for the second 12 hours. Thus, the control temperature of the thermostat is defined by the time function:

$$T_c(t) = \begin{cases} T_H\,, & \text{if } \operatorname{mod}(t,24) < 12 \\ T_L\,, & \text{if } \operatorname{mod}(t,24) \ge 12 \end{cases} \tag{22.4.30}$$

where the modulo operation, $\operatorname{mod}(t,24)$, reduces the time t modulo 24, i.e., it finds the remainder of the division of t by 24, so that it is always in the range $0 \le \operatorname{mod}(t,24) < 24$.

If the room temperature falls below the prescribed control temperature (T_H or T_L), the thermostat turns the furnace on until the control temperature is reached and then it turns the furnace off. This can be modeled into Eq. (22.4.29) by choosing the control signal $R(t)$ as follows:

$$R(t) = \begin{cases} R_0\,, & \text{if } T(t) < T_c(t) \\ 0\,, & \text{if } T(t) \ge T_c(t) \end{cases} \tag{22.4.31}$$

Because $R(t)$ depends on $T(t)$ in a nonlinear manner, Eq. (22.4.29) can only be solved numerically. To this end, time is discretized in small equal-step increments, $t_n = n\Delta t$, $n = 1, 2, 3, \dots$, where Δt is a small step size. The time-derivative in Eq. (22.4.29) can be approximated as a ratio of differences, resulting in the following difference equation:

$$\frac{T(t_{n+1}) - T(t_n)}{\Delta t} = -k\big[T(t_n) - T_{\text{ext}}(t_n)\big] + R(t_n)$$

Using the simplified notation $T(n)$ to denote $T(t_n)$,[‡] and similarly for $R(t_n)$ and $T_c(t_n)$, this difference equation can be rearranged into:

[†] For a more realistic version, see the paper by P. S. Sansgiry and C. C. Edwards, “A Home Heating Model for Calculus Students,” *Coll. Math. J.*, **27**, 395 (1996).

[‡] n is a MATLAB index, and $T(n)$, a MATLAB array.

$$T(n+1)=T(n)-k\,\Delta t\,\big[T(n)-T_{\text{ext}}(n)\big]+\Delta t\,R(n)\,,\quad n\ge 1 \tag{22.4.32}$$

where

$$R(n)=\begin{cases}R_0\,, & \text{if } T(n)<T_c(n)\\ 0\,, & \text{if } T(n)\ge T_c(n)\end{cases} \tag{22.4.33}$$

with

$$T_c(n)=\begin{cases}T_H\,, & \text{if } \operatorname{mod}(t_n,24)<12\\ T_L\,, & \text{if } \operatorname{mod}(t_n,24)\ge 12\end{cases} \tag{22.4.34}$$

The initial value in Eq. (22.4.32) will be assumed given, i.e., $T(1)=T_0$. For the external temperature, we will assume a simple sinusoidal model with 24-hr periodicity:

$$T_{\text{ext}}(n)=A-B\cos\left(\frac{2\pi t_n}{24}\right) \tag{22.4.35}$$

Consider the following realistic numerical values:

$$A=40\ {}^{\text{o}}\text{F}\,,\quad B=10\ {}^{\text{o}}\text{F}$$

$$k=0.35\ \text{hr}^{-1}\,,\quad R_0=20\ {}^{\text{o}}\text{F/hr}$$

$$T_H=70\ {}^{\text{o}}\text{F}\,,\quad T_L=60\ {}^{\text{o}}\text{F}\,,\quad T_0=35\ {}^{\text{o}}\text{F}$$

Define the time vector t_n to span a 48-hr period and sampled every 3 seconds:

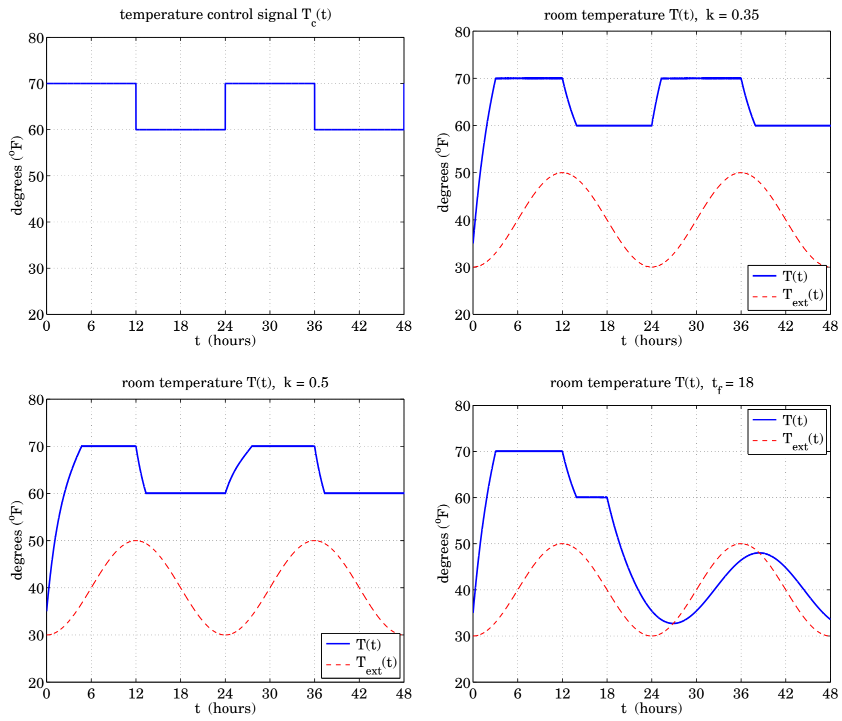
```
Tmax = 48; Dt = 3/3600;      % units of hours
tn = Dt:Dt:Tmax;
```

a. Use a for-loop to calculate the control signal $T_c(n)$ and plot it versus t_n. Within the same for-loop, calculate also the actual room temperature $T(n)$, and on separate graph plot it versus t_n together with the external temperature $T_{\text{ext}}(n)$.

   Observe the initial transients starting from T_0, and the ability of the thermostat system to follow the prescribed high/low settings, switching between the two at every 12-hr period.

b. Repeat the calculation and plotting of $T(n)$ using the value $k=0.25$, corresponding to a well-insulated house, and then using $k=0.50$ for a poorly insulated one.

c. For the case $k=0.35$, assume that there is a power failure at time $t_f=18$ and that from then on the furnace stops operating. Calculate and plot the room temperature and observe how it eventually follows the external temperature variations (with some lag.)



The essential MATLAB code for this problem is listed below:

```
A = 40; B = 10;
T0 = 35;
TH = 70; TL = 60;
R0 = 20;

Tmax = 48;                     % hours
Dt = 3/3600;                   % 3 sec in units of hours
t = Dt:Dt:Tmax;                % units of hours

Te = A - B*cos(pi*t/12);       % external temperature

Tc = TH*(mod(t,24)<12) + TL*(mod(t,24)>=12);   % control temperature
                                               % control, R(n) = R0*(T(n)<Tc(n))
figure; plot(t,Tc,’b-’);
xlabel(’t  (hours)’); ylabel(’degrees (^oF)’);
title(’temperature control signal T_c(t)’);
yaxis(20,80,20:10:80); xaxis(0,48,0:6:48); grid on;

for k = [0.35, 0.25, 0.50]    % generate graphs for three values of k
   T(1) = T0;
   for n=1:length(t)-1
      R = R0*(T(n)<Tc(n));                         % control signal
      T(n+1) = T(n) - k*Dt*(T(n)-Te(n)) + Dt*R;
   end
   figure; plot(t,T,'b-', t,Te,'r--');
   xlabel('t  (hours)'); ylabel('degrees (^oF)');
   title(['room temperature T(t),  k = ',num2str(k)]);
   yaxis(20,80,20:10:80); xaxis(0,48,0:6:48); grid on;
end

k = 0.35;
t_f = 18;            % control signal is non-zero only for t(n)<=tf

T(1) = T0;
for n=1:length(t)-1
   R = R0*(T(n)<Tc(n) & t(n)<=t_f);                % control signal
   T(n+1) = T(n) - k*Dt*(T(n)-Te(n)) + Dt*R;
end

figure; plot(t,T,'b-', t,Te,'r--');
xlabel('t  (hours)'); ylabel('degrees (^oF)');
title(['room temperature T(t),  t_f = ',num2str(t_f)]);
yaxis(20,80,20:10:80); xaxis(0,48,0:6:48); grid on;
```

23

Local Polynomial Filters

23.1 Introduction

We mentioned in Sec. 15.5 that there are limits to the applicability of the plain FIR averager filter—in order to achieve a high degree of noise reduction, its length N may be required to be so large that the filter's passband becomes smaller than the signal bandwidth, causing the removal of useful high frequencies from the desired signal.

In other words, in its attempt to smooth out the noise v_n, the filter begins to smooth out the desired signal x_n to an unacceptable degree. For example, if x_n contains some short-duration peaks, corresponding to the higher frequencies present in x_n, and the filter's length N is longer than the duration of the peaks, the filter will tend to smooth the peaks too much, broadening them and reducing their height.

In this chapter,[†] we discuss local polynomial smoothing filters [620–683] which are generalizations of the FIR averager filter that can preserve better the higher frequency content of the desired signal, at the expense of not removing as much noise as the averager. They can be characterized in three equivalent ways:

1. They are the optimal lowpass filters that minimize the NRR, subject to additional constraints than the DC unity-gain condition (15.5.1)—the constraints being equivalent to the requirement that polynomial input signals go through the filter unchanged.
2. They are the optimal filters that minimize the NRR whose frequency response $H(\omega)$ satisfies certain *flatness* constraints at DC.
3. They are the filters that optimally fit, in a least-squares sense, a set of data points to polynomials of different degrees.

Local polynomial smoothing (LPSM) filters have a long history and have been rediscovered repeatedly in different contexts. They were originally derived in 1866 by the Italian astronomer Schiaparelli [620] who formulated the problem as the minimization of the NRR subject to polynomial-preserving constraints and derived the filters in complete generality, discussing also the case of even-length filters. They were rederived in 1871 by De Forest [649] who generalized them further to include the case of "minimum-roughness" or minimum-R_s filters. Subsequently, they were rediscovered many times

[†] adapted from the author's book on *Applied Optimum Signal Processing* [45]

and used extensively in actuarial applications, for example, by Gram, Hardy, Sheppard, Henderson, and others. See Refs. [652–659] for the development and history of these filters. In the actuarial context, smoothing is referred to as the process of "graduation." They were revived again in the 1960s by Savitzky and Golay [626] and have been applied widely in chemistry and spectroscopy [626–637] known in that context as *Savitzky-Golay filters*. They, and their minimum-R_s versions [649–683] known typically as *Henderson filters*, are used routinely for trend extraction in financial, business, and census applications.

Some recent incarnations also include predictive FIR interpolation, differentiation, fractional-delay, and maximally-flat filters [736–771], and applications to the representation of speech and images in terms of orthogonal-polynomial moments [721–734].

The least-squares polynomial fitting approach also has a long history. Chebyshev [688] derived in 1864 the discrete Chebyshev orthogonal polynomials,[‡] also known as Gram polynomials, which provide convenient and computationally efficient bases for the solution of the least-squares problem and the design of local polynomial filters. Several applications and reviews of the discrete Chebyshev orthogonal polynomials may be found in [688–735]. The minimum-R_s Henderson filters also admit similar efficient representations in terms of the Hahn orthogonal polynomials, a special case of which are the discrete Chebyshev polynomials. We discuss Henderson filters in Sec. 23.12 and orthogonal polynomial bases in Sec. 23.13.

23.2 Local Polynomial Fitting

We begin with the least-squares polynomial fitting approach. We assume that the signal model for the observations is:

$$y_n = x_n + v_n$$

where v_n is white noise and x_n is a smooth signal to be estimated. Fig. 23.2.1 shows five noisy signal samples $[y_{-2}, y_{-1}, y_0, y_1, y_2]$ positioned symmetrically about the origin. Later on, we will shift them to an arbitrary position along the time axis. Polynomial smoothing of the five samples is equivalent to replacing them by the values that lie on smooth polynomial curves drawn between the noisy samples. In Fig. 23.2.1, we consider fitting the five data to a constant signal, a linear signal, and a quadratic signal.

The corresponding smoothed values are given by the 0th, 1st, and 2nd degree polynomials defined for $m = -2, -1, 0, 1, 2$:

$$\begin{aligned} \hat{y}_m &= c_0 && \text{(constant)} \\ \hat{y}_m &= c_0 + c_1 m && \text{(linear)} \\ \hat{y}_m &= c_0 + c_1 m + c_2 m^2 && \text{(quadratic)} \end{aligned} \qquad (23.2.1)$$

For each choice of the polynomial order, the coefficients c_i must be determined optimally such that the corresponding polynomial curve best fits the given data. This can be accomplished by a *least-squares fit*, which chooses the c_i that minimize the total

[‡] not to be confused with the ordinary Chebyshev polynomials.

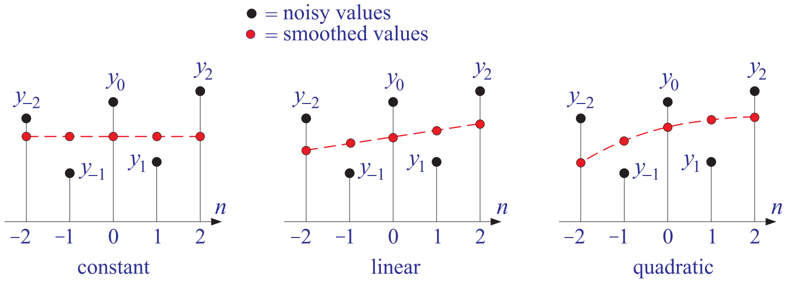


Fig. 23.2.1 Data smoothing with polynomials of degrees $d = 0, 1, 2$.

mean-square error. For example, in the quadratic case, we have the performance index:

$$\mathcal{J} = \sum_{m=-2}^{2} e_m^2 = \sum_{m=-2}^{2} \big(y_m - (c_0 + c_1 m + c_2 m^2)\big)^2 = \min \qquad (23.2.2)$$

where the fitting errors are defined as

$$e_m = y_m - \hat{y}_m = y_m - (c_0 + c_1 m + c_2 m^2)\,, \qquad m = -2, -1, 0, 1, 2$$

It proves convenient to express Eqs. (23.2.1) and (23.2.2) in a vectorial form, which generalizes to higher polynomial orders and to more than five data points. We define the five-dimensional vectors of data, estimates, and errors:

$$\mathbf{y} = \begin{bmatrix} y_{-2} \\ y_{-1} \\ y_0 \\ y_1 \\ y_2 \end{bmatrix}, \qquad \hat{\mathbf{y}} = \begin{bmatrix} \hat{y}_{-2} \\ \hat{y}_{-1} \\ \hat{y}_0 \\ \hat{y}_1 \\ \hat{y}_2 \end{bmatrix}, \qquad \mathbf{e} = \begin{bmatrix} e_{-2} \\ e_{-1} \\ e_0 \\ e_1 \\ e_2 \end{bmatrix} = \mathbf{y} - \hat{\mathbf{y}}$$

Similarly, we define the five-dimensional polynomial *basis vectors* $\mathbf{s}_0$, $\mathbf{s}_1$, $\mathbf{s}_2$, whose components are:

$$s_0(m) = 1, \qquad s_1(m) = m, \qquad s_2(m) = m^2, \qquad -2 \le m \le 2$$

Vectorially, we have:

$$\mathbf{s}_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \qquad \mathbf{s}_1 = \begin{bmatrix} -2 \\ -1 \\ 0 \\ 1 \\ 2 \end{bmatrix}, \qquad \mathbf{s}_2 = \begin{bmatrix} 4 \\ 1 \\ 0 \\ 1 \\ 4 \end{bmatrix} \qquad (23.2.3)$$

In this notation, we may write the third of Eq. (23.2.1) vectorially:

$$\hat{\mathbf{y}} = c_0 \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + c_1 \begin{bmatrix} -2 \\ -1 \\ 0 \\ 1 \\ 2 \end{bmatrix} + c_2 \begin{bmatrix} 4 \\ 1 \\ 0 \\ 1 \\ 4 \end{bmatrix} = c_0 \mathbf{s}_0 + c_1 \mathbf{s}_1 + c_2 \mathbf{s}_2$$

Therefore,

$$\hat{\mathbf{y}} = c_0\mathbf{s}_0 + c_1\mathbf{s}_1 + c_2\mathbf{s}_2 = [\mathbf{s}_0, \mathbf{s}_1, \mathbf{s}_2]\begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} \equiv S\mathbf{c} \qquad (23.2.4)$$

The 5×3 basis matrix S has as columns the three basis vectors $\mathbf{s}_0, \mathbf{s}_1, \mathbf{s}_2$. It is given explicitly as follows:

$$S = [\mathbf{s}_0, \mathbf{s}_1, \mathbf{s}_2] = \begin{bmatrix} 1 & -2 & 4 \\ 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{bmatrix} \qquad (23.2.5)$$

Writing $\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}} = \mathbf{y} - S\mathbf{c}$, we can express the performance index (23.2.2) as the dot product:

$$\mathcal{J} = \mathbf{e}^T\mathbf{e} = (\mathbf{y} - S\mathbf{c})^T(\mathbf{y} - S\mathbf{c}) = \min \qquad (23.2.6)$$

To minimize this expression with respect to $\mathbf{c}$, we set the gradient $\partial\mathcal{J}/\partial\mathbf{c}$ to zero:

$$\frac{\partial\mathcal{J}}{\partial\mathbf{c}} = -2S^T\mathbf{e} = -2S^T(\mathbf{y} - S\mathbf{c}) = -2(S^T\mathbf{y} - S^TS\mathbf{c}) = 0 \qquad (23.2.7)$$

Therefore, the minimization condition gives the so-called *orthogonality equations* and the equivalent *normal equations*:

$$\frac{\partial\mathcal{J}}{\partial\mathbf{c}} = 0 \quad \Leftrightarrow \quad S^T\mathbf{e} = 0 \quad \Leftrightarrow \quad S^TS\mathbf{c} = S^T\mathbf{y} \qquad (23.2.8)$$

with optimal solution:

$$\mathbf{c} = (S^TS)^{-1}S^T\mathbf{y} \equiv G^T\mathbf{y} \qquad (23.2.9)$$

where we defined the 5×3 matrix G by

$$G = S(S^TS)^{-1} \qquad (23.2.10)$$

We note that the solution (23.2.9) is none other than the unique least-squares solution of the full-rank overdetermined linear system $S\mathbf{c} = \mathbf{y}$[†], $\mathbf{c} = S^+\mathbf{y}$, where $S^+ = (S^TS)^{-1}S^T$ is the corresponding pseudoinverse. Inserting the optimal coefficients $\mathbf{c}$ into Eq. (23.2.4), we find the smoothed values:[†]

$$\hat{\mathbf{y}} = S\mathbf{c} = SG^T\mathbf{y} = S(S^TS)^{-1}S^T\mathbf{y} \equiv B^T\mathbf{y} \qquad (23.2.11)$$

where we defined the 5×5 matrix B by

$$B = B^T = SG^T = GS^T = S(S^TS)^{-1}S^T \qquad (23.2.12)$$

[†] see for example by [45]

[†] although B is symmetric, we prefer to write $\hat{\mathbf{y}} = B^T\mathbf{y}$, which generalizes to the non-symmetric case of minimum-roughness filters of Sec. 23.12.

The symmetric 3×3 matrix $F = S^TS$, which appears in the expressions for G and B, has matrix elements that are the *dot products* of the basis vectors, that is, the ijth matrix element is $F_{ij} = (S^TS)_{ij} = \mathbf{s}_i^T\mathbf{s}_j$. Indeed, using Eq. (23.2.5), we find:

$$F = S^TS = \begin{bmatrix} \mathbf{s}_0^T \\ \mathbf{s}_1^T \\ \mathbf{s}_2^T \end{bmatrix}[\mathbf{s}_0, \mathbf{s}_1, \mathbf{s}_2] = \begin{bmatrix} \mathbf{s}_0^T\mathbf{s}_0 & \mathbf{s}_0^T\mathbf{s}_1 & \mathbf{s}_0^T\mathbf{s}_2 \\ \mathbf{s}_1^T\mathbf{s}_0 & \mathbf{s}_1^T\mathbf{s}_1 & \mathbf{s}_1^T\mathbf{s}_2 \\ \mathbf{s}_2^T\mathbf{s}_0 & \mathbf{s}_2^T\mathbf{s}_1 & \mathbf{s}_2^T\mathbf{s}_2 \end{bmatrix} \qquad (23.2.13)$$

Using Eq. (23.2.5), we calculate F and its inverse F^{-1}:

$$F = \begin{bmatrix} 5 & 0 & 10 \\ 0 & 10 & 0 \\ 10 & 0 & 34 \end{bmatrix}, \qquad F^{-1} = \frac{1}{35}\begin{bmatrix} 17 & 0 & -5 \\ 0 & 3.5 & 0 \\ -5 & 0 & 2.5 \end{bmatrix} \qquad (23.2.14)$$

Then, we calculate the 5×3 matrix $G = S(S^TS)^{-1} = SF^{-1}$:

$$G = SF^{-1} = \frac{1}{35}\begin{bmatrix} 1 & -2 & 4 \\ 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{bmatrix}\begin{bmatrix} 17 & 0 & -5 \\ 0 & 3.5 & 0 \\ -5 & 0 & 2.5 \end{bmatrix} \quad \text{or,}$$

$$G = \frac{1}{35}\begin{bmatrix} -3 & -7 & 5 \\ 12 & -3.5 & -2.5 \\ 17 & 0 & -5 \\ 12 & 3.5 & -2.5 \\ -3 & 7 & 5 \end{bmatrix} \equiv [\mathbf{g}_0, \mathbf{g}_1, \mathbf{g}_2] \qquad (23.2.15)$$

As we see below, the three columns of G have useful interpretations as differentiation filters. Next, using Eq. (23.2.12), we calculate the 5×5 matrix B:

$$B = GS^T = \frac{1}{35}\begin{bmatrix} -3 & -7 & 5 \\ 12 & -3.5 & -2.5 \\ 17 & 0 & -5 \\ 12 & 3.5 & -2.5 \\ -3 & 7 & 5 \end{bmatrix}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -2 & -1 & 0 & 1 & 2 \\ 4 & 1 & 0 & 1 & 4 \end{bmatrix} \quad \text{or,}$$

$$B = \frac{1}{35}\begin{bmatrix} 31 & 9 & -3 & -5 & 3 \\ 9 & 13 & 12 & 6 & -5 \\ -3 & 12 & 17 & 12 & -3 \\ -5 & 6 & 12 & 13 & 9 \\ 3 & -5 & -3 & 9 & 31 \end{bmatrix} \equiv [\mathbf{b}_{-2}, \mathbf{b}_{-1}, \mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2] \qquad (23.2.16)$$

Because B is symmetric, its rows are the same as its columns. Thus, we can write it either in column-wise or row-wise form:

$$B = [\mathbf{b}_{-2}, \mathbf{b}_{-1}, \mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2] = \begin{bmatrix} \mathbf{b}_{-2}^T \\ \mathbf{b}_{-1}^T \\ \mathbf{b}_0^T \\ \mathbf{b}_1^T \\ \mathbf{b}_2^T \end{bmatrix} = B^T$$

The five columns or rows of B are the LPSM filters of length 5 and polynomial order 2. The corresponding smoothed values $\hat{\mathbf{y}}$ can be expressed component-wise in terms of these filters, as follows:

$$\begin{bmatrix} \hat{y}_{-2} \\ \hat{y}_{-1} \\ \hat{y}_0 \\ \hat{y}_1 \\ \hat{y}_2 \end{bmatrix} = \hat{\mathbf{y}} = B^T\mathbf{y} = \begin{bmatrix} \mathbf{b}_{-2}^T \\ \mathbf{b}_{-1}^T \\ \mathbf{b}_0^T \\ \mathbf{b}_1^T \\ \mathbf{b}_2^T \end{bmatrix}\mathbf{y} = \begin{bmatrix} \mathbf{b}_{-2}^T\mathbf{y} \\ \mathbf{b}_{-1}^T\mathbf{y} \\ \mathbf{b}_0^T\mathbf{y} \\ \mathbf{b}_1^T\mathbf{y} \\ \mathbf{b}_2^T\mathbf{y} \end{bmatrix}$$

or, for $m = -2, -1, 0, 1, 2$:

$$\hat{y}_m = \mathbf{b}_m^T\mathbf{y} \tag{23.2.17}$$

and more explicitly,

$$\begin{bmatrix} \hat{y}_{-2} \\ \hat{y}_{-1} \\ \hat{y}_0 \\ \hat{y}_1 \\ \hat{y}_2 \end{bmatrix} = \frac{1}{35}\begin{bmatrix} 31 & 9 & -3 & -5 & 3 \\ 9 & 13 & 12 & 6 & -5 \\ -3 & 12 & 17 & 12 & -3 \\ -5 & 6 & 12 & 13 & 9 \\ 3 & -5 & -3 & 9 & 31 \end{bmatrix}\begin{bmatrix} y_{-2} \\ y_{-1} \\ y_0 \\ y_1 \\ y_2 \end{bmatrix} \tag{23.2.18}$$

Thus, the mth filter $\mathbf{b}_m$ dotted into the data vector $\mathbf{y}$ generates the mth smoothed data sample. In a similar fashion, we can express the polynomial coefficients c_i as dot products. Using the solution Eq. (23.2.9), we have

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \mathbf{c} = G^T\mathbf{y} = \begin{bmatrix} \mathbf{g}_0^T \\ \mathbf{g}_1^T \\ \mathbf{g}_2^T \end{bmatrix}\mathbf{y} = \begin{bmatrix} \mathbf{g}_0^T\mathbf{y} \\ \mathbf{g}_1^T\mathbf{y} \\ \mathbf{g}_2^T\mathbf{y} \end{bmatrix}$$

or, explicitly,

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \frac{1}{35}\begin{bmatrix} -3 & 12 & 17 & 12 & -3 \\ -7 & -3.5 & 0 & 3.5 & 7 \\ 5 & -2.5 & -5 & -2.5 & 5 \end{bmatrix}\begin{bmatrix} y_{-2} \\ y_{-1} \\ y_0 \\ y_1 \\ y_2 \end{bmatrix} \tag{23.2.19}$$

Thus, the coefficients c_i can be expressed as the dot products of the columns of G with the data vector $\mathbf{y}$:

$$c_i = \mathbf{g}_i^T\mathbf{y}, \qquad i = 0, 1, 2 \tag{23.2.20}$$

Of the five columns of B, the middle one, $\mathbf{b}_0$, is the most important because it smooths the value y_0, which is symmetrically placed with respect to the other samples in $\mathbf{y}$, as shown in Fig. 23.2.1.

In smoothing a long block of data, the filter $\mathbf{b}_0$ is used during the *steady-state* period, whereas the other columns of B are used only during the input-on and input-off *transients*. We will refer to $\mathbf{b}_0$ and the other columns of B as the *steady-state* and *transient* LPSM filters.

Setting $m = 0$ into Eq. (23.2.1), we note that the middle smoothed value $\hat{y}_0$ is equal to the polynomial coefficient c_0. Using Eqs. (23.2.17) and (23.2.20), we find: $\hat{y}_0 = c_0 = \mathbf{b}_0^T\mathbf{y} = \mathbf{g}_0^T\mathbf{y}$ (the middle column of B and the first column of G are always the same, $\mathbf{b}_0 = \mathbf{g}_0$.)

To express (23.2.18) as a true filtering operation acting on an input sequence y_n, we shift the group of five samples to be centered around the nth time instant, that is, we make the substitution:

$$[y_{-2}, y_{-1}, y_0, y_1, y_2] \longrightarrow [y_{n-2}, y_{n-1}, y_n, y_{n+1}, y_{n+2}]$$

The corresponding five smoothed values will be then:

$$\begin{bmatrix} \hat{y}_{n-2} \\ \hat{y}_{n-1} \\ \hat{y}_n \\ \hat{y}_{n+1} \\ \hat{y}_{n+2} \end{bmatrix} = \frac{1}{35}\begin{bmatrix} 31 & 9 & -3 & -5 & 3 \\ 9 & 13 & 12 & 6 & -5 \\ -3 & 12 & 17 & 12 & -3 \\ -5 & 6 & 12 & 13 & 9 \\ 3 & -5 & -3 & 9 & 31 \end{bmatrix}\begin{bmatrix} y_{n-2} \\ y_{n-1} \\ y_n \\ y_{n+1} \\ y_{n+2} \end{bmatrix} \tag{23.2.21}$$

In particular, the middle sample y_n is smoothed by the filter $\mathbf{b}_0$:

$$\boxed{\hat{x}_n = \frac{1}{35}\big(-3y_{n-2} + 12y_{n-1} + 17y_n + 12y_{n+1} - 3y_{n+2}\big)} \tag{23.2.22}$$

where, in accordance with our assumed model of noisy observations $y_n = x_n + v_n$, we denoted $\hat{y}_n$ by $\hat{x}_n$, i.e., the estimated value of x_n.

The other estimated values $\{\hat{y}_{n+m}\,,\ m = \pm 1, \pm 2\}$, are not used for smoothing, except, as we see later, at the beginning and end of the signal block y_n. They may be used, however, for prediction and interpolation.

The filter (23.2.22) corresponds to fitting every group of five samples $\{y_{n-2}, y_{n-1}, y_n, y_{n+1}, y_{n+2}\}$ to a quadratic polynomial and replacing the middle sample y_n by its smoothed value $\hat{x}_n$. It is a lowpass filter and is normalized to unity gain at DC, because its coefficients add up to one.

Its NRR is the sum of the squared filter coefficients. It can be proved in general that the NRR of any steady-state filter $\mathbf{b}_0$ is equal to the *middle* value of its impulse response, that is, the coefficient $b_0(0)$. Therefore,

$$\mathcal{R} = \mathbf{b}_0^T\mathbf{b}_0 = \sum_{m=-2}^{2} b_0(m)^2 = b_0(0) = \frac{17}{35} = \frac{17/7}{5} = \frac{2.43}{5} = 0.49$$

By comparison, the length-5 FIR averager operating on the same five samples is:

$$\boxed{\hat{x}_n = \frac{1}{5}\big(y_{n-2} + y_{n-1} + y_n + y_{n+1} + y_{n+2}\big)} \tag{23.2.23}$$

with $\mathcal{R} = 1/N = 1/5$. Thus, the length-5 quadratic-polynomial filter performs 2.43 times worse in reducing noise than the FIR averager. However, the higher-order polynomial filters have other advantages to be discussed later.

We saw that the coefficient c_0 represents the smoothed value of y_0 at $m = 0$. Similarly, the coefficient c_1 represents the slope, the derivative, of y_0 at $m = 0$. Indeed, we have from Eq. (23.2.1) by differentiating and setting $m = 0$:

$$\dot{\hat{y}}_0 = \frac{d\hat{y}_m}{dm}\bigg|_0 = c_1\,, \qquad \ddot{\hat{y}}_0 = \frac{d^2\hat{y}_m}{dm^2}\bigg|_0 = 2c_2$$

Thus, c_1 and $2c_2$ represent the polynomial estimates of the first and second derivatives at $m = 0$. Using Eq. (23.2.20) we can express them in terms of the second and third columns of the matrix G:

$$\begin{aligned} \dot{\hat{y}}_0 &= c_1 = \mathbf{g}_1^T\mathbf{y} \\ \ddot{\hat{y}}_0 &= 2c_2 = 2\mathbf{g}_2^T\mathbf{y} \end{aligned} \tag{23.2.24}$$

Shifting these to the nth time sample, and denoting them by $\dot{\hat{x}}_n$ and $\ddot{\hat{x}}_n$, we find the length-5 local polynomial filters for estimating the *first and second derivatives* of x_n:

$$\boxed{\begin{aligned} \dot{\hat{x}}_n &= \frac{1}{35}\big(-7y_{n-2} - 3.5y_{n-1} + 3.5y_{n+1} + 7y_{n+2}\big) \\ \ddot{\hat{x}}_n &= \frac{2}{35}\big(5y_{n-2} - 2.5y_{n-1} - 5y_n - 2.5y_{n+1} + 5y_{n+2}\big) \end{aligned}} \tag{23.2.25}$$

The above designs can be generalized in a straightforward manner to an arbitrary degree d of the fitted polynomial and to an arbitrary length N of the data vector $\mathbf{y}$. We require only that $d \le N - 1$, a restriction to be clarified later. Assuming that N is odd, say, $N = 2M + 1$, the five-dimensional data vector $\mathbf{y} = [y_{-2}, y_{-1}, y_0, y_1, y_2]^T$ is replaced by an N-dimensional one, having M points on either side of y_0:

$$\mathbf{y} = [y_{-M}, \dots, y_{-1}, y_0, y_1, \dots, y_M]^T \tag{23.2.26}$$

The N data samples in $\mathbf{y}$ are then fitted by a polynomial of degree d:

$$\hat{y}_m = c_0 + c_1 m + \cdots + c_d m^d, \qquad -M \le m \le M \tag{23.2.27}$$

In this case, there are $d+1$ polynomial basis vectors $\mathbf{s}_i$, $i = 0, 1, \dots, d$, defined to have components:

$$s_i(m) = m^i, \qquad -M \le m \le M \tag{23.2.28}$$

The corresponding $N\times(d+1)$ basis matrix S is defined to have the $\mathbf{s}_i$ as columns:

$$S = [\mathbf{s}_0, \mathbf{s}_1, \dots, \mathbf{s}_d] \tag{23.2.29}$$

The smoothed values (23.2.27) can be written in the vector form:

$$\hat{\mathbf{y}} = \sum_{i=0}^{d} c_i\mathbf{s}_i = [\mathbf{s}_0, \mathbf{s}_1, \dots, \mathbf{s}_d]\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_d \end{bmatrix} = S\mathbf{c} \tag{23.2.30}$$

The design steps for the LPSM filters can be summarized then as follows:

$$\begin{aligned} F &= S^T S \quad \Leftrightarrow \quad F_{ij} = \mathbf{s}_i^T\mathbf{s}_j, \quad i,j = 0, 1, \dots, d \\ G &= SF^{-1} \equiv [\mathbf{g}_0, \mathbf{g}_1, \dots, \mathbf{g}_d] \\ B &= GS^T = SF^{-1}S^T \equiv [\mathbf{b}_{-M}, \dots, \mathbf{b}_0, \dots, \mathbf{b}_M] \end{aligned} \tag{23.2.31}$$

The corresponding coefficient vector $\mathbf{c}$ and smoothed data vector $\hat{\mathbf{y}}$ will be:

$$\begin{aligned} \mathbf{c} = G^T\mathbf{y} \quad &\Leftrightarrow \quad c_i = \mathbf{g}_i^T\mathbf{y}, \quad i = 0, 1, \dots, d \\ \hat{\mathbf{y}} = B^T\mathbf{y} \quad &\Leftrightarrow \quad \hat{y}_m = \mathbf{b}_m^T\mathbf{y}, \quad -M \le m \le M \end{aligned} \tag{23.2.32}$$

The middle smoothed value $\hat{y}_0$ is given in terms of the middle LPSM filter $\mathbf{b}_0$:

$$\hat{y}_0 = \mathbf{b}_0^T\mathbf{y} = \sum_{k=-M}^{M} b_0(k)y_k$$

The N-dimensional vector $\mathbf{y} = [y_{-M}, \dots, y_{-1}, y_0, y_1, \dots, y_M]^T$ can be shifted to the nth time instant by the replacement:

$$[y_{-M}, \dots, y_{-1}, y_0, y_1, \dots, y_M] \longrightarrow [y_{n-M}, \dots, y_{n-1}, y_n, y_{n+1}, \dots, y_{n+M}]$$

The resulting length-N, order-d, LPSM filter for smoothing a noisy sequence y_n will be, in its steady-state form (denoting again $\hat{x}_n = \hat{y}_n$):

$$\boxed{\hat{x}_n = \hat{y}_n = \sum_{k=-M}^{M} b_0(k)y_{n+k} = \sum_{k=-M}^{M} b_0(-k)y_{n-k}} \tag{23.2.33}$$

The second equation expresses the output in convolutional form.[†] Because the filter $\mathbf{b}_0$ is symmetric about its middle, we can replace $b_0(-k) = b_0(k)$. The non-central estimated values are obtained from the $\mathbf{b}_m$ filters:

$$\boxed{\hat{y}_{n+m} = \sum_{k=-M}^{M} b_m(k)y_{n+k} = \sum_{k=-M}^{M} b_m^R(k)y_{n-k}}\,, \quad -M \le m \le M \tag{23.2.34}$$

These filters satisfy the symmetry property $b_m^R(k) = b_m(-k) = b_{-m}(k)$ and can be used for prediction, as we discuss later.

The $d+1$ columns of the $N\times(d+1)$-dimensional matrix G give the LPSM *differentiation filters*, for derivatives of orders $i = 0, 1, \dots, d$. It follows by differentiating Eq. (23.2.27) i times and setting $m = 0$:

$$\hat{y}_0^{(i)} = \frac{d^i\hat{y}_m}{dm^i}\bigg|_0 = i!\,c_i = i!\,\mathbf{g}_i^T\mathbf{y}$$

Shifting these to time n, gives the differentiation convolutional filtering equations:

$$\hat{x}_n^{(i)} = i!\sum_{m=-M}^{M} g_i^R(m)y_{n-m}, \qquad i = 0, 1, \dots, d \tag{23.2.35}$$

where, $g_i^R(m) = g_i(-m)$ and as in Eq. (23.2.33), we reversed the order of writing the terms, but here the filters $\mathbf{g}_i$ are not necessarily symmetric (actually, they are symmetric for even i, and antisymmetric for odd i.)

[†] We use the notation $\mathbf{b}^R$ to denote the reverse of a double-sided filter $\mathbf{b}$, that is, $b^R(k) = b(-k)$.

Example 23.2.1: We construct the length-5 LPSM filters for the cases $d = 0$ and $d = 1$. For $d = 0$, corresponding to the constant $\hat{y}_m = c_0$ in Eq. (23.2.1), there is only one basis vector $\mathbf{s}_0$ defined in Eq. (23.2.3). The basis matrix $S = [\mathbf{s}_0]$ will have just one column, and the matrix F will be the scalar

$$F = S^T S = \mathbf{s}_0^T \mathbf{s}_0 = [1,1,1,1,1]\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = 5$$

The matrix G will then be

$$G = SF^{-1} = \frac{1}{5}\mathbf{s}_0 = \frac{1}{5}[1,1,1,1,1]^T$$

resulting in the LPSM matrix B:

$$B = GS^T = \frac{1}{5}\mathbf{s}_0\mathbf{s}_0^T = \frac{1}{5}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}[1,1,1,1,1] = \frac{1}{5}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Thus, the steady-state LPSM filter is the length-5 averager:

$$\mathbf{b}_0 = \frac{1}{5}[1,1,1,1,1]^T$$

For the case $d = 1$, corresponding to the linear fit $\hat{y}_m = c_0 + c_1 m$, we have the two basis vectors $\mathbf{s}_0$ and $\mathbf{s}_1$, given in Eq. (23.2.3). We calculate the matrices S, F, and F^{-1}:

$$S = [\mathbf{s}_0, \mathbf{s}_1] = \begin{bmatrix} 1 & -2 \\ 1 & -1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{bmatrix}, \quad F = S^T S = \begin{bmatrix} 5 & 0 \\ 0 & 10 \end{bmatrix}, \quad F^{-1} = \frac{1}{5}\begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix}$$

This gives for G and B:

$$G = SF^{-1} = \frac{1}{5}\begin{bmatrix} 1 & -1 \\ 1 & -0.5 \\ 1 & 0 \\ 1 & 0.5 \\ 1 & 1 \end{bmatrix}, \quad B = GS^T = \frac{1}{5}\begin{bmatrix} 3 & 2 & 1 & 0 & -1 \\ 2 & 1.5 & 1 & 0.5 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0.5 & 1 & 1.5 & 2 \\ -1 & 0 & 1 & 2 & 3 \end{bmatrix}$$

Thus, the steady-state LPSM filter $\mathbf{b}_0$ is still equal to the length-5 FIR averager. It is a general property of LPSM filters, that the filter $\mathbf{b}_0$ is the same for successive polynomial orders, that is, for $d = 0, 1$, $d = 2, 3$, $d = 4, 5$, and so on. However, the transient LPSM filters are different. □

Example 23.2.2: Here, we construct the LPSM filters of length $N = 5$ and order $d = 3$. The smoothed estimates are given by the cubic polynomial:

$$\hat{y}_m = c_0 + c_1 m + c_2 m^2 + c_3 m^3$$

There is an additional basis vector $\mathbf{s}_3$ with components $s_3(m) = m^3$. Therefore, the basis matrix S is:

$$S = [\mathbf{s}_0, \mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3] = \begin{bmatrix} 1 & -2 & 4 & -8 \\ 1 & -1 & 1 & -1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \end{bmatrix} \Rightarrow F = S^T S = \begin{bmatrix} 5 & 0 & 10 & 0 \\ 0 & 10 & 0 & 34 \\ 10 & 0 & 34 & 0 \\ 0 & 34 & 0 & 130 \end{bmatrix}$$

Because of the checkerboard pattern of this matrix, its inverse can be obtained from the inverses of the two 2×2 interlaced submatrices:

$$\begin{bmatrix} 5 & 10 \\ 10 & 34 \end{bmatrix}^{-1} = \frac{1}{70}\begin{bmatrix} 34 & -10 \\ -10 & 5 \end{bmatrix}, \quad \begin{bmatrix} 10 & 34 \\ 34 & 130 \end{bmatrix}^{-1} = \frac{1}{144}\begin{bmatrix} 130 & -34 \\ -34 & 10 \end{bmatrix}$$

Interlacing these inverses, we obtain:

$$F^{-1} = \begin{bmatrix} 34/70 & 0 & -10/70 & 0 \\ 0 & 130/144 & 0 & -34/144 \\ -10/70 & 0 & 5/70 & 0 \\ 0 & -34/144 & 0 & 10/144 \end{bmatrix}$$

Then, we compute the derivative filter matrix G:

$$G = SF^{-1} = \frac{1}{35}\begin{bmatrix} -3 & 35/12 & 5 & -35/12 \\ 12 & -70/3 & -2.5 & 35/6 \\ 17 & 0 & -5 & 0 \\ 12 & 70/3 & -2.5 & -35/6 \\ -3 & -35/12 & 5 & 35/12 \end{bmatrix}$$

and the LPSM matrix B:

$$B = SG^T = \frac{1}{35}\begin{bmatrix} 34.5 & 2 & -3 & 2 & -0.5 \\ 2 & 27 & 12 & -8 & 2 \\ -3 & 12 & 17 & 12 & -3 \\ 2 & -8 & 12 & 27 & 2 \\ -0.5 & 2 & -3 & 2 & 34.5 \end{bmatrix}$$

As mentioned above, the steady-state LPSM filter $\mathbf{b}_0$ is the same as that of case $d = 2$. But, the transient and differentiation filters are different. □

23.3 Exact Design Equations

In practice, the most common values of d are $0, 1, 2, 3, 4$. For these ds and arbitrary filter lengths N, the LPSM matrix B can be constructed in closed form; see references [620–683], as well as the extensive tables in [638]. Denoting the inverse of the $(d+1)\times(d+1)$ matrix $F = S^T S$ by $\Phi = F^{-1}$, we can write

$$B = SF^{-1}S^T = S\Phi S^T = \sum_{i=0}^{d}\sum_{j=0}^{d} \mathbf{s}_i \mathbf{s}_j^T \Phi_{ij} \tag{23.3.1}$$

which gives for the mkth matrix element

$$\boxed{B_{mk}=\sum_{i=0}^{d}\sum_{j=0}^{d}s_i(m)s_j(k)\Phi_{ij}=\sum_{i=0}^{d}\sum_{j=0}^{d}m^ik^j\Phi_{ij}}\,,\qquad -M\le m,k\le M \tag{23.3.2}$$

Because of symmetry, $B_{mk}=B_{km}$, these matrix elements represent the kth component of the LPSM filter $\mathbf{b}_m$ or the mth component of the filter $\mathbf{b}_k$, that is,

$$B_{mk}=B_{km}=b_m(k)=b_k(m)=\sum_{i=0}^{d}\sum_{j=0}^{d}m^ik^j\Phi_{ij} \tag{23.3.3}$$

The matrix Φ can be determined easily for the cases $0\le d\le 4$. The matrix F is a *Hankel matrix*, that is, having the same entries along each antidiagonal line. Therefore, its matrix elements F_{ij} depend only on the sum $i+j$ of the indices. To see this, we write F_{ij} as the inner product:

$$F_{ij}=(S^TS)_{ij}=\mathbf{s}_i^T\mathbf{s}_j=\sum_{m=-M}^{M}s_i(m)s_j(m)=\sum_{m=-M}^{M}m^{i+j}\,,\qquad\text{or,}$$

$$\boxed{F_{ij}=\sum_{m=-M}^{M}m^{i+j}\equiv F_{i+j}}\,,\qquad 0\le i,j\le d \tag{23.3.4}$$

Note that because of the symmetric limits of summation, F_{i+j} will be zero whenever $i+j$ is odd. This leads to the checkerboard pattern of alternating zeros in F that we saw in the above examples. Also, because $d\le 4$, the only values of $i+j$ that we need are: $i+j=0,2,4,6,8$. For those, the summations over m can be done in closed form:

$$\begin{aligned}
F_0&=\sum_{m=-M}^{M}m^0=N=2M+1\\
F_2&=\sum_{m=-M}^{M}m^2=\frac{1}{3}M(M+1)(2M+1)\\
F_4&=\sum_{m=-M}^{M}m^4=\frac{1}{5}(3M^2+3M-1)F_2\\
F_6&=\sum_{m=-M}^{M}m^6=\frac{1}{7}(3M^4+6M^3-3M+1)F_2\\
F_8&=\sum_{m=-M}^{M}m^8=\frac{1}{15}(5M^6+15M^5+5M^4-15M^3-M^2+9M-3)F_2
\end{aligned} \tag{23.3.5}$$

We can express F in terms of these definitions for various values of d. For example, for $d=0,1,2,3$, the F matrices are:

$$[F_0]\,,\quad \begin{bmatrix}F_0&0\\0&F_2\end{bmatrix},\quad \begin{bmatrix}F_0&0&F_2\\0&F_2&0\\F_2&0&F_4\end{bmatrix},\quad \begin{bmatrix}F_0&0&F_2&0\\0&F_2&0&F_4\\F_2&0&F_4&0\\0&F_4&0&F_6\end{bmatrix}$$

The corresponding inverse matrices $\Phi=F^{-1}$ are obtained by interlacing the inverses of the checkerboard submatrices, as in Example 23.2.2. For $d=0,1,2$, we have for Φ:

$$[1/F_0]\,,\quad \begin{bmatrix}1/F_0&0\\0&1/F_2\end{bmatrix},\quad \begin{bmatrix}F_4/D_4&0&-F_2/D_4\\0&1/F_2&0\\-F_2/D_4&0&F_0/D_4\end{bmatrix},$$

and for $d=3$:

$$\Phi=F^{-1}=\begin{bmatrix}F_4/D_4&0&-F_2/D_4&0\\0&F_6/D_8&0&-F_4/D_8\\-F_2/D_4&0&F_0/D_4&0\\0&-F_4/D_8&0&F_2/D_8\end{bmatrix}$$

where the D_4 and D_8 are determinants of the interlaced submatrices:

$$\begin{aligned}
D_4&=F_0F_4-F_2^2=\frac{1}{45}M(M+1)(2M+1)(2M+3)(4M^2-1)\\
D_8&=F_2F_6-F_4^2=\frac{3}{35}M(M+2)(M^2-1)D_4
\end{aligned} \tag{23.3.6}$$

Inserting the above expressions for Φ into Eq. (23.3.3), we determine the corresponding LPSM filters. For $d=0$, we find for $-M\le m,k\le M$:

$$b_m(k)=B_{mk}=\frac{1}{F_0}=\frac{1}{N} \tag{23.3.7}$$

For $d=1$:

$$b_m(k)=B_{mk}=\frac{1}{F_0}+\frac{mk}{F_2} \tag{23.3.8}$$

For $d=2$:

$$b_m(k)=B_{mk}=\frac{F_4}{D_4}+\frac{1}{F_2}mk-\frac{F_2}{D_4}(m^2+k^2)+\frac{F_0}{D_4}m^2k^2 \tag{23.3.9}$$

For $d=3$:

$$\begin{aligned}
b_m(k)=B_{mk}=&\frac{F_4}{D_4}+\frac{F_6}{D_8}mk-\frac{F_2}{D_4}(m^2+k^2)+\frac{F_0}{D_4}m^2k^2\\
&-\frac{F_4}{D_8}(km^3+mk^3)+\frac{F_2}{D_8}m^3k^3
\end{aligned} \tag{23.3.10}$$

The required ratios are given explicitly as follows:

$$
\begin{aligned}
\frac{F_4}{D_4} &= \frac{3\,(3M^2+3M-1)}{(2M+3)\,(4M^2-1)}\\
\frac{F_2}{D_4} &= \frac{15}{(2M+3)\,(4M^2-1)}\\
\frac{F_0}{D_4} &= \frac{45}{M\,(M+1)\,(2M+3)\,(4M^2-1)}\\
\frac{F_6}{D_8} &= \frac{25\,(3M^4+6M^3-3M+1)}{M\,(M+2)\,(M^2-1)\,(2M+3)\,(4M^2-1)}\\
\frac{F_4}{D_8} &= \frac{35\,(3M^2+3M-1)}{M\,(M+2)\,(M^2-1)\,(2M+3)\,(4M^2-1)}\\
\frac{F_2}{D_8} &= \frac{175}{M\,(M+2)\,(M^2-1)\,(2M+3)\,(4M^2-1)}
\end{aligned}
\tag{23.3.11}
$$

In a similar fashion, we also find for the case $d = 4$:

$$
\begin{aligned}
b_m(k) = B_{mk} = \frac{D_{12}}{D} &+ \frac{F_6}{D_8}mk - \frac{D_{10}}{D}(m^2+k^2) + \frac{E_8}{D}m^2k^2\\
&- \frac{F_4}{D_8}(km^3+mk^3) + \frac{F_2}{D_8}m^3k^3 + \frac{D_8}{D}(m^4+k^4)\\
&- \frac{D_6}{D}(m^2k^4+k^2m^4) + \frac{D_4}{D}m^4k^4
\end{aligned}
\tag{23.3.12}
$$

where

$$
\begin{aligned}
D_6 &= F_0F_6 - F_2F_4 & E_8 &= F_0F_8 - F_4^2\\
D_{10} &= F_2F_8 - F_4F_6 & D_{12} &= F_4F_8 - F_6^2\\
D &= F_0D_{12} - F_2D_{10} + F_4D_8
\end{aligned}
\tag{23.3.13}
$$

These are given explicitly as follows:

$$
\begin{aligned}
D_6 &= \frac{1}{7}(6M^2+6M-5)D_4\\
D_{10} &= \frac{1}{21}M\,(M+2)\,(M^2-1)\,(2M^2+2M-3)D_4\\
E_8 &= \frac{1}{5}(4M^4+8M^3-4M^2-8M+1)D_4\\
D_{12} &= \frac{1}{735}M\,(M+2)\,(M^2-1)\,(15M^4+30M^3-35M^2-50M+12)D_4\\
D &= \frac{4}{11025}M\,(M+2)\,(M^2-1)\,(2M+5)\,(4M^2-9)\,(4M^2-1)D_4
\end{aligned}
\tag{23.3.14}
$$

and the required ratios are:

$$
\begin{aligned}
\frac{D_{12}}{D} &= \frac{15\,(15M^4+30M^3-35M^2-50M+12)}{4\,(2M+5)\,(4M^2-1)\,(4M^2-9)}\\
\frac{D_{10}}{D} &= \frac{525\,(2M^2+2M-3)}{4\,(2M+5)\,(4M^2-1)\,(4M^2-9)}\\
\frac{E_8}{D} &= \frac{2205\,(4M^4+8M^3-4M^2-8M+5)}{4M\,(M+2)\,(M^2-1)\,(2M+5)\,(4M^2-1)\,(4M^2-9)}\\
\frac{D_8}{D} &= \frac{945}{4\,(2M+5)\,(4M^2-1)\,(4M^2-9)}\\
\frac{D_6}{D} &= \frac{1575\,(6M^2+6M-5)}{4M\,(M+2)\,(M^2-1)\,(2M+5)\,(4M^2-1)\,(4M^2-9)}\\
\frac{D_4}{D} &= \frac{11025}{4M\,(M+2)\,(M^2-1)\,(2M+5)\,(4M^2-1)\,(4M^2-9)}
\end{aligned}
\tag{23.3.15}
$$

In this case, the matrix F and its two interlaced submatrices are:

$$
F = \begin{bmatrix} F_0 & 0 & F_2 & 0 & F_4\\ 0 & F_2 & 0 & F_4 & 0\\ F_2 & 0 & F_4 & 0 & F_6\\ 0 & F_4 & 0 & F_6 & 0\\ F_4 & 0 & F_6 & 0 & F_8 \end{bmatrix},\quad
\begin{bmatrix} F_0 & F_2 & F_4\\ F_2 & F_4 & F_6\\ F_4 & F_6 & F_8 \end{bmatrix},\quad
\begin{bmatrix} F_2 & F_4\\ F_4 & F_6 \end{bmatrix}
$$

Its inverse—obtained by interlacing the inverses of these two submatrices—can be expressed in terms of the determinant quantities of Eq. (23.3.13):

$$
\Phi = F^{-1} = \begin{bmatrix} D_{12}/D & 0 & -D_{10}/D & 0 & D_8/D\\ 0 & F_6/D_8 & 0 & -F_4/D_8 & 0\\ -D_{10}/D & 0 & E_8/D & 0 & -D_6/D\\ 0 & -F_4/D_8 & 0 & F_2/D_8 & 0\\ D_8/D & 0 & -D_6/D & 0 & D_4/D \end{bmatrix}
$$

Eqs. (23.3.5)–(23.3.15) provide closed-form expressions for the LPSM filters $b_m(k)$ of orders $d = 0, 1, 2, 3, 4$. Setting $m = 0$, we obtain the explicit forms of the steady-state filters $b_0(k)$, $-M \le k \le M$. For $d = 0, 1$:

$$
b_0(k) = \frac{1}{2M+1}
\tag{23.3.16}
$$

for $d = 2, 3$:

$$
b_0(k) = \frac{3\,(3M^2+3M-1-5k^2)}{(2M+3)\,(4M^2-1)}
\tag{23.3.17}
$$

and for $d = 4, 5$:

$$
b_0(k) = \frac{15\big(15M^4+30M^3-35M^2-50M+12-35\,(2M^2+2M-3)k^2+63k^4\big)}{4\,(2M+5)\,(4M^2-1)\,(4M^2-9)}
\tag{23.3.18}
$$

Example 23.3.1: Determine the quadratic/cubic LPSM filters of lengths $N = 5, 7, 9$. Using (23.3.17) with $M = 2, 3, 4$, we find (for $-M \le k \le M$):

$$b_0(k) = \frac{17 - 5k^2}{35} = \frac{1}{35}[-3, 12, 17, 12, -3]$$

$$b_0(k) = \frac{7 - k^2}{21} = \frac{1}{21}[-2, 3, 6, 7, 6, 3, -2]$$

$$b_0(k) = \frac{59 - 5k^2}{231} = \frac{1}{231}[-21, 14, 39, 54, 59, 54, 39, 14, -21]$$

where the coefficients have been reduced to integers as much as possible. □

Example 23.3.2: Determine the quartic and quintic LPSM filters of length $N = 7, 9$. Using Eq. (23.3.18) with $M = 3, 4$, we find:

$$b_0(k) = \frac{131 - 61.25k^2 + 5.25k^4}{231} = \frac{1}{231}[5, -30, 75, 131, 75, -30, 5]$$

$$b_0(k) = \frac{179 - 46.25k^2 + 2.25k^4}{429} = \frac{1}{429}[15, -55, 30, 135, 179, 135, 30, -55, 15]$$

23.4 Geometric Interpretation

The LPSM filters admit a nice *geometric* interpretation, which is standard in least-squares problems. Let $\mathbb{Y}$ be the vector space of the N-dimensional real-valued vectors $\mathbf{y}$, that is, the space $\mathbb{R}^N$, and let $\mathbb{S}$ be the $(d+1)$-dimensional *subspace* spanned by all linear combinations of the basis vectors $\mathbf{s}_i$, $i = 0, 1, \dots, d$.

Thus, the matrix $S = [\mathbf{s}_0, \mathbf{s}_1, \dots, \mathbf{s}_d]$ is a (non-orthogonal) basis of the subspace $\mathbb{S}$. The smoothed vector $\hat{\mathbf{y}}$, being a linear combination of the $\mathbf{s}_i$, belongs to the subspace $\mathbb{S}$. Moreover, because of the orthogonality equations (23.2.8), $\hat{\mathbf{y}}$ is *orthogonal* to the error vector $\mathbf{e}$:

$$\hat{\mathbf{y}}^T\mathbf{e} = (S\mathbf{c})^T\mathbf{e} = \mathbf{c}^T S^T\mathbf{e} = 0$$

Then, the equation $\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}}$ can be rewritten as the *orthogonal decomposition*:

$$\mathbf{y} = \hat{\mathbf{y}} + \mathbf{e} \tag{23.4.1}$$

which expresses $\mathbf{y}$ as a sum of a part that belongs to the subspace $\mathbb{S}$ and a part that belongs to the *orthogonal complement* subspace $\mathbb{S}^\perp$. The decomposition is *unique* and represents the *direct sum* decomposition of the full vector space $\mathbb{Y}$:

$$\mathbb{Y} = \mathbb{S} \oplus \mathbb{S}^\perp$$

This geometric interpretation requires that the dimension of the subspace $\mathbb{S}$ not exceed the dimension of the full space $\mathbb{Y}$, that is, $d + 1 \le N$. The component $\hat{\mathbf{y}}$ that lies in $\mathbb{S}$ is the *projection* of $\mathbf{y}$ onto $\mathbb{S}$. The matrix B in Eq. (23.2.11) is the corresponding *projection matrix*. As such, it will be symmetric, $B^T = B$, and *idempotent*:

$$B^2 = B \tag{23.4.2}$$

The proof is straightforward:

$$B^2 = (SF^{-1}S^T)(SF^{-1}S^T) = SF^{-1}(S^TS)F^{-1}S^T = SF^{-1}S^T = B$$

The matrix $(I - B)$, where I is the N-dimensional identity matrix, is also a projection matrix, projecting onto the orthogonal subspace $\mathbb{S}^\perp$. Thus, the error vector $\mathbf{e}$ belonging to $\mathbb{S}^\perp$ can be obtained from $\mathbf{y}$ by the projection:

$$\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}} = (I - B)\mathbf{y}$$

Because $(I - B)$ is also idempotent and symmetric, $(I - B)^2 = (I - B)$, we obtain for the *minimized* value of the performance index $\mathcal{J}$ of Eq. (23.2.6):

$$\mathcal{J}_{\min} = \mathbf{e}^T\mathbf{e} = \mathbf{y}^T(I - B)^2\mathbf{y} = \mathbf{y}^T(I - B)\mathbf{y} = \mathbf{y}^T\mathbf{y} - \mathbf{y}^T B\mathbf{y} \tag{23.4.3}$$

23.5 Orthogonal Polynomial Bases

Computationally, the non-orthogonal basis $S = [\mathbf{s}_0, \mathbf{s}_1, \dots, \mathbf{s}_d]$ is not the most convenient one. The Gram-Schmidt orthogonalization process may be applied to the columns of S to obtain an orthogonal basis. This procedure amounts to performing the QR-factorization[†] on S, that is,

$$S = QR \tag{23.5.1}$$

where Q is an $N\times(d+1)$ matrix with orthonormal columns, that is, $Q^TQ = I$, and R is a $(d+1)\times(d+1)$ non-singular *upper-triangular* matrix.

The columns of $Q = [\mathbf{q}_0, \mathbf{q}_1, \dots, \mathbf{q}_d]$, correspond to the (orthonormalized) discrete Chebyshev or Gram polynomials $q_i(n)$, $i = 0, 1, \dots, d$, constructed from the monomial basis $s_i(n) = n^i$ by the Gram-Schmidt process. Noting that $S^TS = R^T(Q^TQ)R = R^TR$, the design of the filter matrices B, G can be formulated more efficiently as follows:

$$\begin{aligned} F &= S^TS = R^TR \\ G &= SF^{-1} = QR^{-T} \\ B &= SF^{-1}S^T = QQ^T \end{aligned} \tag{23.5.2}$$

which lead to the explicit construction of the differentiation and LPSM filters in terms of the Chebyshev polynomials $q_i(n)$:

$$\begin{aligned} \mathbf{g}_i &= \sum_{j=0}^{i} \mathbf{q}_j\,(R^{-1})_{ij} \quad\Rightarrow\quad g_i(n) = \sum_{j=0}^{i} q_j(n)\,(R^{-1})_{ij} \\ B &= \sum_{i=0}^{d} \mathbf{q}_i\,\mathbf{q}_i^T \quad\Rightarrow\quad b_m(k) = B_{km} = \sum_{i=0}^{d} q_i(k)\,q_i(m) \end{aligned} \tag{23.5.3}$$

The expression for $b_m(k)$ can be simplified further using the Christoffel-Darboux identity for orthogonal polynomials. We discuss these matters further in Sec. 23.13. The MATLAB function `lpsm` implements (23.5.2). Its inputs are N, d and its outputs B, G:

[†] see [45].

```
[B,G] = lpsm(N,d);        % local polynomial smoothing and differentiation filter design
```

The function constructs the basis matrix S with the help of the function `lpbasis` and carries out its QR-factorization with the help of the built-in function `qr`. The following code fragment illustrates the computational steps:

```
S = lpbasis(N,d);        % construct polynomial basis
[Q,R] = qr(S, 0);        % economy form, R is (d+1)x(d+1) upper triangular
G = Q/R';                % differentiation filters
B = Q*Q';                % smoothing filters
```

23.6 Polynomial Predictive and Interpolation Filters

The case $d+1=N$ or $d=N-1$ is of special interest, corresponding to ordinary polynomial *Lagrange interpolation*. Indeed, in this case, the basis matrix S becomes a square non-singular $N{\times}N$ matrix with an ordinary inverse S^{-1}, which implies that B becomes the identity matrix,

$$B = S(S^TS)^{-1}S^T = S(S^{-1}S^{-T})S^T = I$$

or, equivalently, the subspace $\mathbb{S}$ becomes the full space $\mathbb{Y}$. The optimal polynomial of degree $d=N-1$ fits through all the sample points of the N-dimensional vector $\mathbf{y}$, that is, $\mathbf{e}=0$ or $\hat{\mathbf{y}}=\mathbf{y}=S\mathbf{c}$, with solution $\mathbf{c}=S^{-1}\mathbf{y}$, and interpolates between those samples. This polynomial is defined for any independent variable t by:

$$\hat{y}_t = \sum_{i=0}^{N-1} c_i t^i = \mathbf{c}^T\mathbf{u}_t = \mathbf{y}^T S^{-T}\mathbf{u}_t \equiv \mathbf{y}^T\mathbf{b}_t = \sum_{k=-M}^{M} b_t(k)\,y_k \tag{23.6.1}$$

where we set,

$$\mathbf{u}_t = \begin{bmatrix} 1 \\ t \\ \vdots \\ t^{N-1} \end{bmatrix}, \quad \mathbf{b}_t = S^{-T}\mathbf{u}_t \quad \Rightarrow \quad b_t(k) = \sum_{i=0}^{N-1}(S^{-1})_{ik}\,t^i \tag{23.6.2}$$

The polynomials $b_t(k)$ of degree $(N-1)$ in t are the ordinary Lagrange interpolation polynomials, interpolating through the points y_k. To see this, we note that at each discrete value of t, say $t=m$ with $-M \le m \le M$, we have:

$$b_m(k) = \sum_{i=0}^{N-1}(S^{-1})_{ik}\,m^i = \sum_{i=0}^{N-1}(S^{-1})_{ik}\,S_{mi} = (SS^{-1})_{mk} = I_{mk} = \delta(m-k) \tag{23.6.3}$$

so that the polynomial passes through the signal values at the sampling instants:

$$\hat{y}_t\big|_{t=m} = \sum_{k=-M}^{M} b_m(k)\,y_k = \sum_{k=-M}^{M}\delta(m-k)\,y_k = y_m$$

It is straightforward to show using the property (23.6.3) that $b_t(k)$ is given by the usual Lagrange interpolation formula:

$$b_t(k) = \prod_{\substack{m=-M \\ m\ne k}}^{M}\left(\frac{t-m}{k-m}\right), \qquad -M \le k \le M \tag{23.6.4}$$

Indeed, Eq. (23.6.4) states that the $(2M)$ roots of $b_t(k)$ are the points $t=m$, for $-M \le m \le M$ and $m \ne k$, which fixes the polynomial up to a constant. That constant is determined by the condition $b_k(k)=1$.

Example 23.6.1: For $N=5$ and $d=N-1=4$, the fourth degree Lagrange polynomials, constructed from Eq. (23.6.4), can be expanded in powers of t:

$$\begin{bmatrix} b_t(-2) \\ b_t(-1) \\ b_t(0) \\ b_t(1) \\ b_t(2) \end{bmatrix} = \frac{1}{24}\begin{bmatrix} 0 & 2 & -1 & -2 & 1 \\ 0 & -16 & 16 & 4 & -4 \\ 24 & 0 & -30 & 0 & 6 \\ 0 & 16 & 16 & -4 & -4 \\ 0 & -2 & -1 & 2 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ t^1 \\ t^2 \\ t^3 \\ t^4 \end{bmatrix}$$

The coefficient matrix is recognized as the inverse transposed of the basis matrix S:

$$S = \begin{bmatrix} 1 & -2 & 4 & -8 & 16 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 16 \end{bmatrix} \quad \Rightarrow \quad S^{-T} = \frac{1}{24}\begin{bmatrix} 0 & 2 & -1 & -2 & 1 \\ 0 & -16 & 16 & 4 & -4 \\ 24 & 0 & -30 & 0 & 6 \\ 0 & 16 & 16 & -4 & -4 \\ 0 & -2 & -1 & 2 & 1 \end{bmatrix}$$

which verifies Eq. (23.6.2). □

We note that $b_t(k)$ can be written in the following analytical form, which shows the relation of the Lagrange interpolation filter to the ideal sinc-interpolation filter:

$$b_t(k) = \frac{\Gamma(M+1+t)\,\Gamma(M+1-t)}{\Gamma(M+1+k)\,\Gamma(M+1-k)}\cdot\frac{\sin\big(\pi(t-k)\big)}{\pi(t-k)} \tag{23.6.5}$$

Some alternative expressions are as follows:

$$b_t(k) = (-1)^{M+k}\sum_{m=M+k}^{2M}\binom{M+t}{m}\binom{m}{M+k}(-1)^m \tag{23.6.6}$$

$$b_t(k) = \frac{(-1)^{M+1-k}\,\Gamma(M+1-t)}{(t-k)\,\Gamma(-M-t)\,\Gamma(M+1+k)\,\Gamma(M+1-k)} \tag{23.6.7}$$

and since the $b_t(k)$ sum up to one, we also have [740]:

$$b_t(k) = \left[\sum_{n=-M}^{M}\frac{b_t(n)}{b_t(k)}\right]^{-1} = \left[\sum_{n=-M}^{M}(-1)^{k-n}\,\frac{(M+k)!\,(M-k)!}{(M+n)!\,(M-n)!}\,\frac{t-k}{t-n}\right]^{-1} \tag{23.6.8}$$

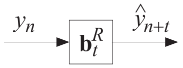
For polynomial orders $d < N-1$, one can still interpolate approximately and smoothly between the samples y_m. In this case, using $\mathbf{c} = G^T\mathbf{y} = (S^TS)^{-1}S^T\mathbf{y}$, we have:

$$\hat{y}_t = \sum_{i=0}^{d} c_i t^i = \mathbf{c}^T\mathbf{u}_t = \mathbf{y}^T G\mathbf{u}_t \equiv \mathbf{y}^T\mathbf{b}_t = \sum_{k=-M}^{M} b_t(k)y_k \qquad (23.6.9)$$

where now

$$\mathbf{u}_t = \begin{bmatrix} 1 \\ t^1 \\ t^2 \\ \vdots \\ t^d \end{bmatrix}, \quad \mathbf{b}_t = G\mathbf{u}_t = S(S^TS)^{-1}\mathbf{u}_t \quad \Rightarrow \quad b_t(k) = \sum_{i=0}^{d} G_{ki}\, t^i \qquad (23.6.10)$$

and shifting the origin $k = 0$ to the arbitrary time instant n, we obtain the interpolation formula for a shift t relative to the time instant n:



$$\boxed{\hat{y}_{n+t} = \sum_{k=-M}^{M} b_t(k)y_{n+k} = \sum_{k=-M}^{M} b_t^R(k)y_{n-k}} \qquad (23.6.11)$$

where $b_t^R(k) = b_t(-k)$. Such formulas can also be used for prediction by choosing $t > M$ so that $n + t > n + M$, that is, it lies beyond the end of the filter range.

We can obtain closed-form expressions for the interpolation filters $b_t(k)$ for $d = 0, 1, 2, 3, 4$ and arbitrary M, by replacing in Eqs. (23.3.7)–(23.3.12) the variable m in $b_m(k)$ by the variable t. For example, for $d = 1, 2, 3, 4$, we have, respectively:

$$\begin{aligned}
b_t(k) &= \frac{1}{F_0} + \frac{tk}{F_2} \\
b_t(k) &= \frac{F_4}{D_4} + \frac{1}{F_2}tk - \frac{F_2}{D_4}(t^2 + k^2) + \frac{F_0}{D_4}t^2k^2 \\
b_t(k) &= \frac{F_4}{D_4} + \frac{F_6}{D_8}tk - \frac{F_2}{D_4}(t^2 + k^2) + \frac{F_0}{D_4}t^2k^2 - \frac{F_4}{D_8}(kt^3 + tk^3) + \frac{F_2}{D_8}t^3k^3 \qquad (23.6.12) \\
b_t(k) &= \frac{D_{12}}{D} + \frac{F_6}{D_8}tk - \frac{D_{10}}{D}(t^2 + k^2) + \frac{E_8}{D}t^2k^2 - \frac{F_4}{D_8}(kt^3 + tk^3) \\
&\quad + \frac{F_2}{D_8}t^3k^3 + \frac{D_8}{D}(t^4 + k^4) - \frac{D_6}{D}(t^2k^4 + k^2t^4) + \frac{D_4}{D}t^4k^4
\end{aligned}$$

where the required coefficient ratios are given by Eqs. (23.3.11) and (23.3.15). The interpolation filter (23.6.10) may be written in terms of the columns of $G = [\mathbf{g}_0, \mathbf{g}_1, \dots, \mathbf{g}_d]$:

$$b_t(k) = \sum_{i=0}^{d} G_{ki}t^i = \sum_{i=0}^{d} g_i(k)t^i \quad \Rightarrow \quad \mathbf{b}_t = \sum_{i=0}^{d} \mathbf{g}_i t^i \qquad (23.6.13)$$

23.7 Farrow Realization Structures

This representation admits a convenient realization, known as a *Farrow structure*, which allows the changing of the parameter t on the fly without having to redesign the filter. It is essentially a block-diagram realization of Eq. (23.6.13) written in nested form using Hörner's rule. For example, if $d = 3$, we have

$$\mathbf{b}_t = \mathbf{g}_0 + \mathbf{g}_1 t + \mathbf{g}_2 t^2 + \mathbf{g}_3 t^3 = \big((\mathbf{g}_3 t + \mathbf{g}_2)t + \mathbf{g}_1\big)t + \mathbf{g}_0 \qquad (23.7.1)$$

Fig. 23.7.1 shows this realization where we replaced $\mathbf{g}_i$ by their reversed versions $\mathbf{g}_i^R$, which appear in the convolutional filtering equations. The parameter t appears only in the lower multipliers and can be independently controlled.

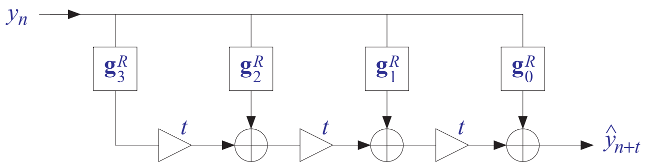


Fig. 23.7.1 Farrow structure for interpolating or predictive FIR filter.

The filtering equation (23.6.11) can also be written in a causal manner by setting $t = M + \tau$ and defining the causal filter, where $N = 2M + 1$:

$$\boxed{h_\tau(k) = b_{M+\tau}(M - k)}, \quad k = 0, 1, \dots, N - 1 \qquad (23.7.2)$$

Replacing $n \to n - M$ and $k \to k - M$, Eq. (23.6.11) is transformed into a *causal* filtering operation that predicts the future sample $y_{n+\tau}$ from the present and past samples y_{n-k}, $k = 0, 1, \dots, N - 1$. The mapping of the time indices is explained in Fig. 23.7.2. The resulting filtering operation reads:

$$\boxed{\hat{y}_{n+\tau} = \sum_{k=0}^{N-1} h_\tau(k)y_{n-k}}, \quad \tau \ge 0 \qquad (23.7.3)$$

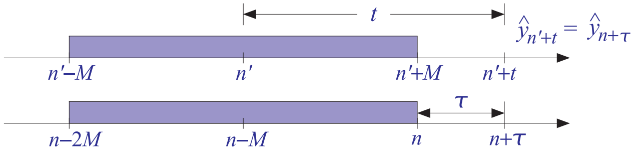


Fig. 23.7.2 Double-sided and causal predictive FIR filters, with $n' = n - M$ and $t = M + \tau$.

Since τ is any real number, the notation $n + \tau$ corresponds to the actual time instant $(n + \tau)T$ in seconds, where T is the sampling time interval. The filter $h_{-\tau}(k)$ may also be used for implementing a *fractional delay* as opposed to prediction, that is,

$$\boxed{\hat{y}_{n-\tau} = \sum_{k=0}^{N-1} h_{-\tau}(k) y_{n-k}} \quad \text{(fractional delay)} \tag{23.7.4}$$

The filters $b_t(k)$ and $h_\tau(k)$ satisfy the following polynomial-preserving moment constraints (being equivalent to $S^T\mathbf{b}_t = \mathbf{u}_t$), where $i = 0, 1, \dots, d$:

$$\sum_{k=-M}^{M} k^i b_t(k) = t^i \quad \Rightarrow \quad \sum_{k=0}^{N-1} k^i h_\tau(k) = (-\tau)^i\,, \quad \sum_{k=0}^{N-1} k^i h_{-\tau}(k) = \tau^i \tag{23.7.5}$$

These constraints imply that Eqs. (23.7.3) and (23.7.4) are exact for polynomials of degree $r \le d$. For any such polynomial $P(n)$, we have:

$$\sum_{k=0}^{N-1} h_{-\tau}(k) P(n-k) = P(n-\tau) \tag{23.7.6}$$

For example, we have for the monomial $P(n) = n^r$ with $r \le d$:

$$\begin{aligned}\sum_{k=0}^{N-1} h_{-\tau}(k)\,(n-k)^r &= \sum_{k=0}^{N-1} h_{-\tau}(k) \sum_{i=0}^{r} \binom{r}{i} n^{r-i}(-1)^i k^i \\ &= \sum_{i=0}^{r} \binom{r}{i} n^{r-i}(-1)^i \sum_{k=0}^{N-1} k^i h_{-\tau}(k) = \sum_{i=0}^{r} \binom{r}{i} n^{r-i}(-1)^i \tau^i = (n-\tau)^r\end{aligned}$$

It is in the sense of Eq. (23.7.6) that we may think of the transfer function of the filter $h_{-\tau}(k)$ as approximating the ideal fractional delay $z^{-\tau}$:

$$\sum_{k=0}^{N-1} h_{-\tau}(k) z^{-k} \simeq z^{-\tau} \tag{23.7.7}$$

Further insight into the nature of the approximation (23.7.7) can be gained by considering the Lagrange interpolation case, $d = N-1$. From the definition of $h_{-\tau}(k) = b_{M-\tau}(M-k)$ and Eqs. (23.6.4) and (23.6.6), we obtain, for $k = 0, 1, \dots, N-1$:

$$h_{-\tau}(k) = \prod_{\substack{i=0 \\ i\ne k}}^{N-1} \left(\frac{\tau - i}{k - i}\right) = \sum_{i=N-1-k}^{N-1} \binom{N-1-\tau}{i} \binom{i}{N-1-k} (-1)^{i-(N-1-k)} \tag{23.7.8}$$

The z-transform of $h_{-\tau}(k)$ is then,

$$\begin{aligned}\sum_{k=0}^{N-1} h_{-\tau}(k) z^{-k} &= \sum_{k=0}^{N-1} \sum_{i=N-1-k}^{N-1} \binom{N-1-\tau}{i} \binom{i}{N-1-k} (-1)^{i-(N-1-k)} z^{-k} \\ &= z^{-(N-1)} \sum_{i=0}^{N-1} \sum_{k=N-1-i}^{N-1} \binom{N-1-\tau}{i} \binom{i}{N-1-k} (-1)^{i-(N-1-k)} z^{N-1-k}\end{aligned}$$

Changing summation variables and using the binomial expansion of $(z-1)^i$, we obtain,

$$\begin{aligned}\sum_{k=0}^{N-1} h_{-\tau}(k) z^{-k} &= z^{-(N-1)} \sum_{i=0}^{N-1} \sum_{j=0}^{m} \binom{N-1-\tau}{i} \binom{i}{j} (-1)^{i-j} z^j \\ &= z^{-(N-1)} \sum_{i=0}^{N-1} \binom{N-1-\tau}{i} (z-1)^i\end{aligned} \tag{23.7.9}$$

Applying the binomial identity,

$$(1+x)^\alpha = \sum_{m=0}^{\infty} \binom{\alpha}{i} x^i \tag{23.7.10}$$

with $x = z-1$ and $\alpha = N-1-\tau$, we have,

$$z^{N-1-\tau} = (1+z-1)^{N-1-\tau} = \sum_{i=0}^{\infty} \binom{N-1-\tau}{i} (z-1)^i \tag{23.7.11}$$

We recognize the sum in Eq. (23.7.9) to be the first N terms of (23.7.11). Thus, taking that sum to approximately represent $z^{N-1-\tau}$, we have,

$$\sum_{k=0}^{N-1} h_{-\tau}(k) z^{-k} \simeq z^{-(N-1)}\, z^{N-1-\tau} = z^{-\tau} \tag{23.7.12}$$

This approximation becomes exact whenever τ is an integer, say $\tau = m$, with $m = 0, 1, \dots, N-1$. Indeed in this case, the summation range $0 \le i \le N-1$ in Eq. (23.7.9) can be restricted to $0 \le i \le N-1-m$ because the binomial coefficient vanishes whenever its (integer) arguments satisfy $N-1-m < i \le N-1$. We then have an ordinary binomial expansion for an integer power:

$$\sum_{k=0}^{N-1} h_{-m}(k) z^{-k} = z^{-(N-1)} \sum_{i=0}^{N-1-m} \binom{N-1-m}{i} (z-1)^i = z^{-(N-1)} (1+z-1)^{N-1-m} = z^{-m}$$

which implies the expected result $h_{-m}(k) = \delta(k-m)$. Eq. (23.7.9) is equivalent to *Newton's forward interpolation* formula. To see this, let us introduce the forward difference operator $\Delta = z-1$, or, $\Delta f_n = f_{n+1} - f_n$, and apply (23.7.9) in the time domain:

$$\hat{y}_{n-\tau} = \sum_{k=0}^{N-1} h_{-\tau}(k) y_{n-k} = \sum_{i=0}^{N-1} \binom{N-1-\tau}{i} \Delta^i y_{n-(N-1)} \tag{23.7.13}$$

This interpolates between the points $[y_{n-(N-1)}, \dots, y_{n-1}, y_n]$ with τ measured backwards from the end-point y_n. We may measure the interpolation distance forward from the first point $y_{n-(N-1)}$ by defining $x = N-1-\tau$. Then, Eq. (23.7.13) reads,

$$\hat{y}_{n-(N-1)+x} = \sum_{i=0}^{N-1} \binom{x}{i} \Delta^i y_{n-(N-1)} \tag{23.7.14}$$

and setting $n = N-1$ so that the data range is $[y_0, y_1, \dots, y_{N-1}]$, we obtain the usual way of writing Newton's polynomial interpolation formula:

$$\hat{y}_x = \sum_{i=0}^{N-1} \binom{x}{i} \Delta^i y_0 = \sum_{i=0}^{N-1} \frac{x(x-1)\cdots(x-i+1)}{i!}\,\Delta^i y_0 \tag{23.7.15}$$

We note also that Eq. (23.7.8) is valid for either even or odd values of N. For $N = 2, 3, 4$, we obtain for the corresponding filter coefficients:

$$\begin{bmatrix} h_{-\tau}(0) \\ h_{-\tau}(1) \end{bmatrix} = \begin{bmatrix} 1-\tau \\ \tau \end{bmatrix}, \quad \begin{bmatrix} h_{-\tau}(0) \\ h_{-\tau}(1) \\ h_{-\tau}(2) \end{bmatrix} = \frac{1}{2}\begin{bmatrix} (\tau-1)(\tau-2) \\ -2\tau(\tau-2) \\ \tau(\tau-1) \end{bmatrix}$$

$$\begin{bmatrix} h_{-\tau}(0) \\ h_{-\tau}(1) \\ h_{-\tau}(2) \\ h_{-\tau}(3) \end{bmatrix} = \frac{1}{6}\begin{bmatrix} -(\tau-1)(\tau-2)(\tau-3) \\ 3\tau(\tau-2)(\tau-3) \\ -3\tau(\tau-1)(\tau-3) \\ \tau(\tau-1)(\tau-2) \end{bmatrix} \tag{23.7.16}$$

and the corresponding interpolation formulas:

$$\begin{aligned} \hat{y}_{n-\tau} &= (1-\tau)y_n + \tau y_{n-1} \\ \hat{y}_{n-\tau} &= \frac{1}{2}(\tau-1)(\tau-2)y_n - \tau(\tau-2)y_{n-1} + \frac{1}{2}\tau(\tau-1)y_{n-2} \\ \hat{y}_{n-\tau} &= -\frac{1}{6}(\tau-1)(\tau-2)(\tau-3)y_n + \frac{1}{2}\tau(\tau-2)(\tau-3)y_{n-1} \\ &\quad - \frac{1}{2}\tau(\tau-1)(\tau-3)y_{n-2} + \frac{1}{6}\tau(\tau-1)(\tau-2)y_{n-3} \end{aligned} \tag{23.7.17}$$

Example 23.7.1: Fig. 23.7.3 shows in the top row an example of a Lagrange fractional-delay filter with $N = 3$ and polynomial order $d = N-1 = 2$ for the delay values $\tau = m/10$, $m = 1, 2, \dots, 10$.

The bottom row is the case $N = 5$ and $d = N-1 = 4$ with delays τ extending over the interval $0 \le \tau \le 2$. This filter interpolates between the samples $[y_{n-4}, y_{n-3}, y_{n-2}, y_{n-1}, y_n]$. The chosen range of τ's spans the gaps between $[y_{n-2}, y_{n-1}, y_n]$. For the subrange $0 \le \tau \le 1$ which spans $[y_{n-1}, y_n]$, the magnitude response is greater than one, while it is less than one for the more central range $1 \le \tau \le 2$ which spans $[y_{n-2}, y_{n-1}]$. The following MATLAB code segment illustrates the generation of the upper two graphs:

```
f = linspace(0,1,1001); w= pi*f;          % frequencies 0 ≤ ω ≤ π
N=3; d=N-1; M = floor(N/2);               % d = N−1 for Lagrange interpolation

Hmag = []; Hdel = [];
for m=1:10,
   tau = m/10;                            % desired delays
   h = flip(lpinterp(N,d,M-tau));         % lpinterp is discussed in Sec. 23.9
   H = freqz(h,1,w);
   Hmag = [Hmag; 10*log10(abs(H))];       % magnitude responses in dB
   Delay = -angle(H)./w; Delay(1) = tau;
   Hdel = [Hdel; Delay];                  % phase delays
```

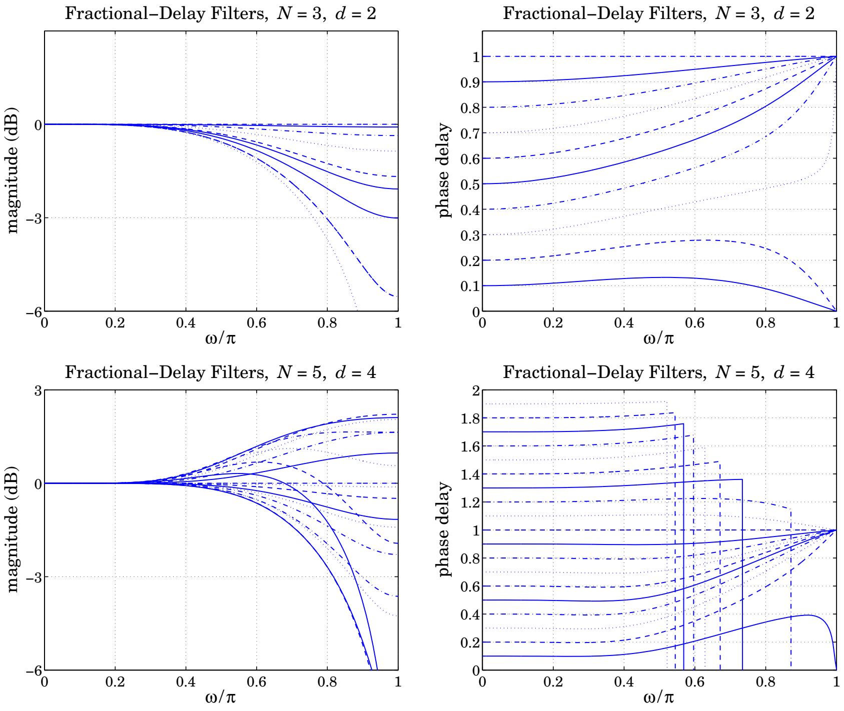


Fig. 23.7.3 Lagrange fractional-delay filters with $N = 3$.

```
end

figure; plot(f,Hmag); figure; plot(f,Hdel);
```

The filters were calculated with the function `lpinterp` (from Sec. 23.9) with arguments $d = N-1$, $t = M - \tau$, with reversed output to account for the definition $h_{-\tau}(k) = b_{M-\tau}(M-k)$.

In both cases, we observe that the useful bandwidth of operation, within which both the phase delays have the correct values and the magnitude response is near unity, is fairly narrow extending to about $\omega = 0.2\pi$, or $f = f_s/10$ in units of the sampling rate f_s. □

References [736–757] contain further information on predictive FIR and fractional-delay filters. See also [758–771] for alternative implementations of fractional delay using maximally-flat and allpass filters. Ref. [746] provides a nice review of various approaches to the fractional-delay problem.

23.8 Minimum Variance Filters

Next we discuss the *equivalence* of the least-square polynomial fitting approach to the minimization of the NRR subject to linear moment constraints. In the actuarial context, such designs are referred to as “minimum $\mathcal{R}_0$” or “minimum variance” filters, as opposed to the “minimum $\mathcal{R}_s$” or “minimum roughness” filters— the nomenclature being explained in Sec. 23.12.

The projection properties of B may be used to calculate the NRR. For example, the property mentioned previously that the NRR of the filter $\mathbf{b}_0$ is the equal to the middle value $b_0(0)$ follows from Eq. (23.4.2). Using the symmetry of B, we have

$$B^T = B = B^2 = B^T B$$

Taking matrix elements, we have $B_{km} = (B^T)_{mk} = (B^T B)_{mk}$. But, B_{km} is the kth component of the mth column $\mathbf{b}_m$. Using a similar argument as in Eq. (23.2.13), we also have $(B^T B)_{mk} = \mathbf{b}_m^T \mathbf{b}_k$. Therefore,

$$\mathbf{b}_m^T \mathbf{b}_k = b_m(k)$$

For $k = m$, we have the diagonal elements of $B^T B = B$:

$$\mathcal{R} = \mathbf{b}_m^T \mathbf{b}_m = b_m(m) \tag{23.8.1}$$

These are recognized as the NRRs of the filters $\mathbf{b}_m$. In particular, for $m = 0$, we have $\mathcal{R} = \mathbf{b}_0^T \mathbf{b}_0 = b_0(0)$. Setting $k = 0$ in Eqs. (23.3.16)–(23.3.18), we find that the NRRs of the cases $d = 0, 1$, $d = 2, 3$, and $d = 4, 5$ are given by the coefficient ratios $1/F_0$, F_4/D_4, and D_{12}/D. Therefore:

$$\begin{aligned}
&(d = 0, 1) \qquad \mathcal{R} = \frac{1}{N} \\
&(d = 2, 3) \qquad \mathcal{R} = \frac{3(3M^2 + 3M - 1)}{(2M + 3)(4M^2 - 1)} \\
&(d = 4, 5) \qquad \mathcal{R} = \frac{15(15M^4 + 30M^3 - 35M^2 - 50M + 12)}{4(2M + 5)(4M^2 - 1)(4M^2 - 9)}
\end{aligned} \tag{23.8.2}$$

In the limit of large N or M, we have the approximate asymptotic expressions:

$$\begin{aligned}
&(d = 0, 1) \qquad \mathcal{R} = \frac{1}{N} \\
&(d = 2, 3) \qquad \mathcal{R} \simeq \frac{9/4}{N} = \frac{2.25}{N} \\
&(d = 4, 5) \qquad \mathcal{R} \simeq \frac{225/64}{N} = \frac{3.52}{N}
\end{aligned} \tag{23.8.3}$$

Thus, the noise reductions achieved by the quadratic/cubic and quartic/quintic cases are 2.25 and 3.52 times worse than that of the plain FIR averager of the same length N. Another consequence of the projection nature of B is:

$$B^T S = S, \qquad S^T B = S^T \tag{23.8.4}$$

Indeed, $B^T S = BS = S(S^T S)^{-1} S^T S = S$. Column-wise the first equation states that:

$$B^T[\mathbf{s}_0, \mathbf{s}_1, \dots, \mathbf{s}_d] = [\mathbf{s}_0, \mathbf{s}_1, \dots, \mathbf{s}_d] \qquad \Rightarrow \qquad B^T \mathbf{s}_i = \mathbf{s}_i, \quad i = 0, 1, \dots, d$$

Thus, the basis vectors $\mathbf{s}_i$ remain *invariant* under projection, but that is to be expected because they already lie in $\mathbb{S}$. In fact, any other linear combination of them, such as Eq. (23.2.30), remains invariant under B, that is, $B^T \hat{\mathbf{y}} = \hat{\mathbf{y}}$.

This property answers the question: When are the smoothed values equal to the original ones, $\hat{\mathbf{y}} = \mathbf{y}$, or, equivalently, when is the error zero, $\mathbf{e} = 0$? Because $\mathbf{e} = \mathbf{y} - B^T \mathbf{y}$, the error will be zero if and only if $B^T \mathbf{y} = \mathbf{y}$, which means that $\mathbf{y}$ already *lies* in $\mathbb{S}$, that is, it is a linear combination of $\mathbf{s}_i$. This implies that the samples y_m are already dth order polynomial functions of m, as in Eq. (23.2.27).

The second equation in (23.8.4) implies certain *constraints* on the filters $\mathbf{b}_m$, which can be used to develop an alternative approach to the LPSM filter design problem in terms of minimizing the NRR subject to constraints. To see this, we write the $(d+1)\times N$ transposed matrix S^T column-wise:

$$S^T = [\mathbf{u}_{-M}, \dots, \mathbf{u}_{-1}, \mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_M] \tag{23.8.5}$$

For example, in the $N = 5$, $d = 2$ case, we have:

$$S^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -2 & -1 & 0 & 1 & 2 \\ 4 & 1 & 0 & 1 & 4 \end{bmatrix} \equiv [\mathbf{u}_{-2},\, \mathbf{u}_{-1},\, \mathbf{u}_0,\, \mathbf{u}_1,\, \mathbf{u}_2]$$

It is easily verified that the mth column $\mathbf{u}_m$ is simply

$$\mathbf{u}_m = \begin{bmatrix} 1 \\ m \\ m^2 \\ \vdots \\ m^d \end{bmatrix}, \qquad -M \le m \le M \tag{23.8.6}$$

which is the same as $\mathbf{u}_t$ at $t = m$, in terms of the definition (23.6.10). Using $B = GS^T$, we can express the LPSM filters $\mathbf{b}_m$ in terms of $\mathbf{u}_m$, as follows:

$$[\mathbf{b}_{-M}, \dots, \mathbf{b}_{-1}, \mathbf{b}_0, \mathbf{b}_1, \dots, \mathbf{b}_M] = B = GS^T = G[\mathbf{u}_{-M}, \dots, \mathbf{u}_{-1}, \mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_M]$$

which implies:

$$\mathbf{b}_m = G\mathbf{u}_m = SF^{-1}\mathbf{u}_m \tag{23.8.7}$$

Multiplying by S^T, we find $S^T \mathbf{b}_m = S^T S F^{-1} \mathbf{u}_m = \mathbf{u}_m$, or,

$$S^T \mathbf{b}_m = \mathbf{u}_m \qquad \Rightarrow \qquad \begin{bmatrix} \mathbf{s}_0^T \mathbf{b}_m \\ \mathbf{s}_1^T \mathbf{b}_m \\ \vdots \\ \mathbf{s}_d^T \mathbf{b}_m \end{bmatrix} = \begin{bmatrix} 1 \\ m \\ \vdots \\ m^d \end{bmatrix} \tag{23.8.8}$$

These relationships are the column-wise equivalent of $S^T B = S^T$. Thus, each LPSM filter $\mathbf{b}_m$ satisfies $(d+1)$ linear constraints:

$$\mathbf{s}_i^T \mathbf{b}_m = m^i, \qquad i = 0, 1, \dots, d \tag{23.8.9}$$

Writing the dot products explicitly, we have equivalently:

$$\boxed{\sum_{n=-M}^{M} n^i b_m(n) = m^i}\,,\qquad i = 0, 1, \dots, d \tag{23.8.10}$$

In particular, for the steady-state LPSM filter $\mathbf{b}_0$, we have $\mathbf{u}_0 = [1, 0, 0, \dots, 0]^T$, with ith component $\delta(i)$. Therefore, the constraint $S^T\mathbf{b}_0 = \mathbf{u}_0$ reads component-wise:

$$\boxed{\sum_{n=-M}^{M} n^i b_0(n) = \delta(i)\,,\quad i = 0, 1, \dots, d} \tag{23.8.11}$$

For $i = 0$, this is the usual DC constraint:

$$\sum_{n=-M}^{M} b_0(n) = 1 \tag{23.8.12}$$

and for $i = 1, 2, \dots, d$:

$$\sum_{n=-M}^{M} n^i b_0(n) = 0 \tag{23.8.13}$$

The quantity in the left-hand side of Eq. (23.8.11) is called the ith *moment* of the impulse response $b_0(n)$. Because of the symmetric limits of summation over n and the symmetry of $b_0(n)$ about its middle, the moments (23.8.13) will be zero for odd i, and therefore are not extra constraints. However, for even i, they are nontrivial constraints.

These moments are related to the *derivatives* of the frequency response at $\omega = 0$. Indeed, defining,

$$B_0(\omega) = \sum_{n=-M}^{M} b_0(n)e^{-j\omega n}$$

and differentiating it i times, we have:

$$j^i B_0^{(i)}(\omega) = j^i \frac{d^i B_0(\omega)}{d\omega^i} = \sum_{n=-M}^{M} n^i b_0(n)e^{-j\omega n}$$

Setting $\omega = 0$, we obtain:

$$j^i B_0^{(i)}(0) = j^i \frac{d^i B_0(\omega)}{d\omega^i}\bigg|_{\omega=0} = \sum_{n=-M}^{M} n^i b_0(n) \tag{23.8.14}$$

Thus, the moment constraints (23.8.12) and (23.8.13) are equivalent to the DC constraint and the *flatness* constraints on the frequency response at $\omega = 0$:

$$\boxed{B_0(0) = 1\,,\qquad B_0^{(i)}(0) = 0\,,\quad i = 1, 2, \dots, d} \tag{23.8.15}$$

The larger the d, the more derivatives vanish at $\omega = 0$, and the flatter the response $B_0(\omega)$ becomes. This effectively increases the cutoff frequency of the lowpass filter—letting through more noise, but at the same time preserving more of the higher frequencies in the desired signal.

Figure 23.8.1 shows the magnitude response $|B_0(\omega)|$ for the cases $N = 7, 15$ and $d = 0, 2, 4$. The quadratic filters are flatter at DC than the plain FIR averager because of the extra constraint $B_0''(0) = 0$. Similarly, the quartic filters are even flatter because they satisfy two flatness conditions: $B_0''(0) = B_0^{(4)}(0) = 0$. The cutoff frequencies are *approximately* doubled and tripled in the cases $d = 2$ and $d = 4$, as compared to $d = 0$.

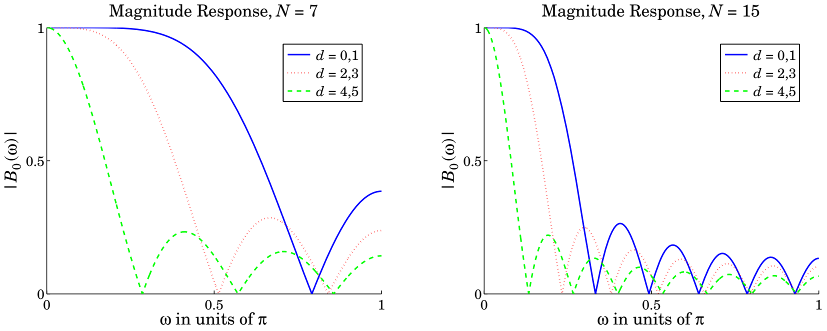


Fig. 23.8.1 LPSM filters of lengths $N = 7, 15$, and orders $d = 0, 2, 4$.

A direct consequence of the moment constraints (23.8.11) is that the moments of the input signal $y(n)$ are *preserved* by the filtering operation (23.2.33), that is,

$$\boxed{\sum_n n^i \hat{x}(n) = \sum_n n^i y(n)\,,\qquad i = 0, 1, \dots, d} \tag{23.8.16}$$

This can be proved easily working in the frequency domain. Differentiating the filtering equation $\hat{X}(\omega) = B_0(\omega)Y(\omega)$ i times, and using the product rules of differentiation, we obtain:

$$\hat{X}^{(i)}(\omega) = \sum_{j=0}^{i} \binom{i}{j} B_0^{(j)}(\omega)Y^{(i-j)}(\omega)$$

Setting $\omega = 0$ and using the moment constraints satisfied by the filter, $B_0^{(j)}(0) = \delta(j)$, we observe that only the $j = 0$ term will contribute to the above sum, giving:

$$\hat{X}^{(i)}(0) = B_0(0)Y^{(i)}(0) = Y^{(i)}(0)\,,\qquad i = 0, 1, \dots, d$$

which implies Eq. (23.8.16), by virtue of Eq. (23.8.14) as applied to $x(n)$ and $y(n)$.

The preservation of moments is a useful property in applications, such as spectroscopic analysis or ECG processing, in which the desired signal has one or more sharp peaks, whose widths must be preserved by the smoothing operation. In particular, the second moment corresponding to $i = 2$ in Eq. (23.8.16) is a measure of the square of the width [626–636,640,642,762].

The above moment constraints can be used in a direct way to design the LPSM filters. We consider first the more general problem of designing an optimum length-N filter that *minimizes* the NRR subject to $d+1$ arbitrary moment constraints. That is, minimize

$$\mathcal{R} = \mathbf{b}^T\mathbf{b} = \sum_{n=-M}^{M} b(n)^2 = \min \tag{23.8.17}$$

subject to the $d+1$ constraints, with a given $\mathbf{u} = [u_0, u_1, \dots, u_d]^T$:

$$\mathbf{s}_i^T\mathbf{b} = \sum_{n=-M}^{M} n^i b(n) = u_i, \quad i = 0, 1, \dots, d \quad \Rightarrow \quad S^T\mathbf{b} = \mathbf{u} \tag{23.8.18}$$

The minimization of Eq. (23.8.17) subject to (23.8.18) can be carried out with the help of Lagrange multipliers, that is, adding the constraint terms to the performance index:

$$\mathcal{J} = \mathbf{b}^T\mathbf{b} + 2\sum_{i=0}^{d} \lambda_i (u_i - \mathbf{s}_i^T\mathbf{b}) = \mathbf{b}^T\mathbf{b} + 2\boldsymbol{\lambda}^T(\mathbf{u} - S^T\mathbf{b}) \tag{23.8.19}$$

The gradient of $\mathcal{J}$ with respect to the unknown filter $\mathbf{b}$ is:

$$\frac{\partial\mathcal{J}}{\partial\mathbf{b}} = 2\mathbf{b} - 2S\boldsymbol{\lambda}$$

Setting the gradient to zero, and solving for $\mathbf{b}$ gives:

$$\mathbf{b} = S\boldsymbol{\lambda} = [\mathbf{s}_0, \mathbf{s}_1, \dots, \mathbf{s}_d]\begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_d \end{bmatrix} = \sum_{i=0}^{d} \lambda_i \mathbf{s}_i$$

Component-wise this means that $b(n)$ has the polynomial form:

$$b(n) = \sum_{i=0}^{d} \lambda_i s_i(n) = \sum_{i=0}^{d} \lambda_i n^i, \qquad -M \le n \le M$$

The Lagrange multiplier vector $\boldsymbol{\lambda}$ is determined by imposing the desired constraint:

$$\mathbf{u} = S^T\mathbf{b} = S^T S\boldsymbol{\lambda} = F\boldsymbol{\lambda} \qquad \Rightarrow \qquad \boldsymbol{\lambda} = F^{-1}\mathbf{u}$$

resulting in the optimum $\mathbf{b}$:

$$\mathbf{b} = S\boldsymbol{\lambda} = SF^{-1}\mathbf{u} = S(S^T S)^{-1}\mathbf{u} = G\mathbf{u} \tag{23.8.20}$$

Since the solution minimizes the norm $\mathbf{b}^T\mathbf{b}$, it is recognized to be the *minimum-norm* solution of the $(d+1)\times N$ full-rank under-determined linear system $S^T\mathbf{b} = \mathbf{u}$, which can be obtained by the pseudoinverse of S^T, that is, $\mathbf{b} = (S^T)^+\mathbf{u}$, $(S^T)^+ = S(S^T S)^{-1}$. In MATLAB, we can simply write $\mathbf{b} = \text{pinv}(S^T)\mathbf{u}$.

Comparing this solution with Eqs. (23.8.7) and (23.8.8), we conclude that the LPSM filters $\mathbf{b}_m$ can be thought of as the optimum filters that have minimum NRR with constraint vectors $\mathbf{u} = \mathbf{u}_m$, that is, the minimization problems,

$$\boxed{\mathcal{R} = \mathbf{b}_m^T\mathbf{b}_m = \min, \quad \text{subject to} \quad S^T\mathbf{b}_m = \mathbf{u}_m} \tag{23.8.21}$$

have solutions,

$$\mathbf{b}_m = SF^{-1}\mathbf{u}_m = G\mathbf{u}_m, \quad -M \le m \le M \tag{23.8.22}$$

and putting these together as the columns of B, we obtain Eq. (23.2.31):

$$B = [\dots, \mathbf{b}_m, \dots] = G[\dots, \mathbf{u}_m, \dots] = GS^T = SF^{-1}S^T \tag{23.8.23}$$

In particular, the steady-state LPSM filter $\mathbf{b}_0$ minimizes the NRR with the constraint vector $\mathbf{u} = \mathbf{u}_0 = [1, 0, \dots, 0]^T$. This was precisely the problem first formulated and solved using Lagrange multipliers by Schiaparelli [620].

Similarly, the interpolating filter $\mathbf{b}_t = G\mathbf{u}_t$ of Eq. (23.6.10) can be thought of as the solution of the constrained minimization problem:

$$\mathcal{R} = \mathbf{b}^T\mathbf{b} = \min, \quad \text{subject to } S^T\mathbf{b} = \mathbf{u}_t, \quad \text{where} \ \ \mathbf{u}_t = [1, t, t^2, \dots, t^d]^T$$

23.9 Predictive Differentiation Filters

Going back to the polynomial fit of Eq. (23.6.9), that is,

$$\hat{y}_t = \sum_{i=0}^{d} c_i t^i = \mathbf{c}^T\mathbf{u}_t = \mathbf{y}^T G\mathbf{u}_t = \mathbf{y}^T\mathbf{b}_t, \quad \text{where} \quad \mathbf{b}_t = G\mathbf{u}_t, \tag{23.9.1}$$

we recall that the differentiation filters (23.2.24) were derived by differentiating (23.9.1) at $t = 0$, and therefore, they correspond to the center of the data vector $\mathbf{y}$:

$$\begin{aligned} \hat{y}_t\big|_{t=0} &= c_0 = \mathbf{b}_0^T\mathbf{y} = \mathbf{g}_0^T\mathbf{y} \\ \dot{\hat{y}}_t\big|_{t=0} &= c_1 = \mathbf{g}_1^T\mathbf{y} \\ \ddot{\hat{y}}_t\big|_{t=0} &= 2c_2 = \mathbf{g}_2^T\mathbf{y}, \quad \text{etc.,} \end{aligned}$$

The first derivative at an arbitrary value of t is given by:

$$\dot{\hat{y}}_t = \mathbf{y}^T\dot{\mathbf{b}}_t, \quad \dot{\mathbf{b}}_t = G\dot{\mathbf{u}}_t$$

where the differentiation operation can be expressed as matrix multiplication:

$$\mathbf{u}_t = \begin{bmatrix} 1 \\ t \\ t^2 \\ \vdots \\ t^d \end{bmatrix} \Rightarrow \dot{\mathbf{u}}_t = \begin{bmatrix} 0 \\ 1 \\ 2t \\ \vdots \\ dt^{d-1} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 \\ 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 2 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \cdots & d & 0 \end{bmatrix} \begin{bmatrix} 1 \\ t \\ t^2 \\ \vdots \\ t^{d-1} \\ t^d \end{bmatrix} \equiv \mathcal{D}\mathbf{u}_t \tag{23.9.2}$$

where $\mathcal{D}$ is the $(d+1)\times(d+1)$ matrix with the sequence of numbers $\{1, 2, \dots, d\}$ along its first subdiagonal and zeros everywhere else. Such a matrix can be constructed trivially in MATLAB, for example, by:

```
D = diag(1:d, -1);
```

It follows that the first-order differentiation filter is $\dot{\mathbf{b}}_t = G\mathcal{D}\mathbf{u}_t$. In particular, the differentiation filter at the sample point $t = m$ is $\dot{\mathbf{b}}_m = G\mathcal{D}\mathbf{u}_m$ and the corresponding estimated derivative:

$$\hat{\dot{y}}_m = \dot{\mathbf{b}}_m^T\mathbf{y} = \mathbf{u}_m^T\mathcal{D}^T G^T\mathbf{y}\,, \quad -M \le m \le M \tag{23.9.3}$$

Stacking these together into a column vector, we obtain:

$$\hat{\dot{\mathbf{y}}} = S\mathcal{D}^T G^T\mathbf{y} = \dot{B}^T\mathbf{y}\,, \quad \text{where} \quad \dot{B} = G\mathcal{D}S^T = SF^{-1}\mathcal{D}S^T \tag{23.9.4}$$

so that $\dot{B}$ has the $\dot{\mathbf{b}}_m$ as columns. Higher-order derivatives correspond to higher powers of the matrix $\mathcal{D}$, for example, $\ddot{\mathbf{u}}_t = \mathcal{D}^2\mathbf{u}_t$, and so on, with the highest non-trivial power being $\mathcal{D}^d$, because $\mathcal{D}^{d+1} = 0$, or equivalently, because the elements of $\mathbf{u}_t$ are monomials up to t^d. Therefore, the order-i differentiation matrix will be:

$$\boxed{B^{(i)} = SF^{-1}\mathcal{D}^i S^T}\,, \quad i = 0, 1, \dots, d \tag{23.9.5}$$

Centering the data vector $\mathbf{y}$ at time n and denoting the m-th column of $B^{(i)}$ by $\mathbf{b}_m^{(i)}$, we obtain the filtering equation for the i-th estimated derivative:

$$\hat{y}_{n+m}^{(i)} = \sum_{k=-M}^{M} b_m^{(i)}(k)y_{n+k} = \sum_{k=-M}^{M} b_m^{(i)}(-k)y_{n-k} \tag{23.9.6}$$

We note that at the data-vector center $m = 0$, we have $\mathbf{b}_0^{(i)} = \mathbf{g}_i$. For arbitrary t, we have $\mathbf{b}_t^{(i)} = G\mathcal{D}^i\mathbf{u}_t$ and we obtain the estimated/interpolated derivative:

$$\boxed{\hat{y}_{n+t}^{(i)} = \sum_{k=-M}^{M} b_t^{(i)}(k)y_{n+k} = \sum_{k=-M}^{M} b_t^{(i)}(-k)y_{n-k}} \tag{23.9.7}$$

As in Eq. (23.7.2), the redefinition $h_\tau^{(i)}(k) = b_{M+\tau}^{(i)}(M-k)$ will result into a causal version of the predictive differentiator filter, with Eq. (23.9.7) transforming into:

$$\boxed{\hat{y}_{n+\tau}^{(i)} = \sum_{k=0}^{N-1} h_\tau^{(i)}(k)y_{n-k}} \qquad \text{(causal predictive differentiator)} \tag{23.9.8}$$

One can easily obtain closed-form expressions for the differentiation filters $b_t^{(i)}(k)$ for $d = 0, 1, 2, 3, 4$ and arbitrary M, by replacing the variable m in Eqs. (23.3.7)–(23.3.12) by the variable t and differentiating i-times with respect to t. For example, for $d = 1, 2, 3, 4$, we differentiate Eqs. (23.6.12) once to get the first derivative:

$$\begin{aligned}
\dot{b}_t(k) &= \frac{k}{F_2}\\
\dot{b}_t(k) &= \frac{1}{F_2}k - \frac{F_2}{D_4}(2t) + \frac{F_0}{D_4}(2tk^2)\\
\dot{b}_t(k) &= \frac{F_6}{D_8}k - \frac{F_2}{D_4}(2t) + \frac{F_0}{D_4}(2tk^2) - \frac{F_4}{D_8}(3t^2k + k^3) + \frac{F_2}{D_8}(3t^2k^3)\\
\dot{b}_t(k) &= \frac{F_6}{D_8}k - \frac{D_{10}}{D}(2t) + \frac{E_8}{D}(2tk^2) - \frac{F_4}{D_8}(k3t^2 + k^3)\\
&\quad + \frac{F_2}{D_8}(3t^2k^3) + \frac{D_8}{D}(4t^3) - \frac{D_6}{D}(2tk^4 + k^2 4t^3) + \frac{D_4}{D}(4t^3k^4)
\end{aligned} \tag{23.9.9}$$

For the causal versions, we have for $d = 1$:

$$\begin{aligned}
h_\tau(k) &= \frac{1}{F_0} + \frac{(M+\tau)(M-k)}{F_2} = \frac{M(M+1) + 3(M+\tau)(M-k)}{M(M+1)(2M+1)}\\
\dot{h}_\tau(k) &= \frac{M-k}{F_2} = \frac{3(M-k)}{M(M+1)(2M+1)}
\end{aligned} \tag{23.9.10}$$

where $k = 0, 1, \dots, N-1$. We note that $\dot{h}_\tau$ can be obtained by differentiating h_τ with respect to τ. The derivative filter is independent of τ because it corresponds to fitting a first-order polynomial. For $d = 2$, we have similarly,

$$\begin{aligned}
h_\tau(k) &= \frac{F_4}{D_4} + \frac{1}{F_2}(M+\tau)(M-k) - \frac{F_2}{D_4}\big((M+\tau)^2 + (M-k)^2\big) + \frac{F_0}{D_4}(M+\tau)^2(M-k)^2\\
\dot{h}_\tau(k) &= \frac{1}{F_2}(M-k) - \frac{F_2}{D_4}2(M+\tau) + \frac{F_0}{D_4}2(M+\tau)(M-k)^2
\end{aligned} \tag{23.9.11}$$

where, we recall from Eq. (23.3.11),

$$\begin{aligned}
\frac{F_4}{D_4} &= \frac{3(3M^2 + 3M - 1)}{(2M+3)(4M^2-1)}\,, \quad \frac{F_2}{D_4} = \frac{15}{(2M+3)(4M^2-1)}\\
\frac{F_0}{D_4} &= \frac{45}{M(M+1)(2M+3)(4M^2-1)}\,, \quad \frac{1}{F_2} = \frac{3}{M(M+1)(2M+1)}
\end{aligned}$$

Example 23.9.1: For the case $N = 5$, $d = 2$, we had found in Eqs. (23.2.5) and (23.2.16) that:

$$S = [\mathbf{s}_0, \mathbf{s}_1, \mathbf{s}_2] = \begin{bmatrix} 1 & -2 & 4\\ 1 & -1 & 1\\ 1 & 0 & 0\\ 1 & 1 & 1\\ 1 & 2 & 4 \end{bmatrix}, \quad G = \frac{1}{35}\begin{bmatrix} -3 & -7 & 5\\ 12 & -3.5 & -2.5\\ 17 & 0 & -5\\ 12 & 3.5 & -2.5\\ -3 & 7 & 5 \end{bmatrix}$$

The corresponding first- and second-order differentiation matrices will be:

$$\dot{B} = G\mathcal{D}^1 S^T = \frac{1}{35}\begin{bmatrix} -27 & -17 & -7 & 3 & 13\\ 6.5 & 1.5 & -3.5 & -8.5 & -13.5\\ 20 & 10 & 0 & -10 & -20\\ 13.5 & 8.5 & 3.5 & -1.5 & -6.5\\ -13 & -3 & 7 & 17 & 27 \end{bmatrix}, \quad \mathcal{D}^1 = \begin{bmatrix} 0 & 0 & 0\\ 1 & 0 & 0\\ 0 & 2 & 0 \end{bmatrix}$$

$$\ddot{B} = G\mathcal{D}^2 S^T = \frac{1}{35}\begin{bmatrix} 10 & 10 & 10 & 10 & 10\\ -5 & -5 & -5 & -5 & -5\\ -10 & -10 & -10 & -10 & -10\\ -5 & -5 & -5 & -5 & -5\\ 10 & 10 & 10 & 10 & 10 \end{bmatrix}, \quad \mathcal{D}^2 = \begin{bmatrix} 0 & 0 & 0\\ 0 & 0 & 0\\ 2 & 0 & 0 \end{bmatrix}$$

The central columns agree with Eq. (23.2.25). The interpolating smoothing and first-order differentiation filters are given by:

$$\mathbf{b}_t = G\mathbf{u}_t = \frac{1}{35}\begin{bmatrix} -3 & -7 & 5\\ 12 & -3.5 & -2.5\\ 17 & 0 & -5\\ 12 & 3.5 & -2.5\\ -3 & 7 & 5 \end{bmatrix}\begin{bmatrix} 1\\ t\\ t^2 \end{bmatrix} = \frac{1}{35}\begin{bmatrix} -3-7t+5t^2\\ 12-3.5t-2.5t^2\\ 17-5t^2\\ 12+3.5t-2.5t^2\\ -3+7t+5t^2 \end{bmatrix}$$

$$\dot{\mathbf{b}}_t = G\mathcal{D}\mathbf{u}_t = \frac{1}{35}\begin{bmatrix} -3 & -7 & 5\\ 12 & -3.5 & -2.5\\ 17 & 0 & -5\\ 12 & 3.5 & -2.5\\ -3 & 7 & 5 \end{bmatrix}\begin{bmatrix} 0 & 0 & 0\\ 1 & 0 & 0\\ 0 & 2 & 0 \end{bmatrix}\begin{bmatrix} 1\\ t\\ t^2 \end{bmatrix} = \frac{1}{35}\begin{bmatrix} -7+10t\\ -3.5-5t\\ -10t\\ 3.5-5t\\ 7+10t \end{bmatrix}$$

where $\dot{\mathbf{b}}_t$ can be obtained either by the indicated matrix multiplication or by simply differentiating $\mathbf{b}_t$ with respect to t. □

The MATLAB function `lpdiff` implements the design of the differentiation matrices:

```
B = lpdiff(N,d,i);          % differentiation filters
```

Like `lpsm`, it carries out a Gram-Schmidt QR-transformation on the monomial basis S and constructs the $B^{(i)}$ by:

$$\begin{aligned} &S = QR\,, \quad Q^TQ = I\,, \quad R = \text{upper triangular}\\ &G = S(S^TS)^{-1} = QR^{-T}\\ &B^{(i)} = GD^iS^T = Q(R^{-T}D^iR^T)Q^T \end{aligned}$$

The predictive/interpolating differentiation filters $\mathbf{b}_t^{(i)}$ are the minimum-norm solution of the under-determined linear system $S^T\mathbf{b} = \mathcal{D}^i\mathbf{u}_t$, or, equivalently the solution of the constrained minimization problem:

$$\mathcal{R} = \mathbf{b}^T\mathbf{b} = \min\,, \quad \text{subject to} \quad S^T\mathbf{b} = \mathcal{D}^i\mathbf{u}_t$$

The MATLAB function `lpinterp` implements the design of predictive and interpolating differentiation filters, essentially carrying out the operation $\mathbf{b} = \text{pinv}(S^T)\mathcal{D}^i\mathbf{u}_t$:

```
b = lpinterp(N,d,t,i);          % local polynomial interpolation and differentiation filters
```

The case $i = 0$ corresponds to the predictive interpolation filters of Sec. 23.6. For the integer values $t = m$, $-M \le m \le M$, the filter $\mathbf{b}$ agrees with the columns of $B^{(i)}$.

Example 23.9.2: Fig. 23.9.1 illustrates the performance of the local polynomial differentiation filters on noiseless and noisy signals.

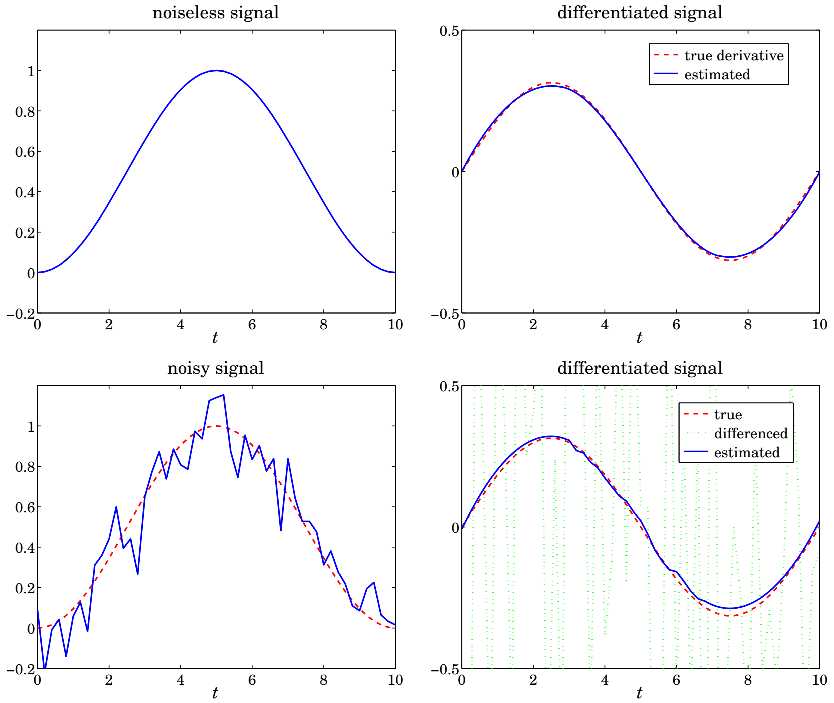


Fig. 23.9.1 Differentiating noisy signals.

The noiseless signal is a raised cosine $s(t) = 0.5 - 0.5\cos(\omega t)$, with $0 \le t \le T$ and $\omega = 2\pi/T$, so that it spans one cycle. Choosing a sampling time interval $\Delta t = T/L$, we can construct a noisy signal sampled at time instants $t_n = n\Delta t = nT/L$, $n = 0, 1, \dots, L$, by adding zero-mean white gaussian noise v_n of variance, say σ^2, so that the noisy observations are:

$$y_n = s(t_n) + v_n\,, \quad n = 0, 1, \dots, L$$

The first derivative of $s(t)$ is $\dot{s}(t) = 0.5\omega\sin(\omega t)$ and its samples, $\dot{s}(t_n) = 0.5\omega\sin(\omega t_n)$. The upper-left graph shows $s(t_n)$ versus t_n, with $T = 10$ and $L = 50$. The upper-right graph shows $\dot{s}(t_n)$ (dashed line) together with the estimated derivative (solid line) of the original signal $s(t_n)$ filtered through an LPSM differentiation filter designed with $N = 31$ and polynomial order $d = 3$. The output of the filter is divided by Δt in order to adjust its dimensions.

The bottom-left graph shows the noisy signal y_n. In the bottom-right graph, the output (solid line) of the same differentiation filter applied to the noisy signal y_n is compared with the true noiseless differentiated signal $\dot{s}_n$, as well as to the differenced signal $\text{diff}(y)/\Delta t$. The following MATLAB code illustrates the generation of the bottom-right graph:

```
T = 10; L = 50; Dt =T/L; w = 2*pi/T; sigma = 0.1;
t = 0:Dt:T;
s = 0.5 - 0.5*cos(w*t);                % noiseless signal

seed=100; randn('state',seed);
y = s + sigma * randn(1,length(s));    % noisy signal

N = 31; d = 3; B1 = lpdiff(N,d,1);     % first-order differentiation filter

sd = 0.5*w*sin(w*t);                   % derivative of s(t)
xd = lpfilt(B1,s)/Dt;                  % estimated derivative of s(t)
x1 = lpfilt(B1,y)/Dt;                  % estimated derivative from the noisy signal
yd = diff(y)/Dt;  td = t(2:end);       % differenced signal estimates the derivative

plot(t,sd,'--', td,yd,':', t,x1,'-');
```

The differencing operation amplifies the noise and renders the estimated derivative useless, whereas the local-polynomial derivative is fairly accurate. The filtering operation is carried out by the function `lpfilt`, which is explained in the next section. □

23.10 Filtering Implementations

In smoothing a length-L signal block y_n, $n = 0, 1, \dots, L-1$, with a double-sided filter h_m, $-M \le m \le M$, the output signal $\hat{x}_n$ is given by the convolutional form:

$$\hat{x}_n = \sum_{m=\max(-M,n-L+1)}^{\min(n,M)} h_m y_{n-m}\,, \quad -M \le n \le L+M-1 \tag{23.10.1}$$

The length of $\hat{x}_n$ is $L+2M$, and the first $2M$ and last $2M$ output samples correspond to the input-on and input-off transients, while the central $L-2M$ points, $M \le n \le L-M-1$, correspond to the steady-state output computed from the steady-state version of Eq. (23.10.1):

$$\hat{x}_n = \sum_{m=-M}^{M} h_m y_{n-m}\,, \quad M \le n \le L-M-1 \tag{23.10.2}$$

The range of the output index n and the limits of summation in (23.10.1) are determined from the inequalities $-M \le m \le M$ and $0 \le n-m \le L-1$ that must be satisfied by the indices of h_m and y_{n-m}. However, only the subrange $\{\hat{x}_n\,,\ 0 \le n \le L-1\}$ is of interest since these output samples represent the smoothed values of the corresponding input samples $\{y_n\,,\ n = 0, 1, \dots, L-1\}$. This is illustrated in Fig. 23.10.1.

The first and last M samples in the subrange $0 \le n \le L-1$ are still parts of the input-on and input-off transients. To clarify these remarks, we consider the case $L = 8$, $M = 2$. The full output (23.10.1) may be represented by the usual convolution matrix of the filter acting on the input signal block:

$$
\begin{bmatrix} \hat{x}_{-2}\\ \hat{x}_{-1}\\ \cdots \\ \hat{x}_0 \\ \hat{x}_1 \\ \cdots \\ \hat{x}_2\\ \hat{x}_3\\ \hat{x}_4\\ \hat{x}_5\\ \cdots\\ \hat{x}_6\\ \hat{x}_7\\ \cdots\\ \hat{x}_8\\ \hat{x}_9 \end{bmatrix}
=
\left[\begin{array}{cccccccc}
h_{-2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
h_{-1} & h_{-2} & 0 & 0 & 0 & 0 & 0 & 0 \\
\hdashline
h_0 & h_{-1} & h_{-2} & 0 & 0 & 0 & 0 & 0 \\
h_1 & h_0 & h_{-1} & h_{-2} & 0 & 0 & 0 & 0 \\
\hdashline
h_2 & h_1 & h_0 & h_{-1} & h_{-2} & 0 & 0 & 0 \\
0 & h_2 & h_1 & h_0 & h_{-1} & h_{-2} & 0 & 0 \\
0 & 0 & h_2 & h_1 & h_0 & h_{-1} & h_{-2} & 0 \\
0 & 0 & 0 & h_2 & h_1 & h_0 & h_{-1} & h_{-2} \\
\hdashline
0 & 0 & 0 & 0 & h_2 & h_1 & h_0 & h_{-1} \\
0 & 0 & 0 & 0 & 0 & h_2 & h_1 & h_0 \\
\hdashline
0 & 0 & 0 & 0 & 0 & 0 & h_2 & h_1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & h_2
\end{array}\right]
\begin{bmatrix} y_0\\ y_1\\ \cdots\\ y_2\\ y_3\\ y_4\\ y_5\\ \cdots\\ y_6\\ y_7 \end{bmatrix}
$$

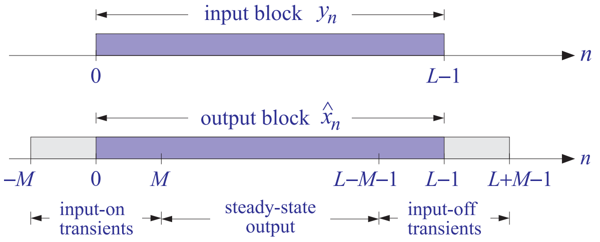


Fig. 23.10.1 Input and output signal blocks from a double-sided filter.

This matrix can be constructed in MATLAB with the built-in function `convmtx`, or with its sparse version `convmat`, or with the function `datamat`, the latter two being part of the OSP toolbox. Defining $\mathbf{h} = [h_{-M}, \dots, h_0, \dots, h_M]^T$, we have the syntax:

```
H = convmtx(h,L);        % built-in convolution matrix
H = convmat(h,L);        % sparse version of convmtx
H = datamat(h,L-1);      % used extensively in [45]
```

Dropping the first and last two outputs, we obtain the outputs in the subrange $0 \le n \le 7$:

$$\hat{\mathbf{x}} = \begin{bmatrix} \hat{x}_0 \\ \hat{x}_1 \\ \cdots \\ \hat{x}_2 \\ \hat{x}_3 \\ \hat{x}_4 \\ \hat{x}_5 \\ \cdots \\ \hat{x}_6 \\ \hat{x}_7 \end{bmatrix} = \begin{bmatrix} h_0 & h_{-1} & h_{-2} & 0 & 0 & 0 & 0 & 0 \\ h_1 & h_0 & h_{-1} & h_{-2} & 0 & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ h_2 & h_1 & h_0 & h_{-1} & h_{-2} & 0 & 0 & 0 \\ 0 & h_2 & h_1 & h_0 & h_{-1} & h_{-2} & 0 & 0 \\ 0 & 0 & h_2 & h_1 & h_0 & h_{-1} & h_{-2} & 0 \\ 0 & 0 & 0 & h_2 & h_1 & h_0 & h_{-1} & h_{-2} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & 0 & h_2 & h_1 & h_0 & h_{-1} \\ 0 & 0 & 0 & 0 & 0 & h_2 & h_1 & h_0 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ \cdots \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ \cdots \\ y_6 \\ y_7 \end{bmatrix} \equiv H\mathbf{y} \qquad (23.10.3)$$

The first two and last two of these outputs are still transient and are being computed with only a subset of the filter coefficients, and therefore, may not adequately represent the corresponding smoothed values. This so-called "end-point problem" has been addressed repeatedly with a number of solutions.

One method that is widely used by the government to process census and business-cycle data (e.g., the X12-ARIMA method) is to backcast and forecast M estimated values at the beginning and end of the length-L input block, so that y_n is now defined over $-M \le n \le L-1+M$, and the desired output samples over the subrange $0 \le n \le L-1$ will be steady-state outputs being computed with the full filter.

Another method is to use different filters for the first M and last M outputs. For example, one can take the outputs $\hat{y}_{n+m}$ of the LPSM filters $b_m(k)$ to estimate the initial and final M transients, while using the central filter $b_0(k)$ for the steady-state outputs. Indeed, the first time index when one can use the steady-state filter $b_0(k)$ is $n = M$:

$$\hat{x}_M = \hat{y}_M = \sum_{k=-M}^{M} b_0(k) y_{M+k}$$

Instead of calculating the previous output $\hat{x}_{M-1}$ using the transient version of $b_0(k)$,

$$\hat{x}_{M-1} = \sum_{k=-(M-1)}^{M} b_0(k) y_{M-1+k}$$

one could estimate $\hat{x}_{M-1}$ using $\hat{y}_{M+m}$ with $m = -1$, that is, using $b_{-1}(k)$, and using $b_{-2}(k), b_{-3}(k), \dots, b_{-M}(k)$ for the other initial M outputs:

$$\begin{aligned} \hat{x}_{M-1} &= \hat{y}_{M-1} = \sum_{k=-M}^{M} b_{-1}(k) y_{M+k} \\ \hat{x}_{M-2} &= \hat{y}_{M-2} = \sum_{k=-M}^{M} b_{-2}(k) y_{M+k} \\ &\vdots \\ \hat{x}_0 &= \hat{y}_{M-M} = \sum_{k=-M}^{M} b_{-M}(k) y_{M+k} \end{aligned} \qquad (23.10.4)$$

Similarly, one can use the filters $b_m(k)$ for $m = 1, 2, \dots, M$ to calculate the last M smoothed outputs, starting with the last steady-state output at $n = L-1-M$ and proceeding to the end $n = L-1$:

$$\begin{aligned} \hat{x}_{L-M} &= \hat{y}_{L-1-M+1} = \sum_{k=-M}^{M} b_1(k) y_{L-1-M+k} \\ \hat{x}_{L-M+1} &= \hat{y}_{L-1-M+2} = \sum_{k=-M}^{M} b_2(k) y_{L-1-M+k} \\ &\vdots \\ \hat{x}_{L-1} &= \hat{y}_{L-1-M+M} = \sum_{k=-M}^{M} b_M(k) y_{L-1-M+k} \end{aligned} \qquad (23.10.5)$$

The following example illustrates the computational steps for the input-on, steady, and input-off output samples, where we denoted $b_{m,k} = b_m(k)$ for simplicity:

$$\hat{\mathbf{x}} = \begin{bmatrix} \hat{x}_0 \\ \hat{x}_1 \\ \cdots \\ \hat{x}_2 \\ \hat{x}_3 \\ \hat{x}_4 \\ \hat{x}_5 \\ \cdots \\ \hat{x}_6 \\ \hat{x}_7 \end{bmatrix} = \begin{bmatrix} b_{-2,-2} & b_{-2,-1} & b_{-2,0} & b_{-2,1} & b_{-2,2} & 0 & 0 & 0 \\ b_{-1,-2} & b_{-1,-1} & b_{-1,0} & b_{-1,1} & b_{-1,2} & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ b_{0,-2} & b_{0,-1} & b_{0,0} & b_{0,1} & b_{0,2} & 0 & 0 & 0 \\ 0 & b_{0,-2} & b_{0,-1} & b_{0,0} & b_{0,1} & b_{0,2} & 0 & 0 \\ 0 & 0 & b_{0,-2} & b_{0,-1} & b_{0,0} & b_{0,1} & b_{0,2} & 0 \\ 0 & 0 & 0 & b_{0,-2} & b_{0,-1} & b_{0,0} & b_{0,1} & b_{0,2} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & b_{1,-2} & b_{1,-1} & b_{1,0} & b_{1,1} & b_{1,2} \\ 0 & 0 & 0 & b_{2,-2} & b_{2,-1} & b_{2,0} & b_{2,1} & b_{2,2} \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ \cdots \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ \cdots \\ y_6 \\ y_7 \end{bmatrix} = H\mathbf{y}$$

In particular, for $N = 5$ and $d = 2$, the convolutional filtering matrix will be:

$$\hat{\mathbf{x}} = \begin{bmatrix} \hat{x}_0 \\ \hat{x}_1 \\ \cdots \\ \hat{x}_2 \\ \hat{x}_3 \\ \hat{x}_4 \\ \hat{x}_5 \\ \cdots \\ \hat{x}_6 \\ \hat{x}_7 \end{bmatrix} = \frac{1}{35} \begin{bmatrix} 31 & 9 & -3 & -5 & 3 & 0 & 0 & 0 \\ 9 & 13 & 12 & 6 & -5 & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ -3 & 12 & 17 & 12 & -3 & 0 & 0 & 0 \\ 0 & -3 & 12 & 17 & 12 & -3 & 0 & 0 \\ 0 & 0 & -3 & 12 & 17 & 12 & -3 & 0 \\ 0 & 0 & 0 & -3 & 12 & 17 & 12 & -3 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & -5 & 6 & 12 & 13 & 9 \\ 0 & 0 & 0 & 3 & -5 & -3 & 9 & 31 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ \cdots \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ \cdots \\ y_6 \\ y_7 \end{bmatrix} = H\mathbf{y}$$

with entries obtained from the matrix B of Eq. (23.2.16):

$$B = \frac{1}{35} \begin{bmatrix} 31 & 9 & -3 & -5 & 3 \\ 9 & 13 & 12 & 6 & -5 \\ -3 & 12 & 17 & 12 & -3 \\ -5 & 6 & 12 & 13 & 9 \\ 3 & -5 & -3 & 9 & 31 \end{bmatrix} = [\mathbf{b}_{-2}, \mathbf{b}_{-1}, \mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2]$$

More generally, given any smoothing (or differentiation) matrix B whose central column contains the (reversed) steady-state filter, and its other columns, the (reversed) filters to be used for the initial and final transients, one can uniquely construct the corresponding $L\times L$ convolutional matrix H for filtering a length-L block of data $\mathbf{y}$.

The procedure is straightforward. First construct the ordinary full $(L+2M)\times L$ convolution matrix for the central filter, then delete its first M and last M rows, and finally, replace the first M and last M rows of the result by the transient filters.

The following MATLAB code segment illustrates the procedure, where the matrix B is assumed to have size $N\times N$, with $N = 2M + 1$, with the central column being the reversed steady-state filter and the other columns, the reversed transient filters:

```
H = convmat(flip(B(:,M+1)), L);         % ordinary (L+2M)×L convolution matrix
H = H(M+1:L+M,:);                       % extract the L×L convolution submatrix
H(1:M, 1:N) = B(:,1:M)';                % redefine upper-left M×L corner
H(L-M+1:L, L-N+1:L) = B(:,M+2:N)';      % redefine lower-right M×L corner
```

The function `flip` reverses the central column of B because `convmat` expects as input the actual filter, not its reverse. The above steps have been incorporated into the function `lpmat` with syntax:

```
H = lpmat(B,L);     % local polynomial filter matrix of size L×L
```

Once the $L\times L$ matrix H is constructed, the actual filtering of a length-L input block $\mathbf{y}$ is straightforward, that is, $\hat{\mathbf{x}} = H\mathbf{y}$, and efficient because H is defined as sparse.

An alternative way to structure the filtering operation is to directly use Eqs. (23.10.4) and (23.10.5) for the transient parts and the following equation for the steady part:

$$\hat{x}_n = \sum_{k=-M}^{M} b_0(k)\, y_{n+k}\,, \quad M \le n \le L-1-M \tag{23.10.6}$$

The following MATLAB code illustrates this approach:

```
y = B(:,1:M)' * x(1:N);                   % first M transient outputs
for n = M+1:L-M,                          % middle L−2M steady-state outputs
   y = [y; B(:,M+1)' * x(n-M:n+M)];       % filtered by central column of B
end
y = [y; B(:,M+2:N)' * x(L-N+1:L)];        % last M transient outputs
```

These steps are implemented in the MATLAB function `lpfilt2`. A faster version is the function `lpfilt`, which uses MATLAB's built-in filtering functions. Thus, three possible ways of computing the filtered output $\hat{\mathbf{x}}$ given a smoothing matrix B are as follows (assuming that $\mathbf{y}$ is a length-L column vector):

```
x_hat = lpmat(B,L)*y;     % use L×L convolution matrix constructed from B
x_hat = lpfilt2(B,y);     % use directly the filtering equations (23.10.4)–(23.10.6)
x_hat = lpfilt(B,y);      % fast version using the function filtdbl
```

The function `lpfilt` internally calls the function `filtdbl`, which uses the built-in function `conv` to implement the FIR filtering by the steady-state double-sided central filter. The following code segment shows the essential part of `lpfilt`:

```
x_hat = filtdbl(flip(B(:,M+1)), y);                  % filter with the central column of B
x_hat(1:M) = B(:,1:M)' * y(1:N);                     % correct the first M transient outputs
x_hat(end-M+1:end) = B(:,M+2:N)' * y(end-N+1:end);   % correct the last M transient outputs
```

where the function `filtdbl` has usage:

```
y = filtdbl(h,x);     % filtering by double-sided FIR filter
```

The function `filtdbl` is essentially the ordinary convolution of the length-$(2M+1)$ filter $\mathbf{h}$ and the length-L signal $\mathbf{x}$, with the first M and last M output points discarded. The result is equivalent to that obtained using the convolution submatrix, as for example, in Eq. (23.10.3). We note, in particular, that the B matrix that gives rise to (23.10.3) is:

$$B = \begin{bmatrix} h_0 & h_1 & h_2 & 0 & 0 \\ h_{-1} & h_0 & h_1 & h_2 & 0 \\ h_{-2} & h_{-1} & h_0 & h_1 & h_2 \\ 0 & h_{-2} & h_{-1} & h_0 & h_1 \\ 0 & 0 & h_{-2} & h_{-1} & h_0 \end{bmatrix}$$

and contains the reversed filter $\mathbf{h}$ in the central column and the transient subfilters in the other columns.

There are other methods of handling the end-point problem, most notably Musgrave's *minimum-revision* method that uses end-point asymmetric filters constructed from a given central filter $\mathbf{h}$. It is discussed in detail in [45]. Here, we note that the output of this method is a B matrix, which can be passed directly into the filtering function `lpfilt`. The MATLAB function `minrev` implements Musgrave's method:

```
B = minrev(h,R);     % Musgrave's minimum revision asymmetric filters
```

where R is a scalar parameter. The method is widely used in the X-11 method of seasonal adjustment and trend extraction.

Example 23.10.1: Schiaparelli was the first one to systematically pose and solve the minimum-NRR filtering problem. He gave the solution to many specific cases, such as filter lengths $N = 5$–13, and polynomial orders $d = 3, 4$.

Here, we reproduce the example from Schiaparelli's paper on smoothing lunar observations, the signal y_n being a measure of the moon's influence on atmospheric effects. Fig. 23.10.2 shows 30 noisy observations (one for each lunar day) and their smoothed versions produced with an LPSM filter of length $N = 13$ and polynomial order $d = 3$ on the left, and $d = 4$ on the right (Schiaparelli's case).

The central filters for the $d = 3$ and $d = 4$ cases are:

$$\mathbf{b}_0 = \frac{1}{143}\big[-11,\ 0,\ 9,\ 16,\ 21,\ 24,\ 25,\ 24,\ 21,\ 16,\ 9,\ 0,\ -11\big]$$

$$\mathbf{b}_0 = \frac{1}{2431}\big[110,\ -198,\ -135,\ 110,\ 390,\ 600,\ 677,\ 600,\ 390,\ 110,\ -135,\ -198,\ 110\big]$$

The following program segment illustrates the computations:

```
Y = loadfile('schiaparelli.dat');     % data file available in the ISP & OSP toolboxes
n = Y(:,1); y = Y(:,2);               % extract n and y_n from the columns of Y

N=13; d=3; M=floor(N/2);              % filter length and polynomial order
B = lpsm(N,d);                        % construct LPSM matrix B
x = lpfilt(B,y);                      % filter noisy observations
b0 = B(:,M+1);                        % middle column of B
x0 = filtdbl(b0,y);                   % filter with b0 only
plot(n,y,'.', n,x,'-', n,x0,'--');
```

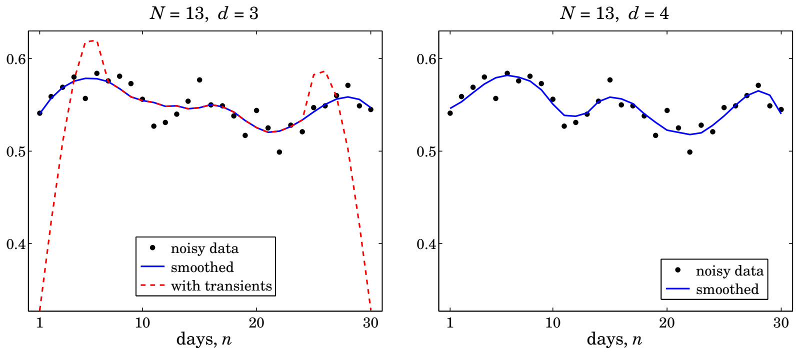


Fig. 23.10.2 Schiaparelli's smoothing example.

where the function `loadfile` extracts only the numerical data from the data file. In the left graph, we have also added the result of filtering with the steady-state filter $\mathbf{b}_0$, which illustrates the end-point problem. The two filtered curves differ only in their first 6 and last 6 points. □

Example 23.10.2: *Global Warming Trends.* Fig. 23.10.3 shows the annual average temperature anomalies (i.e., the differences with respect to the average of the period 1961–90) over the period 1856–2005 in the northern hemisphere. The data are available from the web site: `https://crudata.uea.ac.uk/cru/data/crutem2/`.

Five trend extraction methods are compared. In the upper left, a local polynomial smoothing filter was used of length $N = 65$ and polynomial order $d = 3$. The following MATLAB code illustrates the generation of that graph:

```
Y = loadfile('tavenh2v.dat');      % data file available in the ISP & OSP toolboxes
n = Y(:,1); y = Y(:,14);           % extract n and y_n from Y

N = 65; d = 3; B = lpsm(N,d);      % design the LPSM matrix B
x = lpfilt(B,y);                   % smooth the data vector y

figure; plot(n,y,':', n,x,'-');
```

In the upper-right graph, a minimum-roughness, or minimum-R_s, Henderson filter was used with length $N = 65$, polynomial order $d = 3$, and smoothing order $s = 2$. Such filters are discussed in Sec. 23.12. The resulting trend is noticeably smoother than that of the LPSM filter on the upper-left.

The middle-left graph uses the SVD signal enhancement method [45], with embedding order $M = 10$ and rank $r = 2$, with $K = 40$ iterations. The middle-right graph uses the Whittaker-Henderson smoothing method, discussed in Sec. 26.2, with smoothing order $s = 2$ and smoothing parameter $\lambda = 10^4$.

The lower left and right graphs use the Whittaker-Henderson method with the L_1 criterion with differentiation orders $s = 2$ and $s = 3$ and smoothing parameter $\lambda = 10$, implemented with the CVX package.† The $s = 2$ case represents the smoothed signal in piece-wise linear form, and the $s = 3$ case, in piece-wise parabolic form. This is further discussed in Sec. 26.7.

† `http://cvxr.com/cvx/`

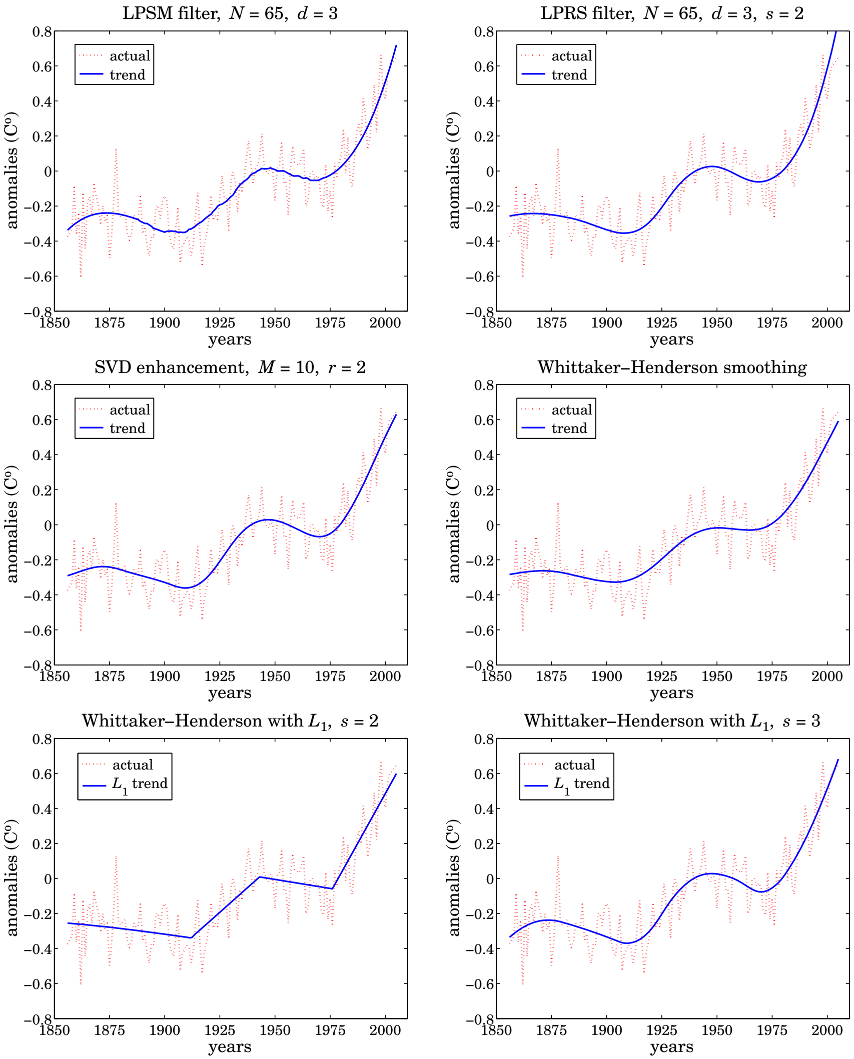


Fig. 23.10.3 Temperature trends determined by five methods.

The following MATLAB code segment illustrates the computation of the corresponding smoothed signals for these four methods:

```
N=65; d=3; s=2; x = lpfilt(lprs(N,d,s), y);   % minimum-R_s Henderson filter
M=10; r=2; K=40; x = svdenh(y,M,r,K);          % SVD enhancement method
la = 10000; s=2; x = whsm(y,la,s);             % Whittaker-Henderson smoothing
```

```
s = 2; la = 10; N = length(y);              % Whittaker-Henderson with L1
D = diff(eye(N),s);                         % s-fold differentiation matrix

cvx_begin                                   % use CVX package
    variable x(N)
    minimize( sum_square(y-x) + la * norm(D*x,1) )
cvx_end
```

All methods adequately handle the end-point problem. Repeating the same filtering operation several times results in even smoother trend signals. For example, Fig. 23.10.4 shows the result of repeating the filtering operation two additional times. The following MATLAB code illustrates the generation of the left graph:

```
N = 65; d=3; B = lpsm(N,d); x = y;
for i=1:3, x = lpfilt(B,x); end
figure;  plot(n,y,':', n,x,'-');
```

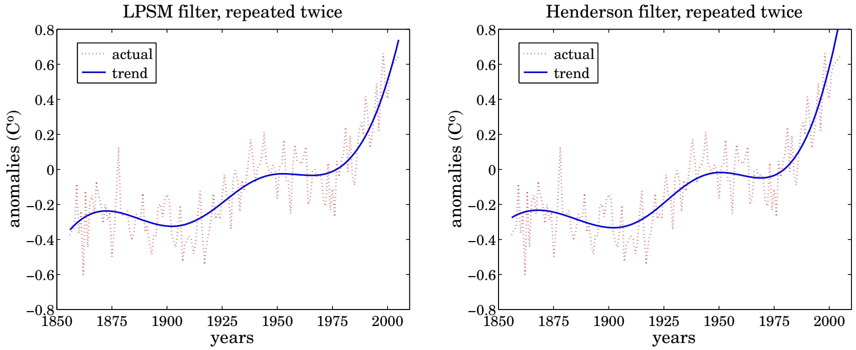


Fig. 23.10.4 Filtering repeated two additional times.

For the steady-state filters $B_0(\omega)$, filtering a total of K times is equivalent to an overall filter $[B_0(\omega)]^K$, an operation which makes a flat passband even flatter and a small stopband even smaller. The properties of iterated smoothing by local polynomial filters has been studied by De Forest, Schoenberg, and Greville [651,667,670].

Fig. 23.10.5 shows the estimated derivatives (solid line) of the temperature signal obtained by filtering it with the LPSM derivative filters, and compares them with the ordinary differencing operation, diff(y), in MATLAB notation. Clearly, differencing is simply too noisy to give any usable results.

The upper two graphs compute the first derivative of the input by $\hat{\dot{x}} = \text{lpfilt}(B_1, y)$ with the differentiator matrix obtained from $B_1 = \text{lpdiff}(N, d, i)$ with $N = 65$ and $i = 1$, and with $d = 1$ in the upper-left, and $d = 2$ in the upper-right graph. During the two periods of almost linear growth from 1910–1940 and 1970-2005, the derivative signal becomes an almost flat positive constant (i.e., the slope). During the other periods, the temperature signal has a very slow upward or downward trend and the derivative signal is almost zero.

We note the flat end-points in the case $d = 1$, which are due to the fact that the asymmetric derivative filters are the same at the end-points ranges as shown in the first equation of (23.9.9). The case $d = 2$ estimates the end-point derivatives better and possibly indicates a faster than linear growth in recent years.

The lower-left graph uses a minimum-R_s derivative filter with $N = 65$, $d = 2$, and smoothness order $s = 3$, resulting in a noticeably smoother estimated derivative than the LPSM case (the W input in `lpdiff` is discussed in the next section.) Finally, the lower-right graph shows the second derivative computed with the filter $B_2 = \text{lpdiff}(N, d, i)$ with $i = 2$, and compares it with the second difference signal, diff(diff(y)), which is even more noisy than the first difference. The following MATLAB code illustrates the computations:

```
d=1; i=1; B1 = lpdiff(N,d,i);                               % LPSM differentiation filters
plot(n, lpfilt(B1,y), n(2:end), diff(y),':');               % upper-left graph

d=2; i=1; B1 = lpdiff(N,d,i);
plot(n, lpfilt(B1,y), n(2:end), diff(y),':');               % upper-right graph

s=3; W = diag(hend(N,s));                                   % Henderson weighting matrix
d=2; i=1; B1 = lpdiff(N,d,i,W);                             % LPRS differentiation filters
plot(n, lpfilt(B1,y), n(2:end), diff(y),':');               % lower-left graph

d=2; i=2; B2 = lpdiff(N,d,i);                               % second derivative filters
plot(n, lpfilt(B2,y), n(3:end), diff(y,2),':');             % lower-right graph
```

The second derivative is essentially zero, being consistent with piecewise linear trends. Derivative signals can also be estimated for the SVD and Whittaker-Henderson methods. Since the outputs $\hat{x}_n$ of these methods are smooth signals, the corresponding derivatives can be simply computed as the difference signals, diff$(\hat{x}_n)$, with comparable results as the local polynomial methods. □

23.11 Minimum Roughness Weighted Polynomial Filters

The design of the LPSM filters was based on a least-squares criterion, such as (23.2.2), where all error terms were equally weighted within the filter's window:

$$\mathcal{J} = \sum_{m=-M}^{M} e_m^2 = \sum_{m=-M}^{M} (y_m - \hat{y}_m)^2 = \sum_{m=-M}^{M} \Big(y_m - \sum_{i=0}^{d} c_i m^i \Big)^2 = \min$$

This can be generalized by using unequal positive weights, w_m, $-M \le m \le M$:

$$\mathcal{J} = \sum_{m=-M}^{M} w_m e_m^2 = \sum_{m=-M}^{M} w_m \Big(y_m - \sum_{i=0}^{d} c_i m^i \Big)^2 = \min \qquad (23.11.1)$$

Introducing the diagonal matrix $W = \text{diag}([w_{-M}, \dots, w_0, \dots, w_M])$, we may write Eq. (23.11.1) compactly as:

$$\mathcal{J} = \mathbf{e}^T W \mathbf{e} = (\mathbf{y} - S\mathbf{c})^T W (\mathbf{y} - S\mathbf{c}) = \min \qquad (23.11.2)$$

where $\mathbf{y}, S, \mathbf{c}$ have the same meaning as in Eqs. (23.2.26)–(23.2.30). Differentiating with respect to $\mathbf{c}$ gives the orthogonality and normal equations:

$$S^T W \mathbf{e} = S^T W (\mathbf{y} - S\mathbf{c}) = 0 \quad \Leftrightarrow \quad (S^T W S)\mathbf{c} = S^T W \mathbf{y} \qquad (23.11.3)$$

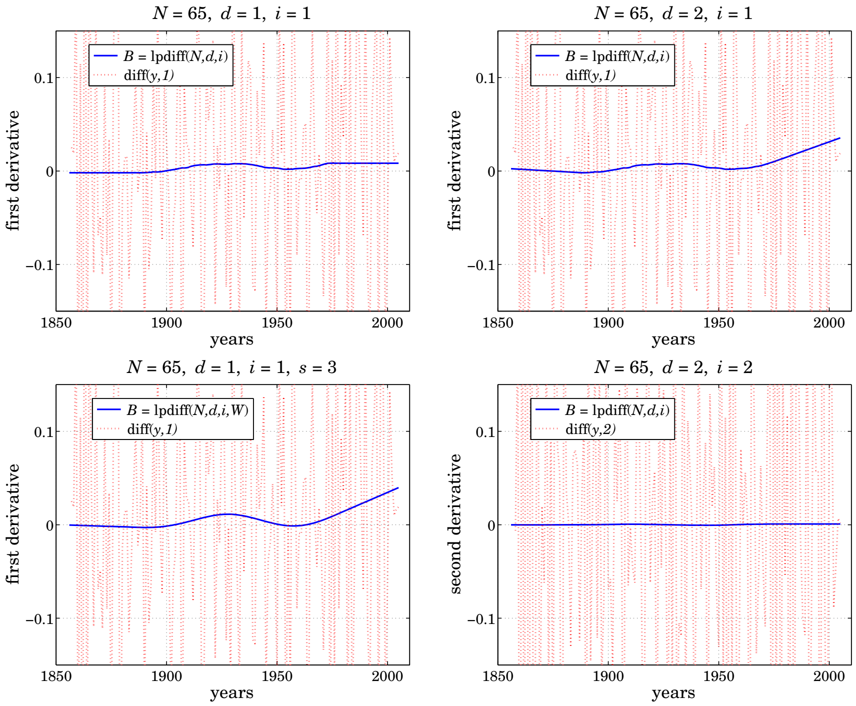


Fig. 23.10.5 Differentiated temperature signal.

with solution for **c** and the estimate $\hat{\mathbf{y}} = S\mathbf{c}$:

$$\begin{aligned} \mathbf{c} &= (S^TWS)^{-1}S^TW\mathbf{y} = G^T\mathbf{y} \\ \hat{\mathbf{y}} &= S\mathbf{c} = S(S^TWS)^{-1}S^TW\mathbf{y} = B^T\mathbf{y} \end{aligned} \tag{23.11.4}$$

where we defined

$$\boxed{\begin{aligned} G &= WS(S^TWS)^{-1} \\ B &= GS^T = WS(S^TWS)^{-1}S^T \end{aligned}} \tag{23.11.5}$$

The matrix B satisfies the following properties:

$$\begin{aligned} &S^TB = S^T \\ &B^T = W^{-1}BW \\ &BWB^T = BW = WB^T \end{aligned} \tag{23.11.6}$$

The first implies the usual polynomial-preserving moment constraints $S^T\mathbf{b}_m = \mathbf{u}_m$, for $-M \le m \le M$, where $\mathbf{b}_m$ is the mth column of B. The second shows that B is no longer symmetric, and the third may be used to simplify the minimized value of the performance index. Indeed, using the orthogonality property, we obtain:

$$\mathcal{J}_{\text{min}} = \mathbf{e}^TW\mathbf{e} = \mathbf{y}^TW\mathbf{y} - \mathbf{y}^TBW\mathbf{y} - \mathbf{y}^TWB^T\mathbf{y} + \mathbf{y}^TBWB^T\mathbf{y} = \mathbf{y}^TW\mathbf{y} - \mathbf{y}^TBW\mathbf{y}$$

A fourth property follows if we assume that the weights w_m are symmetric about their middle, $w_m = w_{-m}$, or more generally if W is assumed to be positive-definite, symmetric, and centro-symmetric, which implies that it remains invariant under reversal of its rows and its columns. The centro-symmetric property can be stated concisely as $JW = WJ$, where J is the column-reversing matrix consisting of ones along its anti-diagonal, that is, the reverse of a column vector is $\mathbf{b}^R = J\mathbf{b}$. Under this assumption on W, it can be shown that B is also centro-symmetric:

$$JB = BJ \quad \Rightarrow \quad \mathbf{b}_m^R = \mathbf{b}_{-m}\,, \quad -M \le m \le M \tag{23.11.7}$$

This can be derived by noting that reversing the basis vector $\mathbf{s}_i$ simply multiplies it by the phase factor $(-1)^i$, so that $JS = S\Omega$, where Ω is the diagonal matrix of phase factors $(-1)^i$, $i = 0, 1, \dots, d$. This then implies Eq. (23.11.7). Similarly one can show that $JG = G\Omega$, so that the reverse of each differentiation filter is $\mathbf{g}_i^R = (-1)^i\mathbf{g}_i$.

The filtering equations (23.2.33) and (23.2.34) retain their form. Among the possible weighting matrices W, we are interested in those such that the polynomial fitting problem (23.11.2) has an equivalent characterization as the minimization of the NRR subject to the polynomial-preserving constraints $S^T\mathbf{b}_m = \mathbf{u}_m$. To this end, we consider the constrained minimization of a generalized or "prefiltered" NRR:

$$\boxed{\mathcal{R} = \mathbf{b}^TV\mathbf{b} = \min\,, \quad \text{subject to} \quad S^T\mathbf{b} = \mathbf{u}} \tag{23.11.8}$$

for a given $(d+1)$-dimensional vector **u**. The $N\times N$ matrix V, where $N = 2M+1$, is assumed to be strictly positive-definite, symmetric, and Toeplitz. We may write component-wise:

$$\mathcal{R} = \sum_{n,m=-M}^{M} b(n)V_{n-m}b(m) = \frac{1}{2\pi}\int_{-\pi}^{\pi}|B(\omega)|^2V(\omega)d\omega \tag{23.11.9}$$

where we set $V_{nm} = V_{n-m}$ because of the Toeplitz property, and introduced the corresponding DTFTs:

$$B(\omega) = \sum_{n=-M}^{M} b(n)e^{-j\omega n}\,, \quad V(\omega) = \sum_{k=-\infty}^{\infty} V_k e^{-j\omega k} \tag{23.11.10}$$

One way to guarantee a positive-definite V is to take $V(\omega)$ to be the power spectrum of a given filter, say, $D(\omega)$, that is, choose $V(\omega) = |D(\omega)|^2$, so that $\mathcal{R}$ will be the ordinary NRR of the cascaded filter $F(\omega) = D(\omega)B(\omega)$ or $F(z) = D(z)B(z)$:

$$\mathcal{R} = \frac{1}{2\pi}\int_{-\pi}^{\pi}|B(\omega)|^2V(\omega)d\omega = \frac{1}{2\pi}\int_{-\pi}^{\pi}|B(\omega)D(\omega)|^2d\omega \tag{23.11.11}$$

The minimum-R_s or minimum-roughness filters discussed in Sec. 23.12 correspond to the choice $D(z) = (1-z^{-1})^s$, for some integer s. For a general V and $\mathbf{u}$, the solution of the problem (23.11.8) is obtained by introducing a Lagrange multiplier vector $\boldsymbol{\lambda}$:

$$\mathcal{J} = \mathbf{b}^T V\mathbf{b} + 2\boldsymbol{\lambda}^T(\mathbf{u} - S^T\mathbf{b}) = \min$$

leading to the solution:

$$\begin{aligned} \boldsymbol{\lambda} &= (S^T V^{-1} S)^{-1}\mathbf{u} \\ \mathbf{b} &= V^{-1}S\boldsymbol{\lambda} = V^{-1}S(S^T V^{-1}S)^{-1}\mathbf{u} \end{aligned} \tag{23.11.12}$$

If we choose $\mathbf{u}_m = [1, m, m^2, \dots, m^d]^T$ as the constraint vectors and put together the resulting solutions as the columns of a matrix B, then,

$$B = [\cdots \mathbf{b}_m \cdots] = V^{-1}S(S^T V^{-1}S)^{-1}[\cdots \mathbf{u}_m \cdots]$$

or, because $S^T = [\cdots \mathbf{u}_m \cdots]$,

$$\boxed{B = V^{-1}S(S^T V^{-1}S)^{-1}S^T} \tag{23.11.13}$$

This solution appears to be different from the solution (23.11.5) of the least-squares problem, $B = WS(S^T WS)^{-1}S^T$. Can the two solutions be the same? The trivial choice $V = W = I$ corresponds to the LPSM filters. The choice $V = W^{-1}$ is not acceptable because with V assumed Toeplitz, and W assumed diagonal, it would imply that all the weights are equal, which is again the LPSM case. A condition that guarantees the equivalence is the following [707,683]:

$$VWS = SC \quad\Rightarrow\quad WS = V^{-1}SC \tag{23.11.14}$$

where C is an invertible $(d+1)\times(d+1)$ matrix. Indeed, then $S^T WS = S^T V^{-1}SC$, and,

$$G = WS(S^T WS)^{-1} = V^{-1}S(S^T V^{-1}S)^{-1} \tag{23.11.15}$$

so that

$$B = WS(S^T WS)^{-1}S^T = V^{-1}S(S^T V^{-1}S)^{-1}S^T \tag{23.11.16}$$

For the minimum-R_s filters, the particular choices for W, V do indeed satisfy condition (23.11.14) with an *upper-triangular* matrix C. With the equivalence of the polynomial-fitting and minimum-NRR approaches at hand, we can also derive the corresponding predictive/interpolating differentiation filters. Choosing $\mathbf{u} = \mathcal{D}^i\mathbf{u}_t$ as the constraint vector in (23.11.12), we obtain,

$$\mathbf{b}_t^{(i)} = V^{-1}S(S^T V^{-1}S)^{-1}\mathcal{D}^i\mathbf{u}_t = WS(S^T WS)^{-1}\mathcal{D}^i\mathbf{u}_t \tag{23.11.17}$$

and at the sample values $t = m$, $-M \le m \le M$, or, at $\mathbf{u}_t = \mathbf{u}_m$, we obtain the differentiation matrix having the $\mathbf{b}_m^{(i)}$ as columns, $B^{(i)} = [\cdots \mathbf{b}_m^{(i)} \cdots]$:

$$B^{(i)} = WS(S^T WS)^{-1}\mathcal{D}^i S^T = V^{-1}S(S^T V^{-1}S)^{-1}\mathcal{D}^i S^T \tag{23.11.18}$$

Computationally, it is best to orthogonalize the basis S. Let $W = U^T U$ be the Cholesky factorization of the positive-definite symmetric matrix W, where U is an $N\times N$ upper-triangular factor. Then, performing the QR-factorization on the $N\times(d+1)$ matrix US, the above computations become:

$$\begin{aligned} W &= U^T U \\ US &= Q_0 R_0\,, \quad \text{with } Q_0^T Q_0 = I\,, \quad R_0 = (d+1)\times(d+1) \text{ upper-triangular} \\ B &= U^T Q_0 Q_0^T U^{-T} \\ B^{(i)} &= U^T Q_0 (R_0^{-T}\mathcal{D}^i R_0^T) Q_0^T U^{-T} \\ \mathbf{b}_t^{(i)} &= U^T Q_0 R_0^{-T}\mathcal{D}^i\mathbf{u}_t \end{aligned} \tag{23.11.19}$$

The MATLAB functions `lpsm`, `lpdiff`, `lpinterp` have the weighting matrix W as an additional input, which if omitted defaults to $W = I$. They implement Eqs. (23.11.19) and their full usage is:

```
[B,G] = lpsm(N,d,W);
B = lpdiff(N,d,i,W);
b = lpinterp(N,d,t,i,W);
```

The factorizations in Eq. (23.11.19) lead naturally to a related implementation in terms of discrete polynomials that are orthogonal with respect to the weighted inner product:

$$\mathbf{a}^T W\mathbf{b} = \sum_{m=-M}^{M} w_m\, a(m)b(m) \tag{23.11.20}$$

Such polynomials may be constructed from the monomials $s_i(m) = m^i$, $i = 0, 1, \dots, d$ via Gram-Schmidt orthogonalization applied with respect to the above inner product. The result of orthogonalizing the basis $S = [\mathbf{s}_0, \mathbf{s}_1, \dots, \mathbf{s}_d]$ is $Q = [\mathbf{q}_0, \mathbf{q}_1, \dots, \mathbf{q}_d]$ whose columns $q_i(m)$ are polynomials of order i in the variable m that are mutually orthogonal, that is, up to an overall normalization:

$$\mathbf{q}_i^T W\mathbf{q}_j = \delta_{ij}D_i\,, \quad i,j = 0, 1, \dots, d \quad\Rightarrow\quad Q^T WQ = D \tag{23.11.21}$$

where $D = \text{diag}([D_0, D_1, \dots, D_d])$ is the diagonal matrix of the (positive) normalization factors D_i. These factors can be selected to be unity if so desired. For the minimum-roughness filters, these polynomials are special cases of the Hahn orthogonal polynomials, whose properties are discussed in Sec. 23.13. For unity weights $w_m = 1$, the polynomials reduce to the discrete Chebyshev/Gram polynomials.

Numerically, these polynomials can be constructed from the factorization (23.11.19). Since D is positive-definite, we may define $D^{1/2} = \text{diag}([D_0^{1/2}, D_1^{1/2}, \dots, D_d^{1/2}])$ to be its square root. Then we construct Q, R in terms of the factors U, Q_0, R_0:

$$Q = U^{-1}Q_0 D^{1/2}\,, \quad R = D^{-1/2}R_0 \tag{23.11.22}$$

where R is still upper-triangular. Then, we have $Q^T WQ = D$ and

$$QR = U^{-1}Q_0 D^{1/2}D^{-1/2}R_0 = U^{-1}Q_0 R_0 = U^{-1}US = S$$

which is equivalent to the Gram-Schmidt orthogonalization of the basis S, and leads to the following equivalent representation of Eq. (23.11.19):

$$\begin{aligned}
&S = QR\,, \quad \text{with } Q^TWQ = D\,, \quad R = (d{+}1)\times(d{+}1) \text{ upper-triangular}\\
&B = WQD^{-1}Q^T\\
&B^{(i)} = WQD^{-1}(R^{-T}\mathcal{D}^iR^T)Q^T\\
&\mathbf{b}_t^{(i)} = WQD^{-1}R^{-T}\mathcal{D}^i\mathbf{u}_t
\end{aligned} \tag{23.11.23}$$

Since $Q = [\mathbf{q}_0, \mathbf{q}_1, \dots, \mathbf{q}_d]$, the matrix B can be expressed as,

$$B = WQD^{-1}Q^T = W\sum_{r=0}^{d} D_r^{-1}\mathbf{q}_r\mathbf{q}_r^T \tag{23.11.24}$$

and for diagonal W, we have component-wise:

$$\boxed{b_m(k) = B_{km} = w_k\sum_{r=0}^{d}\frac{q_r(k)q_r(m)}{D_r}} \qquad -M \le m, k \le M \tag{23.11.25}$$

The sum in (23.11.25) can be simplified further using the Christoffel-Darboux identity discussed in Sec. 23.13. The polynomial predictive interpolation filters $\mathbf{b}_t^{(i)}$ can also be expressed in a similar summation form:

$$\boxed{b_t^{(i)}(k) = w_k\sum_{r=0}^{d}\frac{q_r(k)q_r^{(i)}(t)}{D_r}} \tag{23.11.26}$$

where $q_r^{(i)}(t)$ is the ith derivative of the polynomial $q_r(t)$ obtained from $q_r(m)$ by replacing the discrete variable m by t. This can be justified as follows. The mth rows of the matrices S and Q are the $(d{+}1)$-dimensional vectors:

$$\begin{aligned}
\mathbf{u}_m^T &= [s_0(m), s_1(m), \dots, s_d(m)] = [1, m, \dots, m^d]\\
\mathbf{p}_m^T &= [q_0(m), q_1(m), \dots, q_d(m)]
\end{aligned} \tag{23.11.27}$$

and since $S = QR$, they are related by $\mathbf{u}_m^T = \mathbf{p}_m^TR$. Replacing m by t preserves this relationship, so that $\mathbf{u}_t^T = \mathbf{p}_t^TR$, or,

$$\mathbf{u}_t = R^T\mathbf{p}_t\,, \quad \text{where } \mathbf{p}_t = [q_0(t), q_1(t), \dots, q_d(t)]^T \tag{23.11.28}$$

Differentiating i times, we obtain

$$\mathcal{D}^i\mathbf{u}_t = \mathbf{u}_t^{(i)} = R^T\mathbf{p}_t^{(i)} \quad\Rightarrow\quad \mathbf{p}_t^{(i)} = R^{-T}\mathcal{D}^i\mathbf{u}_t \tag{23.11.29}$$

and therefore $\mathbf{b}_t^{(i)}$ from Eq. (23.11.23) can be written in the following form, which implies Eq. (23.11.26):

$$\boxed{\mathbf{b}_t^{(i)} = WQD^{-1}\mathbf{p}_t^{(i)}} \tag{23.11.30}$$

As in the case of the LPSM filters, for the special case $d = N-1$, the interpolation filters correspond to Lagrange interpolation. In this case Q becomes an invertible $N{\times}N$ matrix satisfying the weighted unitarity property $Q^TWQ = D$, which implies

$$Q^{-1} = D^{-1}Q^TW \tag{23.11.31}$$

from which we obtain the *completeness* property:

$$\boxed{QD^{-1}Q^T = W^{-1}} \tag{23.11.32}$$

which shows that $B = I$. Similarly, using $WQD^{-1} = Q^{-T}$, we obtain from (23.11.23) the usual Lagrange interpolation polynomials:

$$\mathbf{b}_t = WQD^{-1}R^{-T}\mathbf{u}_t = Q^{-T}R^{-T}\mathbf{u}_t = S^{-T}\mathbf{u}_t \tag{23.11.33}$$

With $d = N-1$, the matrix Q is an orthogonal basis for the full space $\mathbb{R}^N$. One of the applications of Eq. (23.11.31) is the representation of signals, such as images or speech in terms of *orthogonal-polynomial moments* [721–734].

Given an N-dimensional signal block $\mathbf{y}$, such as a row in a scanned image, we define the N-dimensional vector of moments with respect to the polynomials Q,

$$\boldsymbol{\mu} = D^{-1}Q^TW\mathbf{y} \quad\Rightarrow\quad \mu_r = \frac{1}{D_r}\sum_{n=-M}^{M} q_r(n)w_ny_n\,, \quad r = 0, 1, \dots, N-1 \tag{23.11.34}$$

Because of Eq. (23.11.31), we have $\boldsymbol{\mu} = Q^{-1}\mathbf{y}$, which allows the reconstruction of $\mathbf{y}$ from its moments:

$$\mathbf{y} = Q\boldsymbol{\mu} \quad\Rightarrow\quad y_n = \sum_{r=0}^{N-1} q_r(n)\mu_r\,, \quad -M \le n \le M \tag{23.11.35}$$

23.12 Henderson Filters

All the results of the previous section find a concrete realization in the minimum-R_s filters that we discuss here. Consider the order-s backward difference filter and its impulse response defined by:

$$D_s(z) = (1-z^{-1})^s \quad\Leftrightarrow\quad d_s(k) = (-1)^k\binom{s}{k}, \quad k = 0, 1, \dots, s \tag{23.12.1}$$

This follows from the binomial expansion:

$$(1-z^{-1})^s = \sum_{k=0}^{s}(-1)^k\binom{s}{k}z^{-k} \tag{23.12.2}$$

The operation of the filter $D_s(z)$ on a signal f_n, with output g_n, is usually denoted in terms of the backward difference operator $\nabla f_n = f_n - f_{n-1}$ as follows:

$$g_n = \nabla^sf_n = \sum_{k=0}^{s} d_s(k)f_{n-k} = \sum_{k=0}^{s}(-1)^k\binom{s}{k}f_{n-k} \tag{23.12.3}$$

If the signal f_n is restricted over the range $-M \le n \le M$, then because $0 \le k \le s$ and $-M \le n-k \le M$, the above equation can be written in the more precise form:

$$g_n = \nabla^s f_n = \sum_{k=\max(0,n-M)}^{\min(s,n+M)} (-1)^k \binom{s}{k} f_{n-k}\,, \quad -M \le n \le M+s \tag{23.12.4}$$

Eq. (23.12.4) gives the full convolutional output $g_n = (d_s * f)_n$, while (23.12.3) is the corresponding steady-state output, obtained by restricting the output index n to the range $-M+s \le n \le M$. Defining the $(N+s)$-dimensional output vector $\mathbf{g}$ and N-dimensional input vector $\mathbf{f}$, where $N = 2M+1$,

$$\mathbf{g} = [g_{-M}, \dots, g_M, \dots, g_{M+s}]^T\,, \quad \mathbf{f} = [f_{-M}, \dots, f_M]^T\,,$$

we may write the full filtering equation (23.12.4) in matrix form:

$$\mathbf{g} = D_s \mathbf{f} \tag{23.12.5}$$

where D_s is the full $(N+s)\times N$ convolutional matrix of the filter $d_s(k)$ defined by its matrix elements:

$$(D_s)_{nm} = d_s(n-m)\,, \quad -M \le n \le M+s, \quad -M \le m \le M \tag{23.12.6}$$

and subject to the restriction that only the values $0 \le n-m \le s$ will result in a non-zero matrix element. The MATLAB functions `binom` and `diffmat` allow the calculation of the binomial coefficients $d_s(k)$ and the convolution matrix D_s:

```
d = binom(s,k);      % binomial coefficients d_s(k)
D = diffmat(s,N);    % (N+s)×N difference convolution matrix
```

For example, the convolution matrix for $N = 7$ and $s = 3$ is:

$$D_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 1 & 0 & 0 & 0 & 0 & 0 \\ 3 & -3 & 1 & 0 & 0 & 0 & 0 \\ -1 & 3 & -3 & 1 & 0 & 0 & 0 \\ 0 & -1 & 3 & -3 & 1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -3 & 1 & 0 \\ 0 & 0 & 0 & -1 & 3 & -3 & 1 \\ 0 & 0 & 0 & 0 & -1 & 3 & -3 \\ 0 & 0 & 0 & 0 & 0 & -1 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

The function `diffmat` is simply a call to `convmat`:

```
D = convmat(binom(s),N);
```

A minimum-R_s filter $B(z)$ is defined to minimize the NRR of the cascaded filter $F(z) = D_s(z)B(z)$ subject to the $d{+}1$ linear constraints $S^T\mathbf{b} = \mathbf{u}$, for a given constraint vector $\mathbf{u}$, where $\mathbf{b}$ denotes the impulse response of $B(z)$ assumed to be double-sided, that is, b_n, $-M \le n \le M$.

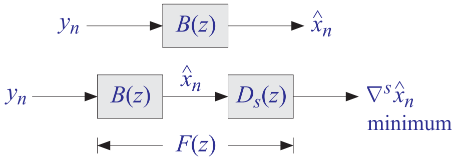


Fig. 23.12.1 Design and smoothing by minimum-R_s filter.

The actual smoothing of data is carried out by the filter $B(z)$ itself, whereas the filter $F(z)$ is used to design $B(z)$. This is depicted in Fig. 23.12.1 in which the filtered output is $\hat{x}_n$, and the output of $F(z)$ is the differenced signal $\nabla^s\hat{x}_n$ whose mean-square value may be taken as a measure of smoothness to be minimized.

Letting $f_n = \nabla^s b_n$ be the impulse response of the filter $F(z)$, or in matrix form $\mathbf{f} = D_s\mathbf{b}$, the corresponding cascaded NRR will be:

$$\mathcal{R}_s = \mathbf{f}^T\mathbf{f} = \sum_{n=-M}^{M+s} f_n^2 = \sum_{n=-M}^{M+s} (\nabla^s b_n)^2 = \frac{1}{2\pi}\int_{-\pi}^{\pi} |D_s(\omega)B(\omega)|^2\, d\omega$$

Since $\mathbf{f}^T\mathbf{f} = \mathbf{b}^T(D_s^T D_s)\mathbf{b}$, we can state the design condition of the minimum-R_s filters as

$$\boxed{\mathcal{R}_s = \sum_{n=-M}^{M+s} (\nabla^s b_n)^2 = \mathbf{b}^T(D_s^T D_s)\mathbf{b} = \min\,, \quad \text{subject to} \quad S^T\mathbf{b} = \mathbf{u}} \tag{23.12.7}$$

This has exactly the same form as Eq. (23.11.8) with $V = D_s^T D_s$. The minimization of $\mathcal{R}_s$ justifies the name "minimum-R_s " filters. The minimum-R_0 LPSM filters of Sec. 23.8 correspond to $s = 0$. In the actuarial literature, the following criterion is used instead, which differs from $\mathcal{R}_s$ by a normalization factor:

$$R_s = \frac{\mathbf{b}^T(D_s^T D_s)\mathbf{b}}{\mathbf{d}_s^T\mathbf{d}_s} = \frac{\mathcal{R}_s}{\mathbf{d}_s^T\mathbf{d}_s} = \min \tag{23.12.8}$$

where R_s is referred as the "smoothing coefficient", $\mathbf{d}_s$ is the impulse response vector of the filter $D_s(z)$, and $\mathbf{d}_s^T\mathbf{d}_s$ is the NRR of $D_s(z)$. Using a binomial identity (a special case of (23.12.13) for $k = 0$), we have,

$$\mathbf{d}_s^T\mathbf{d}_s = \sum_{m=0}^{s} d_s^2(m) = \sum_{m=0}^{s}\binom{s}{m}^2 = \binom{2s}{s} \tag{23.12.9}$$

The criterion (23.12.7) provides a measure of smoothness. To see this, let $\hat{x}_n$ be the result of filtering an arbitrary stationary signal y_n through the filter $B(z)$. If $S_{yy}(\omega)$ is the power spectrum of y_n , then the power spectra of the filtered output $\hat{x}_n$ and of the differenced output $\nabla^s\hat{x}_n$ will be $|B(\omega)|^2 S_{yy}(\omega)$ and $|D_s(\omega)B(\omega)|^2 S_{yy}(\omega)$, respectively. Therefore, the mean-square value of $\nabla^s\hat{x}_n$ will be:

$$E\big[(\nabla^s\hat{x}_n)^2\big] = \frac{1}{2\pi}\int_{-\pi}^{\pi} |D_s(\omega)B(\omega)|^2 S_{yy}(\omega)\, d\omega \tag{23.12.10}$$

If y_n is white noise of variance σ^2, or if we assume that $S_{yy}(\omega)$ is bounded from above by a constant, such as $S_{yy}(\omega)\le\sigma^2$, then we obtain:

$$E\big[\big(\nabla^s\hat{x}_n\big)^2\big] \le \frac{1}{2\pi}\int_{-\pi}^{\pi}|D_s(\omega)B(\omega)|^2\sigma^2\,d\omega = \mathcal{R}_s\sigma^2 \tag{23.12.11}$$

For white noise, $S_{yy}(\omega)=\sigma^2$, Eq. (23.12.11) becomes an equality. Thus, minimizing $\mathcal{R}_s$ will minimize $E\big[\big(\nabla^s\hat{x}_n\big)^2\big]$ and tend to result in a smoother filtered signal $\hat{x}_n$. This property justifies the term "minimum-roughness" filters.

The choice $s=2$ is preferred in smoothing financial and business-cycle data, and is used also by the related method of the Whittaker-Henderson or Hodrick-Prescott filter. The choice $s=3$ is standard in the actuarial literature. The choice $s=4$ is not common but it was used by De Forest [649–652] who was the first to formulate and solve the minimum-R_s problem in 1871. Others, like Hardy and Henderson have considered the minimum-R_3 problem, while Sheppard [660] solved the minimum-R_s problem in general.

Henderson [663] was the first to show the *equivalence* between the NRR minimization problem (23.12.7) with $V=D_s^TD_s$ and the weighted least-squares polynomial fitting problem (23.11.1) using the so-called Henderson weights w_m. Therefore, the minimum-R_s filters are often referred to as *Henderson filters*. They are used widely in seasonal-adjustment, census, and business-cycle extraction applications. We discuss this equivalence next, following essentially Henderson's method.

The elements of the $N\times N$ matrix $V=D_s^TD_s$ are $(D_s^TD_s)_{nm}=V_{nm}=V_{n-m}$, where V_k is the autocorrelation function of the power spectrum $V(\omega)=|D_s(\omega)|^2$. Working in the z-domain, we have the spectral density:

$$V(z)=D_s(z)D_s(z^{-1})=(1-z^{-1})^s(1-z)^s=(-1)^sz^s(1-z^{-1})^{2s} \tag{23.12.12}$$

which shows that $V(z)$ effectively acts as the $(2s)$-difference operation ∇^{2s}. Taking inverse z-transforms of both sides of (23.12.12), we obtain:

$$V_k=\sum_{m=\max(0,k)}^{\min(s,k+s)}d_s(m)d_s(m-k)=(-1)^sd_{2s}(k+s)\,,\quad -s\le k\le s \tag{23.12.13}$$

or, explicitly in terms of the definition of d_s:

$$V_k=(-1)^k\sum_{m=\max(0,k)}^{\min(s,k+s)}\binom{s}{m}\binom{s}{m-k}=(-1)^k\binom{2s}{s+k}\,,\quad -s\le k\le s \tag{23.12.14}$$

or,

$$V_k=(-1)^k\frac{(2s)!}{(s+k)!\,(s-k)!}\,,\quad -s\le k\le s \tag{23.12.15}$$

The V matrix is a banded Toeplitz matrix with bandwidth ±s, whose central row or central column consist of the numbers V_k, $-s\le k\le s$, with V_0 positioned at the center of the matrix. As an example,

$$V=D_3^TD_3=\begin{bmatrix} 20 & -15 & 6 & -1 & 0 & 0 & 0\\ -15 & 20 & -15 & 6 & -1 & 0 & 0\\ 6 & -15 & 20 & -15 & 6 & -1 & 0\\ -1 & 6 & -15 & 20 & -15 & 6 & -1\\ 0 & -1 & 6 & -15 & 20 & -15 & 6\\ 0 & 0 & -1 & 6 & -15 & 20 & -15\\ 0 & 0 & 0 & -1 & 6 & -15 & 20 \end{bmatrix}$$

with central column or central row:

$$V_k=\{-1,\;6,\;-15,\;20,\;-15,\;6,\;-1\}\quad\text{for}\quad k=\{-2,-1,0,1,2\}$$

To understand the action of V as the difference operator ∇^{2s}, let $\mathbf{f}$ be an N dimensional vector indexed for $-M\le m\le M$, and form the output N-dimensional vector:

$$\mathbf{g}=V\mathbf{f}\quad\Rightarrow\quad g_n=\sum_{m=-M}^{M}V_{n-m}f_m\,,\quad -M\le n\le M \tag{23.12.16}$$

where $n-m$ is further restricted such that $-s\le n-m\le s$. Next, consider an extended version of $\mathbf{f}$ obtained by padding s zeros in front and s zeros at the end, so that the extended vector $\mathbf{f}^{\text{ext}}$ will be indexed over, $-(M+s)\le m\le(M+s)$:

$$\mathbf{f}^{\text{ext}}=\big[\underbrace{0,\dots,0}_{s},f_{-M},\dots,f_0,\dots,f_M,\underbrace{0,\dots,0}_{s}\big]^T$$

Then, the summation in Eq. (23.12.16) can be extended as,

$$g_n=\sum_{m=-M-s}^{M+s}V_{n-m}f_m^{\text{ext}}\,,\quad -M\le n\le M \tag{23.12.17}$$

But because of the restriction $-s\le n-m\le s$, the above summation can be restricted to be over $n-s\le m\le n+s$, which is a subrange of the range $-(M+s)\le m\le(M+s)$ because we assumed $-M\le n\le M$. Thus, we may write:

$$g_n=\sum_{m=-n-s}^{n+s}V_{n-m}f_m^{\text{ext}}\,,\quad -M\le n\le M$$

or, changing to $k=n-m$,

$$g_n=\sum_{k=-s}^{s}V_kf_{n-k}^{\text{ext}}=(-1)^s\sum_{k=-s}^{s}d_{2s}(s+k)f_{n-k}^{\text{ext}}=(-1)^s\sum_{i=0}^{2s}d_{2s}(i)f_{n+s-i}^{\text{ext}} \tag{23.12.18}$$

but that is precisely the ∇^{2s} operator:

$$g_n=(-1)^s\nabla^{2s}f_{n+s}^{\text{ext}}\,,\quad -M\le n\le M \tag{23.12.19}$$

If f_m^{ext} is a polynomial of degree $(2s+i)$, then the $(2s)$-differencing operation will result into a polynomial of degree i. Suppose that we start with the weighted monomial:

$$f_m = w_m m^i\,, \quad -M \le m \le M \tag{23.12.20}$$

where the weighting function w_m is itself a polynomial of degree $2s$, then in order for the extended vector f_m^{ext} to vanish over $M < |m| \le M+s$, the function w_m must have zeros at these points, that is,

$$w_m = 0\,, \quad \text{for} \;\; m = \pm(M+1), \pm(M+2), \dots, \pm(M+s)$$

This condition fixes w_m uniquely, up to a normalization constant:

$$\boxed{w_m = \prod_{i=1}^{s}\big[(M+i)^2 - m^2\big]} \qquad \text{(Henderson weights)} \tag{23.12.21}$$

These are called *Henderson weights.* Because the extended signal f_m^{ext} is a polynomial of degree $(2s+i)$, it follows that the signal g_n will be a polynomial of degree i.

Defining the $N\times N$ diagonal matrix $W = \text{diag}\big([w_{-M}, \dots, w_0, \dots, w_M]\big)$, we can write (23.12.20) vectorially in terms of the monomial basis vector $\mathbf{s}_i$ as $\mathbf{f} = W\mathbf{s}_i$. We showed that the matrix operation $\mathbf{g} = V\mathbf{f} = VW\mathbf{s}_i$ results into a polynomial of degree i, which therefore can be expanded as a linear combination of the monomials $\mathbf{s}_0, \mathbf{s}_1, \dots, \mathbf{s}_i$ up to order i, that is,

$$VW\mathbf{s}_i = \sum_{j=0}^{i} \mathbf{s}_j C_{ji} \tag{23.12.22}$$

for appropriate coefficients C_{ji}, which may thought of as the matrix elements of an upper-triangular matrix. Applying this result to each basis vector of $S = [\mathbf{s}_0, \mathbf{s}_1, \dots, \mathbf{s}_d]$ up to order d, it follows that

$$VWS = SC\,, \quad C = (d{+}1)\times(d{+}1) \text{ upper-triangular} \tag{23.12.23}$$

But, this is exactly the condition (23.11.14). Thus, we have shown the equivalence of the NRR minimization problem (23.12.7) with $V = D_s^T D_s$ and the weighted least-squares polynomial fitting problem (23.11.1) with the Henderson weights w_m. The rest of the results of Sec. 23.11 then carry through unchanged.

The MATLAB function `lprs` implements the design. It constructs the W matrix from the Henderson weights and passes it into the function `lpsm`:

```
[B,G] = lprs(N,d,s);          % local polynomial minimum-Rs filters
```

The Henderson weights w_m, $-M \le m \le M$ are calculated by the function `hend`:

```
w = hend(N,s);          % Henderson weights
```

In the next section, we derive closed-form expressions for the Henderson filters using Hahn orthogonal polynomials. Analytical expressions can also be derived working with the non-orthogonal monomial basis S. It follows from $B = WS(S^TWS)^{-1}S^T$ that the kth component of the mth filter will be:

$$b_m(k) = B_{km} = w_k \sum_{i,j=0}^{d} k^i m^j \Phi_{ij} = w_k\, \mathbf{u}_k^T \Phi \mathbf{u}_m \tag{23.12.24}$$

where $\mathbf{u}_k = [1, k, k^2, \dots, k^d]^T$ and Φ is the inverse of the Hankel matrix $F = S^TWS$ whose matrix elements are the weighted inner products:

$$F_{ij} = (S^TWS)_{ij} = \mathbf{s}_i^T W \mathbf{s}_j = \sum_{m=-M}^{M} w_m m^{i+j} \equiv F_{i+j}\,, \quad i,j = 0,1,\dots,d \tag{23.12.25}$$

Except for the factor w_k and the different values of Φ_{ij} the expressions are similar to those of the LPSM filters of Sec. 23.3. The matrix Φ has a similar checkerboard structure. For example, we have for the commonly used case $d = 3$ and $s = 3$:

$$b_m(k) = w_k\,[1,\ k,\ k^2,\ k^3]\begin{bmatrix} \Phi_{00} & 0 & \Phi_{02} & 0 \\ 0 & \Phi_{11} & 0 & \Phi_{13} \\ \Phi_{20} & 0 & \Phi_{22} & 0 \\ 0 & \Phi_{31} & 0 & \Phi_{33} \end{bmatrix}\begin{bmatrix} 1 \\ m \\ m^2 \\ m^3 \end{bmatrix} \tag{23.12.26}$$

where

$$w_k = \big[(M+1)^2 - k^2\big]\big[(M+2)^2 - k^2\big]\big[(M+3)^2 - k^2\big] \tag{23.12.27}$$

and

$$F = \begin{bmatrix} F_0 & 0 & F_2 & 0 \\ 0 & F_2 & 0 & F_4 \\ F_2 & 0 & F_4 & 0 \\ 0 & F_4 & 0 & F_6 \end{bmatrix} \quad \Rightarrow \quad \Phi = F^{-1} = \begin{bmatrix} \Phi_{00} & 0 & \Phi_{02} & 0 \\ 0 & \Phi_{11} & 0 & \Phi_{13} \\ \Phi_{20} & 0 & \Phi_{22} & 0 \\ 0 & \Phi_{31} & 0 & \Phi_{33} \end{bmatrix}$$

where we obtain from the checkerboard submatrices:

$$\begin{bmatrix} \Phi_{00} & \Phi_{02} \\ \Phi_{20} & \Phi_{22} \end{bmatrix} = \begin{bmatrix} F_0 & F_2 \\ F_2 & F_4 \end{bmatrix}^{-1}, \quad \begin{bmatrix} \Phi_{11} & \Phi_{13} \\ \Phi_{31} & \Phi_{33} \end{bmatrix} = \begin{bmatrix} F_2 & F_4 \\ F_4 & F_6 \end{bmatrix}^{-1} \tag{23.12.28}$$

The corresponding F-factors for $s = 3$ are:

$$\begin{aligned} F_0 &= \frac{2}{35}(2M+7)(2M+5)(2M+3)(2M+1)(M+3)(M+2)(M+1) \\ F_2 &= \frac{1}{9}M(M+4)F_0 \\ F_4 &= \frac{1}{11}(3M^2 + 12M - 4)F_2 \\ F_6 &= \frac{1}{143}(15M^4 + 120M^3 + 180M^2 - 240M + 68)F_2 \end{aligned}$$

which give rise to the matrix elements of Φ:

$$\begin{aligned}
\Phi_{00} &= 315\,(3M^2+12M-4)\,/D_1\\
\Phi_{02} &= -3465/D_1\\
\Phi_{22} &= 31185/D_1\\
\Phi_{11} &= 1155\,(15M^4+120M^3+180M^2-240M+68)\,/D_2\\
\Phi_{13} &= -15015\,(3M^2+12M-4)\,/D_2\\
\Phi_{33} &= 165165/D_2
\end{aligned}$$

with the denominator factors:

$$\begin{aligned}
D_1 &= 8\,(2M+9)\,(2M+7)\,(2M+5)\,(2M+3)\,(M+3)\,(M+2)\,(M+1)\,(4M^2-1)\\
D_2 &= 8M\,(M-1)\,(M+4)\,(M+5)\,D_1
\end{aligned}$$

In particular, setting $m = 0$ we find the central filter $b_0(k)$, which for the case $d = 3$ and $s = 3$, is referred to as "Henderson's ideal formula:"

$$b_0(k) = w_k\,(\Phi_{00} + k^2\Phi_{02})$$

or, with $w_k = \big[(M{+}1)^2{-}k^2\big]\big[(M{+}2)^2{-}k^2\big]\big[(M{+}3)^2{-}k^2\big]$:

$$b_0(k) = \frac{315\,(3M^2+12M-4-11k^2)\,w_k}{8\,(2M{+}9)\,(2M{+}7)\,(2M{+}5)\,(2M{+}3)\,(M{+}3)\,(M{+}2)\,(M{+}1)\,(4M^2{-}1)} \qquad (23.12.29)$$

The corresponding predictive/interpolating differentiation filters $b_t^{(i)}(k)$ are given by a similar expression:

$$b_t^{(i)}(k) = w_k\mathbf{u}_k^T\Phi\mathcal{D}^i\mathbf{u}_t \qquad (23.12.30)$$

or, explicitly, for the $d = s = 3$ case and differentiation order $i = 0, 1, 2, 3$:

$$b_t^{(i)}(k) = w_k\,[1,\ k,\ k^2,\ k^3]\begin{bmatrix}\Phi_{00} & 0 & \Phi_{02} & 0\\ 0 & \Phi_{11} & 0 & \Phi_{13}\\ \Phi_{20} & 0 & \Phi_{22} & 0\\ 0 & \Phi_{31} & 0 & \Phi_{33}\end{bmatrix}\begin{bmatrix}0&0&0&0\\1&0&0&0\\0&2&0&0\\0&0&3&0\end{bmatrix}^i\begin{bmatrix}1\\t\\t^2\\t^3\end{bmatrix} \qquad (23.12.31)$$

Example 23.12.1: *USD/Euro exchange rate.* Consider four methods of smoothing the USD/Euro foreign exchange rate for the years 1999-08. The monthly data are available from the web site: `http://research.stlouisfed.org/fred2/series/EXUSEU`

The upper-left graph in Fig. 23.12.2 shows the smoothing by an LPSM filter of length $N = 19$ and polynomial order $d = 3$. In the upper-right graph a minimum-R_s Henderson filter was used with $N = 19$, $d = 3$, and smoothness order $s = 3$.

The middle-left graph uses the SVD signal enhancement method with embedding order $M = 8$ and rank $r = 2$.

The middle-right graph uses the Whittaker-Henderson, or Hodrick-Prescott filter with smoothing parameter $\lambda = 100$ and smoothness order $s = 3$.

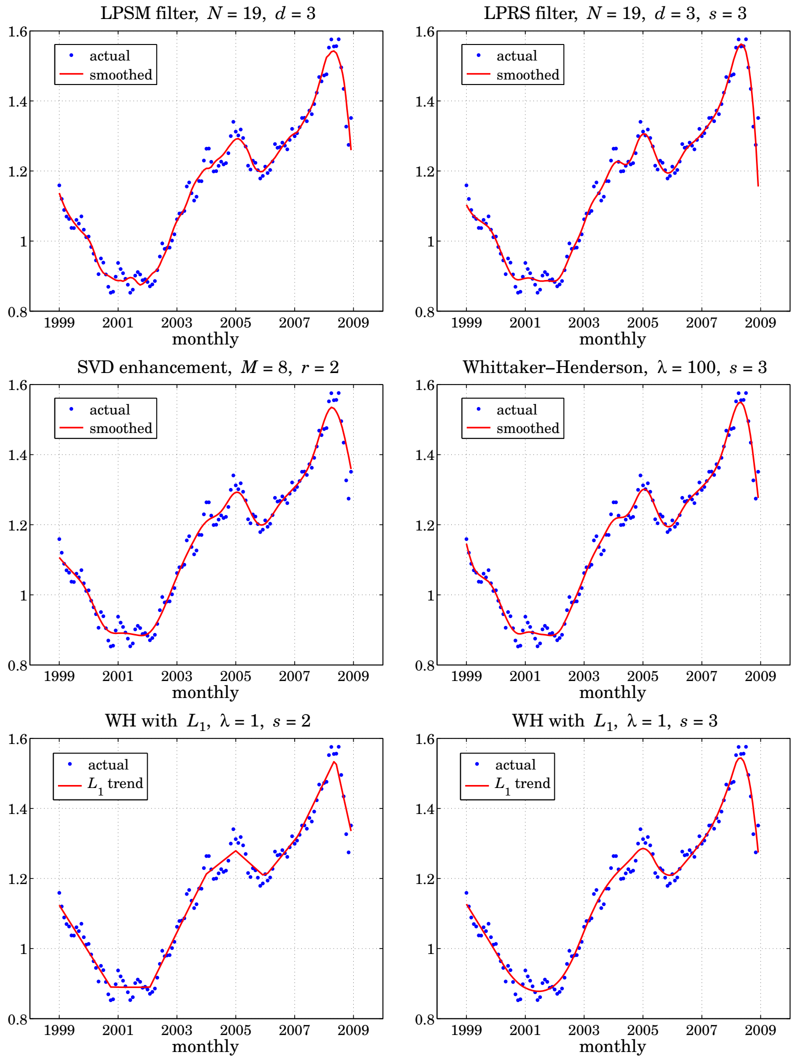


Fig. 23.12.2 Smoothing of USD/Euro exchange rate.

The lower left and right graphs use the Whittaker-Henderson regularization filter with the L_1 criterion with differentiation orders $s = 2$ and $s = 3$ and smoothing parameter $\lambda = 1$, implemented with the CVX package.[†]. The $s = 2$ case represents the smoothed signal in piece-wise linear form. The L_1 case is discussed further in Sec. 26.7.

[†] `http://cvxr.com/cvx/`

The following MATLAB code illustrates the generation of the four graphs:

```
Y = loadfile('exuseu.dat');              % data file available in the OSP toolbox
y = Y(:,4); t = taxis(y,12,1999);        % extract signal y_n from data file
                                         % the function taxis defines the t-axis
N=19; d=3; x1 = lpfilt(lpsm(N,d),y);     % LPSM filter
s=3; x2 = lpfilt(lprs(N,d,s),y);         % LPRS filter
M=8; r=2; x3 = svdenh(y,M,r);            % SVD enhancement
la=100; s=3; x4 = whsm(y,la,s);          % Whittaker-Henderson

s = 2; la = 1; N = length(y);            % Whittaker-Henderson with L_1 criterion
D = diff(eye(N),s);                      % for x6, use s = 3

cvx_begin                                % use CVX package to solve the L_1 problem
    variable x5(N)
    minimize( sum_square(y-x5) + la * norm(D*x5,1) )
cvx_end

figure; plot(t,y,'.', t,x1,'-'); figure; plot(t,y,'.', t,x2,'-');
figure; plot(t,y,'.', t,x3,'-'); figure; plot(t,y,'.', t,x4,'-');
figure; plot(t,y,'.', t,x5,'-'); figure; plot(t,y,'.', t,x6,'-');
```

All methods have comparable performance and can handle the end-point problem. □

The computational procedures implemented into the function `lprs` were outlined in Eq. (23.11.19). The related orthogonalized basis Q defined in Eq. (23.11.23) will be realized in terms of the Hahn orthogonal polynomials.

A direct consequence of upper-triangular nature of the matrix C in Eq. (23.12.23) is that the basis Q becomes an *eigenvector basis* for the matrix VW [707,683]. To see this, substitute $S = QR$ into (23.12.23),

$$VWQR = QRC \quad \Rightarrow \quad VWQ = Q\Lambda\,, \quad \Lambda = RCR^{-1} \tag{23.12.32}$$

Multiplying both sides by Q^TW and using the property $Q^TWQ = D$, we obtain:

$$Q^TWVWQ = Q^TWQ\Lambda = D\Lambda \tag{23.12.33}$$

Because R and C are both upper-triangular, so will be Λ and $D\Lambda$. But the left-hand side of (23.12.33) is a symmetric matrix, and so must be the right-hand side $D\Lambda$. This requires that $D\Lambda$ and hence Λ be a diagonal matrix, e.g., $\Lambda = \text{diag}([\lambda_0, \lambda_1, \dots, \lambda_d])$. This means that the rth column of Q is an eigenvector:

$$VW\mathbf{q}_r = \lambda_r \mathbf{q}_r\,, \quad r = 0, 1, \dots, d \tag{23.12.34}$$

Choosing $d = N-1$ would produce all the eigenvectors of VW. In this case, we have $Q^{-1} = D^{-1}Q^TW$ and we obtain the decomposition:

$$VW = Q\Lambda Q^{-1} = Q\Lambda D^{-1}Q^TW \quad \Rightarrow \quad V = Q(\Lambda D^{-1})Q^T$$

We also find for the inverse of $V = D_s^T D_s$:

$$V^{-1} = WQ\Lambda^{-1}D^{-1}Q^TW$$

There exist [677–679] similar and efficient ways to calculate $V^{-1} = (D_s^T D_s)^{-1}$. The eigenvalues λ_r can be shown to be [707]:

$$\lambda_r = \frac{(2s+r)!}{r!} = \prod_{i=1}^{2s}(r+i)\,, \quad r = 0, 1, \dots, d \tag{23.12.35}$$

As we see in the next section, the rth column $q_r(n)$ of Q is a Hahn polynomial of degree r in n, and hence $W\mathbf{q}_r$, or component-wise, $w_n q_r(n)$, will be a polynomial of degree $2s{+}r$. Moreover, because of the zeros of w_n, the polynomial $f_n = w_n q_r(n)$ can be extended to be over the range $-M-s \le n \le M+s$. Using the same reasoning as in Eq. (23.12.19), it follows that (23.12.34) can be written as

$$(-1)^s\,\nabla^{2s} f^{\rm ext}_{n+s} = \lambda_r q_r(n)\,, \quad -M \le n \le M$$

Since this is valid as an identity in n, it is enough to match the highest powers of n from both sides, that is, n^r. Thus, on the two sides we have

$$f^{\rm ext}_{n+s} = w_{n+s}q_r(n+s) = \underbrace{(-1)^s[(n+s)^{2s}+\cdots]}_{w_{n+s}}\underbrace{[a_{rr}(n+s)^r+\cdots]}_{q_r(n+s)}, \quad \text{or,}$$

$$(-1)^s f^{\rm ext}_{n+s} = a_{rr}n^{2s+r} + \cdots, \quad \text{and also,} \quad q_r(n) = a_{rr}n^r + \cdots$$

where a_{rr} is the highest coefficient of $q_r(n)$ and the dots indicate lower powers of n. Dropping the a_{rr} constant, the eigenvector condition then becomes:

$$\nabla^{2s}[n^{2s+r} + \cdots] = \lambda_r[n^r + \cdots]$$

Each operation of ∇ on n^i lowers the power by one, that is, $\nabla(n^i) = i\,n^{i-1} + \cdots$, $\nabla^2(n^i) = i(i-1)n^{i-2} + \cdots$, $\nabla^3(n^i) = i(i-1)(i-2)n^{i-3} + \cdots$, etc. Thus, we have:

$$\nabla^{2s}[n^{2s+r} + \cdots] = (2s+r)(2s+r-1)(2s+r-2)\cdots(r+1)\,n^r + \cdots$$

which yields Eq. (23.12.35).

23.13 Hahn Orthogonal Polynomials

Starting with Chebyshev [688], the discrete Chebyshev/Gram polynomials have been used repeatedly in the least-squares polynomial fitting problem, LPSM filter design, and other applications [688–735]. Bromba and Ziegler [707] were the first to establish a similar connection between the Hahn orthogonal polynomials and the minimum-R_s problem. For a review of the Hahn polynomials, see Karlin and McGregor [697].

The Hahn polynomials $Q_r(x)$ of a discrete variable $x = 0, 1, 2, \dots, N-1$ and orders $r \le N-1$ satisfy a weighted orthogonality property of the form:

$$\sum_{x=0}^{N-1} w(x)Q_r(x)Q_m(x) = D_r\delta_{rm}\,, \quad r, m = 0, 1, \dots, N-1$$

where the weighting function $w(x)$ depends on two parameters α, β and is defined up to a normalization constant as follows:

$$w(x)=\frac{(\alpha+x)!}{x!}\cdot\frac{(\beta+N-1-x)!}{(N-1-x)!}, \quad x=0,1,\dots,N-1 \tag{23.13.1}$$

The length N can be even or odd, but here we will consider only the odd case and set as usual $N=2M+1$. The interval $[0,N-1]$ can be mapped onto the symmetric interval $[-M,M]$ by making the change of variables $x=n+M$, with $-M\le n\le M$. Then, the weighting function becomes,

$$w(n)=\frac{(\alpha+M+n)!}{(M+n)!}\cdot\frac{(\beta+M-n)!}{(M-n)!}, \quad -M\le n\le M \tag{23.13.2}$$

Defining $q_r(n)=Q_r(x)\big|_{x=n+M}$, the orthogonality property now reads:

$$\boxed{\sum_{n=-M}^{M} w(n)q_r(n)q_m(n)=D_r\delta_{rm}}, \quad r,m=0,1,\dots,N-1 \tag{23.13.3}$$

The minimum-R_s problem corresponds to the particular choice $\alpha=\beta=s$. In this case, the weighting function $w(n)$ reduces to the Henderson weights of Eq. (23.12.21):

$$w(n)=\frac{(s+M+n)!}{(M+n)!}\cdot\frac{(s+M-n)!}{(M-n)!}=\prod_{i=1}^{s}(M+n+i)\cdot\prod_{i=1}^{s}(M-n+i), \quad \text{or,}$$

$$w(n)=\prod_{i=1}^{s}\big[(M+i)^2-n^2\big], \quad -M\le n\le M \tag{23.13.4}$$

For $s=0$, the weights reduce to $w(n)=1$ corresponding to the discrete Chebyshev/Gram polynomials. Because the weights are unity, the Chebyshev/Gram polynomials can be regarded as discrete-time versions of the Legendre polynomials. In fact, they tend to the latter in the limit $N\to\infty$ [717]. Similarly, the Hahn polynomials may be regarded as discrete versions of the Jacobi polynomials. At the opposite limit, $s\to\infty$, the Hahn polynomials tend to the *Krawtchouk* polynomials [717], which are discrete versions of the Hermite polynomials [714]. We review Krawtchouk polynomials and their application to the design of maximally flat filters in Sec. 23.14.

In general, the Hahn polynomials are given in terms of the hypergeometric function ${}_3F_2(a_1,a_2,a_3;b_1,b_2;z)$. For $\alpha=\beta=s$, they take the following explicit form:

$$\boxed{q_r(n)=Q_r(x)=\sum_{k=0}^{r}a_{rk}x^{[k]}\Big|_{x=n+M}=\sum_{k=0}^{r}a_{rk}\,(n+M)^{[k]}}, \quad -M\le n\le M \tag{23.13.5}$$

where $x^{[k]}$ denotes the falling-factorial power,

$$x^{[k]}=x(x-1)\cdots(x-k+1)=\frac{x!}{(x-k)!}=\frac{\Gamma(x+1)}{\Gamma(x-k+1)} \tag{23.13.6}$$

The polynomial coefficients are:

$$\boxed{a_{rk}=(-1)^k\prod_{m=1}^{k}\left[\frac{(r-m+1)\,(2s+r+m)}{(N-m)\,(s+m)m}\right]}, \quad k=0,1,\dots,r \tag{23.13.7}$$

where $a_{r0}=1$. Expanding the product we have:

$$a_{rk}=\frac{(-1)^k r(r-1)\cdots(r-k+1)\cdot(2s+r+1)\,(2s+r+2)\cdots(2s+r+k)}{(N-1)\,(N-2)\cdots(N-k)\cdot(s+1)\,(s+2)\cdots(s+k)\cdot k!} \tag{23.13.8}$$

The polynomials satisfy the symmetry property ,

$$q_r(-n)=(-1)^r q_r(n) \tag{23.13.9}$$

The orthogonality property (23.13.3) is satisfied with the following values of D_r:

$$\boxed{D_r=\frac{(s!)^2}{(2M)!}\cdot\frac{r!\,(2M-r)!}{(2M)!}\cdot\frac{(2s+r+1)\,(2s+r+2)\cdots(2s+r+N)}{2s+2r+1}} \tag{23.13.10}$$

For minimum-R_s filter design with polynomial order $d\le N-1$, only polynomials up to order d are needed, that is, $q_r(n)$, $r=0,1,\dots,d$. Arranging these as the columns of the $N\times(d+1)$ matrix $Q=[\mathbf{q}_0,\mathbf{q}_1,\dots,\mathbf{q}_d]$, the orthogonality property can be expressed as $Q^TWQ=D$, where $D=\text{diag}\big([D_0,D_1,\dots,D_d]\big)$.

The relationship to the monomial basis $S=[\mathbf{s}_0,\mathbf{s}_1,\dots,\mathbf{s}_d]$ is through an upper-triangular invertible matrix R, that is, $S=QR$. This can be justified by noting that the power series of $q_r(n)$ in n is a linear combination of the monomials $s_i(n)=n^i$ for $i=0,1,\dots,r$. In fact, R can be easily constructed from the Hahn coefficients a_{rk} and the Stirling numbers.

Thus, the construction of the minimum-R_s filters outlined in Eq. (23.11.23) is explicitly realized by the Hahn polynomial basis matrix Q:

$$B=WQD^{-1}Q^T \tag{23.13.11}$$

or, component-wise,

$$b_m(n)=B_{nm}=w(n)\sum_{r=0}^{d}\frac{q_r(n)q_r(m)}{D_r}, \quad -M\le n,m\le M \tag{23.13.12}$$

A more direct derivation of (23.13.11) is to perform the local polynomial fit in the Q-basis. The desired degree-d polynomial can be expanded in the linear combination:

$$\hat{y}_m=\sum_{i=0}^{d}c_i m^i=\sum_{r=0}^{d}a_r q_r(m) \quad\Rightarrow\quad \hat{\mathbf{y}}=S\mathbf{c}=Q\mathbf{a}$$

Then, minimize the weighted performance index with respect to $\mathbf{a}$:

$$\mathcal{J}=(\mathbf{y}-Q\mathbf{a})^TW(\mathbf{y}-Q\mathbf{a})=\min$$

Using the condition $Q^TWQ = D$, the solution leads to the same B:

$$\mathbf{a} = D^{-1}Q^TW\mathbf{y} \quad\Rightarrow\quad \hat{\mathbf{y}} = Q\mathbf{a} = QD^{-1}Q^TW\mathbf{y} = B^T\mathbf{y} \tag{23.13.13}$$

The computation of the basis Q is facilitated by the following MATLAB functions. We note first that the falling factorial powers are related to ordinary powers by the *Stirling numbers* of the first and second kind:

$$x^{[k]} = \sum_{i=0}^{k} S_1(k,i)x^i \quad\Leftrightarrow\quad x^k = \sum_{i=0}^{k} S_2(k,i)x^{[i]} \tag{23.13.14}$$

These numbers may be arranged into lower-triangular matrices S_1 and S_2, which are inverses of each other. For example, we have for $k = 0,1,2,3$:

$$\begin{bmatrix} x^{[0]} \\ x^{[1]} \\ x^{[2]} \\ x^{[3]} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 2 & -3 & 1 \end{bmatrix} \begin{bmatrix} x^0 \\ x^1 \\ x^2 \\ x^3 \end{bmatrix} \quad\Leftrightarrow\quad \begin{bmatrix} x^0 \\ x^1 \\ x^2 \\ x^3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 3 & 1 \end{bmatrix} \begin{bmatrix} x^{[0]} \\ x^{[1]} \\ x^{[2]} \\ x^{[3]} \end{bmatrix}$$

$$S_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 2 & -3 & 1 \end{bmatrix}, \qquad S_2 = S_1^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 3 & 1 \end{bmatrix}$$

The MATLAB function `stirling` generates these matrices up to a desired order:

```
S = stirling(d,kind);        % Stirling numbers up to order d of kind = 1,2
```

A polynomial can be expressed in falling factorial powers or in ordinary powers. The corresponding coefficient vectors are related by the Stirling numbers:

$$P(x) = \sum_{k=0}^{d} a_k x^{[k]} = \sum_{i=0}^{d} c_i x^i \quad\Rightarrow\quad \mathbf{c} = S_1^T\mathbf{a}\,, \quad \mathbf{a} = S_2^T\mathbf{c}$$

The function `polval` allows the evaluation of a polynomial in falling (or rising) factorial powers or in ordinary powers at any vector of x values:

```
P = polval(a,z,type);        % polynomial evaluation in factorial powers
```

The functions `hahncoeff`, `hahnpol`, and `hahnbasis` allow the calculation of the Hahn coefficients (23.13.7), the evaluation of the polynomial $Q_r(x)$ at any vector of x's, and the construction of the Hahn basis $Q = [\mathbf{q}_0, \mathbf{q}_1, \dots, \mathbf{q}_d]$:

```
[a,c] = hahncoeff(N,r,s);     % Hahn polynomial coefficients a_rk
Q = hahnpol(N,r,s,x);         % evaluate Hahn polynomial Q_r(x)
[Q,D,L] = hahnbasis(N,d,s);   % Hahn basis Q = [q_0,q_1,...,q_d]
```

Like all orthogonal polynomials, the Hahn polynomials satisfy a three-term recurrence relation of the form:

$$nq_r(n) = \alpha_r q_{r+1}(n) + \beta_r q_r(n) + \gamma_r q_{r-1}(n) \tag{23.13.15}$$

that starts with $r = 0$ and $q_{-1}(n) = 0$ and ends at $r = N-2$. The recurrence relation is a direct consequence of the property (which follows from (23.13.3)) that the order-r polynomial $q_r(n)$ is orthogonal to *every* polynomial of degree strictly less than r. Let us denote the weighted inner product by

$$(a,b) = \sum_{n=-M}^{M} w(n)a(n)b(n) \tag{23.13.16}$$

Then, since the polynomial $nq_r(n)$ has degree $r+1$, it can be expanded as a linear combination of the polynomials $q_i(n)$ up to degree $r+1$:

$$nq_r(n) = \sum_{i=0}^{r+1} c_i q_i(n)$$

The coefficients are determined using the orthogonality property by

$$(nq_r, q_i) = \sum_{j=0}^{r+1} c_j(q_j, q_i) = \sum_{j=0}^{r+1} c_j D_i \delta_{ij} = D_i c_i \quad\Rightarrow\quad c_i = \frac{(nq_r, q_i)}{D_i} \tag{23.13.17}$$

This implies that $c_i = 0$ for $i \le r-2$, therefore, only the terms $i = r+1, r, r-1$ will survive, which is the recurrence relation. Indeed, we note that $(nq_r, q_i) = (q_r, nq_i)$ and that $nq_i(n)$ has degree $(i+1)$. Therefore, as long as $i+1 < r$, or, $i \le r-2$, this inner product will be zero. It follows from (23.13.17) that:

$$\alpha_r = \frac{(nq_r, q_{r+1})}{D_{r+1}}\,, \quad \beta_r = \frac{(nq_r, q_r)}{D_r}\,, \quad \gamma_r = \frac{(nq_r, q_{r-1})}{D_{r-1}} \tag{23.13.18}$$

Because the weights $w(n)$ are symmetric, $w(n) = w(-n)$, and the polynomials satisfy, $q_r(-n) = (-1)^r q_r(n)$, it follows immediately that $\beta_r = 0$. The coefficient γ_r can be related to α_{r-1} by noting that

$$\alpha_{r-1} = \frac{(nq_{r-1}, q_r)}{D_r} = \frac{(nq_r, q_{r-1})}{D_r} \quad\Rightarrow\quad (nq_r, q_{r-1}) = D_r\alpha_{r-1}\,, \quad \text{and hence,}$$

$$\gamma_r = \frac{(nq_r, q_{r-1})}{D_{r-1}} = \frac{D_r\alpha_{r-1}}{D_{r-1}} \tag{23.13.19}$$

Moreover, α_r is related to the leading coefficients a_{rr} of the $q_r(n)$ polynomial. From the definition (23.13.5), we can write

$$q_r(n) = a_{rr}n^r + p_{r-1}(n)\,, \quad q_{r+1}(n) = a_{r+1,r+1}n^{r+1} + p_r(n)$$

where $p_{r-1}(n)$ and $p_r(n)$ are polynomials of degree $r-1$ and r, respectively. Since $D_{r+1} = (q_{r+1}, q_{r+1})$, we have,

$$\alpha_r = \frac{(nq_r, q_{r+1})}{(q_{r+1}, q_{r+1})} = \frac{(a_{rr}n^{r+1} + np_{r-1},\, q_{r+1})}{(a_{r+1,r+1}n^{r+1} + p_r,\, q_{r+1})} = \frac{a_{rr}(n^{r+1},\, q_{r+1})}{a_{r+1,r+1}(n^{r+1},\, q_{r+1})} = \frac{a_{rr}}{a_{r+1,r+1}}$$

where we used the orthogonality of $q_{r+1}(n)$ with $np_{r-1}(n)$ and $p_r(n)$, both of which have order r. Thus,

$$\alpha_r = \frac{a_{rr}}{a_{r+1,r+1}} \tag{23.13.20}$$

Using Eqs. (23.13.7) and (23.13.10), the expressions for α_r and γ_r simplify into:

$$\alpha_r = -\frac{(2M-r)(2s+r+1)}{2(2s+2r+1)}\,, \quad \gamma_r = -\frac{r(2M+2s+r+1)}{2(2s+2r+1)} \tag{23.13.21}$$

These satisfy the constraint $\alpha_r + \gamma_r = -M$, which follows from the recurrence relation and the conditions $q_r(-M) = a_{r0} = 1$ for all r. Next, we derive the Christoffel-Darboux identity which allows the simplification of the sum in (23.13.12). Setting $\beta_r = 0$, replacing $\gamma_r = \alpha_{r-1}D_r/D_{r-1}$ and dividing by D_r, the recurrence relation reads:

$$\frac{nq_r(n)}{D_r} = \frac{\alpha_r}{D_r}\,q_{r+1}(n) + \frac{\alpha_{r-1}}{D_{r-1}}\,q_{r-1}(n) \tag{23.13.22}$$

Multiplying by $q_r(m)$, interchanging the roles of n, m, and subtracting, we obtain:

$$\begin{aligned}
\frac{nq_r(n)q_r(m)}{D_r} &= \frac{\alpha_r}{D_r}\,q_{r+1}(n)q_r(m) + \frac{\alpha_{r-1}}{D_{r-1}}\,q_{r-1}(n)q_r(m)\\
\frac{mq_r(m)q_r(n)}{D_r} &= \frac{\alpha_r}{D_r}\,q_{r+1}(m)q_r(n) + \frac{\alpha_{r-1}}{D_{r-1}}\,q_{r-1}(m)q_r(n)\\
\frac{(n-m)q_r(n)q_r(m)}{D_r} &= \frac{\alpha_r}{D_r}\big[q_{r+1}(n)q_r(m) - q_r(n)q_{r+1}(m)\big] -\\
&\quad - \frac{\alpha_{r-1}}{D_{r-1}}\big[q_r(n)q_{r-1}(m) - q_{r-1}(n)q_r(m)\big]
\end{aligned}$$

Summing up over r, and using $q_{-1}(n) = 0$, the successive terms on the right-hand side cancel except for the last one, resulting in the Christoffel-Darboux identity:

$$(n-m)\sum_{r=0}^{d}\frac{q_r(n)q_r(m)}{D_r} = \frac{\alpha_d}{D_d}\big[q_{d+1}(n)q_d(m) - q_d(n)q_{d+1}(m)\big]\,, \quad \text{or,}$$

$$\boxed{\sum_{r=0}^{d}\frac{q_r(n)q_r(m)}{D_r} = \frac{\alpha_d}{D_d}\,\frac{q_{d+1}(n)q_d(m) - q_d(n)q_{d+1}(m)}{n-m}} \tag{23.13.23}$$

Using this identity into the filter equations (23.13.12), we find

$$\boxed{b_m(n) = w(n)\,\frac{\alpha_d}{D_d}\,\frac{q_{d+1}(n)q_d(m) - q_d(n)q_{d+1}(m)}{n-m}} \tag{23.13.24}$$

This is valid for $-M \le n, m \le M$ and for orders $0 \le d \le N-2$. At $n = m$, the numerator vanishes, so that the numerator and denominator have a common factor $n - m$, which cancels resulting in a polynomial of degree d in n and m. In particular, the central Henderson filters are:

$$\boxed{b_0(n) = w(n)\,\frac{\alpha_d}{D_d}\,\frac{q_{d+1}(n)q_d(0) - q_d(n)q_{d+1}(0)}{n}} \tag{23.13.25}$$

where either $q_d(0)$ or $q_{d+1}(0)$ is zero depending on whether d is odd or even. In fact for the two successive values $d = 2r$ and $d = 2r+1$, while the asymmetric filters $b_m(n)$ are different, the central filters are the same and given by:

$$b_0(n) = \frac{\alpha_{2r}}{D_{2r}}\,q_{2r}(0)\,\frac{q_{2r+1}(n)}{n} = -\frac{\alpha_{2r+1}}{D_{2r+1}}\,q_{2r+2}(0)\,\frac{q_{2r+1}(n)}{n} \tag{23.13.26}$$

the equality of the coefficients following by setting $d = 2r+1$ and $n = 0$ in Eq. (23.13.22).

Next, we derive explicit formulas for some specific cases. The first few Hahn polynomials of orders $d = 0, 1, 2, 3, 4, 5$ and arbitrary M and s are, for $-M \le n \le M$:

$$\begin{aligned}
q_0(n) &= 1\\
q_1(n) &= -\frac{n}{M}\\
q_2(n) &= \frac{(2s+3)n^2 - M(M+s+1)}{M(2M-1)(s+1)}\\
q_3(n) &= -\frac{(2s+5)n^3 - \big[3M^2 + (s+1)(3M-1)\big]n}{M(M-1)(2M-1)(s+1)}\\
q_4(n) &= \frac{(2s+5)(2s+7)n^4 - (2s+5)\big(6M^2 + 6(s+1)M - 4s-5\big)n^2}{M(M-1)(2M-1)(2M-3)(s+1)(s+2)}\\
&\quad + \frac{3M(M-1)(s+M+1)(s+M+2)}{M(M-1)(2M-1)(2M-3)(s+1)(s+2)}\\
q_5(n) &= -\frac{(2s+7)(2s+9)n^5 - 5(2s+7)\big(2M^2 + 2(s+1)M - 2s-3\big)n^3}{M(M-1)(M-2)(2M-1)(2M-3)(s+1)(s+2)}\\
&\quad - \frac{\big[15M^4 + 30(s+1)M^3 + 5(3s^3+s-7)M^2 - (s+1)(s+2)(25M-6)\big]n}{M(M-1)(M-2)(2M-1)(2M-3)(s+1)(s+2)}
\end{aligned} \tag{23.13.27}$$

They are normalized such that $q_r(-M) = 1$. Setting $s = 0$, we obtain the corresponding *discrete Chebyshev/Gram* polynomials:

$$\begin{aligned}
q_0(n) &= 1\\
q_1(n) &= -\frac{n}{M}\\
q_2(n) &= \frac{3n^2 - M(M+1)}{M(2M-1)}\\
q_3(n) &= -\frac{5n^3 - (3M^2+3M-1)n}{M(M-1)(2M-1)}\\
q_4(n) &= \frac{35n^4 - 5(6M^2+6M-5)n^2 + 3M(M^2-1)(M+2)}{2M(M-1)(2M-1)(2M-3)}\\
q_5(n) &= -\frac{63n^5 - 35(2M^2+2M-3)n^3 + (15M^4+30M^3-35M^2-50M+12)n}{2M(M-1)(M-2)(2M-1)(2M-3)}
\end{aligned} \tag{23.13.28}$$

The central Henderson filters for the cases $d = 0, 1$, $d = 2, 3$, and $d = 4, 5$ are as follows for general M and s. For $d = 0, 1$:

$$b_0(n) = \frac{(2s+1)!\,(2M)!}{(s!)^2\,(2M+2s+1)!}\,w(n) \tag{23.13.29}$$

where $w(n)$ is given by Eq. (23.13.4). For $d = 2, 3$, we have:

$$b_0(n) = \frac{(M+s+1)\,(2s+3)!\,(2M)!\big(3M^2 + (s+1)\,(3M-1) - (2s+5)\,n^2\big)}{(2M-1)\,(s!)^2\,(2M+2s+3)!}\,w(n) \tag{23.13.30}$$

This generalizes Henderson's ideal formula (23.12.29) to arbitrary s. For $s = 1, 2$, it simplifies into:

$$\begin{aligned} s = 1, \quad b_0(n) &= \frac{15\,(3M^2 + 6M - 2 - 7n^2)\,w_1(n)}{2\,(M+1)\,(2M+3)\,(2M+5)\,(4M^2-1)} \\ s = 2, \quad b_0(n) &= \frac{105\,(M^2 + 3M - 1 - 3n^2)\,w_2(n)}{2\,(M+1)\,(M+2)\,(2M+3)\,(2M+5)\,(2M+7)\,(4M^2-1)} \end{aligned}$$

where $w_1(n)$ and $w_2(n)$ correspond to (23.13.4) with $s = 1$ and $s = 2$. The case $s = 0$ is, of course, the same as Eq. (23.3.17). For the case $d = 4, 5$, we find:

$$\begin{aligned} b_0(n) = {} & \frac{(M+s+1)\,(M+s+2)\,(2s+5)!\,(2M)!}{2\,(2M-1)\,(2M-3)\,\big((s+2)!\big)^2\,(2M+2s+5)!} \cdot w(n) \cdot \\ & \cdot \Big[(2s+7)\,(2s+9)\,n^4 - 5\,(2s+7)\,\big(2M^2 + 2\,(s+1)\,M - 2s-3\big)\,n^2 + \\ & + 15M^4 + 30\,(s+1)\,M^3 + 5\,(3s^2+s-7)\,M^2 + (s+1)\,(s+2)\,(25M-6) \Big] \end{aligned} \tag{23.13.31}$$

Eqs. (23.13.29)–(23.13.31), as well as the case $d = 6, 7$, have been implemented into the MATLAB function `lprs2`, with usage:

```
b0 = lprs2(N,d,s);
```

% exact forms of the Henderson filters $b_0(n)$ for $0 \le d \le 6$

The asymmetric interpolation filters $b_t(n)$ can be obtained by replacing the discrete variable m by t in Eqs. (23.13.12) and (23.13.24):

$$b_t(n) = w(n)\sum_{r=0}^{d}\frac{q_r(n)q_r(t)}{D_r} = w(n)\,\frac{\alpha_d}{D_d}\,\frac{q_{d+1}(n)q_d(t) - q_d(n)q_{d+1}(t)}{n-t} \tag{23.13.32}$$

Some specific cases are as follows. For $d = 0$, we have:

$$b_t(n) = \frac{(2s+1)!\,(2M)!}{(s!)^2\,(2M+2s+1)!}\,w(n) \tag{23.13.33}$$

For $d = 1$,

$$b_t(n) = \frac{4\,(2s+1)!\,(2M-1)!}{(s!)^2\,(2M+2s+2)!}\,w(n)\,\big[M^2 + (s+1)M + (2s+3)\,nt\big] \tag{23.13.34}$$

For $d = 2$:

$$\begin{aligned} b_t(n) = {} & \frac{4\,(2s+1)!\,(2M-1)!}{(s!)^2\,(2M+2s+2)!}\,w(n)\Big[M\,(M+s+1)\,\big[3M^2 + 3\,(s+1)\,M - s-1\big] \\ & + (s+1)\,(2M-1)\,(2M+2s+3)\,nt - M\,(M+s+1)\,(2s+5)\,(n^2+t^2) \\ & + (s+1)\,(2M-1)\,(2M+2s+3)\,n^2t^2\Big] \end{aligned} \tag{23.13.35}$$

The corresponding predictive differentiation filters are obtained by differentiating with respect to t.

The above closed-form expressions were obtained with the following simple Maple procedures that define the Hahn coefficients a_{rk}, the Hahn polynomials $q_r(n)$ and their norms D_r, and the interpolation filters $b_t(n)$:

```
factpow := proc(x,k) product((x-m), m=0..k-1); end proc;

a := proc(M,r,s,k)
        (-1)^k * product((r-m+1)*(2*s+r+m)/(2*M+1-m)/(s+m)/m, m=1..k);
     end proc;

Q := proc(M,r,s,n) if r=0 then 1; else
        sum(a(M,r,s,k)*factpow(n+M,k), k=0..r);
     end if; end proc;

Dr := proc(M,r,s) GAMMA(s+1)^2 * GAMMA(r+1) * GAMMA(2*M+1-r)
        * product(2*s+r+i, i=1..(2*M+1)) / GAMMA(2*M+1)^2 / (2*s+2*r+1);
      end proc;

B := proc(M,d,s,n,t)
        sum(Q(M,r,s,n)/Dr(M,r,s)*Q(M,r,s,t), r=0..d);
     end proc;
```

where `factpow` defines the falling-factorial powers, and it is understood that the result from the procedure `B(M,d,s,n,t)` must be multiplied by the Henderson weights $w(n)$.

There are other useful choices for the weighting function $w(n)$, such as binomial, which are similar to gaussian weights and lead to the Krawtchouk orthogonal polynomials, or exponentially decaying $w(n) = \lambda^n$, with $n \ge 0$ and $0 < \lambda < 1$, leading to the discrete Laguerre polynomials [719,720] and exponential smoothers. However, these choices do not have an equivalent minimum-NRR characterization. Even so, the smoothing filters are efficiently computed in the orthogonal polynomial basis by:

$$B = WS(S^TWS)^{-1}S^T = WQD^{-1}Q^T\,, \quad Q^TWQ = D \tag{23.13.36}$$

23.14 Maximally-Flat Filters

Greville [668] has shown that in the limit $s \to \infty$ the minimum-R_s filters tend to maximally flat FIR filters that satisfy the usual flatness constraints at dc, that is, $B^{(i)}(\omega)\big|_{\omega=0} = \delta(i)$, for $i = 0, 1, \dots, d$, but also have monotonically decreasing magnitude responses and satisfy $(2M-d)$ additional flatness constraints at the Nyquist frequency, $\omega = \pi$. They are identical to the well-known maximally flat filters introduced by Herrmann [758].

Bromba and Ziegler [707,762] have shown that their impulse responses are given in terms of the Krawtchouk orthogonal polynomials [693,714,717]. Meer and Weiss [724] have derived the corresponding differentiation filters based on the Krawtchouk polynomials for application to images. Here, we look briefly at these properties.

The Krawtchouk polynomials are characterized by a parameter p such that $0 < p < 1$ and are defined over the symmetric interval $-M \le n \le M$ by [717]

$$\bar{q}_r(n) = \sum_{k=0}^{r} \frac{(-1)^k r(r-1)\cdots(r-k+1)p^{-k}}{(N-1)(N-2)\cdots(N-k)\cdot k!}(n+M)^{[k]} \tag{23.14.1}$$

where $N = 2M+1$ and $r = 0, 1, \dots, N-1$. They satisfy the orthogonality property,

$$\sum_{n=-M}^{M} \bar{w}(n)\bar{q}_r(n)\bar{q}_m(n) = \bar{D}_r\delta_{rm} \tag{23.14.2}$$

with the following binomial weighting function and norms, where $q = 1-p$:

$$\begin{aligned} \bar{w}(n) &= \binom{2M}{M+n} p^{M+n}q^{M-n} = \frac{(2M)!}{2^{2M}(M+n)!\,(M-n)!}\,p^{M+n}q^{M-n} \\ \bar{D}_r &= \frac{r!\,(2M-r)!}{(2M)!}\,\frac{q^r}{p^r} \end{aligned} \tag{23.14.3}$$

In the limit $s \to \infty$, the Hahn polynomials tend to the special Krawtchouk polynomials with the parameter $p = q = 1/2$. To see this, we note that the Hahn polynomials are normalized such that $q_r(-M) = 1$, and we expect that they would have a straightforward limit as $s \to \infty$. Indeed, it is evident that the limit of the Hahn coefficients (23.13.8) is

$$\bar{a}_{rk} = \lim_{s\to\infty} a_{rk} = \frac{(-1)^k r(r-1)\cdots(r-k+1)\cdot 2^k}{(N-1)(N-2)\cdots(N-k)\cdot k!} \tag{23.14.4}$$

and therefore, the Hahn polynomials will tend to

$$\bar{q}_r(n) = \sum_{k=0}^{r} \frac{(-1)^k r(r-1)\cdots(r-k+1)\cdot 2^k}{(N-1)(N-2)\cdots(N-k)\cdot k!}(n+M)^{[k]} \tag{23.14.5}$$

which are recognized as a special case of (23.14.1) with $p = 1/2$. The Henderson weights (23.13.4) and norms (23.13.10) diverge as $s \to \infty$, but we may normalize them by a common factor, such as $s^{2M}(s!)^2$, so that they will converge. The limits of the rescaled weights and norms are:

$$\begin{aligned} \bar{w}(n) &= \lim_{s\to\infty}\left[\frac{(2M)!\,w(n)}{2^{2M}s^{2M}(s!)^2}\right] = \lim_{s\to\infty}\left[\frac{(2M)!\,(s+M+n)!\,(s+M-n)!}{2^{2M}s^{2M}(s!)^2\;(M+n)!\,(M-n)!}\right] \\ \bar{D}_r &= \lim_{s\to\infty}\left[\frac{(2M)!\,D_r}{2^{2M}s^{2M}(s!)^2}\right] \\ &= \lim_{s\to\infty}\left[\frac{r!\,(2M-r)!}{(2M)!}\cdot\frac{(2s+r+1)(2s+r+2)\cdots(2s+r+N)}{2^{2M}s^{2M}(2s+2r+1)}\right] \end{aligned}$$

They are easily seen to lead to Eqs. (23.14.3) with $p = 1/2$, that is,

$$\begin{aligned} \bar{w}(n) &= \frac{1}{2^{2M}}\binom{2M}{M+n} = \frac{(2M)!}{2^{2M}(M+n)!\,(M-n)!} \\ \bar{D}_r &= \frac{r!\,(2M-r)!}{(2M)!} \end{aligned} \tag{23.14.6}$$

The first few of the Krawtchouk polynomials are:

$$\begin{aligned} \bar{q}_0(n) &= 1 \\ \bar{q}_1(n) &= -\frac{n}{M} \\ \bar{q}_2(n) &= \frac{2n^2 - M}{M(2M-1)} \\ \bar{q}_3(n) &= -\frac{2n^3 - (3M-1)n}{M(M-1)(2M-1)} \\ \bar{q}_4(n) &= \frac{4n^4 - (12M-8)n^2 + 3M(M-1)}{M(M-1)(2M-1)(2M-3)} \\ \bar{q}_5(n) &= -\frac{4n^5 - 20(M-1)n^3 + (15M^2-25M+6)n}{M(M-1)(M-2)(2M-1)(2M-3)} \end{aligned} \tag{23.14.7}$$

These polynomials satisfy the three-term recurrence relation:

$$n\bar{q}_r(n) = \bar{\alpha}_r\bar{q}_{r+1}(n) + \bar{\gamma}_r\bar{q}_{r-1}(n)\,, \quad \bar{\alpha}_r = -\frac{2M-r}{2}\,, \quad \bar{\gamma}_r = -\frac{r}{2} \tag{23.14.8}$$

with the coefficients $\bar{\alpha}_r, \bar{\gamma}_r$ obtained from Eq. (23.13.21) in the limit $s \to \infty$. The three-term relations lead to the usual Christoffel-Darboux identity from which we may obtain the asymmetric predictive filters:

$$\bar{b}_t(n) = \bar{w}(n)\sum_{r=0}^{d}\frac{\bar{q}_r(n)\bar{q}_r(t)}{\bar{D}_r} = \bar{w}(n)\,\frac{\bar{\alpha}_d}{\bar{D}_d}\,\frac{\bar{q}_{d+1}(n)\bar{q}_d(t) - \bar{q}_d(n)\bar{q}_{d+1}(t)}{n-t} \tag{23.14.9}$$

Differentiation with respect to t gives the corresponding predictive differentiation filters. Some examples are as follows. For $d = 0$ and $d = 1$, we have, respectively

$$\bar{b}_t(n) = \bar{w}(n)\,, \quad \bar{b}_t(n) = \bar{w}(n)\,\frac{2nt + M}{M} \tag{23.14.10}$$

For $d = 2$, the smoothing and first-order differentiation filters are:

$$\begin{aligned} \bar{b}_t(n) &= \bar{w}(n)\,\frac{4n^2t^2 - 2M(n^2+t^2) + 2(2M-1)nt + M(3M-1)}{M(2M-1)} \\ \dot{\bar{b}}_t(n) &= \bar{w}(n)\,\frac{2(2M-1)n - 4Mt + 8n^2t}{M(2M-1)} \end{aligned} \tag{23.14.11}$$

and setting $t = 0$, the central filters simplify into:

$$\bar{b}_0(n) = \bar{w}(n)\,\frac{3M-1-2n^2}{2M-1}\,, \qquad \dot{\bar{b}}_0(n) = \bar{w}(n)\,\frac{2n}{M} \tag{23.14.12}$$

For $d = 3$, we have:

$$\begin{aligned}\bar{b}_t(n) = \frac{\bar{w}(n)}{3M(M-1)(2M-1)}\Big[&8n^3t^3 - 4(3M-1)(n^3t + nt^3) + 12(M-1)n^2t^2 \\ &- 6M(M-1)(n^2+t^2) + (30M^2-30M+8)nt - 3M(M-1)(3M-1)\Big]\end{aligned} \tag{23.14.13}$$

As expected, setting $t = 0$ produces the same result as the $d = 2$ case. Numerically, the smoothing and differentiation filters can be calculated by passing the Krawtchouk weights $\bar{w}(n)$ into the functions `lpsm`, `lpdiff`, and `lpinterp`:

```
W  = diag(hend(N,inf));        % Krawtchouk weights
B  = lpsm(N,d,W);              % smoothing filters
Bi = lpdiff(N,d,i,W);          % i-th derivative filters
b  = lpinterp(N,d,t,i,W);      % interpolation filters b_t
```

The function `hend`(N,s), with $s = \infty$, calculates the Krawtchouk weights of Eq. (23.14.6). In turn, the filter matrices B or $B^{(i)}$ may be passed into the filtering function `lpfilt`. Alternatively, one can call `lprs` with $s = \infty$:

```
B = lprs(N,d,inf);     % LPRS with Krawtchouk weights, maximally-flat filters
```

It is well-known [668,758–771] that the maximally-flat FIR filters of length $N = 2M+1$ and polynomial order $d = 2r+1$ have frequency responses given by the following equivalent expressions:

$$\begin{aligned}B_0(\omega) &= \sum_{i=0}^{r}\binom{M}{i}x^i(1-x)^{M-i} = 1 - \sum_{i=r+1}^{M}\binom{M}{i}x^i(1-x)^{M-i} \\ &= (1-x)^{M-r}\sum_{i=0}^{r}\binom{M-r+i-1}{i}x^i\,, \qquad \text{where } x = \sin^2\Big(\frac{\omega}{2}\Big)\end{aligned} \tag{23.14.14}$$

Near $\omega \simeq 0$ and near $\omega \simeq \pi$, the second and third expressions have the following expansions that exhibit the desired flatness constraints [707]:

$$\begin{aligned}\omega \simeq 0 \quad &\Rightarrow \quad B_0(\omega) \simeq 1 - (\text{const.})\,\omega^{2r+2} = 1 - (\text{const.})\,\omega^{d+1} \\ \omega \simeq \pi \quad &\Rightarrow \quad B_0(\omega) \simeq (\text{const.})\,(\omega-\pi)^{2M-2r} = (\text{const.})\,(\omega-\pi)^{2M-d+1}\end{aligned} \tag{23.14.15}$$

The first implies the flatness constraints at dc, $B_0^{(i)}(0) = \delta(i)$, for $i = 0, 1, \dots, d$, and the second, the flatness constraints at Nyquist, $B_0^{(i)}(\pi) = 0$, for $i = 0, 1, \dots, 2M-d$.

Example 23.14.1: For $d = 2$ or $r = 1$, the z-transform of $b_0(n)$ in Eq. (23.14.12) can be calculated explicitly resulting in:

$$B_0(z) = \bigg[\frac{(1+z^{-1})(1+z)}{4}\bigg]^{M-1}\frac{1}{4}\big[2(M+1) - (M-1)(z+z^{-1})\big]$$

With $z = e^{j\omega}$ we may write

$$\begin{aligned}x = \sin^2\Big(\frac{\omega}{2}\Big) &= \frac{(1-z^{-1})(1-z)}{4} = \frac{2-z-z^{-1}}{4} \quad \Rightarrow \quad \frac{z+z^{-1}}{4} = \frac{1}{2} - x \\ 1 - x = \cos^2\Big(\frac{\omega}{2}\Big) &= \frac{(1+z^{-1})(1+z)}{4}\end{aligned}$$

Thus, we may express $B_0(z)$ in terms of the variable x:

$$B_0(z) = (1-x)^{M-1}\big[1 + (M-1)x\big]$$

which corresponds to Eq. (23.14.14) for $r = 1$. □

Example 23.14.2: Fig. 23.14.1 shows the frequency responses $B_0(\omega)$ for the values $N = 13$, $r = 2$, ($d = 4, 5$), and the smoothness parameter values: $s = 3$, $s = 6$, $s = 9$, and $s = \infty$.

Because $b_0(n)$ is symmetric about $n = 0$, the quantities $B_0(\omega)$ are real-valued. In the limit $s \to \infty$, the response becomes positive and monotonically decreasing. The following MATLAB code illustrates the generation of the bottom two graphs and verifies Eq. (23.14.14):

```
N=13; r=2; d = 2*r+1; M = floor(N/2);

B = lprs(N,d,9);   b9   = B(:,M+1);              % LPRS filter with s = 9
B = lprs(N,d,inf); binf = B(:,M+1);              % LPRS with Krawtchouk weights

f = linspace(0,1,1001); w = pi*f; x = sin(w/2).^2;
B9   = real(exp(j*w*M) .* freqz(b9,1,w));          % frequency responses
Binf = real(exp(j*w*M) .* freqz(binf,1,w));

Bth = 0;
for i=0:r,
   Bth = Bth + nchoosek(M,i) * x.^i .* (1-x).^(M-i);     % Eq. (23.14.14)
end

norm(Bth-Binf)                               % compare Eq. (23.14.14) with output of LPSM

figure; plot(f,B9); figure; plot(f,Binf);
```

The calls to `lprs` and `lpsm` return the full smoothing matrices B from which the central column $\mathbf{b}_0$ is extracted.

The frequency response function `freqz` expects its filter argument to be causal. The factor $e^{j\omega M}$ compensates for that, corresponding to a time-advance by M units. □

Finally, we note that the Krawtchouk binomial weighting function $\bar{w}(n)$ tends to a gaussian for large M, which is a consequence of the De Moivre-Laplace theorem,

$$\bar{w}(n) = \frac{(2M)!}{2^{2M}(M+n)!\,(M-n)!} \simeq \frac{1}{\sqrt{\pi M}}\,e^{-n^2/M}\,, \quad -M \le n \le M \tag{23.14.16}$$

In fact, the two sides of (23.14.16) are virtually indistinguishable for $M \ge 10$.

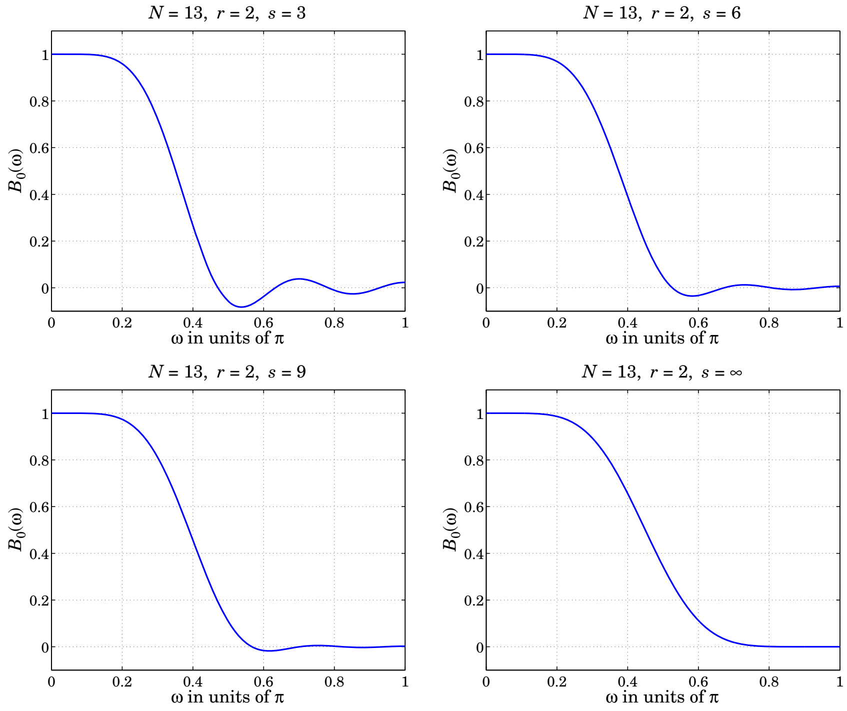


Fig. 23.14.1 Frequency responses of minimum-R_s and maximally-flat filters.

23.15 Missing Data and Outliers

The presence of outliers in the observed signal can cause large distortions in the smoothed signal. The left graph of Fig. 23.15.1 shows what can happen. The two vertical lines indicate the region in which there are four strong outliers, which cause the smoothed curve to deviate drastically from the desired signal.

One possible solution [637,749] is to ignore the outliers and estimate the smoothed values from the surrounding available samples using a filter window that spans the outlier region. The same procedure can be used if some data samples are missing. Once the outliers or missing values have been interpolated, one can apply the weighted LPSM filters as usual. The right graph in Fig. 23.15.1 shows the four adjusted interpolated samples. The resulting smoothed signal now estimates the desired signal more accurately.

This solution can be implemented by replacing the outliers or the missing data by zeros (or, any other values), and assign zero weights to them in the least-squares polynomial fitting problem.

Given a long observed signal y_n, $n = 0, 1, \dots, L-1$, let us assume that in the vicinity of $n = n_0$ there is an outlier or missing sample at the time instant n_0+m, where m lies in the interval $-M \le m \le M$, as shown in Fig. 23.15.2. Several outliers or missing data may be present, not necessarily adjacent to each other, each being characterized by a similar index m.

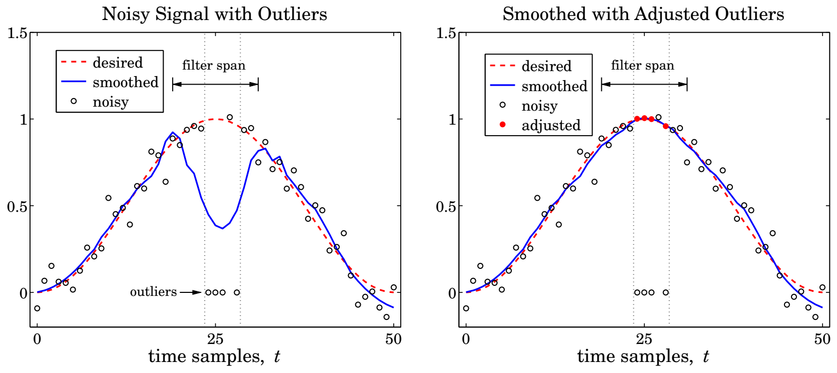


Fig. 23.15.1 Smoothing with missing data or outliers.

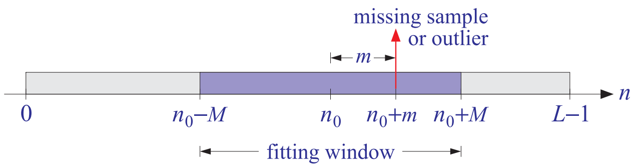


Fig. 23.15.2 Missing sample or outlier and the data window used for estimating it.

The outlier samples y_{n_0+m} can be replaced by zeros and their estimated values, $\hat{y}_{n_0+m}$, can be calculated from the surrounding samples using a filter of length $N = 2M+1$. The corresponding least-squares polynomial-fitting problem is defined by

$$\mathcal{J} = \sum_{m=-M}^{M} p_m w_m \left(y_{n_0+m} - \sum_{i=0}^{d} c_i m^i \right)^2 = \min \qquad (23.15.1)$$

where w_m are the usual Henderson weights and the p_m are zero at the indices for the missing data, and unity otherwise. Let $\mathbf{y} = [y_{n_0-M}, \dots, y_{n_0}, \dots, y_{n_0+M}]^T$, and denote by $W, \mathcal{P}$ the corresponding diagonal matrices of the weights w_m, p_m. Then, (23.15.1) reads:

$$\mathcal{J} = (\mathbf{y} - S\mathbf{c})^T \mathcal{P} W (\mathbf{y} - S\mathbf{c}) = \min, \qquad (23.15.2)$$

leading to the orthogonality conditions and the solution for $\mathbf{c}$:

$$S^T W \mathcal{P} (\mathbf{y} - S\mathbf{c}) = 0 \quad \Rightarrow \quad \mathbf{c} = (S^T \mathcal{P} W S)^{-1} S^T W \mathcal{P} \mathbf{y} \qquad (23.15.3)$$

where we assumed that $S^T\mathcal{P}WS$ is invertible.[†] The estimated samples will be:

$$\hat{\mathbf{y}} = S\mathbf{c} = S(S^T\mathcal{P}WS)^{-1}S^TW\mathcal{P}\mathbf{y} = B^T\mathbf{y} \tag{23.15.4}$$

with the filter matrix,

$$B = \mathcal{P}WS(S^T\mathcal{P}WS)^{-1}S^T \tag{23.15.5}$$

We note that $\mathcal{P}$ is a projection matrix ($\mathcal{P}^T = \mathcal{P}$ and $\mathcal{P}^2 = \mathcal{P}$) and commutes with W, $\mathcal{P}W = W\mathcal{P}$, because both are diagonal. Defining $\mathcal{Q} = I - \mathcal{P}$ to be the complementary projection matrix, the estimated signal can be decomposed in two parts: $\hat{\mathbf{y}} = \mathcal{P}\hat{\mathbf{y}} + \mathcal{Q}\hat{\mathbf{y}}$, with $\mathcal{Q}\hat{\mathbf{y}}$ being the part that contains the estimated missing values or adjusted outliers.

The quantity $\mathcal{P}\mathbf{y}$ represents the samples that are being used to make the estimates, whereas $\mathcal{Q}\mathbf{y}$ corresponds to the missing samples and can be set to zero or to an arbitrary vector $\mathcal{Q}\mathbf{y}_{\text{arb}}$, in other words, we may replace $\mathbf{y}$ by $\mathcal{P}\mathbf{y} + \mathcal{Q}\mathbf{y}_{\text{arb}}$ without affecting the solution of Eq. (23.15.4). This so because $\mathcal{P}(\mathcal{P}\mathbf{y} + \mathcal{Q}\mathbf{y}_{\text{arb}}) = \mathcal{P}\mathbf{y}$.

Once the estimated missing values have been obtained, we may replace $Q\mathbf{y}_{\text{arb}}$ by $\mathcal{Q}\hat{\mathbf{y}}$ and recompute the ordinary W-weighted least-squares estimate from the adjusted vector $\mathcal{P}\mathbf{y} + \mathcal{Q}\hat{\mathbf{y}}$. This produces the same $\hat{\mathbf{y}}$ as in (23.15.4). Indeed, one can show that,

$$\hat{\mathbf{y}} = S(S^T\mathcal{P}WS)^{-1}S^TW\mathcal{P}\mathbf{y} = S(S^TWS)^{-1}S^TW(\mathcal{P}\mathbf{y} + \mathcal{Q}\hat{\mathbf{y}}) \tag{23.15.6}$$

To see this, start with the orthogonality equation (23.15.3), and replace $\mathcal{P}\hat{\mathbf{y}} = \hat{\mathbf{y}} - \mathcal{Q}\hat{\mathbf{y}}$:

$$S^TW\mathcal{P}(\mathbf{y} - \hat{\mathbf{y}}) = 0 \quad \Rightarrow \quad S^TW\mathcal{P}\mathbf{y} = S^TW\mathcal{P}\hat{\mathbf{y}} = S^TW(\hat{\mathbf{y}} - \mathcal{Q}\hat{\mathbf{y}})\,, \quad \text{or,}$$

$$S^TW(\mathcal{P}\mathbf{y} + \mathcal{Q}\hat{\mathbf{y}}) = S^TW\hat{\mathbf{y}} = S^TWS(S^T\mathcal{P}WS)^{-1}W\mathcal{P}\mathbf{y}$$

from which Eq. (23.15.6) follows by multiplying both sides by $S(S^TWS)^{-1}$. The MATLAB function `lpmissing` implements the calculation of B in (23.15.5):

```
B = lpmissing(N,d,m,s);        % filter matrix for missing data
```

The following MATLAB code illustrates the generation of Fig. 23.15.1:

```
t = (0:50)'; x0 = (1-cos(2*pi*t/50))/2;          % desired signal

seed=2005; randn('state',seed);
y = x0 + 0.1 * randn(51,1);                      % noisy signal

n0 = 25; m = [-1 0 1 3];                         % four outlier indices relative to n0
y(n0+m+1) = 0;                                   % four outlier or missing values

N= 13; d = 2; s = 0; M=(N-1)/2;                  % filter specs
x = lpfilt(lprs(N,d,s),y);                       % distorted smoothed signal

B = lpmissing(N,d,m,s);                          % missing-data filter B

yhat = B'*y(n0-M+1:n0+M+1);                      % apply B to the block n0-M ≤ n ≤ n0+M
ynew = y; ynew(n0+m+1) = yhat(M+1+m);            % new signal with interpolated outlier values

xnew = lpfilt(lprs(N,d,s),ynew);                 % recompute smoothed signal

figure; plot(t,x0,'--', t,y,'o', t,x,'-');               % left graph
figure; plot(t,x0,'--', t,y,'o', t,xnew,'-');            % right graph
hold on; plot(n0+m,yhat(M+1+m),'.');
```

[†] This requires that the number of outliers within the data window be at most $N - d - 1$.

The above method of introducing zero weights at the outlier locations can be automated and applied to the entire signal. Taking a cue from Cleveland's LOESS method [776] discussed in the next section, we may apply the following procedure.

Given a length-L signal y_n, $n = 0, 1, \dots, L-1$, with $L \geq N$, an LPSM or LPRS filter with design parameters N, d, s can be applied to y_n to get a preliminary estimate of the smoothed signal $\hat{x}_n$, and compute the error residuals $e_n = y_n - \hat{x}_n$, that is,

$$\begin{aligned} B &= \text{lprs}(N, d, s) \\ \hat{\mathbf{x}} &= \text{lpfilt}(B, \mathbf{y}) \\ \mathbf{e} &= \mathbf{y} - \hat{\mathbf{x}} \end{aligned} \tag{23.15.7}$$

From the error residual $\mathbf{e}$, one may compute a set of "robustness" weights r_n by using the median of $|e_n|$ as a normalization factor in the bisquare function:

$$\mu = \text{median}(|e_n|)\,, \quad r_n = W\left(\frac{e_n}{K\mu}\right)\,, \quad n = 0, 1, \dots, L-1 \tag{23.15.8}$$

where K is a constant such as $K = 2$–6, and $W(u)$ is the bisquare function,

$$W(u) = \begin{cases} (1-u^2)^2, & \text{if } |u| \leq 1 \\ 0, & \text{otherwise} \end{cases} \tag{23.15.9}$$

If a residual e_n deviates too far from the median, that is, $|e_n| > K\mu$, then the robustness weight r_n is set to zero. A new estimate $\hat{x}_n$ can be calculated at each time n by defining the diagonal matrix P in terms of the robustness weights in the neighborhood of n, and then calculating the estimate using the c_0 component of the vector $\mathbf{c}$ in Eq. (23.15.3), that is,

$$\begin{aligned} P_n &= \text{diag}\big([r_{n-M}, \dots, r_n, \dots, r_{n+M}]\big) \\ \hat{x}_n &= c_0 = \mathbf{u}_0^T(S^TP_nWS)^{-1}S^TWP_n\mathbf{y}(n) \end{aligned} \tag{23.15.10}$$

where $\mathbf{u}_0 = [1, 0, \dots, 0]^T$ and $\mathbf{y}(n) = [y_{n-M}, \dots, y_n, \dots, y_{n+M}]^T$. Eq. (23.15.10) may be used for $M \leq n \leq L-1-M$. For $0 \leq n < M$ and $L-1-M < n \leq L-1$ the values of $\hat{x}_n$ can be obtained from the first M and last M outputs of $\hat{\mathbf{y}}$ in (23.15.4) applied to the first and last length-N data vectors and robustness weights:

$$\mathbf{y} = [y_0, y_1, \dots, y_{N-1}]^T\,, \quad P = \text{diag}\big([r_0, r_1, \dots, r_{N-1}]\big)$$

$$\mathbf{y} = [y_{L-N},\, y_{L-N+1},\, \dots,\, y_{L-1}]^T\,, \quad P = \text{diag}\big([r_{L-N},\, r_{L-N+1},\, \dots,\, r_{L-1}]\big)$$

From the new estimates $\hat{x}_n$, one can compute the new residuals $e_n = y_n - \hat{x}_n$, and repeat the procedure of Eqs. (23.15.8)–(23.15.10) a few more times. A total of 3–4 iterations is typically adequate. The MATLAB function `rlpfilt` implements the above steps:

```
[x,r] = rlpfilt(y,N,d,s,Nit)        % robust local polynomial filtering
```

Its outputs are the estimated signal $\hat{x}_n$ and the robustness weights r_n. The median scaling factor K is an additional optional input, which otherwise defaults to $K = 6$.

If the residuals e_n are gaussian-distributed with variance σ^2, then $\mu = 0.6745\sigma$. The default value $K = 6$ (Cleveland [776]) corresponds to allowing through 99.99 percent of the residuals. Other possible values are $K = \sqrt{6} = 2.44$ (Loader [808]) and $K = 4$ allowing respectively 90 and 99 percent of the values.

Fig. 23.15.3 shows the effect of increasing the number of robustness iterations. It is the same example as that in Fig. 23.15.1, but we have added another four outliers in the vicinity of $n = 10$. The upper-left graph corresponds to ordinary filtering without any robustness weights. One observes the successive improvement of the estimate as the number of iterations increases.

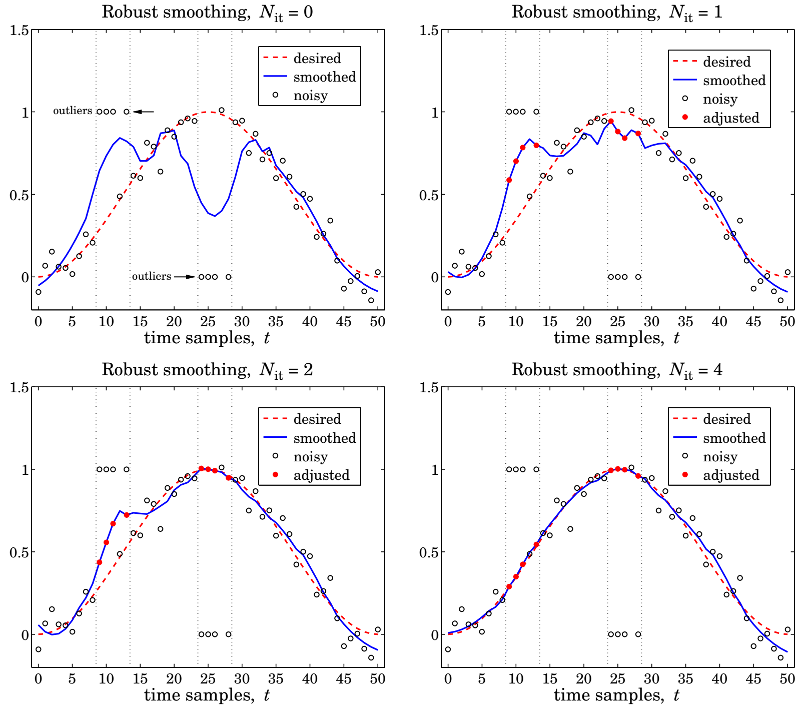


Fig. 23.15.3 Robust smoothing with outliers.

The following MATLAB code illustrates the generation of the lower-right graph. The signal y_n is generated exactly as in the previous example; the outlier values are then introduced around $n = 10$ and $n = 25$:

```
n1=10; n2=25; m = [-1 0 1 3];                  % outlier indices relative to n1 and n2
y(n1+m+1)=1;  y(n2+m+1)=0;                     % outlier values

Nit=4; K=4; x = rlpfilt(y,N,d,s,Nit,K);        % robust LP filtering
```

```
plot(t,x0,'--', t,y,'o', t,x,'-', n1+m,x(n1+m+1),'.', n2+m,x(n2+m+1),'.');
```

23.16 Weighted Local Polynomial Modeling

The methods of weighted least-squares local polynomial modeling and robust filtering can be generalized to unequally-spaced data in a straightforward fashion. Such methods provide enough flexibility to model a wide variety of data, including surfaces, and have been explored widely in recent years [772–815]. For equally-spaced data, the weighted performance index centered at time n was:

$$\mathcal{J}_n = \sum_{m=-M}^{M} \big(y_{n+m} - p(m)\big)^2 w(m) = \min\,, \quad p(m) = \sum_{r=0}^{d} c_i m^r \tag{23.16.1}$$

The value of the fitted polynomial $p(m)$ at $m = 0$ represents the smoothed estimate of y_n, that is, $\hat{x}_n = c_0 = p(0)$. Changing summation indices to $k = n + m$, Eq. (23.16.1) may be written in the form:

$$\mathcal{J}_n = \sum_{k=n-M}^{n+M} \big(y_k - p(k-n)\big)^2 w(k-n) = \min\,, \quad p(k-n) = \sum_{r=0}^{d} c_i (k-n)^r \tag{23.16.2}$$

For a set of N unequally-spaced observations $\{t_k, y(t_k)\}$, $k = 0, 1, \dots, N-1$, we wish to interpolate smoothly at some time instant t, not necessarily coinciding with one of the observation times t_k, but lying in the interval $t_0 \le t \le t_{N-1}$. A generalization of the performance index (23.16.2) is to introduce a t-dependent window bandwidth h_t, and use only the observations that lie within that window, $|t_k - t| \le h_t$, to perform the polynomial fit:

$$\mathcal{J}_t = \sum_{|t_k - t| \le h_t} \big(y(t_k) - p(t_k - t)\big)^2 w(t_k - t) = \min\,, \quad p(t_k - t) = \sum_{r=0}^{d} c_r (t_k - t)^r \tag{23.16.3}$$

The estimated/interpolated value at t will be $\hat{x}_t = c_0 = p(0)$, and the estimated first derivative, $\hat{\dot{x}}_t = c_1 = \dot{p}(0)$, and so on for the higher derivatives, with $r!\,c_r$ representing the rth derivative. As illustrated in Fig. 23.16.1, the fitted polynomial,

$$p(x-t) = \sum_{r=0}^{d} c_r (x-t)^r\,, \qquad t - h_t \le x \le t + h_t$$

is local in the sense that it fits the observations only within the local window $[t-h_t, t+h_t]$. The quantity $\hat{y}_k = p(t_k - t)$ represents the estimated value of the kth observation y_k within that window.

The weighting function $w(t_k - t)$ is chosen to have bandwidth $\pm h_t$. This can be accomplished by using a function $W(u)$ with finite support over the standardized range $-1 \le u \le 1$, and setting $u = (t_k - t)/h_t$:

$$w(t_k - t) = W\left(\frac{t_k - t}{h_t}\right) \tag{23.16.4}$$

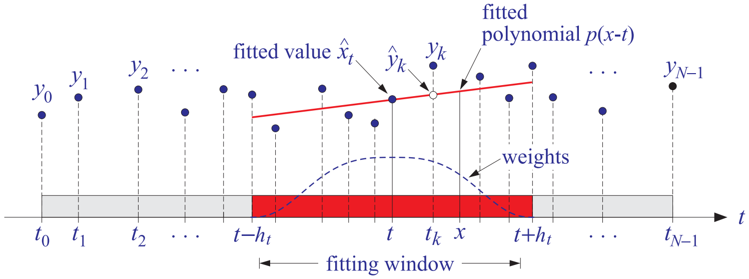


Fig. 23.16.1 Local polynomial modeling.

Some typical choices for $W(u)$ are as follows [808]:

$$
\begin{aligned}
&\text{1. Tricube,} && W(u) = (1-|u|^3)^3 \\
&\text{2. Bisquare,} && W(u) = (1-u^2)^2 \\
&\text{3. Triweight,} && W(u) = (1-u^2)^3 \\
&\text{4. Epanechnikov,} && W(u) = 1-u^2 \\
&\text{5. Gaussian,} && W(u) = e^{-\alpha^2 u^2/2} \\
&\text{6. Exponential,} && W(u) = e^{-\alpha|u|} \\
&\text{7. Rectangular,} && W(u) = 1
\end{aligned}
\tag{23.16.5}
$$

where all types have support $|u| \le 1$ and vanish for $|u| > 1$. A typical value for α in the gaussian and exponential cases is $\alpha = 2.5$. The curve shown in Fig. 23.16.1 is the tricube function; because it vanishes at $u = \pm 1$, the observations that fall exactly at the edges of the window do not contribute to the fit. The MATLAB function `locw` generates the above functions at any vector of values of u:

```
W = locw(u,type);        % local polynomial weighting functions W(u)
```

where `type` takes the values 1-7 as listed in Eq. (23.16.5). The bisquare, triweight, and Epanechnikov functions are special cases of the more general $W(u) = (1-u^2)^s$, which may be thought of as the large-M limit of the Henderson weights; in the limit $s \to \infty$ they tend to a gaussian, as in the Krawtchouk case. The various window functions are depicted in Fig. 23.16.2.

Because of the assumed finite extent of the windows, the summation in Eq. (23.16.3) can be extended to run over all N observations, as is often done in the literature:

$$
\mathcal{J}_t = \sum_{k=0}^{N-1} \big(y(t_k) - p(t_k - t)\big)^2 w(t_k - t) = \min\,, \quad p(t_k - t) = \sum_{r=0}^{d} c_r (t_k - t)^r \tag{23.16.6}
$$

We prefer the form of Eq. (23.16.3) because it emphasizes the local nature of the fitting window. Let N_t be the number of observations that fall within the interval $[t - h_t, t + h_t]$. We may cast the performance index (23.16.3) in a compact matrix form by defining the $N_t \times 1$ vector of observations $\mathbf{y}_t$, the $N_t \times (d+1)$ basis matrix S_t, and the $N_t \times N_t$ diagonal matrix of weights by

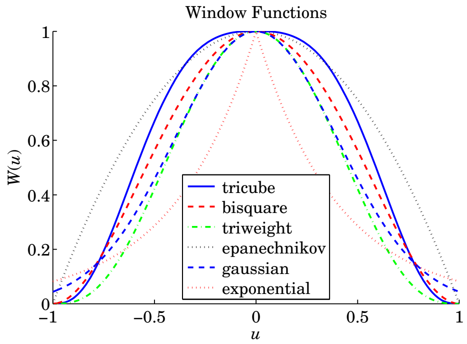


Fig. 23.16.2 Window functions.

$$
\mathbf{y}_t = [\cdots, y(t_k), \cdots]^T\,, \quad \text{for } t - h_t \le t_k \le t + h_t
$$

$$
S_t = \begin{bmatrix} \vdots & \vdots & & \vdots & & \vdots \\ 1 & (t_k - t) & \cdots & (t_k - t)^r & \cdots & (t_k - t)^d \\ \vdots & \vdots & & \vdots & & \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \mathbf{u}^T(t_k - t) \\ \vdots \end{bmatrix} \tag{23.16.7}
$$

$$
W_t = \text{diag}\big([\cdots, w(t_k - t), \cdots]\big)
$$

where $\mathbf{u}^T(t_k - t)$ is the k-th row of S_t, defined in terms of the $(d+1)$-dimensional vector $\mathbf{u}^T(\tau) = [1, \tau, \tau^2, \dots, \tau^d]$. For example, if $t - h_t < t_3 < t_4 < t_5 < t_6 < t + h_t$, then $N_t = 4$ and for a polynomial order $d = 2$, we have:

$$
\mathbf{y}_t = \begin{bmatrix} y(t_3) \\ y(t_4) \\ y(t_5) \\ y(t_6) \end{bmatrix}, \quad S_t = \begin{bmatrix} 1 & (t_3 - t) & (t_3 - t)^2 \\ 1 & (t_4 - t) & (t_4 - t)^2 \\ 1 & (t_5 - t) & (t_5 - t)^2 \\ 1 & (t_6 - t) & (t_6 - t)^2 \end{bmatrix}
$$

$$
W_t = \begin{bmatrix} w(t_3 - t) & 0 & 0 & 0 \\ 0 & w(t_4 - t) & 0 & 0 \\ 0 & 0 & w(t_5 - t) & 0 \\ 0 & 0 & 0 & w(t_6 - t) \end{bmatrix}
$$

With these definitions, Eq. (23.16.3) can be written as

$$
\boxed{\mathcal{J}_t = (\mathbf{y}_t - S_t \mathbf{c})^T W_t (\mathbf{y}_t - S_t \mathbf{c}) = \min} \tag{23.16.8}
$$

with solution for the coefficient vector $\mathbf{c} = [c_0, c_1, \dots, c_d]^T$:

$$
\boxed{\mathbf{c} = (S_t^T W_t S_t)^{-1} S_t^T W_t \mathbf{y}_t} \tag{23.16.9}
$$

The quantity $\hat{\mathbf{y}}_t = S_t\mathbf{c}$ represents the polynomial estimate of the local observation vector $\mathbf{y}_t$. It can be written as

$$\boxed{\hat{\mathbf{y}}_t = B_t^T\mathbf{y}_t\,,\quad B_t = W_tS_t(S_t^TW_tS_t)^{-1}S_t^T} \tag{23.16.10}$$

where the $N_t{\times}N_t$ matrix B_t generalizes (23.11.5), and satisfies a similar polynomial-preserving property as (23.11.6),

$$B_t^TS_t = S_t \tag{23.16.11}$$

Defining the usual $(d{+}1)$-dimensional unit vector $\mathbf{u}_0 = [1,0,\dots,0]^T$, we obtain the estimated value at time t by $\hat{x}_t = c_0 = \mathbf{u}_0^T\mathbf{c}$, and the first derivative by $\hat{\dot{x}}_t = c_1 = \mathbf{u}_1^T\mathbf{c}$, where $\mathbf{u}_1 = [0,1,0,\dots,0]^T$,

$$\boxed{\begin{aligned}\hat{x}_t &= \mathbf{u}_0^T(S_t^TW_tS_t)^{-1}S_t^TW_t\mathbf{y}_t\\ \hat{\dot{x}}_t &= \mathbf{u}_1^T(S_t^TW_tS_t)^{-1}S_t^TW_t\mathbf{y}_t\end{aligned}} \tag{23.16.12}$$

Thus, the effective estimation weights are:

$$\mathbf{h}(t) = W_tS_t(S_t^TW_tS_t)^{-1}\mathbf{u}_0\,,\quad \hat{x}_t = \mathbf{h}^T(t)\mathbf{y}_t \tag{23.16.13}$$

Component-wise, we can write:

$$\hat{x}_t = \mathbf{h}^T(t)\mathbf{y}_t = \sum_{|t_k-t|\le h_t} h_k(t)y_k \tag{23.16.14}$$

where $y_k = y(t_k)$ and

$$\boxed{h_k(t) = w(t_k-t)\mathbf{u}^T(t_k-t)\,(S_t^TW_tS_t)^{-1}\mathbf{u}_0} \tag{23.16.15}$$

We note that $\mathbf{u}_0, \mathbf{u}_1$ are related to the vector $\mathbf{u}(\tau)$ and its derivative by $\mathbf{u}_0 = \mathbf{u}(0)$ and $\mathbf{u}_1 = \dot{\mathbf{u}}(0)$. We also have,

$$S_t^TW_tS_t = \sum_{|t_k-t|\le h_t}\mathbf{u}(t_k-t)\mathbf{u}^T(t_k-t)w(t_k-t) \tag{23.16.16}$$

or, component-wise,

$$(S_t^TW_tS_t)_{ij} = \sum_{|t_k-t|\le h_t}(t_k-t)^{i+j}w(t_k-t)\,,\quad i,j = 0,1,\dots,d \tag{23.16.17}$$

The solution is particularly easy in the special cases $d = 0$, corresponding to local constant fitting, and $d = 1$, corresponding to local linear fits. The case $d = 0$ leads to the so-called *kernel smoothing* approach first proposed by Nadaraya and Watson [772,773]. In this case $\mathbf{u}(\tau) = [1]$ and we find:

$$S_t^TW_tS_t = \sum_{|t_k-t|\le h_t}w(t_k-t)\,,\quad h_k(t) = \frac{w(t_k-t)}{\displaystyle\sum_{|t_k-t|\le h_t}w(t_k-t)}$$

$$\hat{x}_t = \sum_{|t_k-t|\le h_t}h_k(t)y_k = \frac{\displaystyle\sum_{|t_k-t|\le h_t}w(t_k-t)y_k}{\displaystyle\sum_{|t_k-t|\le h_t}w(t_k-t)}\qquad\text{(kernel smoothing)} \tag{23.16.18}$$

For $d = 1$, we have $\mathbf{u}(\tau) = [1,\tau]^T$, and we obtain

$$S_t^TW_tS_t = \sum_{|t_k-t|\le h_t}\begin{bmatrix}1 & (t_k-t)\\ (t_k-t) & (t_k-t)^2\end{bmatrix}w(t_k-t) \equiv \begin{bmatrix}s_0(t) & s_1(t)\\ s_1(t) & s_2(t)\end{bmatrix}$$

$$(S_t^TW_tS_t)^{-1} = \frac{1}{s_0(t)s_2(t)-s_1^2(t)}\begin{bmatrix}s_2(t) & -s_1(t)\\ -s_1(t) & s_0(t)\end{bmatrix}$$

which gives for the filter weights $h_k(t)$:

$$h_k(t) = w(t_k-t)\,\frac{s_2(t)-(t_k-t)s_1(t)}{s_0(t)s_2(t)-s_1^2(t)}\qquad\text{(locally linear fits)} \tag{23.16.19}$$

In general, the invertibility of $S_t^TW_tS_t$ requires that $N_t \ge d+1$. The QR factorization can be used to implement the above computations efficiently. If the weight function $W(u)$ vanishes at the end-points $u = \pm 1$, as in the tricube case, then the window interval must exclude those end-points. In other words, the diagonal entries of W_t are assumed to be strictly positive. Defining U to be the diagonal square root factor of W_t and carrying out the QR factorization of the matrix US_t, we obtain the efficient algorithm:

$$\begin{aligned}&U = \mathrm{sqrt}(W_t)\,,\quad U\text{ is diagonal so that }U^T = U\text{ and }W_t = U^TU = U^2\\ &US_t = QR\,,\quad Q^TQ = I_{d+1}\,,\quad R = (d{+}1){\times}(d{+}1)\text{ upper-triangular}\\ &\mathbf{c} = R^{-1}Q^TU\mathbf{y}_t\end{aligned} \tag{23.16.20}$$

The above steps are equivalent to reducing the problem to an ordinary unweighted least-squares problem, that is, $\mathbf{c}$ is recognized to be the unique least-squares solution of the full-rank, overdetermined, $N_t{\times}(d{+}1)$-dimensional system $(US_t)\mathbf{c} = U\mathbf{y}_t$. Indeed,[†] that $\mathbf{c}$ is given by:

$$\mathbf{c} = \big[(US_t)^T(US_t)\big]^{-1}(US_t)^T(U\mathbf{y}_t) = (S_t^TW_tS_t)^{-1}S_t^TW_t\mathbf{y}_t \tag{23.16.21}$$

where $\big[(US_t)^T(US_t)\big]^{-1}(US_t)^T$ is the pseudoinverse of US_t. The corresponding performance indices are equivalent:

$$\mathcal{J}_t = (\mathbf{y}_t - S_t\mathbf{c})^TW_t(\mathbf{y}_t - S_t\mathbf{c}) = \|U\mathbf{y}_t - US_t\mathbf{c}\|^2 = \min$$

In MATLAB the least-squares solution (23.16.21) can be obtained by the backslash operation: $\mathbf{c} = (US_t)\backslash(U\mathbf{y}_t)$. The construction of the quantities $\mathbf{y}_t, S_t, W_t$ is straightforward. Given the column vectors of observation times and observations,

$$t_{\text{obs}} = [t_0, t_1, \dots, t_{N-1}]^T\,,\quad y_{\text{obs}} = \big[y(t_0), y(t_1), \dots, y(t_{N-1})\big]^T \tag{23.16.22}$$

we may determine, with the help of `locw`, the column vector of indices k for which t_k lies in the local window, and then carry out the procedure (23.16.21):

[†] see Ref. [45]

```
w = locw((tobs - t)/h_t, type);     % weights of all observation times relative to a given t and h_t
k = find(w);                        % column vector of indices of nonzero weights within window
yt = yobs(k);                       % column vector of corresponding local observations y_t
Wt = diag(w(k));                    % diagonal matrix of nonzero local weights W_t
St = [];
for r=0:d,
   St = [St, (tobs(k) - t).^r];     % construct local polynomial basis S_t column-wise
end
U = sqrt(Wt);                       % diagonal square root of W_t
c = (U*St)\(U*yt);                  % least-squares solution
```

Most of the w's obtained from the first line of code are zero, except for those t_k that lie within the local window $t \pm h_t$. The second line, `k = find(w)`, finds the latter. These steps have been incorporated into the MATLAB function `locpol`:

```
[xhat,C] = locpol(tobs,yobs,t,h,d,type);     % local polynomial modeling
```

where `tobs,yobs` are as in (23.16.22), `t,h` are L-dimensional vectors of times and bandwidths at which to carry out the fit, and `d,type` are the polynomial order and window type, with default values $d = 1$, `type` $= 1$. The output `xhat` is the L-dimensional vector of estimates $\hat{x}_t$, and `C` is an $L\times(d+1)$ matrix, whose ith row is the vector $[c_0, c_1, \dots, c_d]$ of polynomial coefficients corresponding to the ith fitting time and bandwidth $t(i), h(i)$. Thus, the first column of `C` is the same as `xhat`, while the second column contains the first derivatives. Separating the first column of `C` into `xhat` is done only for convenience in using the function.

The choice of the bandwidth h_t is an important consideration that influences the quality of the estimate $\hat{x}_t$. Too large an h_t will oversmooth the signal but reduce the noise (i.e., increasing bias but lowering variance), and too small an h_t will undersmooth the signal and not reduce the noise as much (i.e., reducing bias and increasing variance).

Two simple bandwidth choices are the *fixed* and the *nearest-neighbor* bandwidths. In the fixed case, one chooses the same bandwidth at each fitting time, that is, $h_t = h$, for all t. In the nearest-neighbor case, one chooses a fixed number, say K, of observations to lie within each local window, where K is a fraction of the total number of observations N, that is, $K = \lfloor \alpha N \rfloor$, truncated to an integer, where $\alpha \le 1$. Typical values are $\alpha = 0.2$–0.8. Given K, one calculates the distances of all the observation times from t, that is, $|t_k - t|$, $k = 0, 1, \dots, N-1$, then sorts them in increasing order, and picks h_t to be the Kth shortest distance, and therefore, there will be K observations satisfying $|t_k - t| \le h_t$. In summary, the fixed case selects $h_t = h$ but with varying N_t, and the nearest-neighbor case selects varying h_t but with fixed $N_t = K$.

The MATLAB function `locband` may be used to calculate the bandwidths h_t at each t, using either the fixed method, or the nearest-neighbor method:

```
h = locband(tobs,t,alpha,h0);     % bandwidth for local polynomial regression
```

where if $\alpha = 0$, the fixed bandwidth h_0 is selected, and if $0 < \alpha < 1$, the K-nearest bandwidths are selected, where t is a length-L vector of fitting times.

Example 23.16.1: As an example, consider the following 16 observation times t_{obs}, and 5 fitting times t, and choose $\alpha = 0.25$ so that $K = \alpha N = 0.25\times 16 = 4$:

$$t_{\text{obs}} = [0.5,\ 0.8,\ 1.1,\ 1.2,\ 1.8,\ 2.4,\ 2.5,\ 3.4,\ 3.5,\ 3.7,\ 4.0,\ 4.2,\ 4.9,\ 5.0,\ 5.1,\ 6.2]$$

$$t = [0.5, \qquad\qquad 1.5, \qquad\qquad 2.9, \qquad 3.6, \qquad\qquad\qquad\qquad 5.1]$$

then one finds the corresponding bandwidths for each of the five t's

$$\texttt{h = locband(tobs,t,0.25,0)} = [0.7,\ 0.7,\ 0.6,\ 0.6,\ 0.9]$$

and the corresponding local intervals, each containing $K = 4$ observation times:

h_t	$t - h_t$	t	$t + h_t$	included t_ks
0.7	−0.2	0.5	1.2	0.5, 0.8, 1.1, 1.2
0.7	0.8	1.5	2.2	0.8, 1.1, 1.2, 1.8
0.6	2.3	2.9	3.5	2.4, 2.5, 3.4, 3.5
0.6	3.5	4.1	4.7	3.5, 3.7, 4.0, 4.2
0.9	4.2	5.1	6.0	4.2, 4.9, 5.0, 5.1

By contrast, had we chosen a fixed bandwidth, say $h = 0.7$ (the average of the above five), then the corresponding intervals and included observation times would have been:

h_t	$t - h_t$	t	$t + h_t$	included t_ks
0.7	−0.2	0.5	1.2	0.5, 0.8, 1.1, 1.2
0.7	0.8	1.5	2.2	0.8, 1.1, 1.2, 1.8
0.7	2.2	2.9	3.6	2.4, 2.5, 3.4, 3.5
0.7	2.9	3.6	4.3	3.4, 3.5, 3.7, 4.0, 4.2
0.7	4.4	5.1	5.8	4.9, 5.0, 5.1

where now the number N_t of included observations depends on t. As can be seen from this example, both the nearest-neighbor and fixed bandwidth choices adapt well at the end-points of the available observations. □

Choosing t to be one of the observation times, $t = t_i$, Eq. (23.16.12) can be written in the simplified notation:

$$\hat{x}_i = \mathbf{u}_0^T (S_i^T W_i S_i)^{-1} S_i^T W_i \mathbf{y}_i \equiv \mathbf{h}_i^T \mathbf{y}_i\,, \quad \mathbf{h}_i^T = \mathbf{u}_0^T (S_i^T W_i S_i)^{-1} S_i^T W_i \tag{23.16.23}$$

where $\hat{x}_i, S_i, W_i, \mathbf{y}_i$ are the quantities $\hat{x}_t, S_t, W_t, \mathbf{y}_t$ evaluated at $t = t_i$. Component-wise,

$$\hat{x}_i = \sum_{|t_j - t_i| \le h_i} \mathbf{u}_0^T (S_i^T W_i S_i)^{-1} \mathbf{u}(t_j - t_i)\, w(t_j - t_i)\, y_j = \sum_{|t_j - t_i| \le h_i} H_{ij}\, y_j \tag{23.16.24}$$

where the matrix elements H_{ij} are,

$$H_{ij} = h_j(t_i) = \mathbf{u}_0^T (S_i^T W_i S_i)^{-1} \mathbf{u}(t_j - t_i)\, w(t_j - t_i) \tag{23.16.25}$$

Similarly, one may express $S_i^T W_i S_i$ and $S_i^T W_i \mathbf{y}_i$ as,

$$\begin{aligned} S_i^T W_i S_i &= \sum_{|t_j - t_i| \le h_i} \mathbf{u}(t_j - t_i)\mathbf{u}^T(t_j - t_i)\, w(t_j - t_i) \\ S_i^T W_i \mathbf{y}_i &= \sum_{|t_j - t_i| \le h_i} \mathbf{u}(t_j - t_i)\, w(t_j - t_i)\, y_j \end{aligned} \tag{23.16.26}$$

Because the factor $w(t_j - t_i)$ vanishes outside the local window $t_i \pm h_i$, the summations in (23.16.24) and (23.16.26) over t_j can be extended to run over all N observations.

Defining the N-dimensional vectors $\hat{\mathbf{x}} = [\hat{x}_0, \hat{x}_1, \dots, \hat{x}_{N-1}]^T$ and $\mathbf{y} = [y_0, y_1, \dots, y_{N-1}]^T$, we may write (23.16.24) in the compact matrix form:

$$\hat{\mathbf{x}} = H\mathbf{y} \tag{23.16.27}$$

The filtering matrix H is also known as the "hat" matrix or the "smoothing" matrix. Its diagonal elements H_{ii} play a special role in bandwidth selection, where $w_0 = w(0)$,[†]

$$H_{ii} = h_i(t_i) = w_0\, \mathbf{u}_0^T (S_i^T W_i S_i)^{-1} \mathbf{u}_0 \tag{23.16.28}$$

23.17 Bandwidth Selection with CV and GCV

There exist various automatic schemes for choosing the bandwidth. Such schemes may at best be used as guidelines. Ultimately, one must rely on one's judgment in making the final choice.

A popular bandwidth selection method is the so-called *cross-validation* criterion that selects the bandwidth h that minimizes the sum of squared prediction errors:

$$\mathrm{CV}(h) = \frac{1}{N}\sum_{i=0}^{N-1} (y_i - \hat{x}_i^-)^2 = \min \tag{23.17.1}$$

where $\hat{x}_i^-$ is the estimate or prediction of the sample $x_i = x(t_i)$ obtained by *deleting* the ith observation y_i and basing the estimation on the remaining observations, where we are assuming the usual additive-noise model $y(t_i) = x(t_i) + v(t_i)$ with $x(t_i)$ representing the desired signal to be extracted from $y(t_i)$. We show below that the predicted estimate $\hat{x}_i^-$ is related to the estimate $\hat{x}_i$ based on all observations by the relationship:

$$\hat{x}_i^- = \frac{\hat{x}_i - H_{ii} y_i}{1 - H_{ii}} \tag{23.17.2}$$

where H_{ii} is given by (23.16.28). It follows from (23.17.2) that the corresponding estimation errors will be related by:

$$y_i - \hat{x}_i^- = \frac{y_i - \hat{x}_i}{1 - H_{ii}} \tag{23.17.3}$$

and therefore, the CV index can be expressed as:

$$\mathrm{CV}(h) = \frac{1}{N}\sum_{i=0}^{N-1} (y_i - \hat{x}_i^-)^2 = \frac{1}{N}\sum_{i=0}^{N-1} \left(\frac{y_i - \hat{x}_i}{1 - H_{ii}}\right)^2 = \min \tag{23.17.4}$$

A related selection criterion is based on the *generalized cross-validation* index, which replaces H_{ii} by its average over i, that is,

$$\mathrm{GCV}(h) = \frac{1}{N}\sum_{i=0}^{N-1} \left(\frac{y_i - \hat{x}_i}{1 - \bar{H}}\right)^2 = \min\,, \quad \bar{H} = \frac{1}{N}\sum_{i=0}^{N-1} H_{ii} = \frac{1}{N}\,\mathrm{tr}(H) \tag{23.17.5}$$

[†] $w_0 = 1$ for all the windows in Eq. (23.16.5), but any other normalization can be used.

If the bandwidth is to be selected by the nearest-neighbor method, then, the CV and GCV indices may be regarded as functions of the fractional parameter α and minimized. Similarly, one could consider minimizing with respect to the polynomial order d, although in practice d is usually chosen to be 1 or 2.

Eq. (23.17.2) can be shown as follows. If the $t_j = t_i$ observation is deleted from Eq. (23.16.23), the corresponding optimum polynomial coefficients and optimum estimate will be given by

$$\mathbf{c}_- = (S_i^T W_i S_i)_-^{-1} (S_i^T W_i \mathbf{y}_i)_-\,, \quad \hat{x}_i^- = \mathbf{u}_0^T \mathbf{c}_-$$

where the minus subscripts indicate that the $t_j = t_i$ terms are to be omitted. It follows from Eq. (23.16.26) that

$$\begin{aligned} S_i^T W_i S_i &= (S_i^T W_i S_i)_- + w_0 \mathbf{u}_0 \mathbf{u}_0^T \\ S_i^T W_i \mathbf{y}_i &= (S_i^T W_i \mathbf{y}_i)_- + w_0 \mathbf{u}_0 y_i \end{aligned} \tag{23.17.6}$$

and then,

$$\mathbf{c}_- = [S_i^T W_i S_i - w_0 \mathbf{u}_0 \mathbf{u}_0^T]^{-1} [S_i^T W_i \mathbf{y}_i - w_0 \mathbf{u}_0 y_i] \tag{23.17.7}$$

Setting $F_i = S_i^T W_i S_i$ and noting that $\mathbf{c} = F_i^{-1} S_i^T W_i \mathbf{y}_i$ or $S_i^T W_i \mathbf{y}_i = F_i \mathbf{c}$, we may write,

$$\mathbf{c}_- = [F_i - w_0 \mathbf{u}_0 \mathbf{u}_0^T]^{-1} [F_i \mathbf{c} - w_0 \mathbf{u}_0 y_i]$$

Using the matrix inversion lemma, we have,

$$[F_i - w_0 \mathbf{u}_0 \mathbf{u}_0^T]^{-1} = F_i^{-1} + \frac{w_0 F_i^{-1} \mathbf{u}_0 \mathbf{u}_0^T F_i^{-1}}{1 - w_0 \mathbf{u}_0^T F_i^{-1} \mathbf{u}_0} \tag{23.17.8}$$

Noting that $H_{ii} = w_0 \mathbf{u}_0^T F_i^{-1} \mathbf{u}_0$, we obtain,

$$\begin{aligned} \mathbf{c}_- &= \left[F_i^{-1} + \frac{w_0 F_i^{-1} \mathbf{u}_0 \mathbf{u}_0^T F_i^{-1}}{1 - H_{ii}}\right] [F_i \mathbf{c} - w_0 \mathbf{u}_0 y_i] \\ &= \left[I + \frac{w_0 F_i^{-1} \mathbf{u}_0 \mathbf{u}_0^T}{1 - H_{ii}}\right] [\mathbf{c} - w_0 F_i^{-1} \mathbf{u}_0 y_i] \\ &= \mathbf{c} - w_0 F_i^{-1} \mathbf{u}_0 y_i + \frac{w_0 F_i^{-1} \mathbf{u}_0 [\mathbf{u}_0^T \mathbf{c} - w_0 \mathbf{u}_0^T F_i^{-1} \mathbf{u}_0 y_i]}{1 - H_{ii}} \end{aligned}$$

and since $\hat{x}_i = \mathbf{u}_0^T \mathbf{c}$, we find,

$$\mathbf{c}_- = \mathbf{c} - w_0 F_i^{-1} \mathbf{u}_0 y_i + \frac{w_0 F_i^{-1} \mathbf{u}_0 [\hat{x}_i - H_{ii} y_i]}{1 - H_{ii}} = \mathbf{c} + w_0 F_i^{-1} \mathbf{u}_0 \frac{\hat{x}_i - y_i}{1 - H_{ii}}$$

from which we find for $\hat{x}_i^- = \mathbf{u}_0^T \mathbf{c}_-$,

$$\hat{x}_i^- = \hat{x}_i + \frac{H_{ii}(\hat{x}_i - y_i)}{1 - H_{ii}} = \frac{\hat{x}_i - H_{ii} y_i}{1 - H_{ii}} \tag{23.17.9}$$

In practice, the CV and GCV indices are evaluated over a certain range of the smoothing parameter h or α to look for a minimum. The MATLAB function `locgcv` evaluates these indices at any vector of parameter values:

```
[GCV,CV] = locgcv(tobs,yobs,d,type,b,btype);     % CV and GCV evaluation
```

where `type` is the window type, `b` is either a vector of hs or a vector of αs at which to evaluate CV and GCV, and the string `btype` takes the values `'f'` or `'nn'` specifying whether the parameter vector `b` corresponds to a fixed or nearest-neighbor bandwidth.

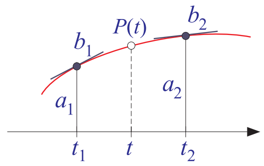
23.18 Local Polynomial Interpolation

The primary advantage of local polynomial modeling is its flexibility and ease of smoothing unequally-spaced data. Its main disadvantage is the potentially high computational cost, that is, the calculations (23.16.12) must be performed for each t, and generally a dense set of such t's might be required in order to get a visually smooth curve.

One way to cut down the cost is to evaluate the smoothed values $\hat{x}_t$ at a less dense grid of ts, and then interpolate smoothly between the computed points. This is akin to what plotting programs do by connecting the dots by straight-line segments (linearly interpolating)—the result being a visually continuous curve. But here, we can do better than just connecting the dots because we have available the slopes at each grid point. These slopes are contained in the second column of the fitting matrix C resulting from `locpol`, assuming of course that $d \geq 1$.

Consider two time instants t_1, t_2 at which the fitted signal values are a_1, a_2 with corresponding slopes b_1, b_2, as shown below. The lowest-degree polynomial $P(t)$ interpolating between the two points t_1, t_2 that matches the fitted values and their slopes at t_1 and t_2 is a cubic polynomial—the method being known as cubic Hermite interpolation. The four polynomial coefficients are fixed uniquely by the four conditions:

$$P(t_1) = a_1\,, \quad \dot{P}(t_1) = b_1$$
$$P(t_2) = a_2\,, \quad \dot{P}(t_2) = b_2$$



which result into the cubic polynomial, where $T = t_2 - t_1$,

$$\begin{aligned} P(t) = &\left(\frac{t-t_2}{T}\right)^2\left[a_1 + (Tb_1 + 2a_1)\left(\frac{t-t_1}{T}\right)\right] \\ &+ \left(\frac{t-t_1}{T}\right)^2\left[a_2 + (Tb_2 - 2a_2)\left(\frac{t-t_2}{T}\right)\right] \end{aligned} \tag{23.18.1}$$

For local-polynomial orders $d \geq 1$, we use Eq. (23.18.1) to interpolate at a denser grid of points between the less dense grid of fitted points. For the special case, $d = 0$, the slopes are not available and we can only use linear interpolation, that is,

$$P(t) = a_1 + (a_2 - a_1)\left(\frac{t-t_1}{T}\right) \tag{23.18.2}$$

The MATLAB function `locval` takes the output matrix `C` from `locpol` corresponding to a grid of fitting points t, and computes the interpolated points y_{grid} at the denser grid of points t_{grid}:

```
ygrid = locval(C,t,tgrid);     % interpolating local polynomial fits
```

The auxiliary function `locgrid` helps establish a uniform grid between the t points:

```
tgrid = locgrid(t, Ngrid);     % uniform grid with respect to t
```

which is simply a shorthand for,

```
tgrid = linspace(min(t), max(t), Ngrid);
```

Example 23.18.1: The motorcycle acceleration dataset [815] has served as a benchmark in many studies of local polynomial modeling and spline smoothing. The ordinate represents head acceleration (in units of g) during impact, and the abscissa is the time (in msec).

Fig. 23.18.1 shows a plot of the GCV index as a function of the nearest-neighbor fractional parameter α on the left, and as a function of the fixed bandwidth h on the right, for the two polynomial orders $d = 1, 2$.

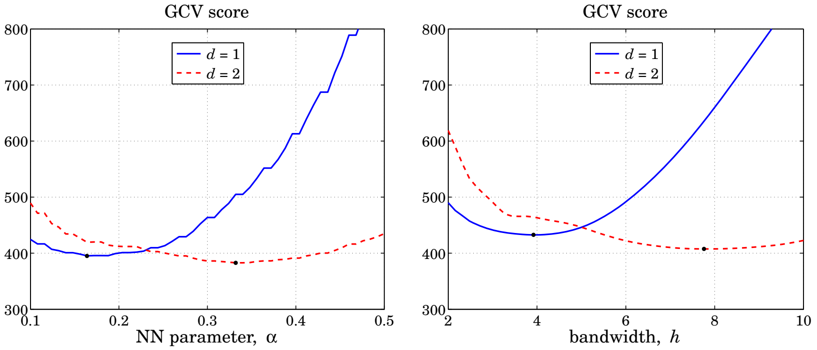


Fig. 23.18.1 GCV score for nearest-neighbor (left) and fixed bandwidths (right).

The "optimal" values of these parameters that minimize the GCV (and indicated by dots on the graphs) are as follows, where the subscripts indicate the value of d:

$$\alpha_1 = 0.16\,, \quad \alpha_2 = 0.33\,, \quad h_1 = 3.9\,, \quad h_2 = 7.8$$

The graphs (for $d = 1$) were produced by the MATLAB code:

```
Y = loadfile('mcyc.dat');                            % file included in the OSP toolbox
tobs = Y(:,1);  yobs = Y(:,2);                       % 133 data points

alpha = linspace(0.1, 0.5, 51);                      % vary over 0.1 ≤ α ≤ 0.5

d=1; type=1;
gcv = locgcv(tobs,yobs,d,type,alpha,'nn');           % GCV as function of α
[F,i] = min(gcv);  alpha1 = alpha(i);                % minimum at α = α1

figure; plot(alpha,gcv);                             % left graph

h = linspace(2, 10, 51);                             % vary over 2 ≤ h ≤ 10
gcv = locgcv(tobs,yobs,d,type,h,'f');                % GCV as function of h
[F,i] = min(gcv);  h1 = h(i);                        % minimum at h = h1
```

Fig. 23.18.2 shows the local polynomial fits corresponding to the above optimal parameter values. The left graph shows the nearest-neighbor cases for $d = 1, 2$, and the right graph, the fixed bandwidth cases. The tricube window was used (`type=1`).

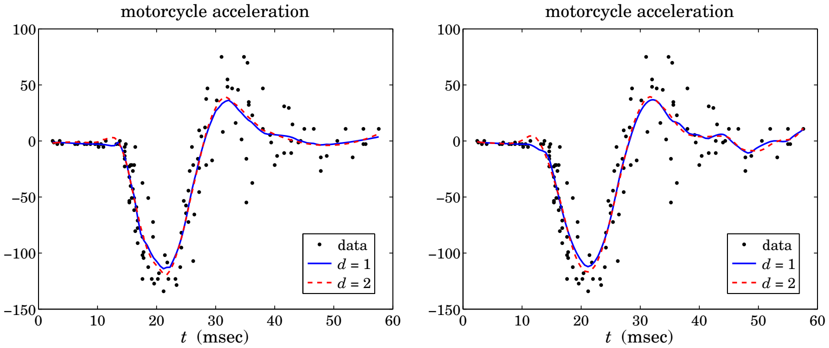


Fig. 23.18.2 Nearest-neighbor (left) and fixed bandwidths (right).

In all cases, the actual fitting was performed at 100 equally-spaced points t within the observation range t_{obs} and were connected by straight-line segments by the plotting program, instead of being interpolated by `locval`. Continuing with the above MATLAB code, the graphs were generated by

```
t = locgrid(tobs,101);                              % equally-spaced fitting times

h = locband(tobs, t, alpha1, 0);                    % NN bandwidths at each t
x1 = locpol(tobs,yobs,t,h,1,type);                  % fit at times t with d = 1

h = locband(tobs, t, alpha2, 0);
x2 = locpol(tobs,yobs,t,h,2,type);                  % fit at times t with d = 2

figure; plot(tobs,yobs,'.', t,x1,'-', t,x2,'--');   % left graph

h = locband(tobs, t, 0, h1);                        % fixed bandwidths at each t
x1 = locpol(tobs,yobs,t,h,1,type);                  % fit at times t with d = 1

h = locband(tobs, t, 0, h2);
x2 = locpol(tobs,yobs,t,h,2,type);                  % fit at times t with d = 2

figure; plot(tobs,yobs,'.', t,x1,'-', t,x2,'--');   % right graph
```

Fig. 23.18.3 demonstrates the Hermite interpolation procedure. The fitting times are 20 equally-spaced points spanning the observation interval t_{obs}. The 20 fitted points are then interpolated at 100 equally-spaced points over t_{obs}. The interpolated curves are essentially identical to those fitted earlier at 100 points.

The polynomial order was $d = 1$ and the bandwidth parameters were $\alpha_1 = 0.21$ for the left graph and $h_1 = 4.4$ for the right one. The left graph was generated by the code segment:

```
tf = locgrid(tobs,21);                        % fitting times
```

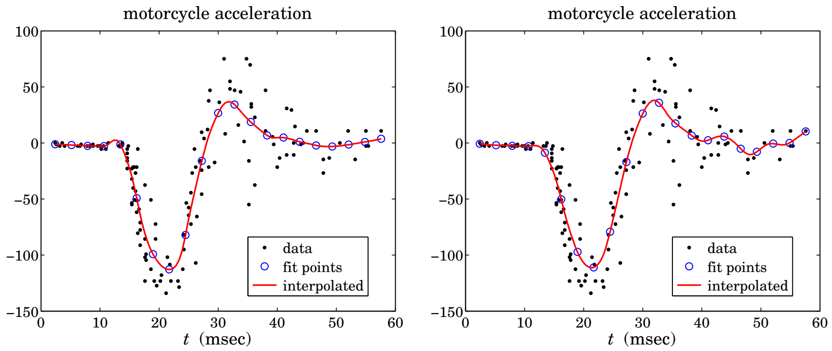


Fig. 23.18.3 Nearest-neighbor (left) and fixed bandwidths (right).

```
h = locband(tobs, tf, alpha1, 0);             % NN bandwidths at tf
[xf,C] = locpol(tobs,yobs,tf,h,1,type);       % fitted values and derivatives

tint = locgrid(tf,101);                       % interpolation times
xint = locval(C, tf, tint);                   % interpolated values

figure; plot(tobs,yobs,'.', tf,xf,'o', tint,xint,'-');
```

Example 23.18.2: The ethanol dataset [814] is also a benchmark example for smoothing techniques. The ordinate NO_x represents nitric oxide concentrations in the engine exhaust gases, and the abscissa E is the equivalence ratio, which is a measure of the richness of the ethanol/air mixture.

The GCV and CV bandwidth selection criteria tend sometimes to result in undersmoothed signals. This can be seen in Fig. 23.18.4 in which the GCV criterion for fixed bandwidth selects the values $h_1 = 0.039$ and $h_2 = 0.058$, for orders $d = 1, 2$.

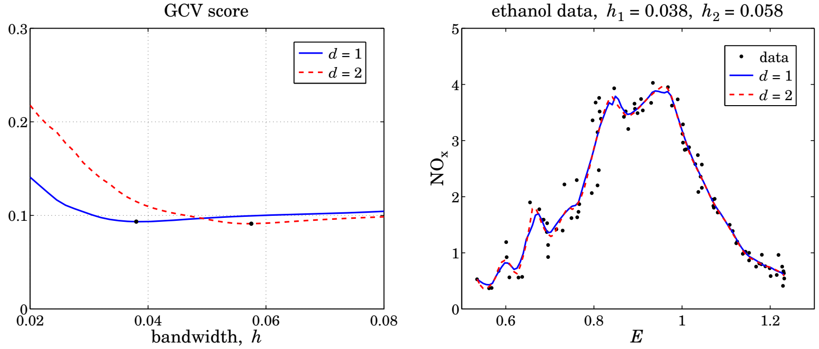


Fig. 23.18.4 GCV and local polynomial fits with $d = 1, 2$.

As can be seen, the resulting fits are jagged, and can benefit form increasing the fitting bandwidth somewhat. The minima of the GCV plot are fairly broad and any neighboring values of the bandwidth would be just as good in terms of the GCV value. A similar effect happens in this example for the nearest-neighbor bandwidth method, in which the GCV criterion selects the value $\alpha = 0.19$ corresponding to jagged graph (not shown). Fig. 23.18.5 shows the fits when the fixed bandwidth is increased to $h = 0.08$ and the nearest-neighbor one to $\alpha = 0.3$. The resulting fits are much smoother.

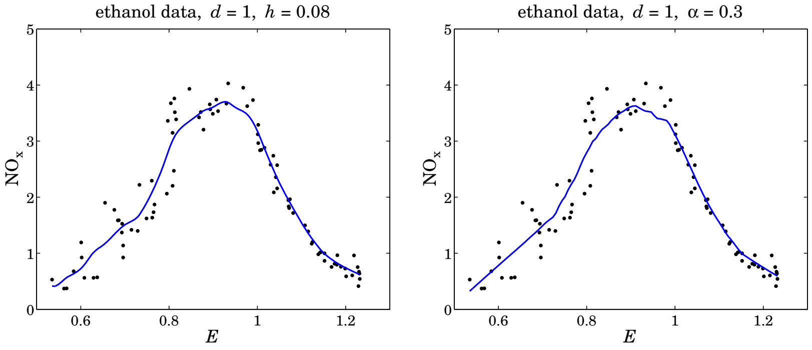


Fig. 23.18.5 Fits with fixed (left) and nearest-neighbor (right) bandwidths.

The MATLAB code for generating the graphs of Fig. 23.18.4 is as follows:

```
Y = loadfile('ethanol.dat');                        % file available in OSP toolbox
tobs = Y(:,1);  yobs = Y(:,2);                      % data

t = locgrid(tobs,101);                              % uniform grid of 101 fitting points

h = linspace(0.02, 0.08, 41);                       % vary h over 0.02 ≤ h ≤ 0.08
gcv1 = locgcv(tobs,yobs,1,1,h,'f');                 % GCV as function of h
gcv2 = locgcv(tobs,yobs,2,1,h,'f');

figure; plot(h,gcv1,'-', h,gcv2,'--');              % left graph

h = locband(tobs, t, 0, h1);                        % fixed bandwidths at t
x1 = locpol(tobs,yobs,t,h,1,1);                     % fit with d = 1 and tricube window

h = locband(tobs, t, 0, h2);
x2 = locpol(tobs,yobs,t,h,2,1);                     % fit with d = 2 and tricube window

figure; plot(tobs,yobs,'.', t,x1,'-', t,x2,'--');      % right graph
```

The MATLAB code for generating Fig. 23.18.5 is as follows:

```
h0 = 0.08; h = locband(tobs, t, 0, h0);               % fixed bandwidth case
x1 = locpol(tobs,yobs,t,h,1,1);                       % order d = 1, tricube window
figure; plot(tobs,yobs,'.', t,x1,'-');                % left graph

alpha = 0.3; h = locband(tobs, t, alpha, 0);          % nearest-neighbor bandwidth case
x1 = locpol(tobs,yobs,t,h,1,1); x1 = C(:,1);          % order d = 1, tricube windowm
figure; plot(tobs,yobs,'.', t,x1,'-');                % right graph
```

Fig. 23.18.6 shows a fit at 10 fitting points and interpolated over 101 points. The fitting parameters are as in the right graph of Fig. 23.18.5. The following code generates Fig. 23.18.6:

```
tf = locgrid(tobs,10);                                   % fitting points
alpha = 0.3; h = locband(tobs, tf, alpha, 0);            % nearest-neighbor bandwidths
[xf,C] = locpol(tobs,yobs,tf,h,1,1);                     % order 1, tricube window

ti = locgrid(tf,101); yi = locval(C,tf,ti);              % interpolated points

figure; plot(tobs,yobs,'.', ti,yi,'-', tf,xf,'o');
```

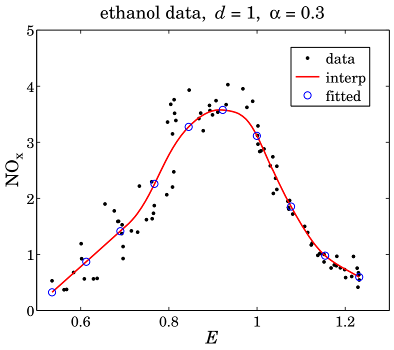


Fig. 23.18.6 Interpolated fits.

23.19 Variable and Adaptive Bandwidth

The issue of selecting the right bandwidth has been studied extensively, with approaches ranging from finding an optimum bandwidth that minimizes a selection criterion such as the GCV to using a locally-adaptive criterion that allows the bandwidth to automatically adapt to the local nature of the signal with different bandwidths being used in different parts of the signal [772–815].

There is no selection criterion that is universally successful or universally agreed upon and one must use one's judgment and visual inspection to decide how much smoothing is satisfactory. The basic idea is always to reduce the bandwidth in regions where the curvature of the signal is high in order not to oversmooth.

The function `locpol` can accept a different bandwidth h_t for each fitting time t. As we saw in the above examples, the function `locband` generates such bandwidths for input to `locpol`. However, `locband` generates either fixed or or nearest-neighbor bandwidths and is not adaptive to the local nature of the signal. One could manually, divide the range of the signal in non-overlapping regions and use a different fixed bandwidth in each region. In some cases, as in the Doppler example below, this is possible but in other cases a more automatic way of adapting is desirable.

A naive, but as we see in the examples below, quite effective way is to estimate the curvature, say κ_t, of the signal and define the bandwidth in terms of a suitable decreasing function $h_t = f(\kappa_t)$. We may define the curvature in terms of the estimate of the second derivative of the signal and normalize it to its maximum value:

$$\kappa_t = \frac{|\hat{\ddot{x}}_t|}{\max_t |\hat{\ddot{x}}_t|} \tag{23.19.1}$$

The second derivative $\hat{\ddot{x}}_t$ can be estimated by performing a local polynomial fit with polynomial order $d \geq 2$ using a fixed bandwidth h_0 or a nearest-neighbor bandwidth α. If one could determine a bandwidth range $[h_{\min}, h_{\max}]$ such that $h_{\max}$ would provide an appropriate amount of smoothing in certain parts of the signal and $h_{\min}$ would be appropriate in regions where the signal appears to have larger curvature, then one may choose $h_{\min} \leq h_0 \leq h_{\max}$, with $h_0 = h_{\max}$ as an initial trial value. An ad hoc but very simple choice for the bandwidth function $f(\kappa_t)$ then could be

$$h_t = h_{\max}\left(\frac{h_{\min}}{h_{\max}}\right)^{\kappa_t} \tag{23.19.2}$$

Other simple choices are possible, for example,

$$h_t = \frac{h_{\max}h_{\min}}{h_{\min} + (h_{\max} - h_{\min})\kappa_t^p}$$

for some power p. Since κ_t varies in $0 \leq \kappa_t \leq 1$, these choices interpolate between $h_{\max}$ at $\kappa_t = 0$ when the curvature is small, and $h_{\min}$ at $\kappa_t = 1$ when the curvature is large.

We illustrate the use of (23.19.2) with the three examples in Figs. 23.19.1–23.19.3, and we make a different bandwidth choice for Fig. 23.19.4. All four examples have been used as benchmarks in studying wavelet denoising methods [569] and we will be discussing them again in that context in Sec. 20.7.

In all cases, we use a second-order polynomial to determine the curvature, and then perform a locally linear fit ($d = 1$) using the variable bandwidth. Fig. 23.19.1 illustrates the test function “bumps” defined by

$$s(t) = \sum_{i=1}^{11} \frac{a_i}{\left[1 + |t - t_i|/w_i\right]^4}, \quad 0 \leq t \leq 1$$

with the parameter values:

$$t_i = [10, 13, 15, 23, 25, 40, 44, 65, 76, 78, 81]/100$$

$$a_i = [40, 50, 30, 40, 50, 42, 21, 43, 31, 51, 42]\cdot 1.0523$$

$$w_i = [5, 5, 6, 10, 10, 30, 10, 10, 5, 8, 5]/1000$$

The function $s(t)$ is sampled at $N = 2048$ equally-spaced points t_n in the interval $[0, 1)$ and zero-mean white gaussian noise of variance $\sigma^2 = 1$ is added so that the noisy signal is $y_n = s_n + v_n$, where $s_n = s(t_n)$. The factor 1.0523 in the amplitudes a_i ensures that the signal-to-noise ratio has the standard benchmark value $\sigma_s/\sigma_v = 7$, where σ_s is the standard deviation of s_n, that is, $\sigma_s = \text{std}(s)$. The bandwidth range is defined by $h_{\max} = 0.01$ and $h_{\min} = 0.00025$. The value for $h_{\max}$ was chosen so that the flat portions of the signal between peaks are adequately smoothed.

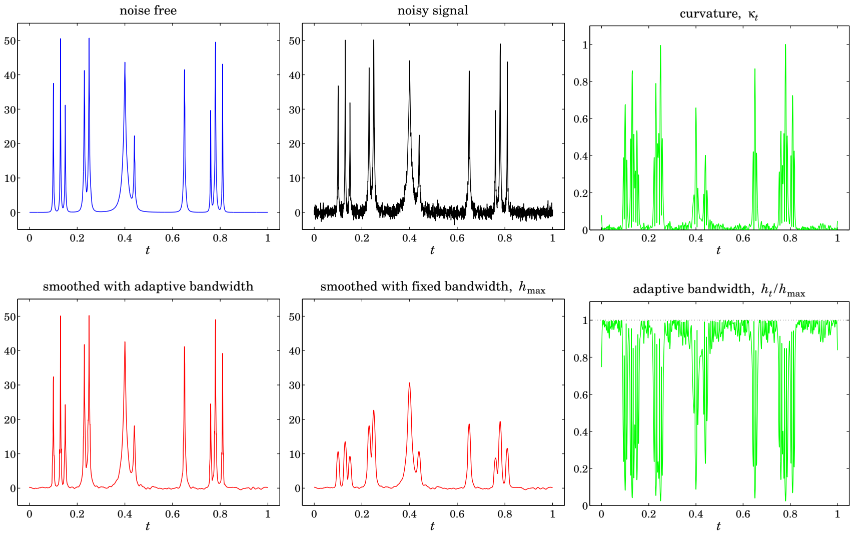


Fig. 23.19.1 Bumps function.

The curvature κ_t, estimated using the bandwidth $h_0 = h_{\max}$, is shown in the upper right graph. The corresponding variable bandwidth h_t derived from Eq. (23.19.2) is shown in the bottom-right graph. The bottom-left graph shows the resulting local linear fit using the variable bandwidth h_t, while the bottom-middle graph shows the fit using the fixed bandwidth $h_{\max}$. Although $h_{\max}$ is adequate for smoothing the valleys of the signal, it is too large for the peaks and results in broadened peaks of reduced heights. On the other hand, the variable bandwidth preserves the peaks fairly well, while achieving comparable smoothing of the valleys. The MATLAB code for this example was as follows:

```
N=2048; t=linspace(0,1,N); s=zeros(size(t));
F = inline('1./(1 + abs(t)).^4');              % bumps function

ti = [10 13 15 23 25 40 44 65 76 78 81]/100;
ai = [40,50,30,40,50,42,21,43,31,51,42] * 1.0523;
wi = [5,5,6,10,10,30,10,10,5,8,5]/1000;

for i=1:length(ai),                            % construct signal
   s = s + ai(i)*F((t-ti(i))/wi(i));
end

hmax=10e-3; hmin=2.5e-4; h0=hmax;              % bandwidth limits

seed=2009; randn('state',seed);                % noisy signal
y = s + randn(size(t));

d=2; type=1;                                   % fit with d = 2 and tricube window
[x,C] = locpol(t,y,t,h0,d,type);               % using fixed bandwidth h0
kt = abs(C(:,3)); kt = kt/max(kt);             % curvature, κt
ht = hmax * (hmin/hmax).^kt;                   % bandwidth, ht
```

```
d=1; type=1;                                    % fit with d = 1
xv = locpol(t,y,t,ht,d,type);                   % use variable bandwidth h_t
xf = locpol(t,y,t,h0,d,type);                   % use fixed bandwidth h_0

figure; plot(t,s);  figure; plot(t,y);  figure; plot(t,kt);          % upper graphs
figure; plot(t,xv); figure; plot(t,xf); figure; plot(t,ht/hmax);     % lower graphs
```

Fig. 23.19.2 shows the “blocks” function defined by

$$s(t)=\sum_{i=1}^{11} a_i F(t-t_i)\,,\quad F(t)=\frac{1}{2}(1+\operatorname{sign} t)\,,\quad 0\le t\le 1$$

with the same delays t_i as above and amplitudes:

$$a_i=[40,-50,30,-40,50,-42,21,43,-31,21,-42]\cdot 0.3655$$

The noisy signal is $y_n = s_n + v_n$ with zero-mean unit-variance white noise. The amplitude factor 0.3655 in a_i is adjusted to give the same SNR as above, $\text{std}(s)/\sigma = 7$. The MATLAB code generating the six graphs is identical to the above, except for the part that defines the signal and the bandwidth limits $h_{\max} = 0.03$ and $h_{\min} = 0.0015$:

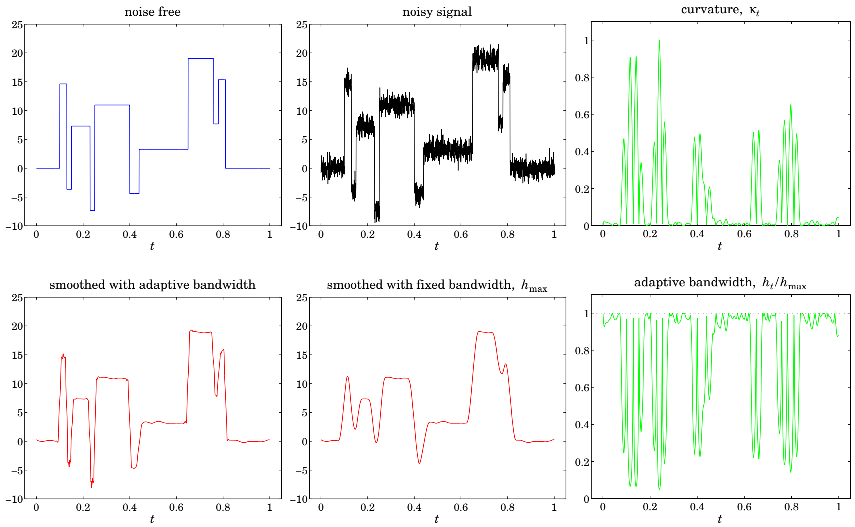


Fig. 23.19.2 Blocks function.

```
N=2048; t=linspace(0,1,N); s=zeros(size(t));

ti = [10 13 15 23 25 40 44 65 76 78 81]/100;
ai = [40,-50,30,-40,50,-42,21,43,-31,21,-42] * 0.3655;
```

```
for i=1:length(ai),
    s = s + ai(i) * (1 + sign(t - ti(i)))/2;          % blocks signal
end

hmax=0.03; hmin=0.0015; h0=hmax;                       % bandwidth limits
```

We observe that the flat parts of the signal are smoothed equally well by the variable and fixed bandwidth choices, but in the fixed case, the edges are smoothed too much. The “HeaviSine” signal shown in Fig. 23.19.3 is defined by

$$s(t)=\big[4\sin(4\pi t)-\operatorname{sign}(t-0.3)-\operatorname{sign}(0.72-t)\big]\cdot 2.357\,,\quad 0\le t\le 1$$

where the factor 2.357 is adjusted to give $\text{std}(s)=7$. The graphs shown in Fig. 23.19.3 are again generated by the identical MATLAB code, except for the parts defining the signal and bandwidths:

```
s = (4*sin(4*pi*t)-sign(t-0.3)-sign(0.72-t))*2.357;          % HeaviSine signal

hmax=0.035; hmin=0.0035; h0=hmax;                              % bandwidth limits
```

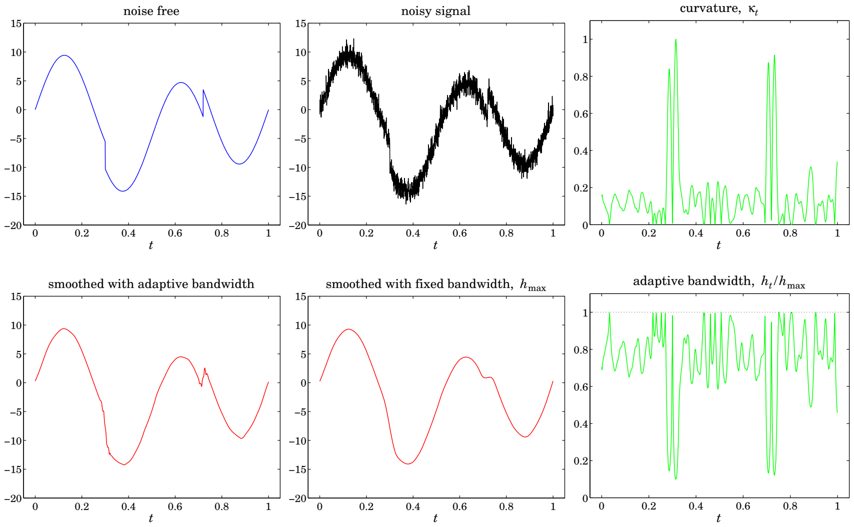


Fig. 23.19.3 HeaviSine function.

We note that the curvature κ_t is significantly large—and the bandwidth h_t is significantly small—only near the discontinuity points. The fixed bandwidth case smoothes the discontinuities too much, whereas the variable bandwidth tends to preserve them while reducing the noise equally well in the rest of the signal.

In the “doppler” example shown in Fig. 23.19.4, noticing that the curvature κ_t is significantly large only in the range $0 \le t \le 0.2$, we have followed a simpler strategy to define a variable bandwidth (although the choice (23.19.2) still works well). We took a fixed but small bandwidth over the range $0 \le t \le 0.2$ and transitioned gradually to a larger bandwidth for $0.2 \le t \le 1$. The signal is defined by

$$s(t) = 24\sqrt{t(1-t)}\,\sin\left(\frac{2.1\pi}{t+0.05}\right), \quad 0 \le t \le 1$$

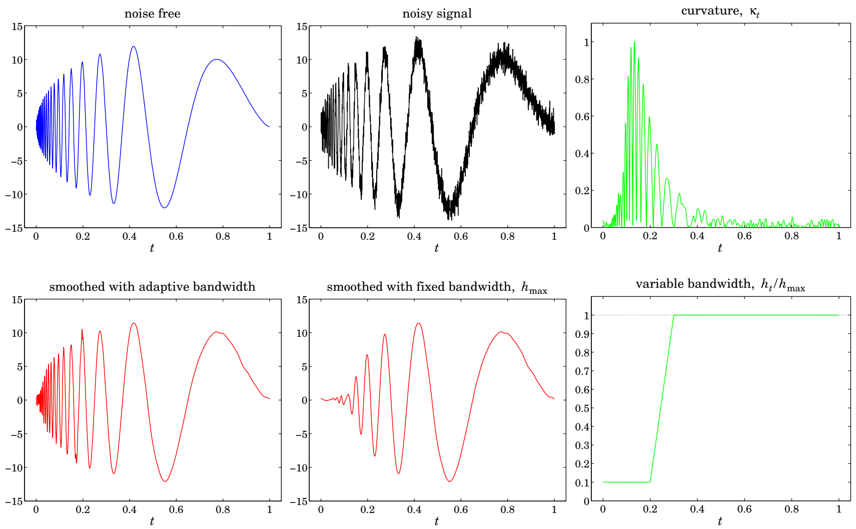


Fig. 23.19.4 Doppler function.

The auxiliary unit-step function `ustep` was used to define the two-step bandwidth with a given rise time. The MATLAB code generating the six graphs was as follows:

```
N = 2048; t = linspace(0,1,N);

s = 24*sqrt(t.*(1-t)) .* sin(2.1*pi./(t+0.05));   % doppler signal

seed=2009; randn('state',seed);                    % noisy signal
y = s + randn(size(t));

hmax=0.02; hmin=0.002; h0=hmax;                    % bandwidth limits

d=2; type=1;                                       % fit with d = 2 and tricube window
[x,C] = locpol(t,y,t,h0,d,type);                   % using fixed bandwidth h0
kt = abs(C(:,3)); kt = kt/max(kt);                 % curvature, κt
ht = hmin + (hmax-hmin) * ustep(t-0.2, 0.1);       % two-step bandwidth, ht
                                                   % ustep is in the OSP toolbox

d=1; type=1;
xv = locpol(t,y,t,ht,d,type);                      % fixed bandwidth h0
xf = locpol(t,y,t,h0,d,type);                      % fixed bandwidth h0

figure; plot(t,s);  figure; plot(t,y);  figure; plot(t,kt);          % upper graphs
figure; plot(t,xv); figure; plot(t,xf); figure; plot(t,ht/hmax);     % lower graphs
```

The local polynomial fitting results from these four benchmark examples are very comparable with the wavelet denoising approach discussed in Sec. 20.7.

23.20 Repeated Observations

Until now we had implicitly assumed that the observations were unique, that is, one observation $y(t_k)$ at each time t_k. However, in experimental data one often has repeated observations at a given t_k, all of which are listed as part of the data set. This is in fact true of both the motorcycle and the ethanol data sets. For example, in the motorcycle data, we have six repeated observations at $t = 14.6$,

k	t_k	y_k
⋮	⋮	⋮
22	14.6	−13.3
23	14.6	−5.4
24	14.6	−5.4
25	14.6	−9.3
26	14.6	−16.0
27	14.6	−22.8
⋮	⋮	⋮

and there other similar instances within the data set. In fact, among the 133 given observations, only 94 correspond to unique observation times.

To handle repeated observations one possibility is to simply keep one and ignore the rest—but which one? A better possibility is to allow all of them to be part of the least-squares performance index. It is easy to see that this is equivalent to replacing each group of repeated observations by their average and modifying the weighting function by the corresponding multiplicity of the group.

Let n_k denote the multiplicity of the observations at time t_k, that is, let $y_i(t_k)$, $i = 1, 2, \dots, n_k$ be the observation values that are given at the unique observation time t_k. Then, the performance index (23.16.3) must be modified to include all of the $y_i(t_k)$:

$$\mathcal{J}_t = \sum_{|t_k-t|\le h_t}\sum_{i=1}^{n_k}\big[y_i(t_k) - \mathbf{u}^T(t_k-t)\,\mathbf{c}\big]^2 w(t_k-t) = \min \tag{23.20.1}$$

Setting the gradient with respect to $\mathbf{c}$ to zero, gives the normal equations:

$$\sum_{|t_k-t|\le h_t}\sum_{i=1}^{n_k} w(t_k-t)\,\mathbf{u}(t_k-t)\,\mathbf{u}^T(t_k-t)\,\mathbf{c} = \sum_{|t_k-t|\le h_t} w(t_k-t)\,\mathbf{u}(t_k-t)\sum_{i=1}^{n_k} y_i(t_k)$$

Defining the average of the n_k observations,

$$\bar{y}(t_k) = \frac{1}{n_k}\sum_{i=1}^{n_k} y_i(t_k)$$

and noting that the left-hand side has no dependence on i, we obtain:

$$\sum_{|t_k-t|\le h_t} n_k w(t_k-t)\mathbf{u}(t_k-t)\mathbf{u}^T(t_k-t)\mathbf{c} = \sum_{|t_k-t|\le h_t} n_k w(t_k-t)\mathbf{u}(t_k-t)\bar{y}(t_k) \qquad (23.20.2)$$

This is recognized to be the solution of an equivalent least-squares local polynomial fitting problem in which each unique t_k is weighted by $n_k w_k(t_k - t)$ with the kth observation replaced by the average $\bar{y}(t_k)$, that is,

$$\bar{\mathcal{J}}_t = \sum_{|t_k-t|\le h_t} \big[\bar{y}(t_k) - \mathbf{u}^T(t_k-t)\mathbf{c}\big]^2 n_k w(t_k-t) = \min \qquad (23.20.3)$$

Internally, the function `locpol` calls the function `avobs`, which takes in the raw data `tobs,yobs` and determines the unique observation times `ta`, averaged observations `ya`, and their multiplicities `na`:

```
[ta,ya,na] = avobs(tobs,yobs);      % average repeated observations
```

For example, if

```
tobs = [ 1   1   1   3   3   5   5   3   4   7   9   9   9   9];
yobs = [20  22  21  11  12  13  15  19  21  25  28  29  31  32];
```

the function first sorts the ts in increasing order,

```
tobs = [ 1   1   1   3   3   3   4   5   5   7   9   9   9   9];
yobs = [20  21  22  11  12  19  21  13  15  25  28  29  31  32];
```

and then returns the output,

```
ta = [ 1   3   4   5   7   9];
ya = [21  14  21  14  25  30];
na = [ 3   3   1   2   1   4];
```

23.21 Loess Smoothing

Loess, which is a shorthand for *lo*cal regr*ess*ion, is a method proposed by Cleveland [776] for handling data with outliers. A version of it was discussed in Sec. 23.15. The method carries out a local polynomial regression using a nearest-neighbor bandwidth and the tricube window function, and then uses the resulting error residuals to iteratively readjust the window weights giving less importance to the outliers.

The method is described as follows [776]. Given the N-dimensional vectors of observation times and observations $t_{\text{obs}}, y_{\text{obs}}$, the nearest-neighbor bandwidth parameter α, and the polynomial order d, the method begins by performing a preliminary fit to all the observation times. For example, in the notation of the `locband` and `locpol` functions:

$$\begin{array}{ll} h = \texttt{locband}(t_{\text{obs}}, t_{\text{obs}}, \alpha, 0); & \text{(find local bandwidths at } t_{\text{obs}}) \\ \hat{x} = \texttt{locpol}(t_{\text{obs}}, y_{\text{obs}}, t_{\text{obs}}, h, d, 1); & \text{(perform fit at all } t_{\text{obs}}) \end{array} \qquad (23.21.1)$$

where the last argument of `locpol` designates the use of the tricube window. From the resulting N-dimensional signal $\hat{x}_k$, $k = 0, 1, \dots, N-1$, we calculate the corresponding error residuals e_k and use their median to calculate "robustness" weights r_k:

$$\begin{aligned} e_k &= y_k - \hat{x}_k\,, \quad k = 0, 1, \dots, N-1 \\ \mu &= \operatorname*{median}_{0\le k\le N-1}\big(|e_k|\big) \\ r_k &= W\left(\frac{e_k}{6\mu}\right) \end{aligned} \qquad (23.21.2)$$

where $W(u)$ is the bisquare function defined in (23.16.5). The local polynomial fitting is now repeated at all observation points t_{obs}, but instead of using the weights $w(t_k - t_{\text{obs}})$ for the kth observation's contribution to the fit, one uses the modified weights $r_k w(t_k - t_{\text{obs}})$. The new residuals are then computed as in (23.21.2) and the process is repeated a few more times or until convergence (i.e., until the estimated signal $\hat{x}_k$ no longer changes).

After the final iteration resulting in the final values of the r_ks, one can carry out the fit at any other time point t, but again using weights $r_k w(t_k - t)$ for the contribution of the kth observation, that is, the weight matrix W_t in Eq. (23.16.7) is replaced by

$$W_t = \operatorname{diag}\big([\cdots, r_k w(t_k - t), \cdots]\big)$$

The MATLAB function `loess` implements these steps:

```
[xhat,C] = loess(tobs,yobs,t,alpha,d,Nit);      % Loess smoothing
```

where t are the final fitting times and `xhat` and `C` have the same meaning as in `locpol`. This function is similar in spirit to the robust local polynomial filtering function `rlpfilt` that was discussed in Sec. 23.15.

Example 23.21.1: Fig. 23.21.1 shows the same example as that of Fig. 23.15.3, with nearest-neighbor bandwidth parameter $\alpha = 0.4$ and an order-2 polynomial. The graphs show the results of $N_{\text{it}} = 0, 2, 4, 6$ iterations—the first one corresponding to ordinary fitting with no robustness iterations. The MATLAB code for the top two graphs was:

```
t = (0:50); x0 = (1 - cos(2*pi*t/50))/2;       % desired signal
seed=2005; randn('state',seed);
y = x0 + 0.1 * randn(size(x0));                % noisy signal

m = [-1 0 1 3];                                % outlier indices
n0=25; y(n0+m+1) = 0;                          % outlier values
n1=10; y(n1+m+1) = 1;

alpha=0.4; d=2;                          % bandwidth parameter and polynomial order

Nit=0; x = loess(t,y,t,alpha,d,Nit);                % loess fit at t
figure; plot(t,x0,'--', t,y,'.', t,x,'-');          % left graph

Nit=2; x = loess(t,y,t,alpha,d,Nit);                % loess fit at t
figure; plot(t,x0,'--', t,y,'.', t,x,'-');          % right graph
```

The loess fit was performed at all t. We observe how successive iterations gradually diminish the distorting influence of the outliers. □

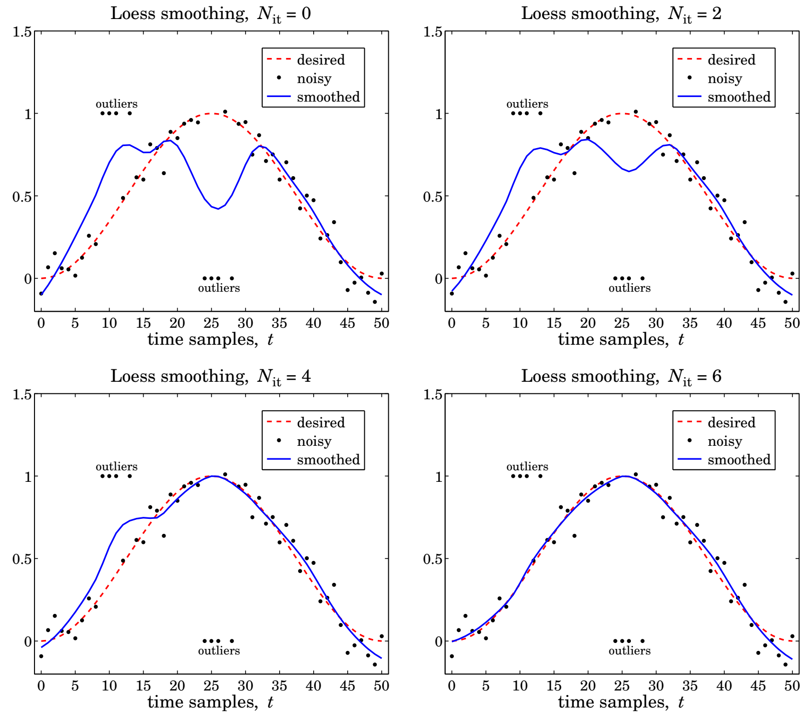


Fig. 23.21.1 Loess smoothing with $d = 2$, $\alpha = 0.4$, and different iterations.

23.22 Problems

23.1 Using binomial identities, prove the equivalence of the three expressions in Eq. (23.14.14) for the maximally-flat filters. Then, show Eq. (23.14.15) and determine the proportionality constants indicated as (const.).

23.2 Prove the matrix inversion lemma identity (23.17.8). Using this identity, show that

$$H_{ii} = \frac{H_{ii}^-}{1 + H_{ii}^-}\,, \quad \text{where}\;\; H_{ii}^- = w_0 \mathbf{u}_0^T F_i^- \mathbf{u}_0\,, \quad F_i^- = (S_i^T W_i S_i)_-$$

then, argue that $0 \le H_{ii} \le 1$.

24

Exponential Moving Average Filters

24.1 Mean Tracking

In this chapter,[†] we discuss *exponential moving average* (EMA) filters, also known as exponentially-weighted moving average (EWMA) filters. They are simple, effective, recursive smoothing filters that can be applied to real-time data. By contrast, the local polynomial modeling approach is typically applied off-line to a block of signal samples that have already been collected.

The EMA filter is used routinely to track stock market data and in forecasting applications such as inventory control where, despite its simplicity, it is highly competitive with other more sophisticated forecasting methods [816–863].

We have already encountered it in Sec. 15.2 and compared it to the plain FIR averager. Here, we view it as a special case of a weighted local polynomial smoothing problem using a causal window and exponential weights, and discuss some of its generalizations. Both the EMA smoother and the FIR averager are applied to data that are assumed to have the typical form:

$$y_n = a_n + v_n \tag{24.1.1}$$

where a_n is a low-frequency trend component, representing an average or estimate of the *local level* of the signal, and v_n a random, zero-mean, broadband component, such as white noise. If a_n is a deterministic signal, then by taking expectations of both sides we see that a_n represents the mean value of y_n, that is, $a_n = E[y_n]$. If y_n is stationary, then a_n is a constant, independent of n.

The output of either the FIR or the EMA filter tracks the signal a_n. To see how such filters arise in the context of estimating the mean level of a signal, consider first the stationary case. The mean $m = E[y_n]$ minimizes the following variance performance index:

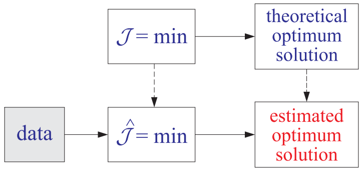
$$\mathcal{J} = E\big[(y_n - a)^2\big] = \min \quad \Rightarrow \quad a_{\text{opt}} = m = E[y_n] \tag{24.1.2}$$

with minimized value $\mathcal{J}_{\min} = \sigma_y^2$. This result is obtained by setting the gradient with respect to a to zero:

$$\frac{\partial a}{\partial \mathcal{J}} = -2E[y_n - a] = 0 \tag{24.1.3}$$

[†] adapted from the author's book on *Applied Optimum Signal Processing* [45]

In general, given a theoretical performance index $\mathcal{J}$, one must replace it in practice by an experimental one, say $\hat{\mathcal{J}}$, expressible in terms of the actual available data. The minimization of $\hat{\mathcal{J}}$ provides then estimates of the parameters or signals to be estimated.



Depending on the index $\hat{\mathcal{J}}$, the estimates may be calculated in a block processing manner using an entire block of data, or, on a sample-by-sample basis with the estimate being updated in real time in response to each new data sample. All adaptive filtering algorithms follow the latter approach.

We illustrate these ideas with the help of the simple performance index (24.1.2). Four possible practical definitions for $\hat{\mathcal{J}}$ that imitate (24.1.2) are:

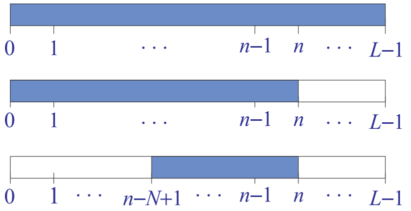
$$\hat{\mathcal{J}} = \sum_{n=0}^{L-1} (y_n - \hat{a})^2 = \min \tag{24.1.4a}$$

$$\hat{\mathcal{J}} = \sum_{k=0}^{n} (y_k - \hat{a})^2 = \min \tag{24.1.4b}$$

$$\hat{\mathcal{J}} = \sum_{k=n-N+1}^{n} (y_k - \hat{a})^2 = \min \tag{24.1.4c}$$

$$\hat{\mathcal{J}} = \sum_{k=0}^{n} \lambda^{n-k} (y_k - \hat{a})^2 = \min \tag{24.1.4d}$$

The first assumes a length-L block of data $[y_0, y_1, \dots, y_{L-1}]$. The last three are suitable for real-time implementations, where n denotes the current time. The second and fourth use the first $n+1$ data $[y_0, y_1, \dots, y_n]$, while the third uses a length-N sliding window $[y_{n-N+1}, \dots, y_{n-1}, y_n]$. The third choice leads to the FIR averager, also known as the *simple moving average* (SMA), and the fourth, to the exponential smoother, or, *exponential moving average* (EMA), where we require that the exponential "forgetting factor" λ be in the range $0 < \lambda < 1$. These time ranges are depicted below.



In order for the $\hat{\mathcal{J}}$s to be unbiased estimates of $\mathcal{J}$, the above expressions should have been divided by the sum of their respective weights, namely, the quantities L, $(n+1)$, N, and $(1 + \lambda + \cdots + \lambda^n)$, respectively. However, such factors do not affect the minimization solutions, which are easily found to be:

$$\hat{a} = \frac{y_0 + y_1 + \cdots + y_{L-1}}{L} \tag{24.1.5a}$$

$$\hat{a}_n = \frac{y_0 + y_1 + \cdots + y_n}{n+1} \tag{24.1.5b}$$

$$\hat{a}_n = \frac{y_n + y_{n-1} + \cdots + y_{n-N+1}}{N} \tag{24.1.5c}$$

$$\hat{a}_n = \frac{y_n + \lambda y_{n-1} + \lambda^2 y_{n-2} + \cdots \lambda^n y_0}{1 + \lambda + \lambda^2 + \cdots + \lambda^n} \tag{24.1.5d}$$

We have tacked on a subscript n to the last three to emphasize their dependence of their performance index on the current time instant n. Eqs. (24.1.4c) and (24.1.5c) tentatively assume that $n \geq N-1$; for $0 \leq n < N-1$, one should use the running average (24.1.4b) and (24.1.5b). Initialization issues are discussed further in Sections 24.6 and 25.6.

All four estimates are *unbiased* estimators of the true mean m. Their quality as estimators can be judged by their variances, which are (assuming that $y_n - m$ are mutually independent):

$$\sigma_{\hat{a}}^2 = E\big[(\hat{a} - m)^2\big] = \frac{\sigma_y^2}{L} \tag{24.1.6a}$$

$$\sigma_{\hat{a}_n}^2 = E\big[(\hat{a}_n - m)^2\big] = \frac{\sigma_y^2}{n+1} \tag{24.1.6b}$$

$$\sigma_{\hat{a}_n}^2 = E\big[(\hat{a}_n - m)^2\big] = \frac{\sigma_y^2}{N} \tag{24.1.6c}$$

$$\sigma_{\hat{a}_n}^2 = E\big[(\hat{a}_n - m)^2\big] = \sigma_y^2 \frac{1-\lambda}{1+\lambda} \cdot \frac{1+\lambda^{n+1}}{1-\lambda^{n+1}} \tag{24.1.6d}$$

The first two, corresponding to ordinary sample averaging, are asymptotically consistent estimators having variances that tend to zero as $L \to \infty$ or $n \to \infty$. The last two are not consistent. However, their variances can be made as small as desired by proper choice of the parameters N or λ.

The exponential smoothing filter may also be derived from a different point of view. The estimates (24.1.5) are the *exact* least-squares solutions of the indices (24.1.4). An alternative to using the exact solutions is to derive an LMS (least-mean-square) type of adaptive algorithm which minimizes the performance index iteratively using a steepest-descent algorithm that replaces the theoretical gradient (24.1.3) by an "instantaneous" one in which the expectation instruction is ignored:

$$\frac{\partial a}{\partial \mathcal{J}} = -2E[y_n - a] \quad \longrightarrow \quad \frac{\widehat{\partial a}}{\partial \mathcal{J}} = -2\big[y_n - \hat{a}_{n-1}\big] \tag{24.1.7}$$

The LMS algorithm then updates the previous estimate by adding a correction in the direction of the negative gradient using a small positive adaptation parameter μ:

$$\Delta a = -\mu \frac{\widehat{\partial a}}{\partial \mathcal{J}}, \qquad \hat{a}_n = \hat{a}_{n-1} + \Delta a \tag{24.1.8}$$

The resulting difference equation is identical to that of the steady-state exponential smoother (see Eq. (24.1.11) below),

$$\hat{a}_n = \hat{a}_{n-1} + 2\mu(y_n - \hat{a}_{n-1})$$

In adaptive filtering applications, the use of the exponentially discounted type of performance index (24.1.4d) leads to the so-called recursive least-squares (RLS) adaptive filters, which are in general different from the LMS adaptive filters. They happened to coincide in this particular example because of the simplicity of the problem.

The sample mean estimators (24.1.5a) and (24.1.5b) are geared to stationary data, whereas (24.1.5c) and (24.1.5d) can track nonstationary changes in the statistics of y_n. If y_n is nonstationary, then its mean $a_n = E[y_n]$ would be varying with n and a good estimate should be able to track it well and efficiently. To see the problems that arise in using the sample mean estimators in the nonstationary case, let us cast Eqs. (24.1.5b) and (24.1.5d) in recursive form. Both can be written as follows:

$$\hat{a}_n = (1-\alpha_n)\hat{a}_{n-1} + \alpha_n y_n = \hat{a}_{n-1} + \alpha_n(y_n - \hat{a}_{n-1}) \qquad (24.1.9)$$

where the gain parameter α_n is given by

$$\alpha_n = \frac{1}{n+1}\,, \qquad \alpha_n = \frac{1}{1+\lambda+\cdots+\lambda^n} = \frac{1-\lambda}{1-\lambda^{n+1}} \qquad (24.1.10)$$

for (24.1.5b) and (24.1.5d), respectively. The last side of Eq. (24.1.9) is written in a so-called "predictor/corrector" Kalman filter form, where the first term $\hat{a}_{n-1}$ is a tentative prediction of $\hat{a}_n$ and the second term is the correction obtained by multiplying the "prediction error" $(y_n - \hat{a}_{n-1})$ by a positive gain factor α_n. This term always corrects in the right direction, that is, if $\hat{a}_{n-1}$ overestimates/underestimates y_n then the error tends to be negative/positive reducing/increasing the value of $\hat{a}_{n-1}$.

There is a dramatic difference between the two estimators. For the sample mean, the gain $\alpha_n = 1/(n+1)$ tends to zero rapidly with increasing n. For stationary data, the estimate $\hat{a}_n$ will converge quickly to the true mean. Once n is fairly large, the correction term becomes essentially unimportant because the gain is so small. If after converging to the true mean the statistics of y_n were to suddenly change with a new value of the mean, the sample-mean estimator $\hat{a}_n$ would have a very hard time responding to such a change and converging to the new value because the new changes are communicated only through the already very small correction term.

On the other hand, for the exponential smoother case (24.1.5d), the gain tends to a constant for large n, that is, $\alpha_n \to \alpha = 1-\lambda$. Therefore, the correction term remains finite and can communicate the changes in the statistics. The price one pays for that is that the estimator is not consistent. Asymptotically, the estimator (24.1.5d) becomes the ordinary exponential smoothing filter described by the difference equation,

$$\hat{a}_n = \lambda\hat{a}_{n-1} + \alpha y_n = \hat{a}_{n-1} + \alpha(y_n - \hat{a}_{n-1}) \qquad (24.1.11)$$

Its transfer function and asymptotic variance are:

$$H(z) = \frac{\alpha}{1-\lambda z^{-1}}\,, \qquad \sigma^2_{\hat{a}_n} = E\big[(\hat{a}_n - m)^2\big] = \sigma_y^2\,\frac{1-\lambda}{1+\lambda} \qquad (24.1.12)$$

The quantity $\sigma^2_{\hat{a}_n}/\sigma^2_y$ is the NRR of this filter. The differences in the behavior of the sample-mean and exponential smoother can be understood by inspecting the corresponding performance indices, which may be written in an expanded form:

$$\begin{aligned}
\hat{\mathcal{J}} &= (y_n - \hat{a})^2 + (y_{n-1} - \hat{a})^2 + (y_{n-2} - \hat{a})^2 + \cdots + (y_0 - \hat{a})^2 \\
\hat{\mathcal{J}} &= (y_n - \hat{a})^2 + \lambda(y_{n-1} - \hat{a})^2 + \lambda^2(y_{n-2} - \hat{a})^2 + \cdots + \lambda^n(y_0 - \hat{a})^2
\end{aligned} \qquad (24.1.13)$$

The first index weighs all terms equally, as it should for stationary data. The second index emphasizes the terms arising from the most recent observation y_n and exponentially forgets, or discounts, the earlier observations and thus can respond more quickly to new changes. Even though the second index appears to have an ever increasing number of terms, in reality, the effective number of terms that are significant is finite and can be estimated by the formula:

$$\bar{n} = \frac{\displaystyle\sum_{n=0}^{\infty} n\lambda^n}{\displaystyle\sum_{n=0}^{\infty}\lambda^n} = \frac{\lambda}{1-\lambda} \qquad (24.1.14)$$

This expression is only a guideline and other possibilities exist. For example, one can define $\bar{n}$ to be the effective time constant of the filter:

$$\lambda^{\bar{n}} = \epsilon \quad\Rightarrow\quad \bar{n} = \frac{\ln\epsilon}{\ln\lambda} \simeq \frac{\ln(\epsilon^{-1})}{1-\lambda}\,, \quad \text{for } \lambda \lesssim 1 \qquad (24.1.15)$$

where ϵ is a small user-specified parameter such as $\epsilon = 0.01$. The sliding window estimator (24.1.5c) is recognized as a length-N FIR averaging filter of the type we considered in Sec. 15.5. It also can track a nonstationary signal at the expense of not being a consistent estimator. Requiring that it achieve the same variance as the exponential smoother gives the conditions:

$$\boxed{\frac{1}{N}\sigma_y^2 = \frac{1-\lambda}{1+\lambda}\sigma_y^2} \quad\Rightarrow\quad \lambda = \frac{N-1}{N+1} \quad\Rightarrow\quad \alpha = 1-\lambda = \frac{2}{N+1} \qquad (24.1.16)$$

Such conditions are routinely used to set the parameters of FIR and exponential smoothing filters in inventory control applications and in tracking stock market data. A similar weighted average as in Eq. (24.1.14) can be defined for any filter by:

$$\boxed{\bar{n} = \frac{\displaystyle\sum_n n h_n}{\displaystyle\sum_n h_n}} \qquad \text{(effective filter lag)} \qquad (24.1.17)$$

where h_n is the filter's impulse response. Eq. (24.1.17) may also be expressed in terms of the filter's transfer function $H(z) = \sum_n h_n z^{-n}$ and its derivative $H'(z) = dH(z)/dz$ evaluated at DC, that is, at $z = 1$:

$$\bar{n} = -\left.\frac{H'(z)}{H(z)}\right|_{z=1} \qquad \text{(effective filter lag)} \qquad (24.1.18)$$

Alternatively, $\bar{n}$ is recognized as the filter's *group delay* at DC, that is, given the frequency response $H(\omega)=\sum_n h_n e^{-j\omega n} = |H(\omega)|e^{j\arg H(\omega)}$, we have (Problem 24.1):

$$\bar{n} = -\frac{d}{d\omega}\arg H(\omega)\Big|_{\omega=0} \qquad \text{(group delay at DC)} \tag{24.1.19}$$

The exponential smoother is a special case of (24.1.17) with $h_n = \alpha\lambda^n u(n)$, where $u(n)$ is the unit-step function. We may apply this definition also to the FIR averager filter that has $h_n = 1/N$, for $n = 0, 1, \dots, N-1$,

$$\bar{n} = \frac{1}{N}\sum_{n=0}^{N-1} n = \frac{N-1}{2}$$

The FIR averager can be mapped into an "equivalent" exponential smoother by equating the $\bar{n}$ lags of the two filters, that is,

$$\boxed{\bar{n} = \frac{N-1}{2} = \frac{\lambda}{1-\lambda}} \tag{24.1.20}$$

This condition is exactly equivalent to condition (24.1.16) arising from matching the NRRs of the two filters. The two equations,

$$E\big[(\hat{a}_n - m)^2\big] = \frac{1-\lambda}{1+\lambda}\sigma_y^2 = \frac{1}{N}\sigma_y^2\,, \qquad \bar{n} = \frac{\lambda}{1-\lambda} = \frac{N-1}{2} \tag{24.1.21}$$

capture the main tradeoff between variance and speed in using an exponential smoother or an equivalent FIR averager, that is, the closer λ is to unity or the larger the N, the smaller the variance and the better the estimate, but the longer the transients and the slower the speed of response.

We summarize the difference equations for the exact exponential smoother (24.1.5d) and the steady-state one (24.1.11),

$$\begin{aligned}
\hat{a}_n &= \frac{\lambda-\lambda^{n+1}}{1-\lambda^{n+1}}\,\hat{a}_{n-1} + \frac{\alpha}{1-\lambda^{n+1}}\,y_n = \hat{a}_{n-1} + \frac{\alpha}{1-\lambda^{n+1}}\,(y_n - \hat{a}_{n-1})\\
\hat{a}_n &= \lambda\hat{a}_{n-1} + \alpha\,y_n = \hat{a}_{n-1} + \alpha\,(y_n - \hat{a}_{n-1})
\end{aligned} \tag{24.1.22}$$

Clearly, the second is obtained in the large-n limit of the first, but in practice the steady one is often used from the start at $n = 0$ because of its simplicity.

To start the recursions at $n = 0$, one needs to specify the initial value $\hat{a}_{-1}$. For the exact smoother, $\hat{a}_{-1}$ can have an arbitrary value because its coefficient vanishes at $n = 0$. This gives for the first smoothed value $\hat{a}_0 = 0\cdot\hat{a}_{-1} + 1\cdot y_0 = y_0$. For the steady smoother it would make sense to also require that $\hat{a}_0 = y_0$, which would imply that $\hat{a}_{-1} = y_0$ because then

$$\hat{a}_0 = \lambda\hat{a}_{-1} + \alpha y_0 = \lambda y_0 + \alpha y_0 = y_0$$

There are other reasonable ways of choosing $\hat{a}_{-1}$, for example one could take it to be the average of a few initial values of y_n. The convolutional solution of the steady smoother with arbitrary nonzero initial conditions is obtained by convolving the filter's impulse response $\alpha\lambda^n u(n)$ with the causal input y_n plus adding a transient term arising from the initial value:

$$\hat{a}_n = \alpha\sum_{k=0}^{n}\lambda^{n-k}y_k + \lambda^{n+1}\hat{a}_{-1} \tag{24.1.23}$$

The influence of the initial value disappears exponentially.

Example 24.1.1: Fig. 24.1.1 illustrates the ability of the sample mean and the exponential smoother to track a sudden level change.

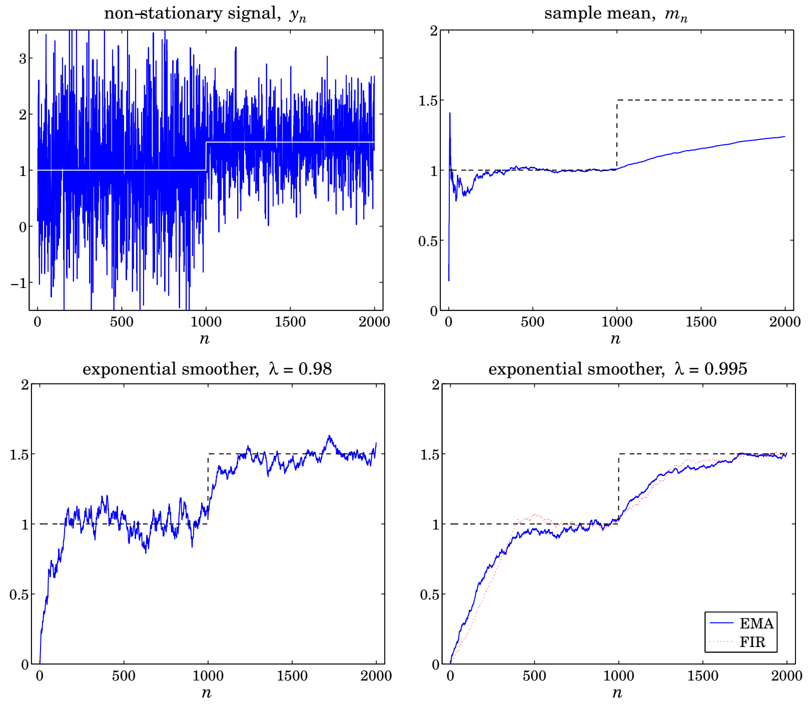


Fig. 24.1.1 Mean tracking with sample mean, exponential smoother, and FIR averager.

The first 1000 samples of the signal y_n depicted on the upper-left graph are independent gaussian samples of mean and variance $m_1 = 1$, $\sigma_1 = 1$. The last 1000 samples are gaussian samples with $m_2 = 1.5$ and $\sigma_2 = 0.5$.

The upper-right graph shows the sample mean computed recursively using (24.1.9) with $\alpha_n = 1/(n+1)$ and initialized at $\hat{a}_{-1} = 0$ (although the initial value does not matter since $\alpha_0 = 1$). We observe that the sample mean converges very fast to the first value of $m_1 = 1$, with its fluctuations becoming smaller and smaller because of its decreasing variance (24.1.6b). But it is extremely slow responding to the sudden change in the mean.

The bottom two graphs show the steady-state exponential smoother initialized at $\hat{a}_{-1} = 0$ with the two values of the forgetting factor $\lambda = 0.98$ and $\lambda = 0.995$. For the smaller λ

the convergence is quick both at the beginning and after the change, but the fluctuations quantified by (24.1.21) remain finite and do not improve even after convergence. For the larger λ, the fluctuations are smaller, but the learning time constant is longer. In the bottom-right graph, we have also added the equivalent FIR averager with N related to λ by (24.1.16), which gives $N = 399$. Its learning speed and fluctuations are comparable to those of the exponential smoother. □

Example 24.1.2: Fig. 24.1.2 shows the daily Dow-Jones Industrial Average (DJIA) from Oct. 1, 2007 to Dec. 31, 2009. In the left graph an exponential smoothing filter is used with $\lambda = 0.9$. In the right graph, an FIR averager with an equivalent length of $N = (1+\lambda)/(1-\lambda) = 19$ is used. The data were obtained from `http://finance.yahoo.com`.

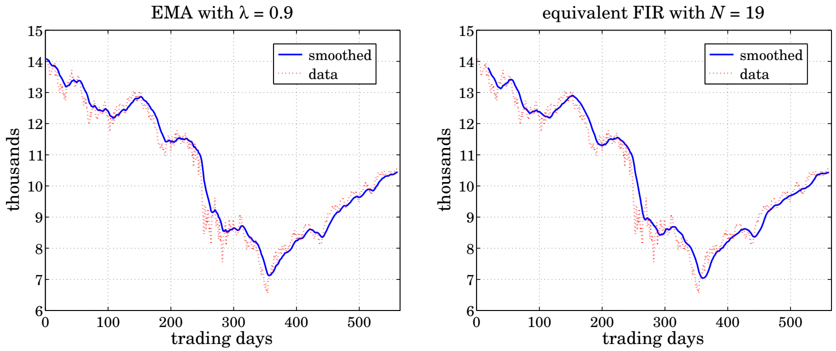


Fig. 24.1.2 Dow-Jones industrial average from 10-Oct-2007 to 31-Dec-2009.

The following code fragment generates the two graphs:

```
Y = loadfile('dow-oct07-dec09.dat');          % data file in OSP toolbox
y = Y(:,4)/1000;                              % extract closing prices
n = (0:length(y)-1);

la = 0.9; al = 1-la;
s0 = la*y(1);                                 % s0 is the initial state
m = filter(al, [1,-la], y, s0);               % filter with initial state
% m = stema(y,0,la, y(1));                    % equivalent calculation

figure; plot(n,m,'-', n,y,':');

N = round((1+la)/(1-la));
h = ones(N,1)/N;                              % FIR averager
x = filter(h,1,y);

figure; plot(n(N:end),x(N:end),'-', n,y,':');  % discard first N−1 outputs
```

The initial value was set such that to get $\hat{a}_0 = y_0$ for the EMA. The built-in function `filter` allows one to specify the initial state. Because `filter` uses the transposed realization, in order to have $\hat{a}_0 = y_0$, the initial state must be chosen as $s_{\text{in}} = \lambda y_0$. This follows from the sample processing algorithm of the transposed realization for the EMA filter (24.1.12), which reads as follows where s is the state:

```
for each input sample $y$ do:
    $\hat{a} = s + \alpha y$
    $s = \lambda\hat{a}$
```

or

$$\boxed{\begin{aligned}\hat{a}_n &= s_n + \alpha y_n\\ s_{n+1} &= \lambda\hat{a}_n\end{aligned}}$$

Thus, in order for the first pass to give $\hat{a}_0 = y_0$, the initial state must be such that $s_0 = \hat{a}_0 - \alpha y_0 = \lambda y_0$. The FIR averager was run with zero initial conditions and therefore, the first $N-1$ outputs were discarded as transients. After $n \geq N$, the EMA and the FIR outputs are comparable since they have the same $\bar{n}$. □

Example 24.1.3: It is evident by inspecting the graphs of the previous example that both the EMA and the FIR filter outputs are lagging behind the data signal. To see this lag more clearly, consider a noiseless signal consisting of three straight-line segments defined by,

$$s_n = \begin{cases} 20 + 0.8n, & 0 \leq n < 75 \\ 80 - 0.3(n-75), & 75 \leq n < 225 \\ 35 + 0.4(n-225), & 225 \leq n \leq 300 \end{cases}$$

Fig. 24.1.3 shows the corresponding output from an EMA with $\lambda = 0.9$ and an equivalent FIR averager with $N = 19$ as in the previous example. The dashed line is the signal s_n and the solid lines, the corresponding filter outputs.

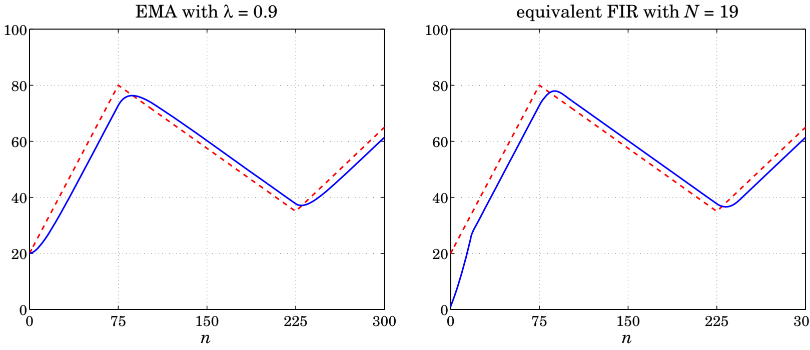


Fig. 24.1.3 Lag introduced by EMA and FIR averager filters.

The EMA was run with initial value $\hat{a}_{-1} = s_0 = 20$. The FIR filter was run with zero initial conditions, and therefore, its first $N-1$ outputs are transients. The amount of delay introduced by the filters is exactly equal to the quantity $\bar{n}$ of Eq. (24.1.20). □

The delay $\bar{n}$ is a consequence of the causality of the filters. Symmetric non-causal filters, such as the LPSM or LPRS filters, do not introduce a delay, that is, $\bar{n} = 0$.

To see how such a delay arises, consider an arbitrary causal filter h_n and a causal input that is a linear function of time, $x_n = a + bn$, for $n \geq 0$. The corresponding convolutional output will be:

$$y_n = \sum_{k=0}^{n} h_k x_{n-k} = \sum_{k=0}^{n} h_k\big[a + b(n-k)\big] = (a+bn)\sum_{k=0}^{n} h_k - b\sum_{k=0}^{n} k h_k$$

For large n, we may replace the upper limit of the summations by $k = \infty$,

$$y_n = (a+bn)\sum_{k=0}^{\infty} h_k - b\sum_{k=0}^{\infty} kh_k = (a+bn)\sum_{k=0}^{\infty} h_k - b\bar{n}\sum_{k=0}^{\infty} h_k = \big[a+b(n-\bar{n})\big]\sum_{k=0}^{\infty} h_k$$

where we used the definition (24.1.17) for $\bar{n}$. For filters that have unity gain at DC, the sum of the filter coefficients is unity, and we obtain,

$$y_n = a + b(n-\bar{n}) = x_{n-\bar{n}} \tag{24.1.24}$$

Such delays are of concern in a number of applications, such as the real-time monitoring of financial data. For FIR filters, the problem of designing noise reducing filters with a prescribed amount of delay $\bar{n}$ has already been discussed in Sec. 23.9. However, we discuss it a bit further in Sec. 24.10 and 25.2 emphasizing its use in stock market trading. The delay $\bar{n}$ can also be controlled by the use of higher-order exponential smoothing discussed in Sec. 24.5.

24.2 Forecasting and State-Space Models

We make a few remarks on the use of the first-order exponential smoother as a forecasting tool. As we already mentioned, the quantity $\hat{a}_{n-1}$ can be viewed as a prediction of y_n based on the past observations $\{y_0, y_1, \dots, y_{n-1}\}$. To emphasize this interpretation, let us denote it by $\hat{y}_{n/n-1} = \hat{a}_{n-1}$, and the corresponding prediction or forecast error by $e_{n/n-1} = y_n - \hat{y}_{n/n-1}$. Then, the steady exponential smoother can be written as,

$$\hat{y}_{n+1/n} = \hat{y}_{n/n-1} + \alpha e_{n/n-1} = \hat{y}_{n/n-1} + \alpha(y_n - \hat{y}_{n/n-1}) \tag{24.2.1}$$

If the prediction is to be optimal, then the prediction error $e_{n/n-1}$ must be a white noise signal, called the *innovations* of the sequence y_n and denoted by $\varepsilon_n = e_{n/n-1}$. It represents that part of y_n that cannot be predicted from its past. This interpretation implies a certain innovations signal model for y_n. We may derive it by working with z-transforms. In the z-domain, Eq. (24.2.1) reads,

$$z\hat{Y}(z) = \hat{Y}(z) + \alpha E(z) = \hat{Y}(z) + \alpha\big(Y(z) - \hat{Y}(z)\big) = \lambda\hat{Y}(z) + \alpha Y(z) \tag{24.2.2}$$

Therefore, the transfer functions from $Y(z)$ to $\hat{Y}(z)$ and from $Y(z)$ to $E(z)$ are,

$$\hat{Y}(z) = \left(\frac{\alpha z^{-1}}{1-\lambda z^{-1}}\right)Y(z)\,, \quad E(z) = \left(\frac{1-z^{-1}}{1-\lambda z^{-1}}\right)Y(z) \tag{24.2.3}$$

In the time domain, using the notation $\nabla y_n = y_n - y_{n-1}$, we may write the latter as

$$\nabla y_n = \varepsilon_n - \lambda\varepsilon_{n-1} \tag{24.2.4}$$

Thus, y_n is an integrated ARMA process, ARIMA(0,1,1), or more simply an integrated MA process, IMA(1,1). In other words, if y_n is such a process, then the exponential smoother forecast $\hat{y}_{n/n-1}$ is optimal in the mean-square sense [826].

The innovations representation model can also be cast in an ordinary Wiener and Kalman filter form of the type discussed in [45]. The state and measurement equations for such a model are:

$$\begin{aligned} x_{n+1} &= x_n + w_n \\ y_n &= x_n + v_n \end{aligned} \tag{24.2.5}$$

where w_n, v_n are zero-mean white-noise signals that are mutually uncorrelated. This model is referred to as a "constant level" state-space model, and represents a random-walk observed in noise. The optimal prediction estimate $\hat{x}_{n/n-1}$ of the state x_n is equivalent to $\hat{a}_{n-1}$. The equivalence between EMA and this model results in the following relationship between the parameters α and $q = \sigma_w^2/\sigma_v^2$:

$$q = \frac{\alpha^2}{1-\alpha} \quad \Rightarrow \quad \alpha = \frac{\sqrt{q^2+4q}-q}{2} \tag{24.2.6}$$

Further discussion of such state-space models may found in [45].

24.3 Higher-Order Polynomial Smoothing Filters

We recall that in fitting a local polynomial of order d to a local block of data $\{y_{n-M}, \dots, y_n, \dots, y_{n+M}\}$, the performance index was

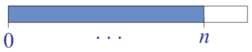
$$\mathcal{J} = \sum_{k=-M}^{M}\big[y_{n+k} - p(k)\big]^2 = \sum_{k=-M}^{M}\big[y_{n+k} - \mathbf{u}_k^T\mathbf{c}\big]^2 = \min$$

where $p(k)$ is a dth degree polynomial, representing the estimate $\hat{y}_{n+k} = p(k)$,

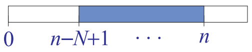
$$p(k) = \mathbf{u}_k^T\mathbf{c} = [1, k, \dots, k^d]\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_d \end{bmatrix} = \sum_{i=0}^{d} c_i k^i$$

and we defined the monomial basis vector $\mathbf{u}_k = [1, k, k^2, \dots, k^d]^T$. The higher-order exponential smoother is obtained by restricting the data range to $\{y_0, y_1, \dots, y_n\}$ and using exponential weights, and similarly, the corresponding FIR version will be restricted to $\{y_{n-N+1}, \dots, y_{n-1}, y_n\}$. The resulting performance indices are then,

$$\mathcal{J}_n = \sum_{k=-n}^{0}\lambda^{-k}\big[y_{n+k} - \mathbf{u}_k^T\mathbf{c}\big]^2 = \min$$



$$\mathcal{J}_n = \sum_{k=-N+1}^{0}\big[y_{n+k} - \mathbf{u}_k^T\mathbf{c}\big]^2 = \min$$



or, replacing the summation index k by $-k$, the performance indices read,

$$\begin{aligned} &\text{(EMA)} \quad \mathcal{J}_n = \sum_{k=0}^{n}\lambda^k\big[y_{n-k} - \mathbf{u}_{-k}^T\mathbf{c}\big]^2 = \min \\ &\text{(FIR)} \quad \mathcal{J}_n = \sum_{k=0}^{N-1}\big[y_{n-k} - \mathbf{u}_{-k}^T\mathbf{c}\big]^2 = \min \end{aligned} \tag{24.3.1}$$

In both cases, we may interpret the quantities $p(\pm\tau)=\mathbf{u}_{\pm\tau}^T\mathbf{c}$ as the estimates $\hat{y}_{n\pm\tau}$. We will denote them by $\hat{y}_{n\pm\tau/n}$ to emphasize their causal dependence only on data up to the current time n. In particular, the quantity $c_0=\mathbf{u}_0^T\mathbf{c}=p(0)$ represents the estimate $\hat{y}_n$, or $\hat{y}_{n/n}$, that is, an estimate of the local level of the signal. Similarly, $c_1=\dot{p}(0)=\dot{\mathbf{u}}_\tau^T\mathbf{c}|_{\tau=0}$ represents the local slope, and $2c_2=\ddot{p}(0)$, the local acceleration. Eqs. (24.1.4d) and (24.1.4c) are special cases of (24.3.1) corresponding to $d=0$.

Both indices in Eq. (24.3.1) can be written in the following compact vectorial form, whose solution we have already obtained in previous chapters:

$$\mathcal{J}=(\mathbf{y}-S\mathbf{c})^TW(\mathbf{y}-S\mathbf{c})=\min \quad\Rightarrow\quad \mathbf{c}=(S^TWS)^{-1}S^TW\mathbf{y} \tag{24.3.2}$$

where the data vector $\mathbf{y}$ is defined as follows in the EMA and FIR cases,

$$\mathbf{y}(n)=\begin{bmatrix} y_n\\ y_{n-1}\\ \vdots\\ y_0\end{bmatrix},\quad \mathbf{y}(n)=\begin{bmatrix} y_n\\ y_{n-1}\\ \vdots\\ y_{n-N+1}\end{bmatrix} \tag{24.3.3}$$

with the polynomial basis matrices S,

$$S_n=\big[\mathbf{u}_0,\mathbf{u}_{-1},\dots,\mathbf{u}_{-n}\big]^T,\quad S_N=\big[\mathbf{u}_0,\mathbf{u}_{-1},\dots,\mathbf{u}_{-N+1}\big]^T=\begin{bmatrix} u_0^T\\ \mathbf{u}_{-1}^T\\ \vdots\\ \mathbf{u}_{-k}^T\\ \vdots\\ \mathbf{u}_{-N+1}^T\end{bmatrix} \tag{24.3.4}$$

with, $\mathbf{u}_{-k}^T=\big[1,(-k),(-k)^2,\dots,(-k)^d\big]$, and weight matrices W in the two cases,

$$W_n=\text{diag}\big([1,\lambda,\dots,\lambda^n]\big),\quad\text{or},\quad W=I_N \tag{24.3.5}$$

The predicted estimates can be written in the filtering form:

$$\boxed{\hat{y}_{n+\tau/n}=\mathbf{u}_\tau^T\mathbf{c}(n)=\mathbf{h}_\tau^T(n)\mathbf{y}(n)} \tag{24.3.6}$$

where in the exponential smoothing case,

$$\boxed{\begin{aligned}\mathbf{c}(n)&=(S_n^TW_nS_n)^{-1}S_n^TW_n\mathbf{y}(n)\\ \mathbf{h}_\tau(n)&=W_nS_n(S_n^TW_nS_n)^{-1}\mathbf{u}_\tau\end{aligned}}\quad\text{(EMA)} \tag{24.3.7}$$

We will see in Eq. (24.5.19) and more explicitly in (24.6.5) that $\mathbf{c}(n)$ can be expressed recursively in the time n. Similarly, for the FIR case, we find:

$$\boxed{\begin{aligned}\mathbf{c}(n)&=(S_N^TS_N)^{-1}S_N^T\mathbf{y}(n)\\ \mathbf{h}_\tau&=S_N(S_N^TS_N)^{-1}\mathbf{u}_\tau\end{aligned}}\quad\text{(FIR)} \tag{24.3.8}$$

We note also that the solution for $\mathbf{c}$ in Eq. (24.3.2) can be viewed as the least-squares solution of the over-determined linear system, $W^{1/2}S\mathbf{c}=W^{1/2}\mathbf{y}$, which is particularly convenient for the numerical solution using MATLAB's backslash operation,

$$\mathbf{c}=(W^{1/2}S)\setminus(W^{1/2}\mathbf{y}) \tag{24.3.9}$$

In fact, this corresponds to an alternative point of view to filtering and is followed in the so-called "linear regression" indicators in financial market trading, as we discuss in Sec. 25.5, where the issue of the initial transients, that is, the evaluation of $\mathbf{c}(n)$ for $0\le n\le N-1$ in the FIR case, is also discussed.

In the EMA case, the basis matrices S_n are full rank for $n\ge d$. For $0\le n<d$, we may restrict the polynomial order d to to $d_n=n$ and thus obtain the first d_n coefficients of the vector $\mathbf{c}(n)$, and set the remaining coefficients to zero. For the commonly used case of $d=1$, this procedure amounts to setting $\mathbf{c}(0)=[y_0,\,0]^T$. Similarly, in the FIR case, we must have $N\ge d+1$ to guarantee the full rank of S_N.

24.4 Linear Trend FIR Filters

The exact solutions of the FIR case have already been found in Sec. 23.9. The $d=1$ and $d=2$ closed-form solutions were given in Eqs. (23.9.10) and (23.9.11). The same expressions are valid for both even and odd N. For example, replacing $M=(N-1)/2$ in (23.9.10), we may express the solution for the $d=1$ case as follows,

$$\boxed{h_\tau(k)=\frac{2(N-1)(2N-1-3k)+6(N-1-2k)\tau}{N(N^2-1)}},\quad k=0,1,\dots,N-1 \tag{24.4.1}$$

A direct derivation of (24.4.1) is as follows. From the definition (24.3.4), we find:

$$\begin{aligned}S_N^TS_N&=\sum_{k=0}^{N-1}\mathbf{u}_{-k}\mathbf{u}_{-k}^T=\sum_{k=0}^{N-1}\begin{bmatrix}1 & -k\\ -k & k^2\end{bmatrix}\\ &=\begin{bmatrix}N & -N(N-1)/2\\ -N(N-1)/2 & N(N-1)(2N-1)/6\end{bmatrix}\\ (S_N^TS_N)^{-1}&=\frac{2}{N(N^2-1)}\begin{bmatrix}(N-1)(2N-1) & 3(N-1)\\ 3(N-1) & 6\end{bmatrix}\end{aligned} \tag{24.4.2}$$

then, from Eq. (24.3.8), because the kth row of S_N is $\mathbf{u}_{-k}^T$, we obtain the kth impulse response coefficient:

$$h_\tau(k)=\mathbf{u}_{-k}^T(S_N^TS_N)^{-1}\mathbf{u}_\tau=\frac{2}{N(N^2-1)}[1,\,-k]\begin{bmatrix}(N-1)(2N-1) & 3(N-1)\\ 3(N-1) & 6\end{bmatrix}\begin{bmatrix}1\\ \tau\end{bmatrix}$$

which leads to Eq. (24.4.1). Thus, we obtain,

$$h_\tau(k)=h_a(k)+h_b(k)\tau,\quad k=0,1,\dots,N-1 \tag{24.4.3}$$

with

$$\boxed{h_a(k)=\frac{2(2N-1-3k)}{N(N+1)}\,,\quad h_b(k)=\frac{6(N-1-2k)}{N(N^2-1)}}\tag{24.4.4}$$

These are the FIR filters that generate estimates of the *local level* and *local slope* of the input signal. Indeed, setting $\mathbf{c}(n)=[a_n,b_n]^T$, where a_n,b_n represent the local level and local slope[†] at time n, we obtain from (24.3.8),

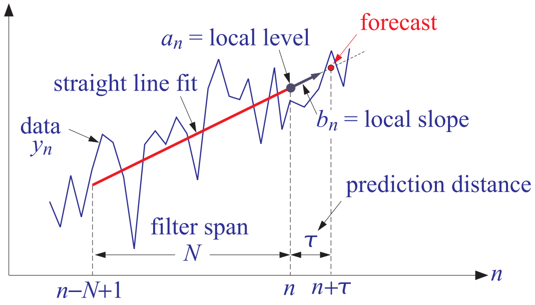
$$\begin{bmatrix}a_n\\ b_n\end{bmatrix}=(S_N^TS_N)^{-1}S_N^T\mathbf{y}(n)=(S_N^TS_N)^{-1}\sum_{k=0}^{N-1}\mathbf{u}_{-k}y_{n-k}$$

which is equivalent, component-wise, to the filtering equations:

$$\begin{aligned}a_n&=\sum_{k=0}^{N-1}h_a(k)y_{n-k}=\text{local level}\\ b_n&=\sum_{k=0}^{N-1}h_b(k)y_{n-k}=\text{local slope}\end{aligned}\tag{24.4.5}$$

Since, $\hat{y}_{n+\tau/n}=a_n+b_n\tau$, it is seen that the local level a_n is equal to $\hat{y}_{n/n}$. Similarly, the sum a_n+b_n is the one-step-ahead forecast $\hat{y}_{n+1/n}$ obtained by extrapolating to time instant $n+1$ by extending the local level a_n along the straight line with slope b_n. This is depicted in the figure below. The sum, a_n+b_n, can be generated directly by the predictive FIR filter, $h_1(k)=h_a(k)+h_b(k)$, obtained by setting $\tau=1$ in (24.4.1):

$$\boxed{h_1(k)=\frac{2(2N-2-3k)}{N(N-1)}\,,\quad k=0,1,\dots,N-1}\quad\text{(predictive FIR filter)}\tag{24.4.6}$$



The filters $h_a(k)$, $h_b(k)$, and $h_1(k)$ find application in the technical analysis of financial markets [864]. Indeed, the filter $h_a(k)$ is equivalent to the so-called *linear regression* indicator, $h_b(k)$ corresponds to the *linear regression slope* indicator, and $h_1(k)$, to the *time series forecast* indicator. We discuss these in more detail, as well as other indicators, in Chap. 25.

[†] a,b are the same as the components c_0,c_1 of the vector $\mathbf{c}$.

More generally, for order d polynomials, it follows from the solution (24.3.8), that the FIR filters $\mathbf{h}_\tau$ satisfy the moment constraints $S_N^T\mathbf{h}_\tau=\mathbf{u}_\tau$, or, component-wise:

$$\sum_{k=0}^{N-1}(-k)^rh_\tau(k)=\tau^r\,,\quad r=0,1,\dots,d\tag{24.4.7}$$

In fact, the solution $\mathbf{h}_\tau=S_N(S_N^TS_N)^{-1}\mathbf{u}_\tau$ is recognized (see [45]) to be the minimum-norm, pseudoinverse, solution of the under-determined system $S_N^T\mathbf{h}=\mathbf{u}_\tau$, that is, it has minimum norm, or, minimum noise-reduction ratio, $\mathcal{R}=\mathbf{h}^T\mathbf{h}=\min$. A direct derivation is as follows. Introduce a $(d+1)\times1$ vector of Lagrange multipliers, $\boldsymbol{\lambda}=[\lambda_0,\lambda_1,\dots,\lambda_d]^T$, and incorporate the constraint into the performance index,

$$\mathcal{J}=\mathbf{h}^T\mathbf{h}+2\boldsymbol{\lambda}^T(\mathbf{u}_\tau-S_N^T\mathbf{h})=\min$$

Then, its minimization leads to,

$$\frac{\partial\mathcal{J}}{\partial\mathbf{h}}=2\mathbf{h}-2S_N\boldsymbol{\lambda}=0\quad\Rightarrow\quad\mathbf{h}=S_N\boldsymbol{\lambda}$$

and, imposing the constraint $S_N^T\mathbf{h}=\mathbf{u}_\tau$ leads to the solutions for $\boldsymbol{\lambda}$ and for $\mathbf{h}$,

$$\mathbf{u}_\tau=S_N^T\mathbf{h}=S_N^TS_N\boldsymbol{\lambda}\quad\Rightarrow\quad\boldsymbol{\lambda}=(S_N^TS_N)^{-1}\mathbf{u}_\tau\quad\Rightarrow\quad\mathbf{h}=S_N\boldsymbol{\lambda}=S_N(S_N^TS_N)^{-1}\mathbf{u}_\tau$$

Returning to Eq. (24.4.3) and setting $\tau=0$, we note that the $d=1$ local-level filter $h_a(k)$ satisfies the explicit constraints:

$$\sum_{k=0}^{N-1}h_a(k)=1\,,\quad\sum_{k=0}^{N-1}kh_a(k)=0\tag{24.4.8}$$

The latter implies that its lag parameter $\bar{n}$ is zero, and therefore, straight-line inputs will appear at the output undelayed (see Example 24.5.1). It has certain limitations as a lowpass filter that we discuss in Sec. 24.10, but its NRR is decreasing with N:

$$\mathcal{R}=\frac{2(2N-1)}{N(N+1)}\tag{24.4.9}$$

A direct consequence of Eq. (24.4.7) is that the filter $h_\tau(k)$ generates the exact predicted value of any polynomial of degree d, that is, for any polynomial $P(x)$ with degree up to d in the variable x, we have the exact convolutional result,

$$\boxed{\sum_{k=0}^{N-1}P(n-k)h_\tau(k)=P(n+\tau)}\,,\quad\text{with }\deg(P)\le d\tag{24.4.10}$$

24.5 Higher-Order Exponential Smoothing

For any value of d, the FIR filters $\mathbf{h}_\tau$ have length N and act on the N-dimensional data vector $\mathbf{y}(n)=[y_n,y_{n-1},\dots,y_{n-N+1}]^T$. By contrast, the exponential smoother weights $\mathbf{h}_\tau(n)$ have an ever increasing length. Therefore, it proves convenient to recast them recursively in time. The resulting recursive algorithm bears a very close similarity to the so-called *exact recursive-least-squares* (RLS) adaptive filters, discussed in [45]. Let us define the quantities,

$$
\begin{aligned}
R_n &= S_n^T W_n S_n = \sum_{k=0}^{n} \lambda^k \mathbf{u}_{-k}\mathbf{u}_{-k}^T = (d+1)\times(d+1) \text{ matrix} \\
\mathbf{r}_n &= S_n^T W_n \mathbf{y}(n) = \sum_{k=0}^{n} \lambda^k \mathbf{u}_{-k} y_{n-k} = (d+1)\times 1 \text{ vector}
\end{aligned}
\tag{24.5.1}
$$

Then, the optimal polynomial coefficients (24.3.7) are:

$$
\mathbf{c}(n) = R_n^{-1}\mathbf{r}_n \tag{24.5.2}
$$

Clearly, the invertibility of R_n requires that $n \ge d$, which we will assume from now on. The sought recursions relate $\mathbf{c}(n)$ to the optimal coefficients $\mathbf{c}(n-1) = R_{n-1}^{-1}\mathbf{r}_{n-1}$ at the previous time instant $n-1$. Therefore, we must actually assume that $n > d$. To proceed, we note that the basis vector $\mathbf{u}_\tau = [1, \tau, \tau^2, \dots, \tau^d]^T$ satisfies the time-propagation property:

$$
\mathbf{u}_{\tau+1} = F\mathbf{u}_\tau \tag{24.5.3}
$$

where F is a $(d+1)\times(d+1)$ unit lower triangular matrix whose ith row consists of the binomial coefficients:

$$
F_{ij} = \binom{i}{j}, \quad 0 \le i \le d, \quad 0 \le j \le i \tag{24.5.4}
$$

This follows from the binomial expansion:

$$
(\tau + 1)^i = \sum_{j=0}^{i} \binom{i}{j} \tau^j
$$

Some examples of the F matrices are for $d = 0, 1, 2$:

$$
F = [1], \quad F = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \quad F = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{24.5.5}
$$

It follows from Eq. (24.5.3) that $\mathbf{u}_\tau = F\mathbf{u}_{\tau-1}$, and inverting $\mathbf{u}_{\tau-1} = F^{-1}\mathbf{u}_\tau$. The inverse matrix $G = F^{-1}$ will also be unit lower triangular with nonzero matrix elements obtained from the binomial expansion of $(\tau - 1)^i$:

$$
G_{ij} = (-1)^{i-j}\binom{i}{j}, \quad 0 \le i \le d, \quad 0 \le j \le i \tag{24.5.6}
$$

For example, we have for $d = 0, 1, 2$,

$$
G = [1], \quad G = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}, \quad G = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 1 & -2 & 1 \end{bmatrix} \tag{24.5.7}
$$

It follows from $\mathbf{u}_{\tau-1} = G\mathbf{u}_\tau$ that $\mathbf{u}_{-k-1} = G\mathbf{u}_{-k}$. This implies the following recursion for R_n:

$$
\begin{aligned}
R_n &= \sum_{k=0}^{n} \lambda^k \mathbf{u}_{-k}\mathbf{u}_{-k}^T = \mathbf{u}_0\mathbf{u}_0^T + \sum_{k=1}^{n} \lambda^k \mathbf{u}_{-k}\mathbf{u}_{-k}^T \\
&= \mathbf{u}_0\mathbf{u}_0^T + \lambda \sum_{k=1}^{n} \lambda^{k-1} \mathbf{u}_{-k}\mathbf{u}_{-k}^T \\
&= \mathbf{u}_0\mathbf{u}_0^T + \lambda \sum_{k=0}^{n-1} \lambda^{k} \mathbf{u}_{-k-1}\mathbf{u}_{-k-1}^T \\
&= \mathbf{u}_0\mathbf{u}_0^T + \lambda G \left(\sum_{k=0}^{n-1} \lambda^{k} \mathbf{u}_{-k}\mathbf{u}_{-k}^T \right) G^T = \mathbf{u}_0\mathbf{u}_0^T + \lambda G R_{n-1} G^T
\end{aligned}
$$

where in the third line we changed summation variables from k to $k-1$, and in the fourth, we used $\mathbf{u}_{-k-1} = G\mathbf{u}_{-k}$. Similarly, we have for $\mathbf{r}_n$,

$$
\begin{aligned}
\mathbf{r}_n &= \sum_{k=0}^{n} \lambda^k \mathbf{u}_{-k} y_{n-k} = \mathbf{u}_0 y_n + \sum_{k=1}^{n} \lambda^k \mathbf{u}_{-k} y_{n-k} \\
&= \mathbf{u}_0 y_n + \lambda \sum_{k=0}^{n-1} \lambda^k \mathbf{u}_{-k-1} y_{n-k-1} \\
&= \mathbf{u}_0 y_n + \lambda G \left(\sum_{k=0}^{n-1} \lambda^k \mathbf{u}_{-k} y_{(n-1)-k} \right) = \mathbf{u}_0 y_n + \lambda G \mathbf{r}_{n-1}
\end{aligned}
$$

Thus, $R_n, \mathbf{r}_n$ satisfy the recursions:

$$
\begin{aligned}
R_n &= \mathbf{u}_0\mathbf{u}_0^T + \lambda G R_{n-1} G^T \\
\mathbf{r}_n &= \mathbf{u}_0 y_n + \lambda G \mathbf{r}_{n-1}
\end{aligned}
\tag{24.5.8}
$$

and they may be initialized to zero, $R_{-1} = 0$ and $\mathbf{r}_{-1} = 0$. Using $\hat{y}_{n+\tau/n} = \mathbf{u}_\tau^T\mathbf{c}(n)$, we may define the smoothed estimates, predictions, and the corresponding errors:

$$
\begin{aligned}
\hat{y}_{n/n} &= \mathbf{u}_0^T\mathbf{c}(n)\,, & e_{n/n} &= y_n - \hat{y}_{n/n} \\
\hat{y}_{n+1/n} &= \mathbf{u}_1^T\mathbf{c}(n) = \mathbf{u}_0^T F^T \mathbf{c}(n)\,, & e_{n+1/n} &= y_{n+1} - \hat{y}_{n+1/n} \\
\hat{y}_{n/n-1} &= \mathbf{u}_1^T\mathbf{c}(n-1) = \mathbf{u}_0^T F^T \mathbf{c}(n-1)\,, & e_{n/n-1} &= y_n - \hat{y}_{n/n-1}
\end{aligned}
\tag{24.5.9}
$$

where we used $\mathbf{u}_1 = F\mathbf{u}_0$. In the language of RLS adaptive filters, we may refer to $\hat{y}_{n/n-1}$ and $\hat{y}_{n/n}$ as the a priori and a posteriori estimates of y_n, respectively. Using the recursions (24.5.8), we may now obtain a recursion for $\mathbf{c}(n)$. Using $\mathbf{c}(n-1) = R_{n-1}^{-1}\mathbf{r}_{n-1}$ and the matrix relationship $GF = I$, we have,

$$
\begin{aligned}
R_n\mathbf{c}(n) &= \mathbf{r}_n = \mathbf{u}_0 y_n + \lambda G \mathbf{r}_{n-1} = \mathbf{u}_0 y_n + \lambda G R_{n-1}\mathbf{c}(n-1) \\
&= \mathbf{u}_0 y_n + \lambda G R_{n-1} G^T F^T \mathbf{c}(n-1) = \mathbf{u}_0 y_n + (R_n - \mathbf{u}_0\mathbf{u}_0^T) F^T \mathbf{c}(n-1) \\
&= R_n F^T \mathbf{c}(n-1) + \mathbf{u}_0\big(y_n - \mathbf{u}_0^T F^T \mathbf{c}(n-1)\big) = R_n F^T \mathbf{c}(n-1) + \mathbf{u}_0(y_n - \hat{y}_{n/n-1}) \\
&= R_n F^T \mathbf{c}(n-1) + \mathbf{u}_0 e_{n/n-1}
\end{aligned}
$$

where in the second line we used $\lambda G R_{n-1} G^T = R_n - \mathbf{u}_0\mathbf{u}_0^T$. Multiplying both sides by R_n^{-1}, we obtain,

$$\mathbf{c}(n) = F^T\mathbf{c}(n-1) + R_n^{-1}\mathbf{u}_0 e_{n/n-1} \tag{24.5.10}$$

Again, in the language of RLS adaptive filters, we define the so-called a posteriori and a priori "Kalman gain" vectors $\mathbf{k}_n$ and $\mathbf{k}_{n/n-1}$,

$$\mathbf{k}_n = R_n^{-1}\mathbf{u}_0\,, \quad \mathbf{k}_{n/n-1} = \lambda^{-1}F^T R_{n-1}^{-1} F\mathbf{u}_0 \tag{24.5.11}$$

and the "likelihood" variables,

$$\nu_n = \mathbf{u}_0^T\mathbf{k}_{n/n-1} = \lambda^{-1}\mathbf{u}_0^T F^T R_{n-1}^{-1} F\mathbf{u}_0 = \lambda^{-1}\mathbf{u}_1^T R_{n-1}^{-1}\mathbf{u}_1\,, \quad \mu_n = \frac{1}{1+\nu_n} \tag{24.5.12}$$

Starting with the recursion $R_n = \mathbf{u}_0\mathbf{u}_0^T + \lambda G R_{n-1} G^T$ and multiplying both sides by R_n^{-1} from the left, then by F^T from the right, then by R_{n-1}^{-1} from the left, and then by F from the right, we may easily derive the equivalent relationship:

$$\lambda^{-1}F^T R_{n-1}^{-1} F = R_n^{-1}\mathbf{u}_0\,\lambda^{-1}\mathbf{u}_0^T F^T R_{n-1}^{-1} F + R_n^{-1} \tag{24.5.13}$$

Multiplying on the right by $\mathbf{u}_0$ and using the definitions (24.5.11), we find

$$\mathbf{k}_{n/n-1} = \mathbf{k}_n\nu_n + \mathbf{k}_n = (1+\nu_n)\mathbf{k}_n\,, \quad \text{or,}$$

$$\mathbf{k}_n = \mu_n\mathbf{k}_{n/n-1} \tag{24.5.14}$$

Substituting this into (24.5.13), we obtain a recursion for the inverse matrix R_n^{-1}, which is effectively a variant of the matrix inversion lemma:

$$R_n^{-1} = \lambda^{-1}F^T R_{n-1}^{-1} F - \mu_n\mathbf{k}_{n/n-1}\mathbf{k}_{n/n-1}^T \tag{24.5.15}$$

This also implies that the parameter μ_n can be expressed as

$$\mu_n = 1 - \mathbf{u}_0^T R_n^{-1}\mathbf{u}_0 = 1 - \mathbf{u}_0^T\mathbf{k}_n \tag{24.5.16}$$

The a priori and a posteriori errors are also proportional to each other. Using (24.5.16), we find,

$$\hat{y}_{n/n} = \mathbf{u}_0^T\mathbf{c}(n) = \mathbf{u}_0^T\big(F^T\mathbf{c}(n-1) + \mathbf{k}_n e_{n/n-1}\big) = \hat{y}_{n/n-1} + (1-\mu_n)e_{n/n-1} = y_n - \mu_n e_{n/n-1}$$

which implies that

$$e_{n/n} = \mu_n e_{n/n-1} \tag{24.5.17}$$

The coefficient updates (24.5.10) may now be expressed as:

$$\mathbf{c}(n) = F^T\mathbf{c}(n-1) + \mathbf{k}_n e_{n/n-1} \tag{24.5.18}$$

We summarize the complete set of computational steps for high-order exponential smoothing. We recall that the invertibility conditions require that we apply the recursions for $n > d$:

$$\boxed{\begin{array}{ll}
1. & \mathbf{k}_{n/n-1} = \lambda^{-1}F^T R_{n-1}^{-1} F\mathbf{u}_0 = \lambda^{-1}F^T R_{n-1}^{-1}\mathbf{u}_1 \\
2. & \nu_n = \mathbf{u}_0^T\mathbf{k}_{n/n-1}\,, \quad \mu_n = 1/(1+\nu_n) \\
3. & \mathbf{k}_n = \mu_n\mathbf{k}_{n/n-1} \\
4. & \hat{y}_{n/n-1} = \mathbf{u}_1^T\mathbf{c}(n-1)\,, \quad e_{n/n-1} = y_n - \hat{y}_{n/n-1} \\
5. & e_{n/n} = \mu_n e_{n/n-1}\,, \quad \hat{y}_n = y_n - e_{n/n} \\
6. & \mathbf{c}(n) = F^T\mathbf{c}(n-1) + \mathbf{k}_n e_{n/n-1} \\
7. & R_n^{-1} = \lambda^{-1}F^T R_{n-1}^{-1} F - \mu_n\mathbf{k}_{n/n-1}\mathbf{k}_{n/n-1}^T
\end{array}} \tag{24.5.19}$$

For $0 \le n \le d$, the fitting may be done with polynomials of varying degree $d_n = n$, and the coefficient estimate computed by explicit matrix inversion, $\mathbf{c}(n) = R_n^{-1}\mathbf{r}_n$. The above computational steps and initialization have been incorporated into the MATLAB function `ema` with usage:

```
C = ema(y,d,lambda);          % exponential moving average - exact version
```

The input `y` is an L-dimensional vector (row or column) of samples to be smoothed, with a total number $L > d$, and `C` is an $L\times(d+1)$ matrix whose nth row is the coefficient vector $\mathbf{c}(n)^T$. Thus, the first column, holds the smoothed estimate, the second column the estimated first derivative, and so on.

To understand the initialization process, consider an input sequence $\{y_0, y_1, y_2, \dots\}$ and the $d = 1$ smoother. At $n = 0$, we use a smoother of order $d_0 = 0$, constructing the quantities $R_0, \mathbf{r}_0$ using the definition (24.5.1):

$$R_0 = [1]\,, \quad \mathbf{r}_0 = [y_0] \quad \Rightarrow \quad \mathbf{c}(0) = R_0^{-1}\mathbf{r}_0 = y_0$$

Next, at $n = 1$ we use a $d_1 = 1$ smoother, and definition (24.5.1) now implies,

$$\left.\begin{array}{l}
R_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + \lambda\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} 1+\lambda & -\lambda \\ -\lambda & \lambda \end{bmatrix} \\[2ex]
\mathbf{r}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} y_1 + \lambda\begin{bmatrix} 1 \\ -1 \end{bmatrix} y_0 = \begin{bmatrix} y_1 + \lambda y_0 \\ -\lambda y_0 \end{bmatrix}
\end{array}\right. \quad \Rightarrow \quad \mathbf{c}(1) = R_1^{-1}\mathbf{r}_1 = \begin{bmatrix} y_1 \\ y_1 - y_0 \end{bmatrix}$$

Starting with $R_1, \mathbf{r}_1$, the recursion (24.5.8) may then be continued for $n \ge d + 1 = 2$. If we had instead $d = 2$, then there is one more initialization step, giving

$$R_2 = \begin{bmatrix} 1+\lambda+\lambda^2 & -\lambda-2\lambda^2 & \lambda+4\lambda^2 \\ -\lambda-2\lambda^2 & \lambda+4\lambda^2 & -\lambda-8\lambda^2 \\ \lambda+4\lambda^2 & -\lambda-8\lambda^2 & \lambda+16\lambda^2 \end{bmatrix}, \quad \mathbf{r}_2 = \begin{bmatrix} y_2 + \lambda y_1 + \lambda^2 y_0 \\ -\lambda y_1 - 2\lambda^2 y_0 \\ \lambda y_1 + 4\lambda^2 y_0 \end{bmatrix}$$

resulting in

$$\mathbf{c}(2) = R_2^{-1}\mathbf{r}_2 = \begin{bmatrix} y_2 \\ 1.5y_2 - 2y_1 + 0.5y_0 \\ 0.5y_2 - y_1 + 0.5y_0 \end{bmatrix} \tag{24.5.20}$$

We note that the first $d + 1$ smoothed values get initialized to the first $d + 1$ values of the input sequence.

Example 24.5.1: Fig. 24.5.1 shows the output of the exact exponential smoother with $d = 1$ and $\lambda = 0.9$ applied on the same noiseless input s_n of Example 24.1.3. In addition, it shows the $d = 1$ FIR filter $h_a(k)$ designed to have zero lag according to Eq. (24.4.4).

Because $d = 1$, both filters can follow a linear signal. The input s_n (dashed curve) is barely visible under the filter outputs (solid curves). The length of the FIR filter was chosen according to the rule $N = (1+\lambda)/(1-\lambda)$.

The following MATLAB code generates the two graphs; it uses the function `upulse` which is a part of the OSP toolbox that generates a unit-pulse of prescribed duration

```
n = 0:300;
s = (20 + 0.8*n) .* upulse(n,75) + ...          % upulse is in the OSP toolbox
    (80 - 0.3*(n-75)) .* upulse(n-75,150) + ...
    (35 + 0.4*(n-225)) .* upulse(n-225,76);

la = 0.9; al = 1-la; d = 1;

C = ema(s,d,la);                          % exact exponential smoother output
x = C(:,1);

N = round((1+la)/(1-la));                 % equivalent FIR length, N=19

k=0:N-1;
ha = 2*(2*N-1-3*k)/N/(N+1);               % zero-lag FIR filter

xh = filter(ha,1,s);                      % FIR filter output

figure; plot(n,s,'--', n,x,'-');          % left graph
figure; plot(n,s,'--', n,xh,'-');         % right graph
```

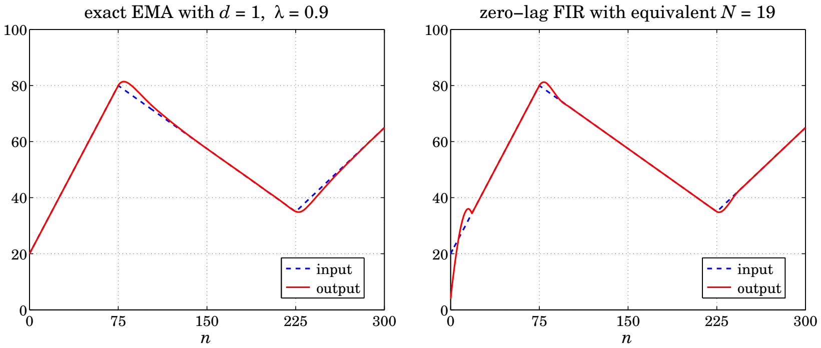


Fig. 24.5.1 Exact EMA with order $d = 1$, and zero-lag FIR filter with equivalent length.

Next, we add some noise $y_n = s_n + 4v_n$, where v_n is zero-mean, unit-variance, white noise. The top two graphs of Fig. 24.5.2 show the noisy signal y_n and the response of the exact EMA with $d = 0$ and $\lambda = 0.9$.

The bottom two graphs show the exact EMA with $d = 1$ as well as the response of the same zero-lag FIR filter to the noisy data. □

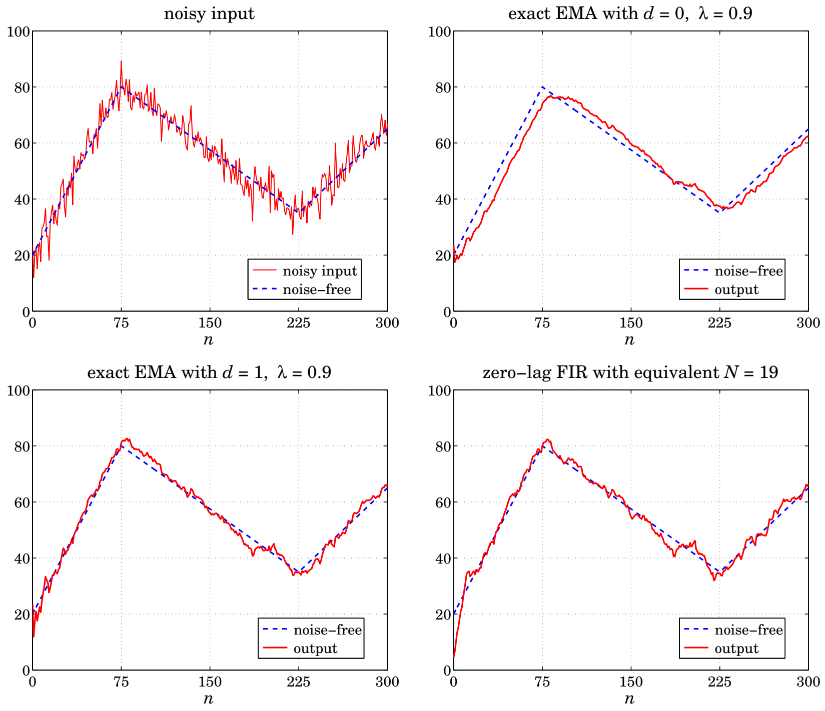


Fig. 24.5.2 EMA with order $d = 1$, and zero-lag FIR filter with equivalent length.

24.6 Steady-State Exponential Smoothing

Next, we look in more detail at the cases $d = 0, 1, 2$, which are the most commonly used in practice, with $d = 1$ providing the best performance and flexibility. We denote the polynomial coefficients by:

$$\mathbf{c}(n) = [a_n], \quad \mathbf{c}(n) = \begin{bmatrix} a_n \\ b_n \end{bmatrix}, \quad \mathbf{c}(n) = \begin{bmatrix} a_n \\ b_n \\ c_n \end{bmatrix} \tag{24.6.1}$$

Then, with $\mathbf{u}_\tau = [1]$, $\mathbf{u}_\tau = [1, \tau]^T$, and $\mathbf{u}_\tau = [1, \tau, \tau^2]^T$, the implied predicted estimates will be for arbitrary τ:

$$\begin{aligned} \hat{y}_{n+\tau/n} &= \mathbf{u}_\tau^T \mathbf{c}(n) = a_n \\ \hat{y}_{n+\tau/n} &= \mathbf{u}_\tau^T \mathbf{c}(n) = a_n + b_n\tau \\ \hat{y}_{n+\tau/n} &= \mathbf{u}_\tau^T \mathbf{c}(n) = a_n + b_n\tau + c_n\tau^2 \end{aligned} \tag{24.6.2}$$

Thus, a_n, b_n represent local estimates of the level and slope, respectively, and $2c_n$

represents the acceleration. The one-step-ahead predictions are,

$$
\begin{aligned}
\hat{y}_{n/n-1} &= \mathbf{u}_1^T\mathbf{c}(n-1) = a_{n-1}\\
\hat{y}_{n/n-1} &= \mathbf{u}_1^T\mathbf{c}(n-1) = a_{n-1} + b_{n-1}\\
\hat{y}_{n/n-1} &= \mathbf{u}_1^T\mathbf{c}(n-1) = a_{n-1} + b_{n-1} + c_{n-1}
\end{aligned}
\tag{24.6.3}
$$

Denoting the a posteriori gains $\mathbf{k}_n$ by,

$$
\mathbf{k}_n = [\alpha(n)]\,, \quad \mathbf{k}_n = \begin{bmatrix} \alpha_1(n) \\ \alpha_2(n) \end{bmatrix}, \quad \mathbf{k}_n = \begin{bmatrix} \alpha_1(n) \\ \alpha_2(n) \\ \alpha_3(n) \end{bmatrix}
\tag{24.6.4}
$$

then, the coefficient updates (24.5.18) take the forms, where $e_{n/n-1} = y_n - \hat{y}_{n/n-1}$,

$$
\begin{aligned}
a_n &= a_{n-1} + \alpha(n)e_{n/n-1}\\
\begin{bmatrix} a_n \\ b_n \end{bmatrix} &= \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} a_{n-1} \\ b_{n-1} \end{bmatrix} + \begin{bmatrix} \alpha_1(n) \\ \alpha_2(n) \end{bmatrix} e_{n/n-1}\\
\begin{bmatrix} a_n \\ b_n \\ c_n \end{bmatrix} &= \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} a_{n-1} \\ b_{n-1} \\ c_{n-1} \end{bmatrix} + \begin{bmatrix} \alpha_1(n) \\ \alpha_2(n) \\ \alpha_3(n) \end{bmatrix} e_{n/n-1}
\end{aligned}
\tag{24.6.5}
$$

Since $\mathbf{k}_n = R_n^{-1}\mathbf{u}_0$, the gains depend only on λ and n and converge to steady-state values for large n. For example, for $d = 0$, we have,

$$
R_n = \sum_{k=0}^{n} \lambda^k = \frac{1-\lambda^{n+1}}{1-\lambda} \quad \Rightarrow \quad k_n = R_n^{-1} = \frac{1-\lambda}{1-\lambda^{n+1}} \to 1-\lambda \equiv \alpha
$$

Thus, the steady-state form of the $d = 0$ EMA smoother is as expected:

$$
\boxed{\begin{aligned} e_{n/n-1} &= y_n - \hat{y}_{n/n-1} = y_n - a_{n-1}\\ a_n &= a_{n-1} + (1-\lambda)e_{n/n-1} \end{aligned}} \quad \text{(single EMA, } d = 0\text{)}
\tag{24.6.6}
$$

initialized as usual at $a_{-1} = y_0$. The corresponding likelihood variable $\mu_n = 1 - \mathbf{u}_0^T\mathbf{k}_n$ tends to $\mu = 1 - (1-\lambda) = \lambda$. Similarly, we find for $d = 1$,

$$
R_n = \sum_{k=0}^{n} \lambda^k \begin{bmatrix} 1 \\ -k \end{bmatrix} [1, -k] = \sum_{k=0}^{n} \lambda^k \begin{bmatrix} 1 & -k \\ -k & k^2 \end{bmatrix} \equiv \begin{bmatrix} R_{00}(n) & R_{01}(n) \\ R_{10}(n) & R_{11}(n) \end{bmatrix}
$$

where

$$
\begin{aligned}
R_{00}(n) &= \frac{1-\lambda^{n+1}}{1-\lambda}\,, \quad R_{01}(n) = R_{10}(n) = \frac{-\lambda + \lambda^{n+1}\big[1 + n(1-\lambda)\big]}{(1-\lambda)^2}\\
R_{11}(n) &= \frac{\lambda(1+\lambda) - \lambda^{n+1}\big[1 + \lambda - 2n(1-\lambda) + n^2(1-\lambda)^2\big]}{(1-\lambda)^3}
\end{aligned}
$$

which have the limit as $n \to \infty$,

$$
\begin{aligned}
R_n \to R &= \frac{1}{(1-\lambda)^3}\begin{bmatrix} (1-\lambda)^2 & -\lambda(1-\lambda) \\ -\lambda(1-\lambda) & \lambda(1+\lambda) \end{bmatrix}\\
R^{-1} &= \begin{bmatrix} 1-\lambda^2 & (1-\lambda)^2 \\ (1-\lambda)^2 & \lambda^{-1}(1-\lambda)^3 \end{bmatrix}
\end{aligned}
\tag{24.6.7}
$$

It follows that the asymptotic gain vector $\mathbf{k} = R^{-1}\mathbf{u}_0$ will be the first column of R^{-1}:

$$
\mathbf{k}_n \to \mathbf{k} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} 1-\lambda^2 \\ (1-\lambda)^2 \end{bmatrix}
\tag{24.6.8}
$$

and the steady-state version of the $d = 1$ EMA smoother becomes:

$$
\boxed{\begin{aligned} & e_{n/n-1} = y_n - \hat{y}_{n/n-1} = y_n - (a_{n-1} + b_{n-1})\\ & \begin{bmatrix} a_n \\ b_n \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} a_{n-1} \\ b_{n-1} \end{bmatrix} + \begin{bmatrix} 1-\lambda^2 \\ (1-\lambda)^2 \end{bmatrix} e_{n/n-1} \end{aligned}} \quad \text{(double EMA, } d = 1\text{)}
\tag{24.6.9}
$$

with estimated level $\hat{y}_{n/n} = a_n$ and one-step-ahead prediction $\hat{y}_{n+1/n} = a_n + b_n$. The corresponding limit of the likelihood parameter is $\mu = 1 - \mathbf{u}_0^T\mathbf{k} = 1 - (1-\lambda^2) = \lambda^2$. The difference equation may be initialized at $a_{-1} = 2y_0 - y_1$ and $b_{-1} = y_1 - y_0$ to agree with the first outputs of the exact smoother. Indeed, iterating up to $n = 1$, we find the same answer for $\mathbf{c}(1)$ as the exact smoother:

$$
\begin{aligned}
\begin{bmatrix} a_0 \\ b_0 \end{bmatrix} &= \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} a_{-1} \\ b_{-1} \end{bmatrix} + \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} e_{0/-1} = \begin{bmatrix} y_0 \\ y_1 - y_0 \end{bmatrix}\\
\begin{bmatrix} a_1 \\ b_1 \end{bmatrix} &= \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} a_0 \\ b_0 \end{bmatrix} + \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} e_{1/0} = \begin{bmatrix} y_1 \\ y_1 - y_0 \end{bmatrix}
\end{aligned}
$$

Of course, other initializations are possible, a common one being to fit a straight line to the first few input samples and choose the intercept and slope as the initial values. This is the default method used by the function `stema` (see below). For the $d = 2$ case, the asymptotic matrix R is

$$
R = \sum_{k=0}^{\infty} \lambda^k \mathbf{u}_{-k}\mathbf{u}_{-k}^T = \sum_{k=0}^{\infty} \lambda^k \begin{bmatrix} 1 & -k & k^2 \\ -k & k^2 & -k^3 \\ k^2 & -k^3 & k^4 \end{bmatrix}
$$

which may be summed to

$$
R = \begin{bmatrix}
\dfrac{1}{1-\lambda} & -\dfrac{\lambda}{(1-\lambda)^2} & \dfrac{\lambda(1+\lambda)}{(1-\lambda)^3} \\[2ex]
-\dfrac{\lambda}{(1-\lambda)^2} & \dfrac{\lambda(1+\lambda)}{(1-\lambda)^3} & -\dfrac{\lambda(1+4\lambda+\lambda^2)}{(1-\lambda)^4} \\[2ex]
\dfrac{\lambda(1+\lambda)}{(1-\lambda)^3} & -\dfrac{\lambda(1+4\lambda+\lambda^2)}{(1-\lambda)^4} & \dfrac{\lambda(1+\lambda)(1+10\lambda+\lambda^2)}{(1-\lambda)^5}
\end{bmatrix}
$$

with an inverse

$$R^{-1} = \begin{bmatrix} 1-\lambda^3 & \dfrac{3}{2}(1+\lambda)(1-\lambda)^2 & \dfrac{1}{2}(1-\lambda)^3 \\ \dfrac{3}{2}(1+\lambda)(1-\lambda)^2 & \dfrac{(1+\lambda)(1-\lambda)^3(1+9\lambda)}{4\lambda^2} & \dfrac{(1-\lambda)^4(1+3\lambda)}{4\lambda^2} \\ \dfrac{1}{2}(1-\lambda)^3 & \dfrac{(1-\lambda)^4(1+3\lambda)}{4\lambda^2} & \dfrac{(1-\lambda)^5}{4\lambda^2} \end{bmatrix}$$

The asymptotic gain vector $\mathbf{k} = R^{-1}\mathbf{u}_0$ and μ parameter are,

$$\mathbf{k} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} 1-\lambda^3 \\ \dfrac{3}{2}(1+\lambda)(1-\lambda)^2 \\ \dfrac{1}{2}(1-\lambda)^3 \end{bmatrix}, \qquad \mu = 1-\alpha_1 = \lambda^3 \tag{24.6.10}$$

and the steady-state $d = 2$ EMA smoother becomes:

$$\boxed{\begin{aligned} & e_{n/n-1} = y_n - \hat{y}_{n/n-1} = y_n - (a_{n-1} + b_{n-1} + c_{n-1}) \\ & \begin{bmatrix} a_n \\ b_n \\ c_n \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{n-1} \\ b_{n-1} \\ c_{n-1} \end{bmatrix} + \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} e_{n/n-1} \end{aligned}} \quad \text{(triple EMA, } d = 2\text{)} \tag{24.6.11}$$

They may be initialized to reach the same values at $n = 2$ as the exact smoother, that is, Eq. (24.5.20). This requirement gives:

$$\begin{bmatrix} a_{-1} \\ b_{-1} \\ c_{-1} \end{bmatrix} = \begin{bmatrix} y_2 - 3y_1 + 3y_0 \\ -1.5y_2 + 4y_1 - 2.5y_0 \\ 0.5y_2 - y_1 + 0.5y_0 \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} a_2 \\ b_2 \\ c_2 \end{bmatrix} = \begin{bmatrix} y_2 \\ 1.5y_2 - 2y_1 + 0.5y_0 \\ 0.5y_2 - y_1 + 0.5y_0 \end{bmatrix}$$

Alternatively, they may be initialized by fitting a second degree polynomial to the first few input samples, as is done by default in the function `stema`, or they may be initialized to a zero vector, or to any other values, for example, $[a_{-1}, b_{-1}, c_{-1}] = [y_0, 0, 0]$.

For arbitrary polynomial order d, the matrix R_n converges to a $(d+1)\times(d+1)$ matrix R that must satisfy the Lyapunov-type equation:

$$R = \mathbf{u}_0\mathbf{u}_0^T + \lambda G R G^T \tag{24.6.12}$$

where G is the backward boost matrix, $G = F^{-1}$. This follows by considering the limit of Eq. (24.5.1) as $n \to \infty$ and using the property $\mathbf{u}_{-k-1} = G\mathbf{u}_{-k}$. Multiplying from the left by F, and noting that $F\mathbf{u}_0 = \mathbf{u}_1$, we have

$$FR = \mathbf{u}_1\mathbf{u}_0^T + \lambda R G^T \tag{24.6.13}$$

Taking advantage of the unit-lower-triangular nature of F and G, this equation can be written component-wise as follows:

$$\sum_{k=0}^{i} F_{ik}R_{kj} = u_1(i)\,u_0(j) + \lambda \sum_{k=0}^{j} R_{ik}G_{jk}\,, \quad 0 \le i,j \le d \tag{24.6.14}$$

Noting that $u_1(i) = 1$ and $u_0(j) = \delta(j)$, and setting first $i = j = 0$, we find

$$R_{00} = 1 + \lambda R_{00} \quad \Rightarrow \quad R_{00} = \frac{1}{1-\lambda} \tag{24.6.15}$$

Then, setting $i = 0$ and $1 \le j \le d$,

$$R_{0j} = \lambda \sum_{k=0}^{j} R_{ik}G_{jk} = \lambda R_{0j} + \lambda \sum_{k=0}^{j-1} R_{0k}G_{jk}$$

which can be solved recursively for R_{0j}:

$$R_{0j} = R_{j0} = \frac{\lambda}{1-\lambda} \sum_{k=0}^{j-1} R_{0k}G_{jk}\,, \quad j = 1, 2, \dots, d \tag{24.6.16}$$

Next, take $i \ge 1$ and $j \ge i$, and use the symmetry of R:

$$R_{ij} + \sum_{k=0}^{i-1} F_{ik}R_{kj} = \lambda R_{ij} + \lambda \sum_{k=0}^{j-1} R_{ik}G_{jk}$$

or, for $\quad i = 1, 2, \dots, d\,, \quad j = i, i+1, \dots, d$,

$$R_{ij} = R_{ji} = \frac{1}{1-\lambda}\left[\lambda \sum_{k=0}^{j-1} R_{ik}G_{jk} - \sum_{k=0}^{i-1} F_{ik}R_{kj}\right] \tag{24.6.17}$$

To clarify the computations, we give the MATLAB code below:

```
R(1,1) = 1/(1-lambda);
for j=2:d+1,
  R(1,j) = lambda * R(1,1:j-1) * G(j,1:j-1)’ / (1-lambda);
  R(j,1) = R(1,j);
end
for i=2:d+1,
  for j=i:d+1,
    R(i,j) = (lambda*R(i,1:j-1)*G(j,1:j-1)’ - F(i,1:i-1)*R(1:i-1,j))/(1-lambda);
    R(j,i) = R(i,j);
  end
end
```

Once R is determined, one may calculate the gain vector $\mathbf{k} = R^{-1}\mathbf{u}_0$. Then, the overall filtering algorithm can be stated as follows, for $n \ge 0$,

$$\boxed{\begin{aligned} & \hat{y}_{n/n-1} = \mathbf{u}_1^T\mathbf{c}(n-1) \\ & e_{n/n-1} = y_n - \hat{y}_{n/n-1} \\ & \mathbf{c}(n) = F^T\mathbf{c}(n-1) + \mathbf{k}\,e_{n/n-1} \end{aligned}} \quad \text{(steady-state EMA)} \tag{24.6.18}$$

which requires specification of the initial vector $\mathbf{c}(-1)$. The transfer function from the input y_n to the signals $\mathbf{c}(n)$ can be determined by taking z-transforms of Eq. (24.6.18):

$$\mathbf{C}(z) = z^{-1}F^T\mathbf{C}(z) + \mathbf{k}\big(Y(z) - z^{-1}\mathbf{u}_1^T\mathbf{C}(z)\big)\,, \quad \text{or,}$$

$$\boxed{\mathbf{H}(z)=\frac{\mathbf{C}(z)}{Y(z)}=\big[I-(F^T-\mathbf{k}\mathbf{u}_1^T)z^{-1}\big]^{-1}\mathbf{k}} \qquad (24.6.19)$$

The computational steps (24.6.18) have been incorporated in the MATLAB function `stema`, with usage,

```
C = stema(y,d,lambda,cinit);      % steady-state exponential moving average
```

where `C,y,d,lambda` have the same meaning as in the function `ema`. The parameter `cinit` is a $(d+1)\times 1$ column vector that represents the initial vector $\mathbf{c}(-1)$. If omitted, it defaults to fitting a polynomial of order d to the first L input samples, where L is the effective length corresponding to λ, that is, $L=(1+\lambda)/(1-\lambda)$. The fitting is carried out with the help of the function `lpbasis` from Chap. 23.1, and is given in MATLAB notation by:

```
cinit = lpbasis(L,d,-1) \ y(1:L);      % fit order-d polynomial to first L inputs
```

where the fit is carried out with respect to the time origin $n=-1$. The length L must be less than the length of the input vector $\mathbf{y}$. If not, another, shorter L can be used. Other initialization possibilities for $\mathbf{c}_{\text{init}}$ are summarized in the help file for `stema`.

To clarify the fitting operation, we note that fitting the first L samples y_n, $n=0,1,\dots,L-1$, to a polynomial of degree d centered at $n=-1$ amounts to the minimization of the performance index:

$$\mathcal{J}=\sum_{n=0}^{L-1}(y_n-p_n)^2=\min\,,\quad p_n=\sum_{i=0}^{d}(n+1)^i c_i=\mathbf{u}_{n+1}^T\mathbf{c}$$

which can be written compactly as

$$\mathcal{J}=\|\mathbf{y}-S\mathbf{c}\|^2=\min\,,\quad S=[\mathbf{u}_1,\mathbf{u}_2,\dots,\mathbf{u}_{n+1},\dots,\mathbf{u}_L]^T$$

with solution $\mathbf{c}=(S^TS)^{-1}S^T\mathbf{y}=S\backslash\mathbf{y}$ in MATLAB notation.[†] The actual fitted values $\mathbf{p}=[p_0,p_1,\dots,p_{L-1}]^T$ are then computed by $\mathbf{p}=S\mathbf{c}$.

Selecting $n=-1$ as the centering time, assumes that the filtering operation will start at $n=0$ requiring therefore the value $\mathbf{c}(-1)$. The centering can be done at any other reference time $n=n_0$, for example, one would choose $n_0=L-1$ if the filtering operation were to start at $n=L$. The performance index would be then,

$$\mathcal{J}=\sum_{n=0}^{L-1}(y_n-p_n)^2=\min\,,\quad p_n=\sum_{i=0}^{d}(n-n_0)^i c_i=\mathbf{u}_{n-n_0}^T\bar{\mathbf{c}}$$

with another set of coefficients $\bar{\mathbf{c}}$. The MATLAB implementation is in this case,

```
cinit = lpbasis(L,d,n0) \ y(1:L);      % fit order-d polynomial to first L inputs
```

From $\mathbf{u}_{n+1}=F\mathbf{u}_n$, we obtain $\mathbf{u}_{n+1}=F^{n_0+1}\mathbf{u}_{n-n_0}$. By requiring that the fitted polynomials be the same, $p_n=\mathbf{u}_{n+1}^T\mathbf{c}=\mathbf{u}_{n-n_0}^T\bar{\mathbf{c}}$, it follows that,

$$\bar{\mathbf{c}}=(F^T)^{n_0+1}\mathbf{c} \qquad (24.6.20)$$

[†] assuming that S has full rank, which requires $L>d$.

In Sec. 24.8, we discuss the connection to conventional multiple exponential smoothing obtained by filtering in cascade through $d+1$ copies of a single exponential smoothing filter $H(z)=\alpha/(1-\lambda z^{-1})$, that is, through $[H(z)]^{d+1}$. Example 24.12.1 illustrates the above initialization methods, as well as how to map the initial values of $\mathbf{c}(n)$ to the initial values of the cascaded filter outputs.

24.7 Smoothing Parameter Selection

The performance of the steady-state EMA may be judged by computing the covariance of the estimates $\mathbf{c}(n)$, much like the case of the $d=0$ smoother. Starting with $\mathbf{c}(n)=R_n^{-1}\mathbf{r}_n$ and $\mathbf{r}_n=S_n^TW_n\mathbf{y}(n)$, we obtain for the correlation matrix,

$$E[\mathbf{c}(n)\mathbf{c}^T(n)]=R_n^{-1}S_n^TW_nE[\mathbf{y}(n)\mathbf{y}^T(n)]W_nS_nR_n^{-1}$$

and for the corresponding covariance matrix,

$$\Sigma_{cc}=R_n^{-1}S_n^TW_n\Sigma_{yy}W_nS_nR_n^{-1} \qquad (24.7.1)$$

Under the typical assumption that y_n is white noise, we have $\Sigma_{yy}=\sigma_y^2I_{n+1}$, where I_{n+1} is the $(n+1)$-dimensional unit matrix. Then,

$$\Sigma_{cc}=\sigma_y^2R_n^{-1}Q_nR_n^{-1}\,,\quad Q_n=S_n^TW_n^2S_n \qquad (24.7.2)$$

In the limit $n\to\infty$, the matrices R_n,Q_n tend to steady-state values, so that

$$\Sigma_{cc}=\sigma_y^2R^{-1}QR^{-1} \qquad (24.7.3)$$

where the limit matrices R,Q are given by

$$R=\sum_{k=0}^{\infty}\lambda^k\mathbf{u}_{-k}\mathbf{u}_{-k}\,,\quad Q=\sum_{k=0}^{\infty}\lambda^{2k}\mathbf{u}_{-k}\mathbf{u}_{-k} \qquad (24.7.4)$$

Since $\hat{y}_{n/n}=\mathbf{u}_0^T\mathbf{c}(n)$ and $\hat{y}_{n+1/n}=\mathbf{u}_1^T\mathbf{c}(n)$, the corresponding variances will be:

$$\sigma^2_{\hat{y}_{n/n}}=\mathbf{u}_0^T\Sigma_{cc}\mathbf{u}_0\,,\quad \sigma^2_{\hat{y}_{n+1/n}}=\mathbf{u}_1^T\Sigma_{cc}\mathbf{u}_1\equiv\sigma^2_{\hat{y}}\,, \qquad (24.7.5)$$

Because y_n was assumed to be an uncorrelated sequence, the two terms in the prediction error $e_{n+1/n}=y_{n+1}-\hat{y}_{n+1/n}$ will be uncorrelated since $\hat{y}_{n+1/n}$ depends only on data up to n. Therefore, the variance of the prediction error $e_{n+1/n}$ will be:

$$\sigma_e^2=\sigma_y^2+\sigma_{\hat{y}}^2=\sigma_y^2\big[1+\mathbf{u}_1^TR^{-1}QR^{-1}\mathbf{u}_1\big] \qquad (24.7.6)$$

For the case $d=0$, we have

$$R=\sum_{k=0}^{\infty}\lambda^k=\frac{1}{1-\lambda}\,,\quad Q=\sum_{k=0}^{\infty}\lambda^{2k}=\frac{1}{1-\lambda^2}$$

which gives the usual results:

$$\sigma_{\hat{y}}^2=\Sigma_{cc}=\frac{1-\lambda}{1+\lambda}\sigma_y^2\,,\quad \sigma_e^2=\sigma_y^2+\sigma_{\hat{y}}^2=\frac{2}{1+\lambda}\sigma_y^2$$

For $d = 1$, we have as in Eq. (24.6.7),

$$R = \frac{1}{(1-\lambda)^3}\begin{bmatrix} (1-\lambda)^2 & -\lambda(1-\lambda) \\ -\lambda(1-\lambda) & \lambda(1+\lambda) \end{bmatrix},$$

with the Q matrix being obtained from R by replacing $\lambda \to \lambda^2$,

$$Q = \frac{1}{(1-\lambda^2)^3}\begin{bmatrix} (1-\lambda^2)^2 & -\lambda^2(1-\lambda^2) \\ -\lambda^2(1-\lambda^2) & \lambda^2(1+\lambda^2) \end{bmatrix}$$

It follows then that

$$\Sigma_{cc} = \sigma_y^2 R^{-1}QR^{-1} = \frac{1-\lambda}{(1+\lambda)^3}\begin{bmatrix} 1+4\lambda+5\lambda^2 & (1-\lambda)(1+3\lambda) \\ (1-\lambda)(1+3\lambda) & 2(1-\lambda)^2 \end{bmatrix} \tag{24.7.7}$$

The diagonal entries are the variances of the level and slope signals a_n, b_n:

$$\sigma_a^2 = \frac{(1-\lambda)(1+4\lambda+5\lambda^2)}{(1+\lambda)^3}\sigma_y^2\,, \qquad \sigma_b^2 = \frac{2(1-\lambda)^3}{(1+\lambda)^3}\sigma_y^2 \tag{24.7.8}$$

For the prediction variance, we find

$$\sigma_{\hat{y}}^2 = \sigma_y^2\mathbf{u}_1^T(R^{-1}QR^{-1})\mathbf{u}_1 = \frac{(1-\lambda)(\lambda^2+4\lambda+5)}{(1+\lambda)^3}\sigma_y^2 \tag{24.7.9}$$

which gives for the prediction error:

$$\sigma_e^2 = \sigma_y^2 + \sigma_{\hat{y}}^2 = \left[1 + \frac{(1-\lambda)(\lambda^2+4\lambda+5)}{(1+\lambda)^3}\right]\sigma_y^2 = \frac{2(3+\lambda)}{(1+\lambda)^3}\sigma_y^2 \tag{24.7.10}$$

In order to achieve an equivalent smoothing performance with a $d = 0$ EMA, one must equate the corresponding prediction variances, or mean-square errors. If λ_0, λ_1 denote the equivalent $d = 0$ and $d = 1$ parameters, the condition reads:

$$\frac{2}{1+\lambda_0} = \frac{2(3+\lambda_1)}{(1+\lambda_1)^3} \quad\Rightarrow\quad \lambda_0 = \frac{(1+\lambda_1)^3}{3+\lambda_1} - 1 \tag{24.7.11}$$

Eq. (24.7.11) may also be solved for λ_1 in terms of λ_0,

$$\lambda_1 = \frac{1}{3}D_0 + \frac{1+\lambda_0}{D_0} - 1\,, \quad D_0 = \left[27(1+\lambda_0) + \sqrt{27(1+\lambda_0)^2(26-\lambda_0)}\right]^{1/3} \tag{24.7.12}$$

Setting $\lambda_0 = 0$ gives $D_0 = (27 + 3\sqrt{78})^{1/3}$ and $\lambda_1 = 0.5214$. For all $\lambda_1 \geq 0.5214$, the equivalent λ_0 is non-negative and the NRR $\sigma_{\hat{y}}^2/\sigma_y^2$ of the prediction filter remains less than unity.

The corresponding FIR averager would have length $N_0 = (1+\lambda_0)/(1-\lambda_0)$, whereas an equivalent zero-lag FIR filter should have length N_1 that matches the corresponding NRRs. We have from Eq. (24.4.9):

$$\frac{2(2N_1 - 1)}{N_1(N_1+1)} = \frac{1-\lambda_0}{1+\lambda_0}$$

which gives,

$$\lambda_0 = \frac{N_1^2 - 3N_1 + 2}{N_1^2 + 5N_1 - 2} \quad\Leftrightarrow\quad N_1 = \frac{3 + 5\lambda_0 + \sqrt{33\lambda_0^2 + 30\lambda_0 + 1}}{2(1-\lambda_0)} \tag{24.7.13}$$

The MATLAB function `emap` implements Eq. (24.7.12),

```
la1 = emap(la0);        % mapping equivalent λ's between d = 0 and d = 1 EMAs
```

The computed λ_1 is an increasing function of λ_0 and varies over $0.5214 \leq \lambda_1 \leq 1$ as λ_0 varies over $0 \leq \lambda_0 \leq 1$.

Example 24.7.1: The lower-right graph of Fig. 24.7.1 shows a zero-lag FIR filter defined by Eq. (24.4.4) with length $N_1 = 100$ and applied to the noisy signal shown on the upper-left graph. The noisy signal was $y_n = 20 + 0.2n + 4v_n$, for $0 \leq n \leq 300$, with zero-mean, unit-variance, white noise v_n.

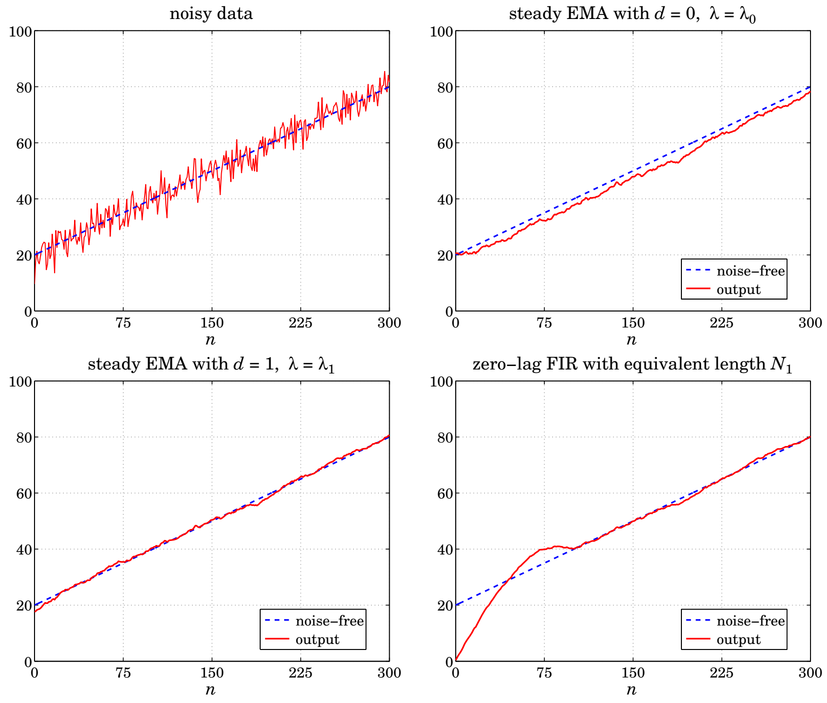


Fig. 24.7.1 Comparison of two equivalent steady-state EMAs with equivalent zero-lag FIR.

The equivalent EMA parameter for $d = 0$ was found from (24.7.13) to be $\lambda_0 = 0.9242$, which was then mapped to $\lambda_1 = 0.9693$ of an equivalent $d = 1$ EMA using Eq. (24.7.12). The upper-right graph shows the $d = 0$ EMA output, and the lower-left graph, the $d = 1$ EMA. The steady-state version was used for both EMAs with default initializations. The following MATLAB code illustrates the calculations:

```
t = 0:300; s = 20 + 0.2*t;
randn('state', 1000);
y = s + 4 * randn(size(t));                   % noisy input

N1 = 100;
la0 = (N1^2-3*N1+2)/(N1^2+5*N1-2);            % equivalent λ0
la1 = emap(la0);                              % equivalent λ1

C = stema(y,0,la0); x0 = C(:,1);              % steady EMA with d = 0, λ = λ0
C = stema(y,1,la1); x1 = C(:,1);              % steady EMA with d = 1, λ = λ1

k=0:N1-1; h = 2*(2*N1-1-3*k)/N1/(N1+1);       % zero-lag FIR of length N1
% h = lpinterp(N1,1,-(N1-1)/2)';              % alternative calculation
xh = filter(h,1,y);

figure; plot(t,y,'-', t,s,'-');    figure; plot(t,s,'--', t,x0,'-');
figure; plot(t,s,'--', t,x1,'-'); figure; plot(t,s,'--', t,xh,'-');
```

We observe that all three outputs achieve comparable noise reduction. The $d = 0$ EMA suffers from the expected delay. Both the $d = 1$ EMA and the zero-lag FIR filter follow the straight-line input with no delay, after the initial transients have subsided. □

The choice of the parameter λ is more of an art than science. There do exist, however, criteria that determine an “optimum” value. Given the prediction $\hat{y}_{n/n-1} = \mathbf{u}_1^T\mathbf{c}(n-1)$ of y_n, and prediction error $e_{n/n-1} = y_n - \hat{y}_{n/n-1}$, the following criteria, to be minimized with respect to λ, are widely used:

$$\boxed{\begin{aligned} &\text{MSE} = \text{mean}(e_{n/n-1}^2)\,, && \text{(mean square error)}\\ &\text{MAE} = \text{mean}\big(|e_{n/n-1}|\big)\,, && \text{(mean absolute error)}\\ &\text{MAPE} = \text{mean}\big(100|e_{n/n-1}/y_n|\big)\,, && \text{(mean absolute percentage error)}\end{aligned}} \qquad (24.7.14)$$

where the mean may be taken over the entire data set or over a portion of it. Usually, the criteria are evaluated over a range of λ's and the minimum is selected. Typically, the criteria tend to underestimate the value of λ, that is, they produce too small a λ to be useful for smoothing purposes. Even so, the optimum λ has been used successfully for forecasting applications. The MATLAB function `emaerr` calculates these criteria for any vector of λ's and determines the optimum λ in that range:

```
[err,lopt] = emaerr(y,d,lambda,type);     % mean error criteria
```

where `type` takes one of the string values `'mse'`, `'mae'`, `'mape'` and `err` is the criterion evaluated at the vector `lambda`, and `lopt` is the corresponding optimum λ.

Example 24.7.2: Fig. 24.7.2 shows the same Dow-Jones data of Example 24.1.2. The MSE criterion was searched over the range $0.1 \le \lambda \le 0.9$. The upper-left graph shows the MSE versus λ. The minimum occurs at $\lambda_{\text{opt}} = 0.61$.

The upper-right graph shows the $d = 1$ exact EMA run with $\lambda = \lambda_{\text{opt}}$. The EMA output is too rough to provide adequate smoothing. The other criteria are even worse. The MAE and MAPE optima both occur at $\lambda_{\text{opt}} = 0.56$. For comparison, the bottom two graphs show the $d = 1$ exact EMA run with the two higher values $\lambda = 0.90$ and $\lambda = 0.95$. The MATLAB code generating these graphs was as follows:

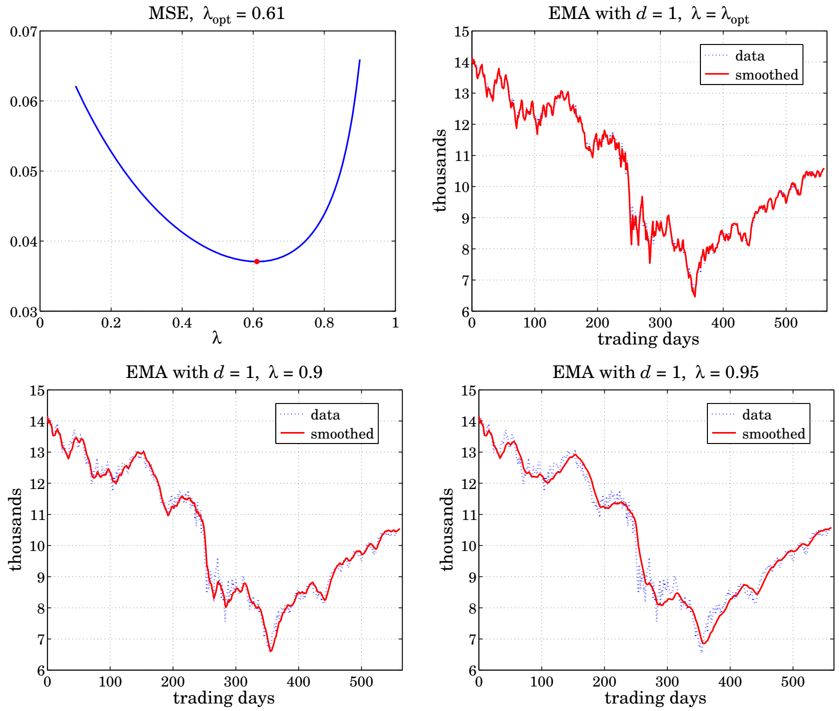


Fig. 24.7.2 MSE criterion for the DJIA data.

```
Y = loadfile('dow-oct07-dec09.dat');         % read data
y = Y(:,1)/1000;   n = (0:length(y)-1)';

d = 1; u1 = ones(d+1,1);                     % polynomial order for EMA

la = linspace(0.1, 0.9, 81);                 % range of λ's to search
[err,lopt] = emaerr(y,d,la,'mse');           % evaluate MSE at this range of λ's

figure; plot(la,err, lopt,min(err),'.');     % upper-left graph

C = ema(y,d,lopt); yhat = C*u1;
figure; plot(n,y,':', n,yhat,'-');           % upper-right graph

la=0.90; C = ema(y,d,la); yhat = C*u1;       % bottom-left graph
figure; plot(n,y,':', n,yhat,'-');           % use la=0.95 for bottom-right
```

We note that the $d = 1$ smoother is more capable in following the signal than the $d = 0$ one. We plotted the forecasted value $\hat{y}_{n+1/n} = \mathbf{c}^T(n)\mathbf{u}_1$ versus n. Because the output matrix `C` from the `ema` function has the $\mathbf{c}^T(n)$ as its rows, the entire vector of forecasted values can be calculated by acting by `C` on the unit vector $\mathbf{u}_1$, that is, `yhat = C*u1`. □

24.8 Single, Double, Triple Exponential Smoothing

Single exponential smoothing is the same as first-order, $d = 0$, steady-state exponential smoothing. We recall its filtering equation and corresponding transfer function:

$$a_n^{[1]} = \lambda a_{n-1}^{[1]} + \alpha y_n\,, \quad H^{[1]}(z) = H(z) = \frac{\alpha}{1-\lambda z^{-1}} \tag{24.8.1}$$

where $\alpha = 1-\lambda$. Double smoothing refers to filtering $a_n^{[1]}$ one more time through the same filter $H(z)$; and triple smoothing, two more times. The resulting filtering equations and transfer functions (from the overall input y_n to the final outputs) are:

$$\begin{aligned} a_n^{[2]} &= \lambda a_{n-1}^{[2]} + \alpha a_n^{[1]}\,, \quad H^{[2]}(z) = \left(\frac{\alpha}{1-\lambda z^{-1}}\right)^2 \\ a_n^{[3]} &= \lambda a_{n-1}^{[3]} + \alpha a_n^{[2]}\,, \quad H^{[3]}(z) = \left(\frac{\alpha}{1-\lambda z^{-1}}\right)^3 \end{aligned} \tag{24.8.2}$$

$$y_n \longrightarrow \boxed{H} \longrightarrow a_n^{[1]} \longrightarrow \boxed{H} \longrightarrow a_n^{[2]} \longrightarrow \boxed{H} \longrightarrow a_n^{[3]}$$

Thus, the filter $H(z)$ acts once, twice, three times, or in general $d+1$ times, in cascade, producing the outputs,

$$y_n \longrightarrow \boxed{H} \longrightarrow a_n^{[1]} \longrightarrow \boxed{H} \longrightarrow a_n^{[2]} \longrightarrow \boxed{H} \longrightarrow a_n^{[3]} \longrightarrow \cdots \longrightarrow a_n^{[d]} \longrightarrow \boxed{H} \longrightarrow a_n^{[d+1]} \tag{24.8.3}$$

The transfer function and the corresponding causal impulse response from y_n to the r-th output $a_n^{[r]}$ are, for $r = 1, 2, \dots, d+1$ with $u(n)$ denoting the unit-step function:

$$H^{[r]}(z) = \big[H(z)\big]^r = \left(\frac{\alpha}{1-\lambda z^{-1}}\right)^r \quad \Leftrightarrow \quad h^{[r]}(n) = \alpha^r \lambda^n \frac{(n+r-1)!}{n!(r-1)!}\,u(n) \tag{24.8.4}$$

Double and triple exponential smoothing are in a sense equivalent to the $d = 1$ and $d = 2$ steady-state EMA filters of Eq. (24.6.9) and (24.6.11). From Eq. (24.6.19), which in this case reads $\mathbf{H}(z) = [H_a(z), H_b(z)]^T$, we may obtain the transfer functions from y_n to the outputs a_n and b_n:

$$H_a(z) = \frac{(1-\lambda)(1+\lambda-2\lambda z^{-1})}{(1-\lambda z^{-1})^2}\,, \quad H_b(z) = \frac{(1-\lambda)^2(1-z^{-1})}{(1-\lambda z^{-1})^2} \tag{24.8.5}$$

It is straightforward to verify that H_a and H_b are related to H and H^2 by

$$\boxed{\begin{aligned} H_a &= 2H - H^2 = 1-(1-H)^2 \\ H_b &= \frac{\alpha}{\lambda}(H - H^2) \end{aligned}} \quad \begin{aligned} &\text{(local level filter)} \\ &\text{(local slope filter)} \end{aligned} \tag{24.8.6}$$

In the time domain this implies the following relationships between the a_n, b_n signals and the cascaded outputs $a_n^{[1]}, a_n^{[2]}$:

$$\boxed{\begin{aligned} a_n &= 2a_n^{[1]} - a_n^{[2]} = \text{local level} \\ b_n &= \frac{\alpha}{\lambda}(a_n^{[1]} - a_n^{[2]}) = \text{local slope} \end{aligned}} \tag{24.8.7}$$

which can be written in a 2×2 matrix form:

$$\begin{bmatrix} a_n \\ b_n \end{bmatrix} = \begin{bmatrix} 2 & -1 \\ \alpha/\lambda & -\alpha/\lambda \end{bmatrix} \begin{bmatrix} a_n^{[1]} \\ a_n^{[2]} \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} a_n^{[1]} \\ a_n^{[2]} \end{bmatrix} = \begin{bmatrix} 1 & -\lambda/\alpha \\ 1 & -2\lambda/\alpha \end{bmatrix} \begin{bmatrix} a_n \\ b_n \end{bmatrix} \tag{24.8.8}$$

Similarly, for the $d = 2$ case, the transfer functions from y_n to a_n, b_n, c_n are:

$$\begin{aligned} H_a(z) &= \frac{\alpha\big[1+\lambda+\lambda^2 - 3\lambda(1+\lambda)z^{-1} + 3\lambda^2 z^{-2}\big]}{(1-\lambda z^{-1})^3} \\ H_b(z) &= \frac{1}{2}\,\frac{\alpha^2(1-z^{-1})\big[3(1+\lambda) - (5\lambda+1)z^{-1}\big]}{(1-\lambda z^{-1})^3} \\ H_c(z) &= \frac{1}{2}\,\frac{\alpha^3(1-z^{-1})^2}{(1-\lambda z^{-1})^3} \end{aligned} \tag{24.8.9}$$

which are related to H, H^2, H^3 by the matrix transformation:

$$\begin{bmatrix} H \\ H^2 \\ H^3 \end{bmatrix} = \begin{bmatrix} 1 & -\lambda/\alpha & \lambda(\lambda+1)/\alpha^2 \\ 1 & -2\lambda/\alpha & 2\lambda(2\lambda+1)/\alpha^2 \\ 1 & -3\lambda/\alpha & 3\lambda(3\lambda+1)/\alpha^2 \end{bmatrix} \begin{bmatrix} H_a \\ H_b \\ H_c \end{bmatrix} \tag{24.8.10}$$

implying the transformation between the outputs:

$$\begin{bmatrix} a_n^{[1]} \\ a_n^{[2]} \\ a_n^{[3]} \end{bmatrix} = \begin{bmatrix} 1 & -\lambda/\alpha & \lambda(\lambda+1)/\alpha^2 \\ 1 & -2\lambda/\alpha & 2\lambda(2\lambda+1)/\alpha^2 \\ 1 & -3\lambda/\alpha & 3\lambda(3\lambda+1)/\alpha^2 \end{bmatrix} \begin{bmatrix} a_n \\ b_n \\ c_n \end{bmatrix} \tag{24.8.11}$$

with corresponding inverse relationships,

$$\begin{bmatrix} H_a \\ H_b \\ H_c \end{bmatrix} = \frac{1}{2\lambda^2} \begin{bmatrix} 6\lambda^2 & -6\lambda^2 & 2\lambda^2 \\ \alpha(1+5\lambda) & -2\alpha(1+4\lambda) & \alpha(1+3\lambda) \\ \alpha^2 & -2\alpha^2 & \alpha^2 \end{bmatrix} \begin{bmatrix} H \\ H^2 \\ H^3 \end{bmatrix} \tag{24.8.12}$$

$$\begin{bmatrix} a_n \\ b_n \\ c_n \end{bmatrix} = \frac{1}{2\lambda^2} \begin{bmatrix} 6\lambda^2 & -6\lambda^2 & 2\lambda^2 \\ \alpha(1+5\lambda) & -2\alpha(1+4\lambda) & \alpha(1+3\lambda) \\ \alpha^2 & -2\alpha^2 & \alpha^2 \end{bmatrix} \begin{bmatrix} a_n^{[1]} \\ a_n^{[2]} \\ a_n^{[3]} \end{bmatrix} \tag{24.8.13}$$

In particular, we have:

$$H_a = 3H - 3H^2 + H^3 = 1-(1-H)^3 \tag{24.8.14}$$

and

$$\hat{y}_{n/n} = a_n = 3a_n^{[1]} - 3a_n^{[2]} + a_n^{[3]} \tag{24.8.15}$$

More generally, for an order-d polynomial EMA, we have [827],

$$H_a = 1-(1-H)^{d+1} \tag{24.8.16}$$

$$\hat{y}_{n/n} = a_n = -\sum_{r=1}^{d+1} (-1)^r \binom{d+1}{r} a_n^{[r]} \tag{24.8.17}$$

24.9 Tukey's Twicing Operation

There is an interesting interpretation [839] of these results in terms of Tukey's *twicing* operation [841] and its generalizations to *thricing*, and so on. To explain twicing, consider a smoothing operation, which for simplicity we may assume that it can be represented by the matrix operation $\hat{\mathbf{y}} = H\mathbf{y}$, or if so preferred, in the z-domain as the multiplication of z-transforms $\hat{Y}(z) = H(z)Y(z)$.

The resulting residual error is $\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}} = (I - H)\mathbf{y}$. In the twicing procedure, the residuals are filtered through the same smoothing operation, which will smooth them further, $\hat{\mathbf{e}} = H\mathbf{e} = H(I - H)\mathbf{y}$, and the result is added to the original estimate to get an improved estimate:

$$\hat{\mathbf{y}}_{\text{impr}} = \hat{\mathbf{y}} + \hat{\mathbf{e}} = \big[H + H(I - H)\big]\mathbf{y} = \big[2H - H^2\big]\mathbf{y} \tag{24.9.1}$$

which is recognized as the operation (24.8.6). The process can be continued by repeating it on the residuals of the residuals, and so on. For example, at the next step, one would compute the new residual $\mathbf{r} = \mathbf{e} - \hat{\mathbf{e}} = (I - H)\mathbf{e} = (I - H)^2\mathbf{y}$, then filter it through H, $\hat{\mathbf{r}} = H\mathbf{r} = H(I - H)^2\mathbf{y}$, and add it to get the "thriced" improved estimate:

$$\hat{\mathbf{y}}_{\text{impr}} = \hat{\mathbf{y}} + \hat{\mathbf{e}} + \hat{\mathbf{r}} = \big[H + H(I - H) + H(I - H)^2\big]\mathbf{y} = \big[3H - 3H^2 + H^3\big]\mathbf{y} \tag{24.9.2}$$

which is Eq. (24.8.14). The process can be continued d times, resulting in,

$$\hat{\mathbf{y}}_{\text{impr}} = H\big[I + (I - H) + (I - H)^2 + \cdots + (I - H)^d\big]\mathbf{y} = \big[I - (I - H)^{d+1}\big]\mathbf{y} \tag{24.9.3}$$

Twicing and its generalizations can be applied with benefit to any smoothing operation, for example, if we used an LPRS filter designed by $B = \text{lprs}(N, d, s)$, the computational steps for twicing would be:

$$\hat{\mathbf{y}} = \text{lpfilt}(B, \mathbf{y}) \;\Rightarrow\; \mathbf{e} = \mathbf{y} - \hat{\mathbf{y}} \;\Rightarrow\; \hat{\mathbf{e}} = \text{lpfilt}(B, \mathbf{e}) \;\Rightarrow\; \hat{\mathbf{y}}_{\text{impr}} = \hat{\mathbf{y}} + \hat{\mathbf{e}}$$

A limitation of twicing is that, while it drastically improves the passband of a lowpass smoothing filter, it worsens its stopband. To see this, we write for the improved transfer function, $H_{\text{impr}}(z) = 1 - \big(1 - H(z)\big)^{d+1}$. In the passband, H is near unity, say $H \approx 1 - \epsilon$, with $|\epsilon| \ll 1$, then,

$$H_{\text{impr}} = 1 - \big(1 - (1 - \epsilon)\big)^{d+1} = 1 - \epsilon^{d+1}$$

thus, making the passband ripple $(d+1)$ orders of magnitude smaller. On the other hand, in the stopband, H is near zero, say $H \approx \pm\epsilon$, resulting in a worsened stopband,

$$H_{\text{impr}} = 1 - (1 \mp \epsilon)^{d+1} \approx 1 - \big(1 \mp (d + 1)\epsilon\big) = \pm(d + 1)\epsilon$$

The twicing procedure has been generalized by Kaiser and Hamming [842] to the so-called "filter sharpening" that improves both the passband and the stopband. For example, the lowest-order filter combination that achieves this is,

$$H_{\text{impr}} = H^2(3 - 2H) = 3H^2 - 2H^3 \tag{24.9.4}$$

where now both the passband and stopband ripples are replaced by $\epsilon \to \epsilon^2$. More generally, it can be shown [842] that the filter that achieves pth order tangency at $H = 0$ and qth order tangency at $H = 1$ is given by

$$H_{\text{impr}} = H^{p+1}\sum_{k=0}^{q}\frac{(p + k)!}{p!\,k!}(1 - H)^k \tag{24.9.5}$$

The multiple exponential moving average case corresponds to $p = 0$ and $q = d$, resulting in $H_{\text{impr}} = 1 - (1 - H)^{d+1}$, whereas Eq. (24.9.4) corresponds to $p = q = 1$.

24.10 Zero-Lag Filters and Twicing

Another advantage of twicing and, more generally, filter sharpening is that the resulting improved smoothing filter always has zero lag, that is, $\bar{n} = 0$.

Indeed, assuming unity DC gain for the original filter, $H(z)\big|_{z=1} = 1$, it is straightforward to show that the general formula (24.9.5) gives for the first derivative:

$$H'_{\text{impr}}(z)\big|_{z=1} = 0 \tag{24.10.1}$$

which implies that its lag is zero, $\bar{n} = 0$, by virtue of Eq. (24.1.18). The twicing procedure, or its generalizations, for getting zero-lag filters is not limited to the exponential moving average. It can be applied to any other lowpass filter. For example, if we apply it to an ordinary length-N FIR averager, we would obtain:

$$H(z) = \frac{1}{N}\sum_{n=0}^{N-1} z^{-n} = \frac{1}{N}\,\frac{1 - z^{-N}}{1 - z^{-1}} \quad\Rightarrow\quad H_a(z) = 2H(z) - H^2(z) \tag{24.10.2}$$

The impulse response of $H_a(z)$ can be shown to be, where $0 \le n \le 2(N - 1)$,

$$h_a(n) = \left(\frac{2N - 1 - n}{N^2}\right)\big[u(n) - 2u(n - N) + u(n - 2N + 1)\big] \tag{24.10.3}$$

It is straightforward to show that $\bar{n}_a = 0$ and that its noise-reduction ratio is

$$\mathcal{R} = \frac{8N^2 - 6N + 1}{3N^3} \tag{24.10.4}$$

Because of their zero-lag property, double and triple EMA filters are used as *trend indicators* in the financial markets [881,882]. The application of twicing to the modified exponential smoother of Eq. (15.2.2) gives rise to a similar indicator called the *instantaneous trendline* [869], and further discussed in Problem 24.8. We discuss such market indicators in Chap. 25.

The zero-lag property for a causal lowpass filter comes at a price, namely, that although its magnitude response is normalized to unity at $\omega = 0$ and has a flat derivative there, it typically bends upwards developing a bandpass peak near DC before it attenuates to smaller values at higher frequencies. See, for example, Fig. 24.10.1.

This behavior might be deemed to be objectionable since it tends to unevenly amplify the low-frequency passband of the filter.

To clarify these remarks, consider a lowpass filter $H(\omega)$ (with real-valued impulse response h_n) satisfying the gain and flatness conditions $H(0)= 1$ and $H'(0)= 0$ at $\omega = 0$. The flatness condition implies the zero-lag property $\bar{n} = 0$. Using these two conditions, it is straightforward to verify that the second derivative of the magnitude response at DC is given by:

$$\frac{d^2}{d\omega^2}|H(\omega)|^2_{\omega=0} = 2\,\mathrm{Re}\big[H''(0)\big] + 2|H'(0)|^2 = 2\,\mathrm{Re}\big[H''(0)\big] = -2\sum_{n=0}^{\infty} n^2 h_n \tag{24.10.5}$$

Because $\bar{n} = \sum_n n h_n = 0$, it follows that some of the coefficients h_n must be negative, which can cause (24.10.5) to become positive, implying that $\omega = 0$ is a local minimum and hence the response will rise for ωs immediately beyond DC. This is demonstrated for example in Problem 24.7 by Eq. (24.14.1), so that,

$$\frac{d^2}{d\omega^2}|H(\omega)|^2_{\omega=0} = -2\sum_{n=0}^{\infty} n^2 h_n = \frac{4\lambda^2}{(1-\lambda)^2}$$

A similar calculation yields the result,

$$\frac{d^2}{d\omega^2}|H(\omega)|^2_{\omega=0} = -2\sum_{n=0}^{\infty} n^2 h_n = \frac{1}{3}(N-1)(N-2)$$

for the optimum zero-lag FIR filter of Eq. (24.4.4),

$$h_a(k) = \frac{2(2N-1-3k)}{N(N+1)}\,, \quad k = 0,1,\dots,N-1 \tag{24.10.6}$$

We note that the first derivative of the magnitude response $|H(\omega)|^2$ is always zero at DC, regardless of whether the filter has zero lag or not. Indeed, it follows from $H(0)= 1$ and the reality of h_n that,

$$\frac{d}{d\omega}|H(\omega)|^2_{\omega=0} = 2\,\mathrm{Re}\big[H'(0)\big] = 0 \tag{24.10.7}$$

Example 24.10.1: *Zero-Lag Filters.* In Problem 24.7, we saw that the double EMA filter has transfer function and magnitude response:

$$H_a(z) = \frac{(1-\lambda)(1+\lambda-2\lambda z^{-1})}{(1-\lambda z^{-1})^2}$$

$$|H_a(\omega)|^2 = \frac{(1-\lambda)^2\big[1+2\lambda+5\lambda^2-4\lambda(1+\lambda)\cos\omega\big]}{\big[1-2\lambda\cos\omega+\lambda^2\big]^2}$$

and that a secondary peak develops at,

$$\cos\omega_{\max} = \frac{1+4\lambda-\lambda^2}{2(1+\lambda)}\,, \quad |H_a(\omega_{\max})|^2 = \frac{(1+\lambda)^2}{1+2\lambda}$$

The left graph of Fig. 24.10.1 shows the magnitude responses for the two cases of $\lambda = 0.8$ and $\lambda = 0.9$. The calculated peak frequencies are $\omega_{\max} = 0.1492$ and $\omega_{\max} = 0.0726$ rads/sample, corresponding to periods of $2\pi/\omega_{\max} = 42$ and 86 samples/cycle. The peak points are indicated by black dots on the graph.

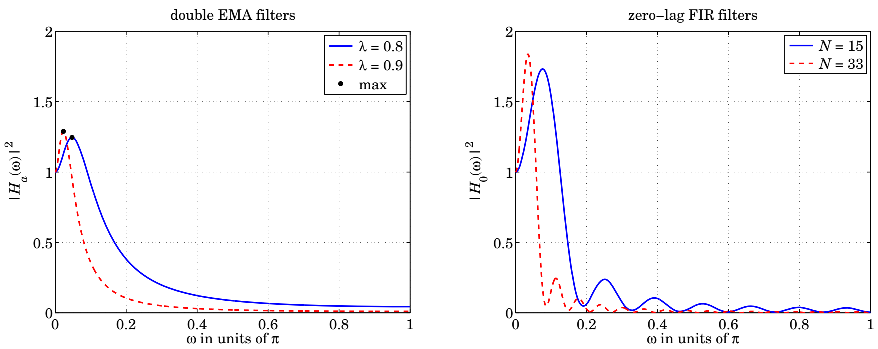


Fig. 24.10.1 Double EMA and zero-lag FIR filter responses.

The right graph shows the magnitude responses of the optimum zero-lag FIR filter $h_a(k)$ of Eq. (24.10.6) for the two lengths $N = 15$ and $N = 33$. The lengths N were calculated to achieve equivalent behavior in the vicinity of DC, i.e., equal second derivatives at $\omega = 0$,

$$\frac{4\lambda^2}{(1-\lambda)^2} = \frac{1}{3}(N-1)(N-2) \quad\Rightarrow\quad N = \frac{3}{2} + \sqrt{\frac{12\lambda^2}{(1-\lambda)^2} + \frac{1}{4}}$$

The magnitude responses were computed from the above formulas for the double EMA cases, and by the following MATLAB code in the FIR cases:

```
w = linspace(0,1,1001);                  % ω in units of π
N = 15; k = (0:N-1);
h = 2*(2*N-1-3*k)/N/(N+1);
H2 = abs(freqz(h,1,pi*w)).^2;            % magnitude response squared
```

We observe from these examples that the zero-lag filters have less than desirable passbands. However, they do act as lowpass filters, attenuating the high frequencies and thereby smoothing the input signal. □

24.11 Local Level, Local Slope, Local Acceleration Filters

Since the twicing operation can be applied to any lowpass filter $H(z)$ resulting in the zero-lag local-level filter, $H_a(z)= 2H(z)-H^2(z)$, it raises the question as to what would be the corresponding local-slope filter $H_b(z)$, in other words, what is the generalization of Eqs. (24.8.6) for an arbitrary filter $H(z)$, and similarly, what is the generalization of Eq. (24.8.12) for the thricing operations.

Starting with an arbitrary causal lowpass filter $H(z)$, with impulse response $h(n)$, and assuming that it has unity gain at DC, the local level, slope, and acceleration filters depend on the following two filter moments:

$$\mu_1 = \bar{n} = \sum_{n=0}^{\infty} n h(n)\,, \quad \mu_2 = \sum_{n=0}^{\infty} n^2 h(n) \tag{24.11.1}$$

In terms of these parameters, the generalization of Eq. (24.8.6) is then,

$$\boxed{\begin{aligned} H_a(z) &= 2H(z) - H^2(z) \\ H_b(z) &= \frac{1}{\mu_1}\big[H(z) - H^2(z)\big] \end{aligned}} \quad \begin{aligned} &\text{(local level filter)} \\ &\text{(local slope filter)} \end{aligned} \tag{24.11.2}$$

while the generalization of (24.8.12) is,

$$\begin{bmatrix} H_a(z) \\ H_b(z) \\ H_c(z) \end{bmatrix} = \frac{1}{2\mu_1^3}\begin{bmatrix} 6\mu_1^3 & -6\mu_1^3 & 2\mu_1^3 \\ \mu_2 + 4\mu_1^2 & -2(\mu_2 + 3\mu_1^2) & \mu_2 + 2\mu_1^2 \\ \mu_1 & -2\mu_1 & \mu_1 \end{bmatrix}\begin{bmatrix} H(z) \\ H^2(z) \\ H^3(z) \end{bmatrix} \tag{24.11.3}$$

and in particular,

$$H_a = 3H - 3H^2 + H^3 = 1 - (1 - H)^3 \tag{24.11.4}$$

For an EMA filter, $h(n) = (1-\lambda)\lambda^n u(n)$, we have,

$$\mu_1 = \frac{\lambda}{1-\lambda}\,, \quad \mu_2 = \frac{\lambda(1+\lambda)}{(1-\lambda)^2}$$

and Eqs. (24.11.2) and (24.11.3) reduce to (24.8.6) and (24.8.12), respectively. To justify (24.11.2), consider a noiseless straight-line input signal, $y_n = a + bn$. Since it is linearly rising and noiseless, the local-level will be itself, and the local-slope will be constant, that is, we may define,

$$a_n \equiv a + bn\,, \quad b_n \equiv b$$

Following the calculation leading to Eq. (24.1.24), we can easily verify that the two successive outputs, $a_n^{[1]}, a_n^{[2]}$, from the twice-cascaded filter $h(n)$, will be,

$$\begin{aligned} a_n^{[1]} &= a + b(n - \mu_1) = (a - \mu_1 b) + bn = a + bn - \mu_1 b = a_n - \mu_1 b_n \\ a_n^{[2]} &= (a - \mu_1 b - \mu_1 b) + bn = (a - 2\mu_1 b) + bn = a_n - 2\mu_1 b_n \end{aligned}$$

These may be solved for a_n, b_n in terms of $a_n^{[1]}, a_n^{[2]}$, leading to the following time-domain relationships, which are equivalent to Eqs. (24.11.2),

$$\begin{aligned} a_n &= 2a_n^{[1]} - a_n^{[2]} \\ b_n &= \frac{1}{\mu_1}\big(a_n^{[1]} - a_n^{[2]}\big) \end{aligned}$$

For the thricing case, consider a noiseless input that is quadratically varying with time, $y_n = a + bn + cn^2$, so that its local level, local slope, and local acceleration may be defined as,[†]

$$a_n \equiv a + bn + cn^2\,, \quad b_n \equiv b + 2cn\,, \quad c_n \equiv c$$

[†] true acceleration would be represented by $2c$.

Upon passing through the first stage of $h(n)$, the output will be,

$$\begin{aligned} a_n^{[1]} &= \sum_k \big[a + b(n-k) + c(n-k)^2\big]h(k) \\ &= \sum_k \big[a + b(n-k) + c(n^2 - 2nk + k^2)\big]h(k) \\ &= a + b(n - \mu_1) + c(n^2 - 2n\mu_1 + \mu_2) \\ &= (a - b\mu_1 + c\mu_2) + (b - 2c\mu_1)n + cn^2 \end{aligned}$$

and applying the same formula to the second and third stages of $h(n)$, we find the outputs,

$$\begin{aligned} a_n^{[1]} &= (a - b\mu_1 + c\mu_2) + (b - 2c\mu_1)n + cn^2 \\ a_n^{[2]} &= (a - 2b\mu_1 + 2c\mu_2 + 2c\mu_1^2) + (b - 4c\mu_1)n + cn^2 \\ a_n^{[3]} &= (a - 3b\mu_1 + 3c\mu_2 + 6c\mu_1^2) + (b - 6c\mu_1)n + cn^2 \end{aligned}$$

which can be re-expressed in terms of the a_n, b_n, c_n signals,

$$\begin{aligned} a_n^{[1]} &= a_n - \mu_1 b_n + \mu_2 c_n \\ a_n^{[2]} &= a_n - 2\mu_1 b_n + 2(\mu_2 + \mu_1^2)c_n \\ a_n^{[3]} &= a_n - 3\mu_1 b_n + 3(\mu_2 + 2\mu_1^2)c_n \end{aligned}$$

Solving these for a_n, b_n, c_n leads to the time-domain equivalent of Eq. (24.11.3),

$$\begin{bmatrix} a_n \\ b_n \\ c_n \end{bmatrix} = \frac{1}{2\mu_1^3}\begin{bmatrix} 6\mu_1^3 & -6\mu_1^3 & 2\mu_1^3 \\ \mu_2 + 4\mu_1^2 & -2(\mu_2 + 3\mu_1^2) & \mu_2 + 2\mu_1^2 \\ \mu_1 & -2\mu_1 & \mu_1 \end{bmatrix}\begin{bmatrix} a_n^{[1]} \\ a_n^{[2]} \\ a_n^{[3]} \end{bmatrix}$$

and in particular,

$$a_n = 3a_n^{[1]} - 3a_n^{[2]} + a_n^{[3]}$$

24.12 Basis Transformations and EMA Initialization

The transformation matrix between the $\mathbf{c}(n) = [c_0(n), c_1(n), \dots, c_d(n)]^T$ basis and the cascaded basis $\mathbf{a}(n) = [a_n^{[1]}, a_n^{[2]}, \dots, a_n^{[d+1]}]^T$ can be written in the general form:

$$\mathbf{a}(n) = M\mathbf{c}(n) \quad \Rightarrow \quad a_n^{[r]} = \sum_{i=0}^{d} M_{ri}c_i(n)\,, \quad r = 1, 2, \dots, d+1 \tag{24.12.1}$$

The matrix elements M_{ri} can be found by looking at the special case when the input is a polynomial of degree d,

$$x_{n+\tau} = \sum_{i=0}^{d} \tau^i c_i(n)$$

The convolutional output of the filter $H^{[r]}(z)$ is then,

$$a_n^{[r]} = \sum_{k=0}^{\infty} h^{[r]}(k)x_{n-k} = \sum_{k=0}^{\infty} h^{[r]}(k)\sum_{i=0}^{d}(-k)^i c_i(n)$$

It follows that,

$$M_{ri} = \sum_{k=0}^{\infty} h^{[r]}(k)(-k)^i = \sum_{k=0}^{\infty} \alpha^r\lambda^k \frac{(k+r-1)!}{k!(r-1)!}(-k)^i \tag{24.12.2}$$

with $1 \le r \le d+1$ and $0 \le i \le d$. The matrices for $d = 1$ and $d = 2$ in Eqs. (24.8.8) and (24.8.11) are special cases of (24.12.2). The function `emat` calculates M numerically,

```
M = emat(d,lambda);      % polynomial to cascaded basis transformation matrix
```

One of the uses of this matrix is to determine the initial condition vector in the $a_n^{[r]}$ basis, $\mathbf{a}_{\text{init}} = M\mathbf{c}_{\text{init}}$, where $\mathbf{c}_{\text{init}}$ is more easily determined, for example, using the default method of the function `stema`.

The function `mema` implements the multiple exponential moving average in cascade form, generating the individual outputs $a_n^{[r]}$:

```
[a,A] = mema(y,d,la,ainit);      % multiple exponential moving average
```

where `A` is an $N\times(d+1)$ matrix whose nth row is $\mathbf{a}(n)^T = [a_n^{[1]}, a_n^{[2]}, \dots, a_n^{[d+1]}]$, and `a` is the a_n output of Eq. (24.8.17). The $(d+1)\times 1$ vector `ainit` represents the initial values of $\mathbf{a}(n)$, that is, $\mathbf{a}_{\text{init}} = [a_{\text{init}}^{[1]}, a_{\text{init}}^{[2]}, \dots, a_{\text{init}}^{[d+1]}]^T$. If the argument `ainit` is omitted, it defaults to the following least-squares fitting procedure:

```
L = round((1+la)/(1-la));           % effective length of single EMA
cinit = lpbasis(L,d,-1) \ y(1:L);   % fit order-d polynomial to first L inputs
M = emat(d,la);                     % transformation matrix to cascaded basis
ainit = M*cinit;                    % (d+1)×1 initial vector
```

The function `mema` calculates the filter outputs using the built-in function `filter` to implement filtering by the single EMA filter $H(z) = \alpha/(1-\lambda z^{-1})$ and passing each output to the next stage, for example, with $a_n^{[0]} = y_n$,

$$a^{[r]} = \texttt{filter}(\alpha,\ [1,-\lambda],\ a^{[r-1]},\ \lambda a_{\text{init}}^{[r]}), \quad r = 1,2,\dots,d+1 \tag{24.12.3}$$

The last argument of `filter` imposes the proper initial state on the filtering operation (the particular form is dictated from the fact that `filter` uses the transposed realization.) Eq. (24.12.3) is equivalent to the operations:

$$a_n^{[r]} = \lambda a_{n-1}^{[r]} + \alpha a_n^{[r-1]}, \quad r = 1,2,\dots,d+1 \tag{24.12.4}$$

Example 24.12.1: *EMA Initialization.* To clarify these operations and the initializations, we consider a small example using $d = 1$ (double EMA) and $\lambda = 0.9$, $\alpha = 1-\lambda = 0.1$. The data vector $\mathbf{y}$ has length 41 and is given in the code segment below.

The top two graphs in Fig. 24.12.1 show the default initialization method in which a linear fit (because $d = 1$) is performed to the first $L = (1+\lambda)/(1-\lambda) = 19$ data samples. In the bottom two graphs, the initialization is based on performing the linear fit to just the first $L = 5$ samples.

In all cases, the linearly-fitted segments are shown on the graphs (short dashed lines). In the left graphs, the initialization parameters $\mathbf{c}_{\text{init}}, \mathbf{a}_{\text{init}}$ were determined at time $n = -1$ and processing began at $n = 0$. In the right graphs, the $\mathbf{c}_{\text{init}}, \mathbf{a}_{\text{init}}$ were recalculated to correspond to time $n = L-1$ (i.e., $n = 18$ and $n = 4$ for the top and bottom graphs), and processing was started at $n = L$. The table below displays the computations for the left and right bottom graphs.

For both the left and right tables, the same five data samples $\{y_n, 0 \le n \le 4\}$ were used to determine the initialization vectors $\mathbf{c}_{\text{init}}$, which were then mapped into $\mathbf{a}_{\text{init}} = M\mathbf{c}_{\text{init}}$. The transformation matrix M is in this example (cf. Eq. (24.8.8)):

$$M = \begin{bmatrix} 1 & -\lambda/\alpha \\ 1 & -2\lambda/\alpha \end{bmatrix} = \begin{bmatrix} 1 & -9 \\ 1 & -18 \end{bmatrix}$$

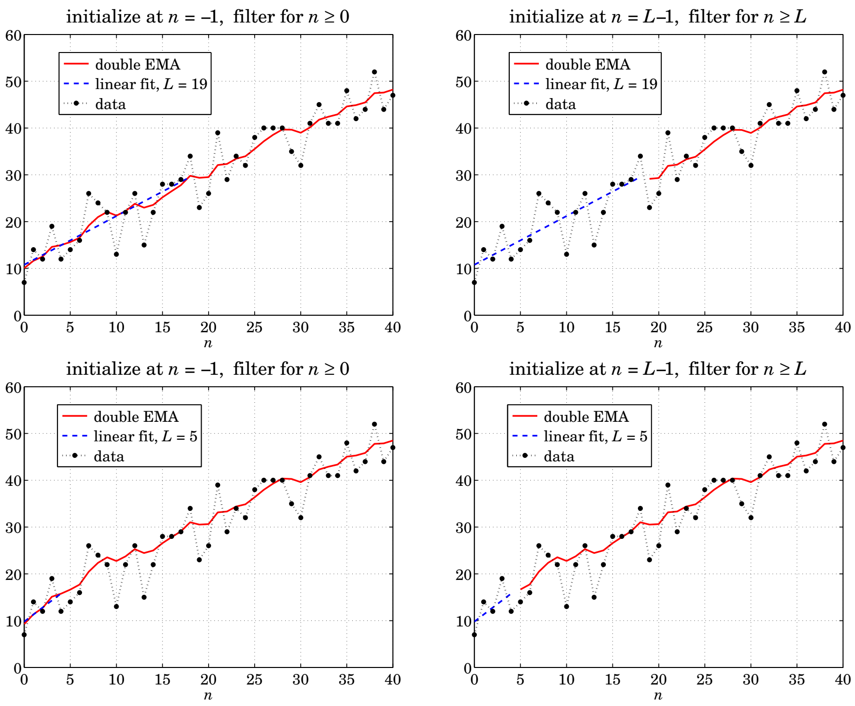


Fig. 24.12.1 Double-EMA initialization examples.

n	y_n	$a_n^{[1]}$	$a_n^{[2]}$
−1		−5.2000	−18.7000
0	7	−3.9800	−17.2280
1	14	−2.1820	−15.7234
2	12	−0.7638	−14.2274
3	19	1.2126	−12.6834
4	12	2.2913	−11.1860
5	14	3.4622	−9.7211
6	16	4.7160	−8.2774
⋮	⋮	⋮	⋮
39	44	39.2235	30.5592
40	47	40.0011	31.5034

n	y_n	$a_n^{[1]}$	$a_n^{[2]}$
−1			
0	7		
1	14		
2	12		
3	19		
4	12	2.3000	−11.2000
5	14	3.4700	−9.7330
6	16	4.7230	−8.2874
⋮	⋮	⋮	⋮
39	44	39.2237	30.5596
40	47	40.0013	31.5037

For the left table, the data fitting relative to $n = -1$ gives:

$$\mathbf{c}_{\text{init}} = \begin{bmatrix} 8.3 \\ 1.5 \end{bmatrix} \quad \Rightarrow \quad \mathbf{a}_{\text{init}} = M\mathbf{c}_{\text{init}} = \begin{bmatrix} 1 & -9 \\ 1 & -18 \end{bmatrix} \begin{bmatrix} 8.3 \\ 1.5 \end{bmatrix} = \begin{bmatrix} -5.2 \\ -18.7 \end{bmatrix}$$

obtained from `cinit = S\y(1:L)`, indeed, with $S = \text{lpbasis}(L, d, -1)$, we find

$$S = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \end{bmatrix} \Rightarrow \mathbf{c}_{\text{init}} = (S^T S)^{-1} S^T \mathbf{y}_{1:L} = \begin{bmatrix} 0.8 & 0.5 & 0.2 & -0.1 & -0.4 \\ -0.2 & -0.1 & 0.0 & 0.1 & 0.2 \end{bmatrix} \begin{bmatrix} 7 \\ 14 \\ 12 \\ 19 \\ 12 \end{bmatrix} = \begin{bmatrix} 8.3 \\ 1.5 \end{bmatrix}$$

These initial values are shown at the top of the left table. The rest of the table entries are computed by cranking the difference equations for $n \ge 0$,

$$a_n^{[1]} = \lambda a_{n-1}^{[1]} + \alpha y_n$$
$$a_n^{[2]} = \lambda a_{n-1}^{[2]} + \alpha a_n^{[1]}$$

for example,

$$a_0^{[1]} = \lambda a_{-1}^{[1]} + \alpha y_0 = (0.9)(-5.2) + (0.1)(7) = -3.980$$
$$a_0^{[2]} = \lambda a_{-1}^{[2]} + \alpha a_0^{[1]} = (0.9)(-18.7) + (0.1)(-3.98) = -17.228$$

For the right table, the initialization coefficients relative to $n = L-1 = 4$ may be determined by boosting those for $n = -1$ by $L = 5$ time units:

$$\bar{\mathbf{c}}_{\text{init}} = (F^T)^L \mathbf{c}_{\text{init}} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}^5 \begin{bmatrix} 8.3 \\ 1.5 \end{bmatrix} = \begin{bmatrix} 1 & 5 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 8.3 \\ 1.5 \end{bmatrix} = \begin{bmatrix} 15.8 \\ 1.5 \end{bmatrix}$$

$$\bar{\mathbf{a}}_{\text{init}} = M\bar{\mathbf{c}}_{\text{init}} = \begin{bmatrix} 1 & -9 \\ 1 & -18 \end{bmatrix} \begin{bmatrix} 15.8 \\ 1.5 \end{bmatrix} = \begin{bmatrix} 2.3 \\ -11.2 \end{bmatrix}$$

Alternatively, $\bar{\mathbf{c}}_{\text{init}}$ can be computed from `cinit = lpbasis(L,d,L-1)\y(1:L)`, i.e.,

$$S = \begin{bmatrix} 1 & -4 \\ 1 & -3 \\ 1 & -2 \\ 1 & -1 \\ 1 & 0 \end{bmatrix} \Rightarrow \bar{\mathbf{c}}_{\text{init}} = (S^T S)^{-1} S^T \mathbf{y}_{1:L} = \begin{bmatrix} -0.2 & 0.0 & 0.2 & 0.4 & 0.6 \\ -0.2 & -0.1 & 0.0 & 0.1 & 0.2 \end{bmatrix} \begin{bmatrix} 7 \\ 14 \\ 12 \\ 19 \\ 12 \end{bmatrix} = \begin{bmatrix} 15.8 \\ 1.5 \end{bmatrix}$$

The $\bar{\mathbf{a}}_{\text{init}}$ values are shown on the right table at the $n = 4$ line. The rest of the table is computed by cranking the above difference equations starting at $n = 5$. For example,

$$a_5^{[1]} = \lambda a_4^{[1]} + \alpha y_5 = (0.9)(2.3) + (0.1)(14) = 3.47$$
$$a_5^{[2]} = \lambda a_4^{[2]} + \alpha a_5^{[1]} = (0.9)(-11.2) + (0.1)(3.47) = -9.733$$

We note that the filtered values at $n = L - 1 = 4$ on the left table and the initial values on the right table are very close to each other, and therefore, the two initialization methods produce very comparable results for the output segments $n \ge L$. The following MATLAB code illustrates the generation of the bottom graphs in Fig. 24.12.1:

```
y = [ 7 14 12 19 12 14 16 26 24 22 13 22 26 15 22 28 28 29 34 23 26 ...
     39 29 34 32 38 40 40 40 35 32 41 45 41 41 48 42 44 52 44 47 ]';
n = (0:length(y)-1)';

d=1; F=binmat(d,1); L=5;                        % F = boost matrix - not needed
la = 0.9; al = 1-la;
% L = round((1+la)/(1-la));                     % use this L for the top two graphs

cinit = lpbasis(L,d,-1)\y(1:L);                 % fit relative to n = −1
M = emat(d,la);                                 % transformation matrix
ainit = M*cinit;                                % initial values for cascade realization

C = stema(y,d,la,cinit);                        % needed for testing purposes only
[a,A] = mema(y,d,la,ainit);                     % filter for n ≥  0

N1 = norm(A-C*M') + norm(a-C(:,1));             % compare stema and mema outputs

t = (0:L-1)'; p = lpbasis(L,d,-1)*cinit;        % initial L fitted values

figure; plot(n,y,'.', n,a,'-', t,p,'--', n,y,':');    % bottom left graph

cinit = lpbasis(L,d,L-1)\y(1:L);                % fit relative to n = L − 1
                                                % or, multiply previous cinit by (F')^L
ainit = M*cinit;                                % initial values for cascade realization

nL = n(L+1:end); yL = y(L+1:end);               % restrict input to n ≥  L

C = stema(yL,d,la,cinit);                       % needed for testing purposes only
[a,A] = mema(yL,d,la,ainit);                    % filter for n ≥  L

N2 = norm(A-C*M') + norm(a-C(:,1));             % compare stema and mema outputs

t = (0:L-1)'; p = lpbasis(L,d,L-1)*cinit;       % initial L fitted values

figure; plot(n,y,'.', nL,a,'-', t,p,'--', n,y,':');   % bottom right graph

Ntot = N1 + N2                                  % overall test – should be zero
```

The first initialization scheme (finding $\mathbf{c}_{\text{init}}, \mathbf{a}_{\text{init}}$ at $n = -1$ and starting filtering at $n = 0$) is generally preferable because it produces an output of the same length as the input. □

An alternative initialization scheme that is more common in financial market trading applications of EMA is discussed in Sec. 25.4.

24.13 Holt's Exponential Smoothing

We recall that the $d = 1$ steady-state EMA was given by

$$\begin{bmatrix} a_n \\ b_n \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a_{n-1} \\ b_{n-1} \end{bmatrix} + \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} (y_n - a_{n-1} - b_{n-1}) \tag{24.13.1}$$

with asymptotic gain factors $\alpha_1 = 1 - \lambda^2$ and $\alpha_2 = (1 - \lambda)^2$, which are both computable from a single λ parameter.

Holt [824] has generalized (24.13.1) to allow arbitrary values for α_1, α_2. The additional flexibility has been found to be effective in applications. There is an alternative way of writing (24.13.1). From the first equation, we have

$$a_n = a_{n-1} + b_{n-1} + \alpha_1 (y_n - a_{n-1} - b_{n-1}) \quad \Rightarrow \quad y_n - a_{n-1} - b_{n-1} = \frac{1}{\alpha_1}(a_n - a_{n-1} - b_{n-1})$$

and substituting into the second,

$$b_n = b_{n-1} + \alpha_2 (y_n - a_{n-1} - b_{n-1}) = b_{n-1} + \frac{\alpha_2}{\alpha_1}(a_n - a_{n-1} - b_{n-1})$$

Defining $\bar{\alpha}_2 = \alpha_2 / \alpha_1$, we may write the new system as follows:

$$\begin{aligned} a_n &= a_{n-1} + b_{n-1} + \alpha_1 (y_n - a_{n-1} - b_{n-1}) \\ b_n &= b_{n-1} + \bar{\alpha}_2 (a_n - a_{n-1} - b_{n-1}) \end{aligned} \tag{24.13.2}$$

and defining the effective λ-parameters $\lambda_1 = 1 - \alpha_1$ and $\bar{\lambda}_2 = 1 - \bar{\alpha}_2$,

$$\boxed{\begin{aligned} a_n &= \lambda_1 (a_{n-1} + b_{n-1}) + \alpha_1 y_n \\ b_n &= \bar{\lambda}_2 b_{n-1} + \bar{\alpha}_2 (a_n - a_{n-1}) \end{aligned}} \quad \text{(Holt's exponential smoothing)} \tag{24.13.3}$$

Eq. (24.13.1) is referred to a exponential smoothing with "local trend". The first equation tracks the local level a_n, and the second, the local slope b_n, which is being determined by smoothing the first-order difference of the local level $a_n - a_{n-1}$.

The predicted value is as usual $\hat{y}_{n/n-1} = a_{n-1} + b_{n-1}$, and for the next time instant, $\hat{y}_{n+1/n} = a_n + b_n$, and τ steps ahead, $\hat{y}_{n+\tau/n} = a_n + b_n \tau$. The MATLAB function `holt` implements Eq. (24.13.1):

```
C = holt(y,a1,a2,cinit);          % Holt's exponential smoothing
```

where C has the same meaning as `stema`, its nth row $\mathbf{c}^T(n) = [a_n, b_n]$ holding the local level and slope at time n. The initialization vector `cinit` can be chosen as in `stema` by a linear fit to the first L samples of $\mathbf{y}$, where $L = (1 + \lambda)/(1 - \lambda)$, with λ determined from α_1 from the relationship $\alpha_1 = 1 - \lambda^2$ or $\lambda = \sqrt{1 - \alpha_1}$. Another possibility is to choose $\mathbf{c}_{\text{init}} = [y_0, 0]^T$, or, $[y_0, y_1 - y_0]^T$.

Like `emaerr`, the MATLAB function **holterr** evaluates the MSE/MAE/MAPE errors over any set of parameter pairs (α_1, α_2) and produces the corresponding optimum pair $(\alpha_{1,\text{opt}}, \alpha_{2,\text{opt}})$:

```
[err,a1opt,a2opt] = holterr(y,a1,a2,type,cinit);          % mean error measures
```

By taking z-transforms of Eq. (24.13.1), we obtain the transfer functions from y_n to the two outputs a_n, b_n:

$$\begin{aligned} H_a(z) &= \frac{\alpha_1 + (\alpha_2 - \alpha_1) z^{-1}}{1 + (\alpha_1 + \alpha_2 - 2) z^{-1} + (1 - \alpha_1) z^{-2}} \\ H_b(z) &= \frac{\alpha_2 (1 - z^{-1})}{1 + (\alpha_1 + \alpha_2 - 2) z^{-1} + (1 - \alpha_1) z^{-2}} \end{aligned} \tag{24.13.4}$$

The transfer function from y_n to the predicted output $\hat{y}_{n+1/n}$ is $H(z) = H_a(z) + H_b(z)$. Making the usual assumption that y_n is a white noise sequence, the variance of $\hat{y}_{n+1/n}$ will be $\sigma_{\hat{y}}^2 = \mathcal{R} \sigma_y^2$, where $\mathcal{R}$ is the NRR of $H(z)$:

$$\mathcal{R} = \sum_{n=0}^{\infty} h^2(n) = \frac{2\alpha_1^2 + \alpha_1 \alpha_2 + 2\alpha_2}{\alpha_1 (4 - 2\alpha_1 - \alpha_2)} \tag{24.13.5}$$

This reduces to Eq. (24.7.9) in the EMA case of $\alpha_1 = 1 - \lambda^2$ and $\alpha_2 = (1 - \lambda)^2$, while Eq. (24.13.4) reduces to (24.8.5). It can be shown that $\mathcal{R}$ remains less than unity for $0 \le \alpha_1 \le 1$ and $0 \le \alpha_2 \le 2\alpha_1 (1 - \alpha_1)/(1 + \alpha_1)$, with R reaching unity at $\alpha_1 = \sqrt{2} - 1 = 0.4142$ and $\alpha_2 = 2\alpha_1 (1 - \alpha_1)/(1 + \alpha_1) = 2(3 - 2\sqrt{2}) = 0.3431$.

There is an extensive literature on exponential smoothing, a small subset of which is [816–863]. There are many other variants (no less than 15), such as multiplicative, seasonal, adaptive versions. A recent review of all cases that emphasizes the state-space point of view is found in [823].

We finish by mentioning the Holt-Winters generalization [825] of Holt's method to seasonal data. In addition to tracking the level and slope signals a_n, b_n the method also tracks the local seasonal component, say s_n. For the additive version, we have:

$$\boxed{\begin{aligned} a_n &= \lambda_1 (a_{n-1} + b_{n-1}) + \alpha_1 (y_n - s_{n-D}) \\ b_n &= \bar{\lambda}_2 b_{n-1} + \bar{\alpha}_2 (a_n - a_{n-1}) \\ s_n &= \lambda_3 s_{n-D} + \alpha_3 (y_n - a_{n-1} - b_{n-1}) \end{aligned}} \quad \text{(Holt-Winters)} \tag{24.13.6}$$

where D is the assumed periodicity of the seasonal data, and α_3 and $\lambda_3 = 1 - \alpha_3$ are the smoothing parameters associated with the seasonal component. The predicted estimate is obtained by

$$\hat{y}_{n+1/n} = a_n + b_n + s_{n-D}$$

24.14 Problems

24.1 Consider a filter with a real-valued impulse response h_n. Let $H(\omega)=M(\omega)e^{-j\theta(\omega)}$ be its frequency response, where $M(\omega)=|H(\omega)|$ and $\theta(\omega)=-\arg H(\omega)$. First, argue that $\theta(0)=0$ and $M'(0)=0$, where $M'(\omega)=dM(\omega)/d\omega$. Then, show that the filter delay $\bar{n}$ of Eq. (24.1.18) is the group delay at DC, that is, show Eq. (24.1.19),

$$\bar{n}=\frac{d\theta(\omega)}{d\omega}\bigg|_{\omega=0}$$

24.2 The lag of a filter was defined by Eqs. (24.1.17) and (24.1.18) to be,

$$\bar{n}=\frac{\sum_n nh_n}{\sum_n h_n}=-\frac{H'(z)}{H(z)}\bigg|_{z=1}$$

If the filter $H(z)$ is the cascade of two filters, $H(z)=H_1(z)H_2(z)$, with individual lags, $\bar{n}_1,\bar{n}_2$, then show that, regardless of whether $H_1(z),H_2(z)$ are normalized to unity gain at DC, the lag of $H(z)$ will be the sum of the lags,

$$\bar{n}=\bar{n}_1+\bar{n}_2$$

24.3 Consider a low-frequency signal $s(n)$ whose spectrum $S(\omega)$ is limited within a narrow band around DC, $|\omega|\le\Delta\omega$, and therefore, its inverse DTFT representation is:

$$s(n)=\frac{1}{2\pi}\int_{-\Delta\omega}^{\Delta\omega}S(\omega)e^{j\omega n}\,d\omega$$

For the purposes of this problem, we may think of the above relationship as defining $s(n)$ also for non-integer values of n. Suppose that the signal $s(n)$ is filtered through a filter $H(\omega)$ with real-valued impulse response whose magnitude response $|H(\omega)|$ is approximately equal to unity over the $\pm\Delta\omega$ signal bandwidth. Show that the filtered output can be written approximately as the delayed version of the input by an amount equal to the group delay at DC, that is,

$$y(n)=\frac{1}{2\pi}\int_{-\Delta\omega}^{\Delta\omega}H(\omega)S(\omega)e^{j\omega n}\,d\omega\approx s(n-\bar{n})$$

24.4 Show that the general filter-sharpening formula (24.9.5) results in the following special cases:

$$p=0,\ q=d\quad\Rightarrow\quad H_{\text{impr}}=1-(1-H)^{d+1}$$

$$p=1,\ q=d\quad\Rightarrow\quad H_{\text{impr}}=1-(1-H)^{d+1}\big[1+(d+1)H\big]$$

24.5 Prove the formulas in Eqs. (24.10.5) and (24.10.7).

24.6 Prove Eq. (24.4.10).

24.7 Consider the single and double EMA filters:

$$H(z)=\frac{1-\lambda}{1-\lambda z^{-1}}\,,\quad H_a(z)=2H(z)-H^2(z)=\frac{(1-\lambda)(1+\lambda-2\lambda z^{-1})}{(1-\lambda z^{-1})^2}$$

a. Show that the impulse response of $H_a(z)$ is:

$$h_a(n)=(1-\lambda)\big[1+\lambda-(1-\lambda)n\big]\lambda^n u(n)$$

b. Show the relationships:

$$\sum_{n=0}^{\infty}nh_a(n)=0,\quad \sum_{n=0}^{\infty}n^2h_a(n)=-\frac{2\lambda^2}{(1-\lambda)^2} \tag{24.14.1}$$

c. Show that the NRR of the filter $H_a(z)$ is:

$$\mathcal{R}=\sum_{n=0}^{\infty}h_a^2(n)=\frac{(1-\lambda)(1+4\lambda+5\lambda^2)}{(1+\lambda)^3}$$

d. Show that the magnitude response squared of $H_a(z)$ is:

$$|H_a(\omega)|^2=\frac{(1-\lambda)^2\big[1+2\lambda+5\lambda^2-4\lambda(1+\lambda)\cos\omega\big]}{\big[1-2\lambda\cos\omega+\lambda^2\big]^2} \tag{24.14.2}$$

e. Show that Eq. (24.14.2) has local minima at $\omega=0$ and $\omega=\pi$, and a local maximum at $\omega=\omega_{\max}$:

$$\cos\omega_{\max}=\frac{1+4\lambda-\lambda^2}{2(1+\lambda)} \tag{24.14.3}$$

and that the corresponding extremal values are:

$$\begin{aligned}|H_a(0)|^2&=1\,,\quad |H_a(\pi)|^2=\frac{(1-\lambda)^2(1+3\lambda)^2}{(1+\lambda)^4}\\ |H_a(\omega_{\max})|^2&=\frac{(1+\lambda)^2}{1+2\lambda}\end{aligned} \tag{24.14.4}$$

24.8 Consider the modified EMA of Eq. (15.2.2) and its twicing,

$$H(z)=\frac{(1-\lambda)(1+z^{-1})}{2(1-\lambda z^{-1})}\,,\quad H_a(z)=2H(z)-H^2(z)=\frac{(1-\lambda)(1+z^{-1})\big(3+\lambda-z^{-1}(1+3\lambda)\big)}{4(1-\lambda z^{-1})^2}$$

a. Show the relationships:

$$\sum_{n=0}^{\infty}nh_a(n)=0,\quad \sum_{n=0}^{\infty}n^2h_a(n)=-\frac{(1+\lambda)^2}{2(1-\lambda)^2}$$

b. Show that the NRR of the filter $H_a(z)$ is:

$$\mathcal{R}=\sum_{n=0}^{\infty}h_a^2(n)=\frac{1}{8}(1-\lambda)(3\lambda+7)$$

24.9 Consider the optimum length-N predictive FIR filter $h_\tau(k)$ of polynomial order $d=1$ given by Eq. (24.4.1).

a. Show that its effective lag is related to the prediction distance τ by $\bar{n}=-\tau$.

b. Show that its NRR is given by

$$\mathcal{R}=\sum_{k=0}^{N-1}h_\tau^2(k)=\frac{1}{N}+\frac{3(N-1+2\tau)^2}{N(N^2-1)}$$

Thus, it is minimized when $\tau=-(N-1)/2$. What is the filter $h_\tau(k)$ in this case?

c. Show that the second-derivative of its frequency response at DC is given by:

$$\frac{d^2}{d\omega^2}H(\omega)_{\omega=0} = -\sum_{n=0}^{\infty} k^2 h_\tau(k) = \frac{1}{6}(N-1)(N-2+6\tau)$$

Determine the range of τs for which $\left|H(\omega)\right|^2$ is sloping upwards or downwards in the immediate vicinity of $\omega = 0$.

d. It is evident from the previous question that the value $\tau = -(N-2)/6$ corresponds to the vanishing of the second-derivative of the magnitude response. Show that in this case the filter is simply,

$$h(k) = \frac{3N(N-1)-2k(2N-1)}{N(N^2-1)}\,, \quad k = 0,1,\dots,N-1$$

and verify explicitly the results:

$$\sum_{k=0}^{N-1} h(k) = 1\,, \quad \sum_{k=0}^{N-1} kh(k) = \frac{N-2}{6}\,, \quad \sum_{k=0}^{N-1} k^2 h(k) = 0\,, \quad \sum_{k=0}^{N-1} h^2(k) = \frac{7N^2-4N-2}{3N(N^2-1)}$$

e. Show that $h_\tau(k)$ interpolates linearly between the $\tau = 0$ and $\tau = 1$ filters, that is, show that for $k = 0,1,\dots,N-1$,

$$h_\tau(k) = (1-\tau)h_a(k) + \tau h_1(k) = h_a(k) + \big[h_1(k) - h_a(k)\big]\tau$$

f. Another popular choice for the delay parameter is $\tau = -(N-1)/3$. Show that,

$$h(k) = \frac{2(N-k)}{N(N+1)}\,, \quad k = 0,1,\dots,N-1$$

and that,

$$\sum_{k=0}^{N-1} h(k) = 1\,, \quad \sum_{k=0}^{N-1} kh(k) = \frac{N-1}{3}\,, \quad \sum_{k=0}^{N-1} k^2 h(k) = \frac{N(N-1)}{6}\,, \quad \sum_{k=0}^{N-1} h^2(k) = \frac{2(2N+1)}{3N(N+1)}$$

In financial market trading, the cases $\tau = -(N-1)/2$ and $\tau = -(N-1)/3$ correspond, respectively, to the so-called “simple” and “weighted” moving average indicators. The case $\tau = -(N-2)/6$ is not currently used, but it provides a useful compromise between reducing the lag while preserving the flatness of the passband. By comparison, the relative lags of three cases are:

$$\frac{1}{6}(N-2) < \frac{1}{3}(N-1) < \frac{1}{2}(N-1)$$

24.10 *Computer Experiment: Response of predictive FIR filters.* Consider the predictive FIR filter $h_\tau(k)$ of the previous problem. For $N = 9$, compute and on the same graph plot the magnitude responses $|H(\omega)|^2$ for the following values of the prediction distance:

$$\tau = -\frac{N-1}{2}\,, \quad \tau = -\frac{N-2}{6}\,, \quad \tau = 0\,, \quad \tau = 1$$

Using the calculated impulse response values $h_\tau(k)$, $0 \le k \le N-1$, and for each value of τ, calculate the filter lag, $\bar{n}$, the NRR, $\mathcal{R}$, and the “curvature” parameter of Eq. (24.10.5). Recall from part (d) of the Problem 24.9 that the second τ should result in zero curvature.

Repeat all the questions for $N = 18$.

24.11 *Moving-Average Filters with Prescribed Moments.* The predictive FIR filter of Eq. (25.3.3) has lag equal to $\bar{n} = -\tau$ by design. Show that its second moment is not independently specified but is given by,

$$\overline{n^2} = \sum_{n=0}^{N-1} n^2 h(n) = -\frac{1}{6}(N-1)(N-2+6\tau) \qquad (24.14.5)$$

The construction of the predictive filters (25.3.3) can be generalized to allow arbitrary specification of the first and second moments, that is, the problem is to design a length-N FIR filter with the prescribed moments,

$$\overline{n^0} = \sum_{n=0}^{N-1} h(n) = 1\,, \quad \overline{n^1} = \sum_{n=0}^{N-1} nh(n) = -\tau_1\,, \quad \overline{n^2} = \sum_{n=0}^{N-1} n^2 h(n) = \tau_2 \qquad (24.14.6)$$

Show that such filter is given by an expression of the form,

$$h(n) = c_0 + c_1 n + c_2 n^2\,, \quad n = 0,1,\dots,N-1$$

where the coefficients c_0, c_1, c_2 are the solutions of the linear system,

$$\begin{bmatrix} S_0 & S_1 & S_2 \\ S_1 & S_2 & S_3 \\ S_2 & S_3 & S_4 \end{bmatrix}\begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -\tau_1 \\ \tau_2 \end{bmatrix}$$

where

$$S_p = \sum_{n=0}^{N-1} n^p\,, \quad p = 0,1,2,3,4$$

Then, show that the S_p are given explicitly by,

$$S_0 = N\,, \quad S_1 = \frac{1}{2}N(N-1)\,, \quad S_2 = \frac{1}{6}N(N-1)(2N-1)$$

$$S_3 = \frac{1}{4}N^2(N-1)^2\,, \quad S_4 = \frac{1}{30}N(N-1)(2N-1)(3N^2-3N-1)$$

and that the coefficients are given by,

$$c_0 = \frac{3(3N^2-3N+2) + 18(2N-1)\tau_1 + 30\tau_2}{N(N+1)(N+2)}$$

$$c_1 = -\frac{18(N-1)(N-2)(2N-1) + 12(2N-1)(8N-11)\tau_1 + 180(N-1)\tau_2}{N(N^2-1)(N^2-4)}$$

$$c_2 = \frac{30(N-1)(N-2) + 180(N-1)\tau_1 + 180\tau_2}{N(N^2-1)(N^2-4)}$$

Finally, show that the condition $c_2 = 0$ recovers the predictive FIR case of Eq. (25.3.3) with second moment given by Eq. (24.14.5).

24.12 Consider the Butterworth filter of Eq. (25.7.2). Show that the lag of the first-order section and the lag of the ith second-order section are given by,

$$\bar{n}_0 = \frac{1}{2\Omega_0}\,, \quad \bar{n}_i = -\frac{\cos\theta_i}{\Omega_0}\,, \quad i = 1,2,\dots,K$$

Using these results, prove Eq. (25.7.8) for the full lag $\bar{n}$, and show that it is valid for both even and odd filter orders M.

25

Filtering Methods in Financial Markets

25.1 Technical Analysis of Financial Markets

Technical analysis of financial markets refers to a family of signal processing methods and indicators used by stock market traders to make sense of the constantly fluctuating market data and arrive at successful "buy" or "sell" decisions.

Both linear and nonlinear filtering methods are used. A comprehensive reference on such methods is the Achelis book [864]. Some additional references are [865–928,935–937].

In this chapter,[†] we look briefly at some widely used indicators arising from FIR or EMA filters, and summarize their properties and their MATLAB implementation. In order to keep the discussion self-contained, some material from the previous sections is repeated.

25.2 Moving Average Filters – SMA, WMA, TMA, EMA

Among the linear filtering methods are smoothing filters that are used to smooth out the daily fluctuations and bring out the trends in the data. The most common filters are the *simple moving average* (SMA) and the *exponentially weighted moving average* (EMA), and variations, such as the *weighted or linear moving average* (WMA) and the *triangular moving average* (TMA). The impulse responses of these filters are:

$$
\begin{aligned}
&(\text{SMA}) && h(n) = \frac{1}{N}\,, \quad 0 \le n \le N-1 \\
&(\text{WMA}) && h(n) = \frac{2(N-n)}{N(N+1)}\,, \quad 0 \le n \le N-1 \\
&(\text{TMA}) && h(n) = \frac{N - |n-N+1|}{N^2}\,, \quad 0 \le n \le 2N-2 \\
&(\text{EMA}) && h(n) = (1-\lambda)\lambda^n\,, \quad 0 \le n < \infty
\end{aligned}
\tag{25.2.1}
$$

[†] adapted from the author's book on *Applied Optimum Signal Processing* [45]

with transfer functions,

$$
\begin{aligned}
&(\text{SMA}) && H(z) = \frac{1 + z^{-1} + z^{-2} + \cdots + z^{-N+1}}{N} = \frac{1}{N}\,\frac{1-z^{-N}}{1-z^{-1}} \\
&(\text{WMA}) && H(z) = \frac{2}{N(N+1)}\,\frac{N-(N+1)z^{-1} + z^{-N-1}}{(1-z^{-1})^2} \\
&(\text{TMA}) && H(z) = \left[\frac{1}{N}\,\frac{1-z^{-N}}{1-z^{-1}}\right]^2 \\
&(\text{EMA}) && H(z) = \frac{\alpha}{1-\lambda z^{-1}}\,, \quad \alpha = 1-\lambda
\end{aligned}
\tag{25.2.2}
$$

where N denotes the filter span for the SMA and WMA cases, while for the EMA case, λ is a forgetting factor such that $0 < \lambda < 1$, which is usually specified in terms an equivalent FIR length N given by the following condition, which implies that the SMA and the EMA filters have the same lag and the same noise reduction ratio, as discussed in Sec. 24.1,

$$
N = \frac{1+\lambda}{1-\lambda} \quad \Rightarrow \quad \lambda = \frac{N-1}{N+1} \quad \Rightarrow \quad \alpha = 1-\lambda = \frac{2}{N+1}
\tag{25.2.3}
$$

The TMA filter has length $2N-1$ and is evidently the convolution of two length-N SMAs. Denoting by y_n the raw data, where n represents the nth trading day (or, weekly, monthly, or quarterly sample periods), we will denote the output of the moving average filters by a_n representing the smoothed *local level* of y_n. The corresponding I/O filtering equations are then,

$$
\begin{aligned}
&(\text{SMA}) && a_n = \frac{y_n + y_{n-1} + y_{n-2} + \cdots + y_{n-N+1}}{N} \\
&(\text{WMA}) && a_n = \frac{2}{N(N+1)} \sum_{k=0}^{N-1} (N-k)\, y_{n-k} \\
&(\text{TMA}) && a_n = \frac{1}{N^2} \sum_{k=0}^{2N-2} \big(N - |k-N+1|\big)\, y_{n-k} \\
&(\text{EMA}) && a_n = \lambda\, a_{n-1} + (1-\lambda)\, y_n
\end{aligned}
\tag{25.2.4}
$$

The typical *trading rule* used by traders is to "buy" when a_n is rising and y_n lies above a_n, and to "sell" when a_n is falling and y_n lies below a_n.

Unfortunately, these widely used filters have an inherent lag, which can often result in false buy/sell signals. The basic tradeoff is that longer lengths N result in longer lags, but at the same time, the filters become more effective in smoothing and reducing noise in the data. The noise-reduction capability of any filter is quantified by its "noise-reduction ratio" defined by,

$$
\mathcal{R} = \sum_{n=0}^{\infty} h^2(n)
\tag{25.2.5}
$$

with smaller $\mathcal{R}$ corresponding to more effective noise reduction. By construction, the above filters are lowpass filters with unity gain at DC, therefore, satisfying the constraint,

$$\sum_{n=0}^{\infty} h(n) = 1 \tag{25.2.6}$$

The "lag" is defined as the group delay at DC which, after using Eq. (25.2.6), is given by,

$$\bar{n} = \sum_{n=0}^{\infty} n\,h(n) \tag{25.2.7}$$

One can easily verify that the noise-reduction ratios and lags of the above filters are:

$$\begin{array}{llll}
\text{(SMA)} & \mathcal{R} = \dfrac{1}{N}\,, & \bar{n} = \dfrac{N-1}{2} & \\[2ex]
\text{(WMA)} & \mathcal{R} = \dfrac{4N+2}{3N(N+1)}\,, & \bar{n} = \dfrac{N-1}{3} & \\[2ex]
\text{(TMA)} & \mathcal{R} = \dfrac{2N^2+1}{3N^3}\,, & \bar{n} = N-1 & \\[2ex]
\text{(EMA)} & \mathcal{R} = \dfrac{1-\lambda}{1+\lambda} = \dfrac{1}{N}\,, & \bar{n} = \dfrac{\lambda}{1-\lambda} = \dfrac{N-1}{2}\,, & \text{for equivalent } N
\end{array} \tag{25.2.8}$$

The tradeoff is evident, with $\mathcal{R}$ decreasing and $\bar{n}$ increasing with N.

We include one more lowpass smoothing filter, the *integrated linear regression slope* (ILRS) filter [891] which is developed in Sec. 25.3. It has unity DC gain and its impulse response, transfer function, lag, and NRR are given by,

$$\begin{aligned}
\text{(IRLS)}\quad h(n) &= \frac{6(n+1)(N-1-n)}{N(N^2-1)}\,, \quad n = 0,1,\dots,N-1 \\
H(z) &= \frac{6}{N(N^2-1)}\,\frac{N(1-z^{-1})(1+z^{-N})-(1-z^{-N})(1+z^{-1})}{(1-z^{-1})^3} \\
\bar{n} &= \frac{N-2}{2}\,, \quad \mathcal{R} = \frac{6(N^2+1)}{5N(N^2-1)}
\end{aligned} \tag{25.2.9}$$

Fig. 25.2.1 compares the frequency responses of the above filters. We note that the ILRS has similar bandwidth as the WMA, but it also has a smaller NRR and more suppressed high-frequency range, thus, resulting in smoother output.

As a small example, we also give for comparison the impulse responses, $\mathbf{h} = [h_0, h_1, \dots]$, of the SMA, WMA, TMA, and ILRS filters for the case $N = 5$,

$$\begin{array}{ll}
\text{(SMA)} & \mathbf{h} = \dfrac{1}{5}[1,\,1,\,1,\,1,\,1] \\[2ex]
\text{(WMA)} & \mathbf{h} = \dfrac{1}{15}[5,\,4,\,3,\,2,\,1] \\[2ex]
\text{(TMA)} & \mathbf{h} = \dfrac{1}{25}[1,\,2,\,3,\,4,\,5,\,4,\,3,\,2,\,1] \\[2ex]
\text{(ILRS)} & \mathbf{h} = \dfrac{1}{10}[2,\,3,\,3,\,2,\,0]
\end{array}$$

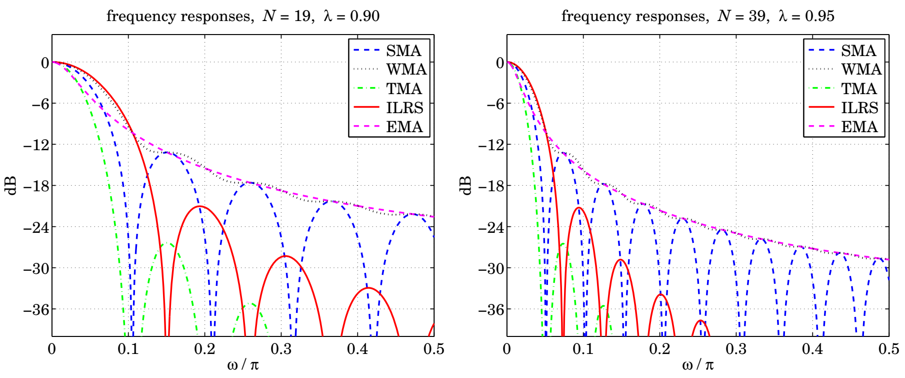


Fig. 25.2.1 Frequency responses of SMA, WMA, TMA, ILRS, and EMA filters.

with the SMA having constant weights, the WMA having linearly decreasing weights, the TMA has triangular weights, and the last coefficient h_{N-1} of the ILRS always being zero.

The following MATLAB functions implement the SMA, WMA, TMA, and ILRS moving averages. The input array $\mathbf{y}$ represents the financial data to be filtered.

```
a = sma(y,N,yin);    % simple moving average
a = wma(y,N,yin);    % weighted moving average
a = tma(y,N,yin);    % triangular moving average
a = ilrs(y,N,yin);   % integrated linear regression slope
```

The string variable yin specifies the way the filters are initialized and can take on the following values as explained further in Sec. 25.6,

```
yin = 'f',     % progressive filtering (default method)
yin = 'n',     % initial transients are NaNs (facilitates plotting)
yin = 'c',     % standard convolutional transients
```

Some comparisons of these and other moving average indicators are presented in Figures 25.5.2 and 25.8.1.

25.3 Predictive Moving Average Filters

The *predictive FIR* and *double EMA* filters discussed in Sects. 24.4 and 24.8 find application in stock market trading. Their main property is the elimination or shortening of the lag, which is accomplished by tracking both the local level and the local slope of the data. More discussion of these filters and their application in the trading context may be found in Refs. [881–892].

The local-level and local-slope FIR filters $h_a(k)$ and $h_b(k)$ were given in Eq. (24.4.4), and their filtering equations by (24.4.5). They define the following market indicators:

$$\boxed{\begin{aligned} a_n &= \sum_{k=0}^{N-1} h_a(k)\,y_{n-k} = \text{linear regression indicator}\\ b_n &= \sum_{k=0}^{N-1} h_b(k)\,y_{n-k} = \text{linear regression slope indicator}\\ a_n + b_n &= \sum_{k=0}^{N-1} h_1(k)\,y_{n-k} = \text{time-series forecast indicator} \end{aligned}} \tag{25.3.1}$$

where $h_1(k) = h_a(k) + h_b(k)$. The quantity $a_n + b_n$, denoted by $\hat{y}_{n+1/n}$, represents the one-step ahead forecast or prediction to time $n+1$ based on the data up to time n. More generally, the prediction τ steps ahead from time n is given by the following indicator, which we will refer to as the *predictive moving average* (PMA),

$$\boxed{\hat{y}_{n+\tau/n} = a_n + \tau b_n = \sum_{k=0}^{N-1} h_\tau(k)\,y_{n-k} \quad \text{(PMA)}} \tag{25.3.2}$$

where, as follows from Eq. (24.4.4), we have for $n = 0, 1, \dots, N-1$,

$$\boxed{h_\tau(n) = h_a(n) + \tau h_b(n) = \frac{2(2N-1-3n)}{N(N+1)} + \tau\,\frac{6(N-1-2n)}{N(N^2-1)}} \tag{25.3.3}$$

The time "advance" τ can be non-integer, positive, or negative. Positive τs correspond to forecasting, negative τs to delay or lag. In fact, the SMA and WMA are special cases of Eq. (25.3.3) for the particular choices of $\tau = -(N-1)/2$ and $\tau = -(N-1)/3$, respectively.

The phrase "linear regression indicators" is justified in Sec. 25.5. The filters $h_\tau(n)$ are very flexible and useful in the trading context, and are actually the *optimal filters* that have *minimum noise-reduction ratio* subject to the two constraints of having unity DC gain and lag equal to $-\tau$, that is, for fixed N, $h_\tau(n)$ is the solution of the optimization problem (for $N = 1$, we ignore the lag constraint to get, $h_\tau(n) = 1$, for $n = 0$, and all τ):

$$\mathcal{R} = \sum_{n=0}^{N-1} h^2(n) = \min, \qquad \text{subject to} \quad \sum_{n=0}^{N-1} h(n) = 1\,, \quad \sum_{n=0}^{N-1} n\,h(n) = -\tau \tag{25.3.4}$$

This was solved in Sec. 24.4. The noise-reduction-ratio of these filters is,

$$\mathcal{R}_\tau = \sum_{n=0}^{N-1} h_\tau^2(n) = \frac{1}{N} + \frac{3(N-1+2\tau)^2}{N(N^2-1)} \tag{25.3.5}$$

We note the two special cases, first for the SMA filter having $\tau = -(N-1)/2$, and second, for the zero-lag filter $h_a(n)$ having $\tau = 0$,

$$\mathcal{R}_{\text{SMA}} = \frac{1}{N}\,, \quad \mathcal{R}_a = \frac{4N-2}{N(N+1)}$$

The transfer functions of the FIR filters $h_a(n), h_b(n)$ are not particularly illuminating, however, they are given below in rational form,

$$H_a(z) = \sum_{n=0}^{N-1} h_a(n) z^{-n} = \frac{2}{N(N+1)}\;\frac{N(1-z^{-1})(2+z^{-N}) - (1+2z^{-1})(1-z^{-N})}{(1-z^{-1})^2}$$

$$H_b(z) = \sum_{n=0}^{N-1} h_b(n) z^{-n} = \frac{6}{N(N^2-1)}\;\frac{N(1-z^{-1})(1+z^{-N}) - (1+z^{-1})(1-z^{-N})}{(1-z^{-1})^2}$$

By a proper limiting procedure, one can easily verify the unity-gain and zero-lag properties, $H_a(z)\big|_{z=1} = 1$, and, $\bar{n} = -H_a'(z)\big|_{z=1} = 0$.

The ILRS filter mentioned in the previous section is defined as the *integration*, or cumulative sum, of the slope filter, which can be evaluated explicitly resulting in (25.2.9),

$$h(n) = \sum_{k=0}^{n} h_b(k) = \sum_{k=0}^{n} \frac{6(N-1-2k)}{N(N^2-1)} = \frac{6(n+1)(N-1-n)}{N(N^2-1)} \tag{25.3.6}$$

where $0 \le n \le N-1$. For $n > N$, since $h_b(k)$ has duration N, the above sum remains constant and equal to zero, i.e., equal to its final value,

$$\sum_{k=0}^{N-1} h_b(k) = 0$$

The corresponding transfer function is the integrated (accumulated) form of $H_b(z)$ and is easily verified to be as in Eq. (25.2.9),

$$H(z) = \frac{H_b(z)}{1 - z^{-1}}$$

The following MATLAB function, **pma**, implements Eq. (25.3.2) and the related indicators, where the input array **y** represents the financial data to be filtered. The function, **pmaimp**, implements the impulse response of Eq. (25.3.3).

```
at = pma(y,N,tau,yin);                % at = a + tau*b, prediction distance tau

 a = pma(y,N,0,yin);                  % local-level indicator
 b = pma(y,N,1,yin)-pma(y,N,0,yin);   % local-slope indicator

af = pma(y,N,1,yin);                  % time-series forecast indicator, af = a + b

ht = pmaimp(N,tau);                   % impulse response of predictive filter
ha = pmaimp(N,0);                     % impulse response of local level filter
hb = pmaimp(N,1)-pmaimp(N,0);         % impulse response of local slope filter
```

and again, the string variable `yin` specifies the way the filters are initialized and can take on the following values,

```
yin = 'f',     % progressive filtering (default method)
yin = 'n',     % initial transients are NaNs (facilitates plotting)
yin = 'c',     % standard convolutional transients
```

A few examples of impulse responses are as follows, for $N = 5, 8, 11$,

$$
\begin{aligned}
N = 5\,, \quad & \mathbf{h}_a = \frac{1}{5}[3,\,2,\,1,\,0,\,-1] && \text{(local level)}\\
& \mathbf{h}_b = \frac{1}{10}[2,\,1,\,0,\,-1,\,-2] && \text{(local slope)}\\
& \mathbf{h}_1 = \frac{1}{10}[8,\,5,\,2,\,-1,\,-4] && \text{(time-series forecast)}\\
N = 8\,, \quad & \mathbf{h}_a = \frac{1}{12}[5,\,4,\,3,\,2,\,1,\,0,\,-1,\,-2]\\
& \mathbf{h}_b = \frac{1}{84}[7,\,5,\,3,\,1,\,-1,\,-3,\,-5,\,-7]\\
& \mathbf{h}_1 = \frac{1}{28}[14,\,11,\,8,\,5,\,2,\,-1,\,-4,\,-7]\\
N = 11\,, \quad & \mathbf{h}_a = \frac{1}{22}[7,\,6,\,5,\,4,\,3,\,2,\,1,\,0,\,-1,\,-2,\,-3]\\
& \mathbf{h}_b = \frac{1}{110}[5,\,4,\,3,\,2,\,1,\,0,\,-1,\,-2,\,-3,\,-4,\,-5]\\
& \mathbf{h}_1 = \frac{1}{55}[20,\,17,\,14,\,11,\,8,\,5,\,2,\,-1,\,-4,\,-7,\,-10]
\end{aligned}
$$

Some comparisons of PMA with other moving average indicators are shown in Figures 25.5.2 and 25.8.1.

25.4 Single, Double, Triple EMA Indicators

As discussed in Sec. 24.6, the single EMA (SEMA), double EMA (DEMA), and triple EMA (TEMA) steady-state exponential smoothing recursions are as follows,

$$
\boxed{\begin{aligned}
e_{n/n-1} &= y_n - \hat{y}_{n/n-1} = y_n - a_{n-1}\\
a_n &= a_{n-1} + (1-\lambda)\,e_{n/n-1}
\end{aligned}} \quad \text{(SEMA)} \qquad (25.4.1)
$$

$$
\boxed{\begin{aligned}
& e_{n/n-1} = y_n - \hat{y}_{n/n-1} = y_n - (a_{n-1} + b_{n-1})\\
& \begin{bmatrix} a_n \\ b_n \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a_{n-1} \\ b_{n-1} \end{bmatrix} + \begin{bmatrix} 1-\lambda^2 \\ (1-\lambda)^2 \end{bmatrix} e_{n/n-1}
\end{aligned}} \quad \text{(DEMA)} \qquad (25.4.2)
$$

$$
\boxed{\begin{aligned}
& e_{n/n-1} = y_n - \hat{y}_{n/n-1} = y_n - (a_{n-1} + b_{n-1} + c_{n-1})\\
& \begin{bmatrix} a_n \\ b_n \\ c_n \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{n-1} \\ b_{n-1} \\ c_{n-1} \end{bmatrix} + \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} e_{n/n-1}
\end{aligned}} \quad \text{(TEMA)} \qquad (25.4.3)
$$

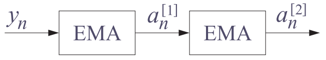
where

$$
\alpha_1 = 1-\lambda^3\,, \quad \alpha_2 = \frac{3}{2}(1-\lambda)(1-\lambda^2)\,, \quad \alpha_3 = \frac{1}{2}(1-\lambda)^3
$$

and $\hat{y}_{n/n-1}$ represents the forecast of y_n based on data up to time $n-1$. More generally, the forecast ahead by a distance τ is given by,

$$
\begin{aligned}
&\text{(SEMA)} && \hat{y}_{n+\tau/n} = a_n && && \hat{y}_{n/n-1} = a_{n-1}\\
&\text{(DEMA)} && \hat{y}_{n+\tau/n} = a_n + b_n\tau && \Rightarrow && \hat{y}_{n/n-1} = a_{n-1} + b_{n-1}\\
&\text{(TEMA)} && \hat{y}_{n+\tau/n} = a_n + b_n\tau + c_n\tau^2 && && \hat{y}_{n/n-1} = a_{n-1} + b_{n-1} + c_{n-1}
\end{aligned} \qquad (25.4.4)
$$

We saw in Sec. 24.8 that an alternative way of computing the local level and local slope signals a_n, b_n in the DEMA case is in terms of the outputs of the cascade of two single EMAs, that is, with $\alpha = 1-\lambda$,



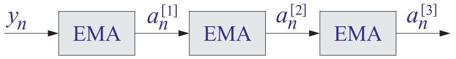
$$
\boxed{\begin{aligned}
a_n^{[1]} &= \lambda a_{n-1}^{[1]} + \alpha y_n\\
a_n^{[2]} &= \lambda a_{n-1}^{[2]} + \alpha a_n^{[1]}
\end{aligned}} \qquad (25.4.5)
$$

$$
\boxed{\begin{aligned}
a_n &= 2a_n^{[1]} - a_n^{[2]} = \text{local level DEMA indicator}\\
b_n &= \frac{\alpha}{\lambda}\,(a_n^{[1]} - a_n^{[2]}) = \text{local slope DEMA indicator}
\end{aligned}} \qquad (25.4.6)
$$

The transfer functions from y_n to the signals a_n, b_n were given in Eq. (24.8.5), and are expressible as follows in terms of the transfer function of a single EMA, $H(z) = \alpha/(1-\lambda z^{-1})$,

$$
\begin{aligned}
H_a(z) &= \frac{\alpha(1+\lambda-2\lambda z^{-1})}{(1-\lambda z^{-1})^2} = 2H(z) - H^2(z) = 1 - \big[1-H(z)\big]^2\\
H_b(z) &= \frac{\alpha^2(1-z^{-1})}{(1-\lambda z^{-1})^2} = \frac{\alpha}{\lambda}\big[H(z) - H^2(z)\big]
\end{aligned} \qquad (25.4.7)
$$

Similarly, in the TEMA case, the signals a_n, b_n, c_n can be computed from the outputs of three successive single EMAs via the following relationships,



$$
\boxed{\begin{aligned}
a_n^{[1]} &= \lambda a_{n-1}^{[1]} + \alpha y_n\\
a_n^{[2]} &= \lambda a_{n-1}^{[2]} + \alpha a_n^{[1]}\\
a_n^{[3]} &= \lambda a_{n-1}^{[3]} + \alpha a_n^{[2]}
\end{aligned}} \qquad (25.4.8)
$$

$$
\begin{bmatrix} a_n \\ b_n \\ c_n \end{bmatrix} = \frac{1}{2\lambda^2} \begin{bmatrix} 6\lambda^2 & -6\lambda^2 & 2\lambda^2 \\ \alpha(1+5\lambda) & -2\alpha(1+4\lambda) & \alpha(1+3\lambda) \\ \alpha^2 & -2\alpha^2 & \alpha^2 \end{bmatrix} \begin{bmatrix} a_n^{[1]} \\ a_n^{[2]} \\ a_n^{[3]} \end{bmatrix} \qquad (25.4.9)
$$

where $\alpha = 1-\lambda$. See also Eqs. (24.8.9)–(24.8.13). In particular, we have,

$$
\boxed{a_n = 3a_n^{[1]} - 3a_n^{[2]} + a_n^{[3]}} \quad \text{(local level TEMA indicator)} \qquad (25.4.10)
$$

Initialization issues for the single EMA, DEMA, and TEMA recursions are discussed in Sec. 25.6. The following MATLAB functions implement the corresponding filtering operations, where the input array $\mathbf{y}$ represents the financial data to be filtered.

```
           a = sema(y,N,yin);   % single exponential moving average
     [a,b,a1,a2] = dema(y,N,yin);   % double exponential moving average
  [a,b,c,a1,a2,a3] = tema(y,N,yin);   % triple exponential moving average
```

The variable `yin` specifies the way the filters are initialized and can take on the following possible values,

```
yin = y(1)   % default for SEMA
yin = ’f’    % fits polynomial to first N samples, default for DEMA, TEMA
yin = ’c’    % cascaded initialization for DEMA, TEMA, described in Sect. 6.19
yin = any vector of initial values of [a], [a;b], or [a;b;c] at n=-1
yin = [0], [0;0], or [0;0;0] for standard convolutional output
```

Even though the EMA filters are IIR filters, traders prefer to specify the parameter λ of the EMA recursions through the SMA-equivalent length N defined as in Eq. (24.1.16),

$$\lambda = \frac{N-1}{N+1} \quad \Leftrightarrow \quad N = \frac{1+\lambda}{1-\lambda} \tag{25.4.11}$$

The use of DEMA and TEMA as market indicators with less lag was first advocated by Mulloy [881,882]. Some comparisons of these with other moving average indicators are shown in Fig. 25.5.2.

25.5 Linear Regression and R-Square Indicators

In the literature of technical analysis, the PMA indicators of Eq. (25.3.1) are usually not implemented as FIR filters, but rather as successive fits of straight lines to the past N data from the current data point, that is, over the time span, $[n-N+1, n]$, for each n. This is depicted Fig. 25.5.1 below.

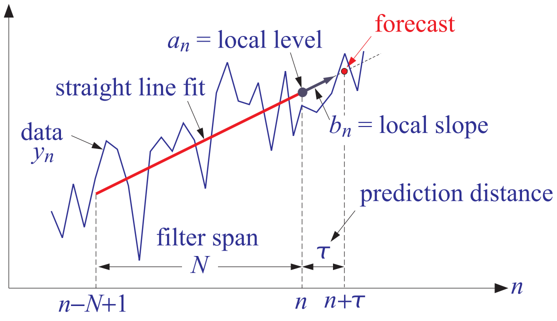


Fig. 25.5.1 Local linear regression and prediction.

They have been rediscovered many times in the past and different names given to them. For example, Lafferty [884] calls them "end-point moving averages", while Rafter [887] refers to them as "moving trends." Their application as a forecasting tool was discussed first by Chande [883].

Because of the successive fitting of straight lines, the signals a_n, b_n are known as the "linear regression" indicator and the "linear regression slope" indicator, respectively. The a_n indicator is also known as "least-squares moving average" (LSMA).

For each $n \geq N-1$, the signals a_n, b_n can be obtained as the *least-squares solution* of the following $N{\times}2$ *overdetermined* system of linear equations in two unknowns:

$$a_n - kb_n = y_{n-k}\,, \quad k = 0, 1, \dots, N-1 \tag{25.5.1}$$

which express the fitting of a straight line, $a + b\tau$, to the data $[y_{n-N+1}, \dots, y_{n-1}, y_n]$, that is, over the time window, $[n-N+1, n]$, where a is the intercept at the end of the line. The overdetermined system (25.5.1) can be written compactly in matrix form by defining the length-N column vectors,

$$\mathbf{u} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \\ \vdots \\ 1 \end{bmatrix}, \quad \mathbf{k} = \begin{bmatrix} 0 \\ 1 \\ 2 \\ \vdots \\ k \\ \vdots \\ N-1 \end{bmatrix}, \quad \mathbf{y}_n = \begin{bmatrix} y_n \\ y_{n-1} \\ y_{n-2} \\ \vdots \\ y_{n-k} \\ \vdots \\ y_{n-N+1} \end{bmatrix} \quad \Rightarrow \quad [\mathbf{u}, -\mathbf{k}] \begin{bmatrix} a_n \\ b_n \end{bmatrix} = \mathbf{y}_n \tag{25.5.2}$$

with the least-squares solution expressed in the following MATLAB-like vectorial notation using the backslash operator,

$$\begin{bmatrix} a_n \\ b_n \end{bmatrix} = [\mathbf{u}, -\mathbf{k}] \setminus \mathbf{y}_n \tag{25.5.3}$$

Indeed, this is the solution for the local level and local slope parameters a, b that minimize the following least-squares performance index, defined for each n,

$$\mathcal{J}_n = \sum_{k=0}^{N-1} \big(a - bk - y_{n-k}\big)^2 = \min \tag{25.5.4}$$

In order to account also for the initial transient period, $0 \leq n \leq N-1$, we may change the upper limit of summation in Eq. (25.5.4) to,

$$\mathcal{J}_n = \sum_{k=0}^{\min(n,N-1)} \big(a - bk - y_{n-k}\big)^2 = \min \tag{25.5.5}$$

which amounts to fitting a straight line to a progressively longer and longer data vector until its length becomes equal to N, that is, starting with $a_0 = y_0$ and $b_0 = 0$,[†] we fit a line to $[y_0, y_1]$ to get a_1, b_1, then fit a line to $[y_0, y_1, y_2]$ to get a_2, b_2, and so on until $n = N-1$, and beyond that, we continue with a length-N data window.

[†] $b_0 = 0$ is an arbitrary choice since b_0 is indeterminate for $N = 1$.

Thus, we may state the complete solution for all $0 \le n \le L-1$, where L is the length of the data vector y_n, using the backslash notation,[‡]

$$\boxed{\begin{aligned} &\text{for each, } n = 0,1,2,\dots,L-1, \text{ do:}\\ &\quad K_n = \min(n, N-1)+1 = \text{fitting length, } K_n = N \text{ when } n \ge N-1\\ &\quad \mathbf{k} = \big[0 : K_n - 1\big]' = \text{column vector, length } K_n\\ &\quad \mathbf{y}_n = y(n-\mathbf{k}) = \text{column vector, } [y_n, y_{n-1}, \dots, y_{n-K_n+1}]^T\\ &\quad \mathbf{u} = \text{ones}(K_n, 1) = \text{column vector}\\ &\quad \begin{bmatrix} a_n \\ b_n \end{bmatrix} = [\mathbf{u}, -\mathbf{k}] \setminus \mathbf{y}_n = \text{linear regression indicators}\\ &\quad R^2(n) = \big(\text{corr}(-\mathbf{k}, \mathbf{y}_n)\big)^2 = 1 - \det\big(\text{corrcoef}(-\mathbf{k}, \mathbf{y}_n)\big) = R^2 \text{ indicator} \end{aligned}} \tag{25.5.6}$$

where we also included the so-called *R-square indicator*,[*] which is the *coefficient of determination* for the linear fit, and quantifies the strength of the linear relationship, that is, higher values of R^2 suggest that the linear fit is statistically significant with a certain degree of confidence (usually taken to be at the 95% confidence level).

The MATLAB function, **r2crit** in the AOSP toolbox, calculates the critical values R_c^2 of R^2 for a given N and given confidence level p, such that if $R^2(n) > R_c^2$, then the linear fit is considered to be statistically significant for the nth segment. Some typical critical values of R_c^2 at the $p = 0.95$ and $p = 0.99$ levels are listed below in Eq. (25.5.7), and were computed with the following MATLAB commands (see also [864]),

```
N = [5, 10, 14, 20, 25, 30, 50, 60, 120];
R2c = r2crit(N,0.95);
R2c = r2crit(N,0.99);
```

N	$p = 0.95$	$p = 0.99$
5	0.7711	0.9180
10	0.3993	0.5846
14	0.2835	0.4374
20	0.1969	0.3152
25	0.1569	0.2552
30	0.1303	0.2143
50	0.0777	0.1303
60	0.0646	0.1090
120	0.0322	0.0549

(25.5.7)

The *standard errors* for the successive linear fits, as well as the standard errors for the quantities a_n, b_n, can be computed by including the following lines within the for-loop in Eq. (25.5.6),

$$\boxed{\begin{aligned} \mathbf{e}_n &= \mathbf{y}_n - [\mathbf{u}, -\mathbf{k}]\begin{bmatrix} a_n \\ b_n \end{bmatrix} = \text{fitting error, column vector}\\ \sigma_e(n) &= \sqrt{\frac{\mathbf{e}_n^T\mathbf{e}_n}{K_n - 2}} = \text{standard error}\\ \sigma_a(n) &= \sqrt{\frac{2(2K_n-1)}{K_n(K_n+1)}}\,\sigma_e(n) = \text{standard error for } a_n\\ \sigma_b(n) &= \sqrt{\frac{12}{K_n(K_n^2-1)}}\,\sigma_e(n) = \text{standard error for } b_n \end{aligned}} \tag{25.5.8}$$

The derivation of the expressions for $\sigma_e, \sigma_a, \sigma_b$ follows from the standard theory of least-squares linear regression. For example, linear regression based on the K pairs, (x_k, y_k), $k = 0, 1, \dots, K-1$, results in the estimates, $\hat{y}_k = a + bx_k$, and error residuals, $e_k = y_k - \hat{y}_k$, from which the standard errors can be calculated from the following expressions [939],

$$\sigma_e^2 = \frac{1}{K-2}\sum_{k=0}^{N-1} e_k^2\,, \quad \sigma_a^2 = \sigma_e^2\,\frac{\overline{x^2}}{K\sigma_x^2}\,, \quad \sigma_b^2 = \frac{\sigma_e^2}{K\sigma_x^2} \tag{25.5.9}$$

For our special case of equally-spaced data, $x_k = -k$, we easily find,

$$\begin{aligned} \bar{x} &= -\bar{k} = -\frac{1}{K}\sum_{k=0}^{K-1} k = -\frac{K-1}{2}\\ \overline{x^2} &= \overline{k^2} = \frac{1}{K}\sum_{k=0}^{K-1} k^2 = \frac{(K-1)(2K-1)}{6} \qquad \Rightarrow \qquad \begin{aligned} \sigma_a^2 &= \frac{2(2K-1)}{K(K+1)}\,\sigma_e^2\\ \sigma_b^2 &= \frac{12}{K(K^2-1)}\,\sigma_e^2 \end{aligned}\\ \sigma_x^2 &= \sigma_k^2 = \overline{k^2} - \bar{k}^2 = \frac{K^2-1}{12} \end{aligned}$$

Standard error bands [913], as well as other types of bands and envelopes, and their use as market indicators, are discussed further in Sec. 25.11. The MATLAB function, **lreg**, implements Eqs. (25.5.6) and (25.5.8) with usage,

```
[a,b,R2,se,sa,sb] = lreg(y,N,init);    % linear regression indicators
```

```
y = data                  a = local level      se = standard error
N = window length         b = local slope      sa = standard error for a
init = initialization     R2 = R-square        sb = standard error for b
```

where `init` specifies the initialization scheme and takes on the following values,

```
init = 'f', progressive linear fitting of initial N-1 samples, default
init = 'n', replacing initial N-1 samples of a,b,R2,se,sa,sb by NaNs
```

[‡] the backslash solution also correctly generates the case $n = 0$, i.e., $a_0 = y_0$ and $b_0 = 0$.

[*] where, **corr**, **det**, and **corrcoef**, are built-in MATLAB functions.

The local level and local slope outputs a_n, b_n are identical to those produced by the function **pma** of Sec. 25.3.

Generally, these indicators behave similarly to the DEMA indicators, but both indicator types should be used with some caution since they are too quick to respond to price changes and sometimes tend to produce false buy/sell signals. In other words, some delay may miss the onset of a trend but provides more safety.

Example 25.5.1: Fig. 25.5.2 compares the SMA, EMA, WMA, PMA/linear regression, DEMA, TEMA indicators. The data are from [889] and represent daily prices for Nicor-Gas over 130 trading days starting on Sept. 1, 2006. The included excel file, **nicor.xls**, contains the open-high-low-close prices in its first four columns. The left graphs were produced by the following MATLAB code, in which the function, **ohlc**, from the AOSP toolbox, makes an OHLC[†] bar chart,

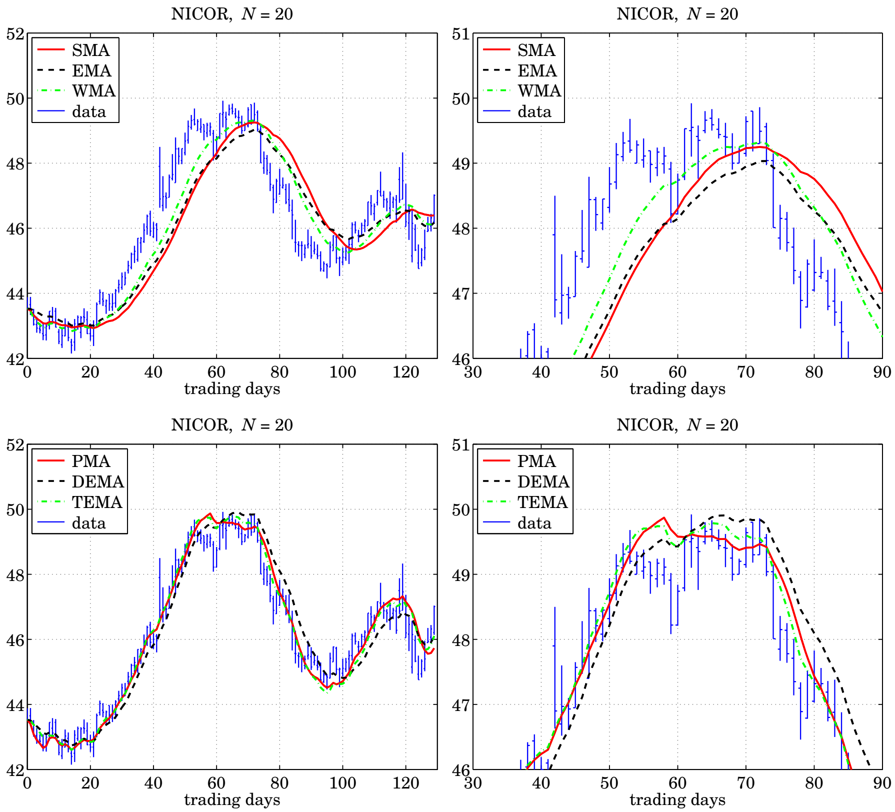


Fig. 25.5.2 Comparison of SMA, EMA, WMA with PMA/LREG, DEMA, TEMA indicators.

```
Y = xlsread('nicor.xls');   % read Nicor-Gas data
```

```
Y = Y(1:130,:);               % keep only 130 trading days
y = Y(:,4);                   % closing prices
t = 0:length(y)-1;            % trading days

N = 20;                       % filter length

figure;                       % SMA, EMA, WMA
plot(t,sma(y,N),'r-', t,sema(y,N),'k--', t,wma(y,N),'g-.');  hold on;
ohlc(t,Y(:,1:4));             % add OHLC bar chart


figure;                       % PMA/lreg, DEMA, TEMA
plot(t,pma(y,N,0),'r-', t,dema(y,N),'k--',  t,tema(y,N),'g-.'); hold on;
ohlc(t,Y(:,1:4));             % add OHLC bar chart
```

The filter length was $N = 20$. The right graphs are an expanded view of the range $[45, 90]$ days and show more clearly the reduced lag of the PMA, DEMA, and TEMA indicators. At about the 57th trading day, these indicators turn downwards but still lie above the data, therefore, they would correctly issue a "sell" signal. By contrast, the SMA, EMA, and WMA indicators are rising and lie below the data, and they would issue a "buy" signal.

Fig. 25.8.1 in Sec. 25.8 compares the PMA with two other indicators of reduced lag, namely, the Hull moving average (HMA), and the exponential Hull moving average (EHMA).

The R-squared and slope indicators are also useful in determining the direction of trend. Fig. 25.5.3 shows the PMA/linear regression indicator, a_n, for the same Nicor data, together with the corresponding $R^2(n)$ signal, and the slope signal b_n, using again a filter length of $N = 20$. They were computed with the MATLAB code:

```
[a,b,R2] = lreg(y,N);          % local level, local slope, and R-squared

% equivalent calculation:
% a = pma(y,N,0);
% b = pma(y,N,1)-pma(y,N,0);
```

For $N = 20$, the critical value of R^2 at the 95% confidence level is $R_c^2 = 0.1969$, determined in Eq. (25.5.7), and is displayed as the horizontal dashed line on the R^2 graph.

When $R^2(n)$ is small, below R_c^2, it indicates lack of a trend with the data moving sideways, and corresponds to slope b_n near zero.

When $R^2(n)$ rises near unity, it indicates a strong trend, but it does not indicate the direction, upwards or downwards. This is indicated by the slope indicator b_n, which is positive when the signal is rising, and negative, when it is falling. More discussion on using these three indicators in conjunction may be found in [889]. □

25.6 Initialization Schemes

In Eq. (25.5.6), one solves a shorter linear fitting problem of progressively increasing length during the transient period, $0 \le n < N - 1$, and then switches to fixed length N for $n \ge N - 1$.

The same idea can be applied to all FIR filters, such as the SMA, WMA, TMA, and the PMA filter, $h_\tau(n)$, that is, to use the same type of filter, but of progressively increasing

[†] Open-High-Low-Close

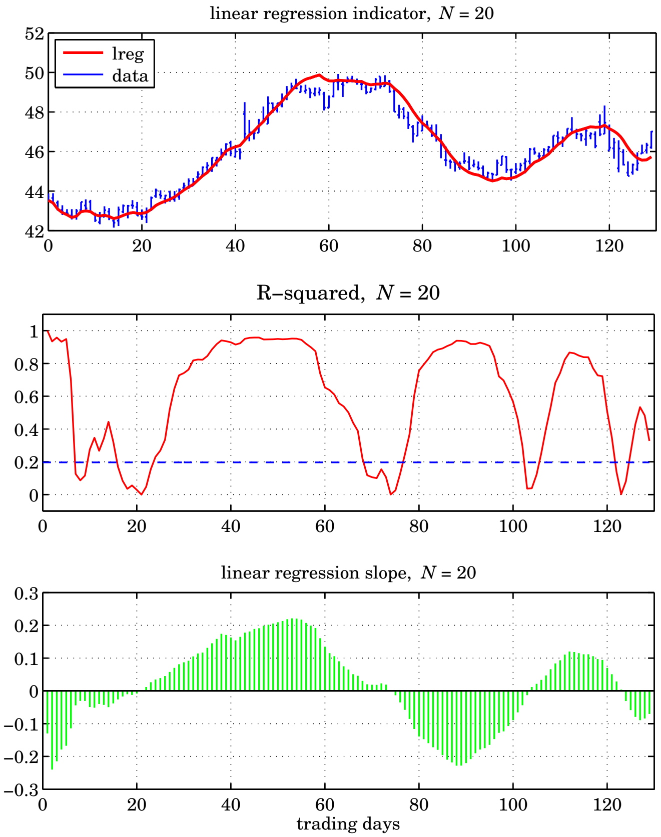


Fig. 25.5.3 PMA/linear regression, R-squared, and linear regression slope indicators.

length, during the period $0 \le n < N-1$, and then switch to using the filters of fixed length N for $n \ge N-1$. The first $N-1$ outputs computed in this manner are not the same as the standard convolutional outputs obtained from the built-in function **filter**, because the latter uses the same length-N filter and assumes zero initial internal states.

To clarify this, consider the SMA case with $N = 5$, then the above procedure and the standard convolutional one compute the outputs in the following manner, agreeing

only after $n \ge N-1 = 4$,

progressive	convolutional
$a_0 = y_0$	$a_0 = \frac{1}{5}\, y_0$
$a_1 = \frac{1}{2}(y_1 + y_0)$	$a_1 = \frac{1}{5}(y_1 + y_0)$
$a_2 = \frac{1}{3}(y_2 + y_1 + y_0)$	$a_2 = \frac{1}{5}(y_2 + y_1 + y_0)$
$a_3 = \frac{1}{4}(y_3 + y_2 + y_1 + y_0)$	$a_3 = \frac{1}{5}(y_3 + y_2 + y_1 + y_0)$
$a_4 = \frac{1}{5}(y_4 + y_3 + y_2 + y_1 + y_0)$	$a_4 = \frac{1}{5}(y_4 + y_3 + y_2 + y_1 + y_0)$
$a_5 = \frac{1}{5}(y_5 + y_4 + y_3 + y_2 + y_1)$	$a_5 = \frac{1}{5}(y_5 + y_4 + y_3 + y_2 + y_1)$
$\cdots$	$\cdots$

Similarly, the local level PMA filters, $\mathbf{h}_a$, of lengths up to $N = 5$ can be determined from Eq. (25.3.3), leading to the following progressive initializations,

$$
\begin{array}{lll}
N = 1\,, & \mathbf{h}_a = [1]\,, & a_0 = y_0 \\
N = 2\,, & \mathbf{h}_a = [1, 0]\,, & a_1 = y_1 \\
N = 3\,, & \mathbf{h}_a = \frac{1}{6}\,[5, 2, -1]\,, & a_2 = \frac{1}{6}(5y_2 + 2y_1 - y_0) \\
N = 4\,, & \mathbf{h}_a = \frac{1}{10}\,[7, 4, 1, -2]\,, & a_3 = \frac{1}{10}(7y_3 + 4y_2 + y_1 - 2y_0) \\
N = 5\,, & \mathbf{h}_a = \frac{1}{5}\,[3, 2, 1, 0, -1]\,, & a_4 = \frac{1}{5}(3y_4 + 2y_3 + y_2 - y_0)
\end{array}
$$

and for the local slope filters $\mathbf{h}_b$,

$$
\begin{array}{lll}
N = 1\,, & \mathbf{h}_b = [0]\,, & b_0 = 0 \\
N = 2\,, & \mathbf{h}_b = [1, -1]\,, & b_1 = y_1 - y_0 \\
N = 3\,, & \mathbf{h}_b = \frac{1}{2}\,[1, 0, -1]\,, & b_2 = \frac{1}{2}(y_2 - y_0) \\
N = 4\,, & \mathbf{h}_b = \frac{1}{10}\,[3, 1, -1, -3]\,, & b_3 = \frac{1}{10}(3y_3 + y_2 - y_1 - 3y_0) \\
N = 5\,, & \mathbf{h}_b = \frac{1}{10}\,[2, 1, 0, -1, -2]\,, & b_4 = \frac{1}{10}(2y_4 + y_3 - y_1 - 2y_0)
\end{array}
$$

where, we arbitrarily set $\mathbf{h}_b = [0]$ for the case $N = 1$, since the slope is meaningless for a single data point. To see the equivalence of these with the least-square criterion of Eq. (25.5.5) consider, for example, the case $N = 5$ and $n = 2$,

$$\mathcal{J}_2 = (a - y_2)^2 + (a - b - y_1)^2 + (a - 2b - y_0)^2 = \min$$

with minimization conditions,

$$\begin{aligned}\frac{\partial J_2}{\partial a} &= 2(a-y_2)+2(a-b-y_1)+2(a-2b-y_0)=0\\ \frac{\partial J_2}{\partial b} &= -2(a-b-y_1)-4(a-2b-y_0)=0\end{aligned} \quad\Rightarrow\quad \begin{aligned}3a-3b &= y_2+y_1+y_0\\ 3a-5b &= y1+2y_0\end{aligned}$$

resulting in the solution,

$$a=\frac{1}{6}(5y_2+2y_1-y_0)\,,\quad b=\frac{1}{2}(y_2-y_0)$$

Similarly we have for the cases $n=0$ and $n=1$,

$$\begin{aligned}\mathcal{J}_0 &= (a-y_0)^2=\min &&\Rightarrow\quad a=y_0\,,\quad b=\text{indeterminate}\\ \mathcal{J}_1 &= (a-y_1)^2+(a-b-y_0)^2=\min &&\Rightarrow\quad a=y_1\,,\quad b=y_1-y_0\end{aligned}$$

EMA Initializations

The single, double, and triple EMA difference equations (25.4.1)–(25.4.3), also need to be properly initialized at $n=-1$. For the single EMA case, a good choice is $a_{-1}=y_0$, which leads to the same value at $n=0$, that is,

$$a_0=\lambda a_{-1}+\alpha y_0=\lambda y_0+\alpha y_0=y_0 \tag{25.6.1}$$

This is the default initialization for our function, **sema**. Another possibility is to choose the mean of the first N data samples, $a_{-1}=\text{mean}([y_0,y_1,\dots,y_{N-1}])$.

For DEMA, if we initialize both the first and the second EMAs as in Eq. (25.6.1), then we must choose, $a_{-1}^{[1]}=y_0$, which leads to $a_0^{[1]}=y_0$, which then would require that, $a_{-1}^{[2]}=a_0^{[1]}=y_0$, thus, in this scheme, we would choose,

$$\begin{bmatrix}a_{-1}^{[1]}\\ a_{-1}^{[2]}\end{bmatrix}=\begin{bmatrix}y_0\\ y_0\end{bmatrix}\quad\Rightarrow\quad\begin{bmatrix}a_{-1}\\ b_{-1}\end{bmatrix}=\begin{bmatrix}2 & -1\\ \alpha/\lambda & -\alpha/\lambda\end{bmatrix}\begin{bmatrix}a_{-1}^{[1]}\\ a_{-1}^{[2]}\end{bmatrix}=\begin{bmatrix}y_0\\ 0\end{bmatrix} \tag{25.6.2}$$

This is the default initialization method for our function, **dema**. Another possibility is to fit a straight line to a few initial data [881,938], such as the first N data, where N is the equivalent SMA length, $N=(1+\lambda)/(1-\lambda)$, and then extrapolate the line backwards to $n=-1$. This can be accomplished in MATLAB-like notation as follows,

$$\begin{aligned}&\mathbf{n}=[1:N]'=\text{column vector}\\ &\mathbf{y}=[y_0,y_1,\dots,y_{N-1}]'=\text{column vector}\\ &\mathbf{u}=\text{ones}(\text{size}(\mathbf{n}))\\ &\begin{bmatrix}a_{-1}\\ b_{-1}\end{bmatrix}=[\mathbf{u},\mathbf{n}]\setminus\mathbf{y}\end{aligned} \tag{25.6.3}$$

If one wishes to use the cascade of two EMAs, then the EMA signals, $a_n^{[1]},a_n^{[2]}$, must be initialized by first applying Eq. (25.6.3), and then using the inverse matrix relationship of Eq. (25.4.6), i.e.,

$$\begin{bmatrix}a_{-1}^{[1]}\\ a_{-1}^{[2]}\end{bmatrix}=\begin{bmatrix}1 & -\lambda/\alpha\\ 1 & -2\lambda/\alpha\end{bmatrix}\begin{bmatrix}a_{-1}\\ b_{-1}\end{bmatrix} \tag{25.6.4}$$

A third possibility [864] is to initialize the first EMA with $a_{-1}^{[1]}=y_0$, then calculate the output at the time instant $n=N-1$ and use it to initialize the second EMA at $n=N$, that is, define $a_{N-1}^{[2]}=a_{N-1}^{[1]}$. This value can be iterated backwards to $n=-1$ to determine the proper initial value $a_{-1}^{[2]}$ such that, if iterated forward, it would arrive at the chosen value $a_{N-1}^{[2]}=a_{N-1}^{[1]}$. Thus, the steps in this scheme are as follows,

$$\begin{array}{l}a_{-1}^{[1]}=y_0\\ \text{for } n=0,1,\dots,N-1,\\ \qquad a_n^{[1]}=\lambda a_{n-1}^{[1]}+\alpha y_n\\ \text{end}\end{array}\quad\Rightarrow\quad\begin{array}{l}a_{N-1}^{[2]}=a_{N-1}^{[1]}\\ \text{for } n=N-1,\dots,1,0,\\ \qquad a_{n-1}^{[2]}=\dfrac{1}{\lambda}(a_n^{[2]}-\alpha a_n^{[1]})\\ \text{end}\end{array} \tag{25.6.5}$$

Upon exit from the second loop, one has $a_{-1}^{[2]}$, then, one can transform the calculated $a_{-1}^{[1]},a_{-1}^{[2]}$ to the a_n,b_n basis in order to get the DEMA recursion started,

$$\begin{bmatrix}a_{-1}\\ b_{-1}\end{bmatrix}=\begin{bmatrix}2 & -1\\ \alpha/\lambda & -\alpha/\lambda\end{bmatrix}\begin{bmatrix}a_{-1}^{[1]}\\ a_{-1}^{[2]}\end{bmatrix}$$

Such cascaded initialization scheme for DEMA (and TEMA below) is somewhat ad hoc since the EMA filters are IIR and there is nothing special about the time $n=N$; one, could just as well wait until about $n=6N$ when typically all transient effects have disappeared. We have found that the schemes described in Eqs. (25.6.2) and (25.6.3) work the best.

Finally, we note that for ordinary convolutional output, one would choose zero initial values,

$$\begin{bmatrix}a_{-1}\\ b_{-1}\end{bmatrix}=\begin{bmatrix}0\\ 0\end{bmatrix}\quad\Rightarrow\quad\begin{bmatrix}a_{-1}^{[1]}\\ a_{-1}^{[2]}\end{bmatrix}=\begin{bmatrix}0\\ 0\end{bmatrix}$$

All of the above initialization choices are incorporated in the function, **dema**. For TEMA, the default initialization is similar to that of Eq. (25.6.2), that is,

$$\begin{bmatrix}a_{-1}^{[1]}\\ a_{-1}^{[2]}\\ a_{-1}^{[3]}\end{bmatrix}=\begin{bmatrix}y_0\\ y_0\\ y_0\end{bmatrix}\quad\Rightarrow\quad\begin{bmatrix}a_{-1}\\ b_{-1}\\ c_{-1}\end{bmatrix}=\begin{bmatrix}y_0\\ 0\\ 0\end{bmatrix} \tag{25.6.6}$$

Alternatively, one can fit a second-order polynomial to the first few data samples, such as the first $2N$ samples [881], and extrapolate them back to $n=-1$. The fitting can be done with the following MATLAB-like code,

$$\mathbf{n} = [1 : 2N-1]' = \text{column vector}$$

$$\mathbf{y} = [y_0, y_1, \dots, y_{2N-1}]' = \text{column vector}$$

$$\mathbf{u} = \text{ones}(\text{size}(\mathbf{n}))$$

$$\begin{bmatrix} a_{-1} \\ b_{-1} \\ c_{-1} \end{bmatrix} = [\mathbf{u}, \mathbf{n}, \mathbf{n}^2] \setminus \mathbf{y}$$

The cascaded initialization scheme is also possible in which the output of the first EMA at time $n = N-1$ serves to initialize the second EMA at $n = N$, and the output of the second EMA at $n = 2N-1$ serves to initialize the third EMA at $n = 2N$. This, as well as the second-order polynomial fitting initialization schemes are incorporated in the function, **tema**.

A special case of the EMA indicator is “Wilder’s Exponential Moving Average” [865], known as WEMA. It is used widely and appears in several other indicators, such as the “Relative Strength Index” (RSI), the “Average True Range” (ATR), and the “Directional Movement System” (±DMI and ADX), discussed in Sec. 25.12. An N-point WEMA is defined to be an ordinary EMA with λ, α parameters,

$$\boxed{\alpha = \frac{1}{N}, \quad \lambda = 1 - \alpha = 1 - \frac{1}{N}} \quad \text{(WEMA parameters)} \tag{25.6.7}$$

It is equivalent to an EMA with effective length, N_e, determined as follows,

$$\lambda = \frac{N_e - 1}{N_e + 1} = 1 - \frac{1}{N} \quad \Rightarrow \quad \boxed{N_e = 2N - 1} \tag{25.6.8}$$

The corresponding filtering equation for calculating the smoothed local-level signal a_n from the input data y_n, will be,

$$a_n = \lambda a_{n-1} + \alpha y_n = a_{n-1} + \alpha(y_n - a_{n-1})$$

or, for $n \geq 0$,

$$\boxed{a_n = a_{n-1} + \frac{1}{N}(y_n - a_{n-1})} \quad \text{(WEMA)} \tag{25.6.9}$$

The required initial value a_{-1} can be chosen in a variety of ways, just as in EMA. However, by convention [865], the default way of fixing it is similar to that in Eq. (25.6.5). It is defined by choosing the value of a_n at time $n = N-1$ to be the mean of first N input values, then, a_{N-1} is back-tracked to time $n = -1$, thus, we have,

```
$a_{N-1} = \frac{1}{N}(y_0 + y_1 + \cdots + y_{N-1})$
for $n = N-1, \dots, 1, 0,$
    $a_{n-1} = \frac{1}{\lambda}(a_n - \alpha y_n)$
end
```
(25.6.10)

Upon exit from the loop, one has the proper starting value of a_{-1}. The following MATLAB function, **wema**, implements WEMA with such default initialization scheme,

```
a = wema(y,N,ain);     % Wilder’s EMA

y   = signal to be smoothed
N   = effective length, (EMA alpha = 1/N, lambda = 1-1/N)
ain = any initial value
    = ’m’, default, as in Eq.(6.19.10)
    = 0, for standard convolutional output

a = smoothed version of y
```

25.7 Butterworth Moving Average Filters

Butterworth moving average (BMA) lowpass filters, are useful alternatives [869] to the first-order EMA filters, and have comparable smoothing properties and shorter lag. Here, we summarize their properties and filtering implementation, give explicit design equations for orders $M = 1, 2, 3$, and derive a general expression for their lag.

Digital Butterworth filters are characterized by two parameters, the filter order M, and the 3-dB cutoff frequency f_0 in Hz, or, the corresponding digital frequency in units of radians per sample, $\omega_0 = 2\pi f_0 / f_s$, where f_s is the sampling rate in Hz. We may also define the period of f_0 in units of samples/cycle, $N = f_s / f_0$, so that, $\omega_0 = 2\pi / N$.

We follow the design method of Chap. 12 based on the bilinear transformation, although the matched z-transform method has also been used [869]. If the filter order is even, say, $M = 2K$, then, there are K second-order sections, and if it is odd, $M = 2K + 1$, there is an additional first-order section. Both cases can be combined into one by writing,

$$M = 2K + r, \quad r = 0, 1 \tag{25.7.1}$$

Then, the transfer function can be expressed in the following cascaded and direct forms,

$$\begin{aligned} H(z) &= \left[\frac{G_0(1 + z^{-1})}{1 + a_{01} z^{-1}}\right]^r \prod_{i=1}^{K} \left[\frac{G_i(1 + z^{-1})^2}{1 + a_{i1} z^{-1} + a_{i2} z^{-2}}\right] \\ &= \frac{G(1 + z^{-1})^M}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_M z^{-M}} \end{aligned} \tag{25.7.2}$$

where the notation $[\ ]^r$ means that the first-order factor is absent if $r = 0$ and present if $r = 1$. The corresponding first-order coefficients are,

$$G_0 = \frac{\Omega_0}{\Omega_0 + 1}, \qquad a_{01} = \frac{\Omega_0 - 1}{\Omega_0 + 1} \tag{25.7.3}$$

The second-order coefficients are , for $i = 1, 2, \dots, K$,

$$
\begin{aligned}
G_i &= \frac{\Omega_0^2}{1-2\Omega_0\cos\theta_i+{\Omega_0}^2}\\
a_{i1} &= \frac{2\,(\Omega_0^2-1)}{1-2\Omega_0\cos\theta_i+{\Omega_0}^2}\,,\quad a_{i2}=\frac{1+2\Omega_0\cos\theta_i+\Omega_0^2}{1-2\Omega_0\cos\theta_i+{\Omega_0}^2}
\end{aligned}
\tag{25.7.4}
$$

where the angles θ_i are defined by,

$$
\boxed{\theta_i=\frac{\pi}{2M}\,(M-1+2i)\,,\quad i=1,2,\dots,K}
\tag{25.7.5}
$$

and the quantity Ω_0 is the equivalent analog 3-dB frequency defined as,

$$
\boxed{\Omega_0=\tan\Big(\frac{\omega_0}{2}\Big)=\tan\Big(\frac{\pi f_0}{f_s}\Big)=\tan\Big(\frac{\pi}{N}\Big)}
\tag{25.7.6}
$$

We note that the filter sections have zeros at $z=-1$, that is, at the Nyquist frequency, $f=f_s/2$, or, $\omega=\pi$. Setting $\Omega=\tan(\omega/2)$, the magnitude response of the designed digital filter can be expressed simply as follows:

$$
|H(\omega)|^2=\frac{1}{1+\big(\Omega/\Omega_0\big)^{2M}}=\frac{1}{1+\big(\tan(\omega/2)/\Omega_0\big)^{2M}}
\tag{25.7.7}
$$

Each section has unity gain at DC. Indeed, setting $z=1$ in Eq. (25.7.2), we obtain the following condition, which can be verified from the given definitions,

$$
\frac{4G_i}{1+a_{i1}+a_{i2}}=1\qquad\text{and}\qquad\frac{2G_0}{1+a_{01}}=1
$$

Moreover, the filter lag can be shown to be (cf. Problem 24.12), for any $M\ge 1$ and $N>2$,

$$
\boxed{\bar{n}=\frac{1}{2\Omega_0\,\sin\Big(\dfrac{\pi}{2M}\Big)}=\frac{1}{2\tan\Big(\dfrac{\pi}{N}\Big)\sin\Big(\dfrac{\pi}{2M}\Big)}\quad\text{(lag)}}
\tag{25.7.8}
$$

For $M\gtrsim 2$ and $N\gtrsim 5$, it can be approximated well by [868],

$$
\bar{n}=\frac{MN}{\pi^2}
$$

The overall numerator gain in the direct form is the product of gains,

$$
G=G_0^r\,G_1G_2\cdots G_K
$$

and the direct-form numerator coefficients are the coefficients of the binomial expansion of $(1+z^{-1})^M$ times the overall gain G. The direct-form denominator coefficients are obtained by convolving the coefficients of the individual sections, that is, setting, $\mathbf{a}=[1]$ if M is even, and, $\mathbf{a}=[1,a_{01}]$ if M is odd, then the vector, $\mathbf{a}=[1,a_1,a_2,\dots,a_M]$, can be constructed recursively by,

$$
\begin{aligned}
&\text{for } i=1,2,\dots,K\\
&\qquad\mathbf{a}=\text{conv}\big(\mathbf{a},\ [1,a_{i1},a_{i2}]\big)
\end{aligned}
\tag{25.7.9}
$$

For example, we have,

$$
\begin{aligned}
&M=2\,,\quad \mathbf{a}=[1,\,a_{11},a_{12}]\\
&M=3\,,\quad \mathbf{a}=\text{conv}\big([1,\,a_{01}],\,[1,\,a_{11},a_{12}]\big)=[1,\,a_{01}+a_{11},\,a_{12}+a_{01}a_{11},\,a_{01}a_{12}]
\end{aligned}
$$

From these, we obtain the following explicit expressions, for $M=2$,

$$
\begin{aligned}
&G=\frac{\Omega_0^2}{\Omega_0^2+\sqrt{2}\Omega_0+1}\,,\quad a_1=\frac{2\,(\Omega_0^2-1)}{\Omega_0^2+\sqrt{2}\Omega_0+1}\,,\quad a_2=\frac{\Omega_0^2-\sqrt{2}\Omega_0+1}{\Omega_0^2+\sqrt{2}\Omega_0+1}\\
&H(z)=\frac{G\,(1+2z^{-1}+z^{-2})}{1+a_1z^{-1}+a_2z^{-2}}\,,\quad \bar{n}=\frac{1}{\sqrt{2}\,\Omega_0}
\end{aligned}
\tag{25.7.10}
$$

and, for $M=3$,

$$
\begin{aligned}
&G=\frac{\Omega_0^3}{(\Omega_0+1)\,(\Omega_0^2+\Omega_0+1)}\,,\quad a_1=\frac{(\Omega_0-1)\,(3\Omega_0^2+5\Omega_0+3)}{(\Omega_0+1)\,(\Omega_0^2+\Omega_0+1)}\\
&a_2=\frac{3\Omega_0^2-5\Omega_0+3}{\Omega_0^2+\Omega_0+1}\,,\quad a_3=\frac{(\Omega_0-1)\,(\Omega_0^2-\Omega_0+1)}{(\Omega_0+1)\,(\Omega_0^2+\Omega_0+1)}\\
&H(z)=\frac{G\,(1+3z^{-1}+3z^{-2}+z^{-3})}{1+a_1z^{-1}+a_2z^{-2}+a_3z^{-3}}\,,\quad \bar{n}=\frac{1}{\Omega_0}
\end{aligned}
\tag{25.7.11}
$$

We note also that the $M=1$ case has lag, $\bar{n}=1/(2\Omega_0)$, and is equivalent to the modified EMA of Eq. (15.2.2). This can be seen by rewriting $H(z)$ in the form,

$$
H(z)=\frac{G_0\,(1+z^{-1})}{1+a_{01}z^{-1}}=\frac{\frac{1}{2}(1-\lambda)\,(1+z^{-1})}{1-\lambda z^{-1}}\,,\quad \lambda=-a_{01}=\frac{1-\Omega_0}{1+\Omega_0}
$$

where $0<\lambda<1$ for $\Omega_0<1$, which requires $N>4$.

The MATLAB function, **bma**, implements the design and filtering operations for any filter order M and any period $N>2$,[†] with usage,

```
[y,nlag,b,a] = bma(x,N,M,yin);   % Butterworth moving average
[y,nlag,b,a] = bma(x,N,M);
```

where

```
x = input signal
N = 3-dB period, need not be integer, but N>2
M = filter order
yin = any Mx1 vector of initial values of the output y
      default, yin = repmat(x(1),M,1)
      yin = 'c' for standard convolutional output

y = output signal
nlag = filter lag
b = [b0, b1, b2, ..., bM], numerator filter coefficients
a = [ 1, a1, a2, ..., aM], denominator filter coefficients
```

[†] the sampling theorem requires, $f_0<f_s/2$, or, $N=f_s/f_0>2$

Fig. 25.7.1 shows the BMA output for Butterworth orders $M = 2,3$ applied to the same Nicor-Gas data of Fig. 25.5.2. It has noticeably shorter lag than SMA. The graphs were produced by the MATLAB code,

```
Y = xlsread('nicor.xls');      % load Nicor-Gas data
Y = Y(1:130,1:4);              % 130 trading days, and [O,H,L,C] prices
y = Y(:,4);                    % closing prices
t = 0:length(y)-1;             % trading days

N = 20;                        % period,    SMA lag = (N-1)/2 = 9.50
[y2,n2] = bma(y,N,2);          % order-2 Butterworth,  lag n2 = 4.46
[y3,n3] = bma(y,N,3);          % order-3 Butterworth,  lag n3 = 6.31

figure; plot(t,sma(y,N), t,y2, t,y3);     % plot SMA, y2, y3
        hold on; ohlc(t,Y,'color','b');   % add OHLC bar chart
```

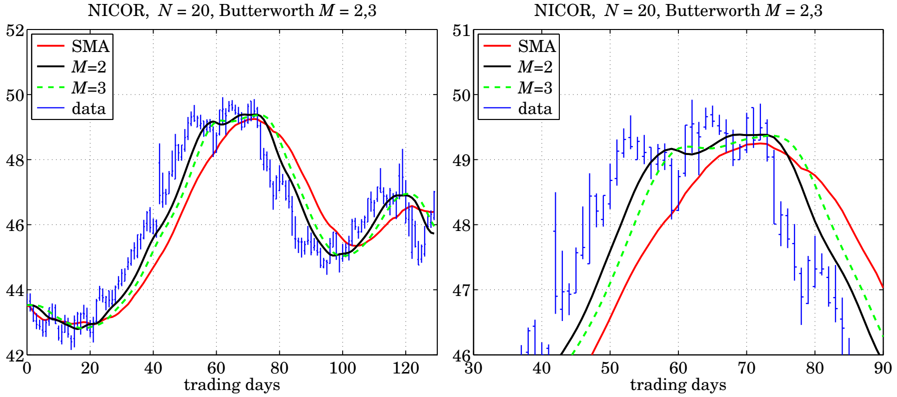


Fig. 25.7.1 Comparison of SMA and Butterworth filters of orders $M = 2,3$.

25.8 Moving Average Filters with Reduced Lag

The PMA/linear regression and the DEMA/TEMA indicators have zero lag by design. There are other useful indicators that are easily implemented and have zero or very much reduced lag. Examples are twicing and Kaiser-Hamming (KH) filter sharpening [644], the Hull moving average (HMA) [893], the zero-lag EMA indicator (ZEMA) [868], the generalized DEMA (GDEMA) [891], and their variants. Here, we discuss a general procedure for constructing such reduced-lag filters, including the corresponding local-slope filters.

Consider three lowpass filters $H_1(z), H_2(z), H_3(z)$ with unity DC gains and lags, $\bar{n}_1, \bar{n}_2, \bar{n}_2$, respectively, and define the following filters for estimating the local level and local slope of the data, generalizing the twicing operations of Eq. (24.11.2),

$$\boxed{\begin{aligned} H_a(z) &= H_1(z)\big[(1+\nu)H_2(z) - \nu H_3(z)\big] = \text{local level} \\ H_b(z) &= \frac{1}{\bar{n}_3 - \bar{n}_2} H_1(z)\big[H_2(z) - H_3(z)\big] = \text{local slope} \end{aligned}} \tag{25.8.1}$$

where ν is a positive constant. One may view $H_a(z)$ as the smoothed, by $H_1(z)$, version of $(1+\nu)H_2(z) - \nu H_3(z)$. The filter $H_a(z)$ will still have unity DC gain as follows by evaluating Eq. (25.8.1) at $z = 1$, that is,

$$H_a(1) = (1+\nu)H_1(1)H_2(1) - \nu H_1(1)H_3(1) = (1+\nu) - \nu = 1$$

Using the fact that the lag of a product of filters is the sum of the corresponding lags (cf. Problem 24.2), we find that the lag of $H_a(z)$ is,

$$\bar{n}_a = (1+\nu)(\bar{n}_1 + \bar{n}_2) - \nu(\bar{n}_1 + \bar{n}_3)\,, \quad \text{or,}$$

$$\boxed{\bar{n}_a = \bar{n}_1 + (1+\nu)\bar{n}_2 - \nu\bar{n}_3} \tag{25.8.2}$$

By appropriately choosing the parameters $\nu, \bar{n}_1, \bar{n}_2, \bar{n}_3$, the lag $\bar{n}_a$ can be made very small, even zero. Indeed, the following choice for ν will generate any particular $\bar{n}_a$,

$$\boxed{\nu = \frac{\bar{n}_1 + \bar{n}_2 - \bar{n}_a}{\bar{n}_3 - \bar{n}_2}} \tag{25.8.3}$$

Below we list a number of examples that are special cases of the above constructions. In this list, the filter $H(z)$, whose lag is denoted by $\bar{n}$, represents any unity-gain lowpass filter, such as WMA, EMA, or SMA and similarly, $H_N(z)$ represents either a length-N FIR filter such as WMA or SMA, or an EMA filter with SMA-equivalent length N. Such filters have a lag related to N via a relationship of the form, $\bar{n} = r\cdot(N-1)$, for example, $r = 1/3$, for WMA, and, $r = 1/2$, for EMA and SMA.

	reduced-lag filters	lag
(twicing)	$H_a(z) = 2H(z) - H^2(z)\,,$	$\bar{n}_a = 0$
(GDEMA)	$H_a(z) = (1+\nu)H(z) - \nu H^2(z)\,,$	$\bar{n}_a = (1-\nu)\bar{n}$
(KH)	$H_a(z) = (1+\nu)H^2(z) - \nu H^3(z)\,,$	$\bar{n}_a = (2-\nu)\bar{n}$
(HMA)	$H_a(z) = H_{\sqrt{N}}(z)\big[2H_{N/2}(z) - H_N(z)\big]\,,$	$\bar{n}_a = r\big[\sqrt{N} - 2\big]$
(ZEMA)	$H_a(z) = 2H(z) - z^{-d}H(z)\,,$	$\bar{n}_a = \bar{n} - d$
(ZEMA)	$H_a(z) = (1+\nu)H(z) - \nu z^{-d}H(z)\,,$	$\bar{n}_a = \bar{n} - \nu d$

(25.8.4)

The corresponding local-slope filters are as follows (they do not depend on ν),

$$
\begin{array}{|ll|}
\hline
 & \text{local-slope filters} \\
\hline
\text{(DEMA/GDEMA)} & H_b(z) = \dfrac{1}{\bar{n}}\big[H(z) - H^2(z)\big] \\[2ex]
\text{(KH)} & H_b(z) = \dfrac{1}{\bar{n}}\big[H^2(z) - H^3(z)\big] \\[2ex]
\text{(HMA)} & H_b(z) = \dfrac{2}{rN}\,H_{\sqrt{N}}(z)\big[H_{N/2}(z) - H_N(z)\big] \\[2ex]
\text{(ZEMA)} & H_b(z) = \dfrac{1}{d}\big[H(z) - z^{-d}H(z)\big] \\
\hline
\end{array}
\qquad (25.8.5)
$$

The standard twicing method, $H_a(z) = 2H(z) - H^2(z)$, coincides with DEMA if we choose $H(z)$ to be a single EMA filter,

$$
H_{\text{EMA}}(z) = \frac{\alpha}{1 - \lambda z^{-1}}\,, \quad \alpha = 1 - \lambda\,, \quad \lambda = \frac{N-1}{N+1}\,, \quad \bar{n} = \frac{N-1}{2} \qquad (25.8.6)
$$

but the filter $H(z)$ can also be chosen to be an SMA, WMA, or BMA filter, leading to what may be called, “double SMA,” or, “double WMA,”, or, ‘double BMA.”

The generalized DEMA, $H_{\text{GDEMA}}(z) = (1+v)H(z) - vH^2(z)$, also has, $H = H_{\text{EMA}}$, and is usually operated in practice with $v = 0.7$. It reduces to standard DEMA for $v = 1$. The so-called Tillson’s *T3 indicator* [891] is obtained by cascading GDEMA three times,

$$
H_{\text{T3}}(z) = \big[H_{\text{GDEMA}}(z)\big]^3 \qquad \text{(T3 indicator)} \qquad (25.8.7)
$$

The Kaiser-Hamming (KH) filter sharpening case is not currently used as an indicator, but it has equally smooth output as GDEMA and T3. It reduces to the standard filter sharpening case with zero lag for $v = 2$.

In the original Hull moving average [893], $H_a(z) = H_{\sqrt{N}}(z)\big[2H_{N/2}(z) - H_N(z)\big]$, the filter H_N is chosen to be a length-N weighted moving average (WMA) filter, as defined in Eqs. (25.2.1) and (25.2.2), and similarly, $H_{N/2}$ and $H_{\sqrt{N}}$ are WMA filters of lengths $N/2$ and $\sqrt{N}$ respectively. Assuming for the moment that these filter lengths are integers, then the corresponding lags of the three WMA filters, $H_{\sqrt{N}}, H_{N/2}, H_N$, will be,

$$
\bar{n}_1 = \frac{\sqrt{N}-1}{3}\,, \quad \bar{n}_2 = \frac{N/2-1}{3}\,, \quad \bar{n}_3 = \frac{N-1}{3}\,,
$$

and setting $v = 1$ in Eq. (25.8.2), we find,

$$
\bar{n}_a = \bar{n}_1 + 2\bar{n}_2 - \bar{n}_3 = \frac{\sqrt{N}-1}{3} + \frac{N-2}{3} - \frac{N-1}{3} = \frac{\sqrt{N}-2}{3} \qquad (25.8.8)
$$

Thus, for larger Ns, the lag is effectively reduced by a factor of $\sqrt{N}$. The extra filter factor $H_{\sqrt{N}}(z)$ provides some additional smoothing. In practice, the filter lengths $N_1 = \sqrt{N}$ and $N_2 = N/2$ are replaced by their rounded values. This changes the lag $\bar{n}_a$ somewhat. If one wishes to maintain the same lag as that given by Eq. (25.8.8), then one can compensate for the replacement of N_1, N_2 by their rounded values by using a slightly different value for v. It is straightforward to show that the following procedure will generate the desired lag value, where the required v is evaluated from Eq. (25.8.3),

$$
\begin{aligned}
N_1 &= \text{round}\big(\sqrt{N}\big)\,, \quad \varepsilon_1 = N_1 - \sqrt{N} = \text{rounding error} \\
N_2 &= \text{round}\left(\frac{N}{2}\right), \quad \varepsilon_2 = N_2 - \frac{N}{2} = \text{rounding error} \\
v &= \frac{\bar{n}_1 + \bar{n}_2 - \bar{n}_a}{\bar{n}_3 - \bar{n}_2} = \frac{N/2 + \varepsilon_1 + \varepsilon_2}{N/2 - \varepsilon_2} \\
\bar{n}_a &= \frac{N_1 - 1}{3} + (1+v)\frac{N_2 - 1}{3} - v\frac{N-1}{3} = \frac{\sqrt{N}-2}{3} \\
\bar{n}_3 - \bar{n}_2 &= \frac{N - N_2}{3}
\end{aligned}
\qquad (25.8.9)
$$

with transfer functions,

$$
\begin{aligned}
H_a(z) &= H_{N_1}(z)\big[(1+v)H_{N_2}(z) - vH_N(z)\big] = \text{local level} \\
H_b(z) &= \frac{1}{\bar{n}_3 - \bar{n}_2}\,H_{N_1}(z)\big[H_{N_2}(z) - H_N(z)\big] = \text{local slope}
\end{aligned}
\qquad (25.8.10)
$$

The WMA filters in the HMA indicator can be replaced with EMA filters resulting in the so-called “exponential Hull moving average” (EHMA), which has been found to be very competitive with other indicators [937]. Because N does not have to be an integer in EMA, it is not necessary to round the lengths $N_1 = \sqrt{N}$ and $N_2 = N/2$, and one can implement the indicator as follows, where H_N denotes the single EMA of Eq. (25.8.6),

$$
\begin{aligned}
&\bar{n}_a = \frac{\sqrt{N}-2}{2}\,, \quad \bar{n}_3 - \bar{n}_2 = \frac{N}{4} \\
&H_a(z) = H_{\sqrt{N}}(z)\big[2H_{N/2}(z) - H_N(z)\big] \\
&H_b(z) = \frac{4}{N}\,H_{\sqrt{N}}(z)\big[H_{N/2}(z) - H_N(z)\big]
\end{aligned}
$$

One can also replace the WMA filters by SMAs leading to the “simple Hull moving average” (SHMA). The filter H_N now stands for a length-N SMA filter, resulting in $\bar{n}_a = (\sqrt{N}-1)/2$, and $\bar{n}_3 - \bar{n}_2 = (N - N_2)/2$. Except for these changes, the computational procedure outlined in Eq. (25.8.9) remains the same.

The following MATLAB code illustrates the computation of the local-level output signal a_n from the data y_n, for the three versions of HMA and a given value of $N > 1$,

```
N1 = round(sqrt(N));  e1 = N1 - sqrt(N);
N2 = round(N/2);      e2 = N2 - N/2;

v = (N/2 + e1 + e2) / (N/2 - e2);

a = wma((1+v)*wma(y,N2) - v*wma(y,N), N1);     % HMA
a = sma((1+v)*sma(y,N2) - v*sma(y,N), N1);     % SHMA
a = sema(2*sema(y,N/2) - sema(y,N), sqrt(N));  % EHMA
```

The functions, **hma**, **shma**, **ehma**, which are discussed below, implement these operations but offer more options, including the computation of the slope signals.

In the zero-lag EMA (ZEMA or ZLEMA) indicator [868], $H_a(z) = 2H(z) - z^{-d}H(z)$, the filter $H(z)$ is chosen to be a single EMA filter of the form of Eq. (25.8.6), and the delay d is chosen to coincide with the filter lag, that is, $d = \bar{n} = (N-1)/2$. It follows from Eq. (25.8.4) that the lag will be exactly zero, $\bar{n}_a = \bar{n} - d = 0$. This assumes that $\bar{n}$ is an integer, which happens only for odd N. For even N, the delay d may be chosen as the rounded-up version of $\bar{n}$, that is,

$$d = \text{round}(\bar{n}) = \text{round}\left(\frac{N-1}{2}\right) = \frac{N}{2}\,, \quad N = \text{even}$$

Then, the lag $\bar{n}_a$ can still be made to be zero by choosing the parameter v such that $\bar{n}_a = \bar{n} - vd = 0$, or, $v = \bar{n}/d = \bar{n}/\text{round}(\bar{n})$. Thus, the generalized form of the ZEMA indicator is constructed by,

$$\begin{aligned}
&\bar{n} = \frac{N-1}{2}\,, \quad d = \text{round}(\bar{n})\,, \quad v = \frac{\bar{n}}{d} \\
&H_a(z) = (1+v)H(z) - vz^{-d}H(z) \\
&H_b(z) = \frac{1}{d}\left[H(z) - z^{-d}H(z)\right]
\end{aligned} \tag{25.8.11}$$

The code segment below illustrates the computation of the local-level ZEMA signal. It uses the function, **delay**, which implements the required delay.

```
nbar = (N-1)/2;
d = round(nbar);
v = nbar/d;
a = (1+v)*sema(y,N) - v*delay(sema(y,N), d);   % ZEMA
```

The following MATLAB functions implement the reduced-lag filters discussed above, where the input array **y** represents the financial data to be filtered, and the outputs **a**, **b** represent the local-level and local-slope signals.

```
[a,b] = hma(y,N,yin);       % Hull moving average
[a,b] = ehma(y,N,yin);      % exponential Hull moving average
[a,b] = shma(y,N,yin);      % simple Hull moving average
[a,b] = zema(y,N,yin);      % zero-lag EMA

y = delay(x,d);             % d-fold delay, y(n) = x(n-d)
a = gdema(y,N,v,yin);       % generalized DEMA
a = t3(y,N,v,yin);          % Tillson's T3
```

The input variable `yin` defines the initialization and defaults to progressive filtering for **hma**, **shma**, and **zema**, `yin='f'`, and to, `yin` $= y_0$, for **ehma**.

Fig. 25.8.1 compares the PMA/linear regression indicator with HMA, EHMA, and ZEMA on the same Nicor-Gas data, with filter length $N = 20$. Fig. 25.8.2 compares the corresponding slope indicators. The MATLAB code below illustrates the computation.

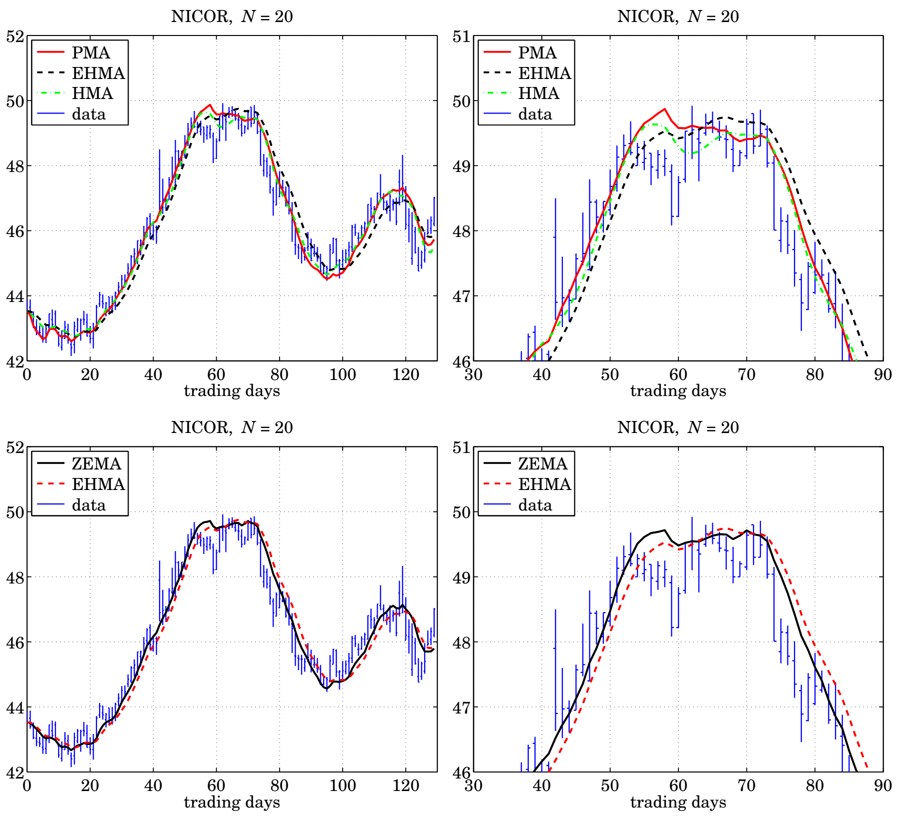


Fig. 25.8.1 Comparison of PMA/LREG, HMA, EHMA, and ZEMA indicators.

```
Y = xlsread('nicor.xls');     % load Nicor-Gas data
Y = Y(1:130,1:4);             % keep 130 trading days, and [O,H,L,C] prices
y = Y(:,4);                   % closing prices
t = 0:length(y)-1;            % trading days

N = 20;                                   % filter length

[al,bl] = lreg(y,N);                      % PMA/LREG
[ah,bh] = hma(y,N);                       % HMA
[ae,be] = ehma(y,N);                      % EHMA
[az,bz] = zema(y,N);                      % ZEMA

figure; plot(t,al, t,ae, t,ah);           % PMA/LREG, EHMA, HMA
hold on; ohlc(t,Y);                       % add OHLC chart

figure; plot(t,az, t,ae);                 % ZEMA, EHMA
hold on; ohlc(t,Y);                       % add OHLC chart
```

```
figure; plot(t,bh, t,be); hold on;     % HMA, EHMA slopes
stem(t,bl,'marker','none');            % plot LREG slope as stem

figure; plot(t,bh, t,bz); hold on;     % HMA, ZEMA slopes
stem(t,bl,'marker','none');
```

We note that the reduced-lag HMA, EHMA, and ZEMA local-level and local-slope filters have comparable performance as the PMA/linear regression and DEMA/TEMA filters, with a comparable degree of smoothness.

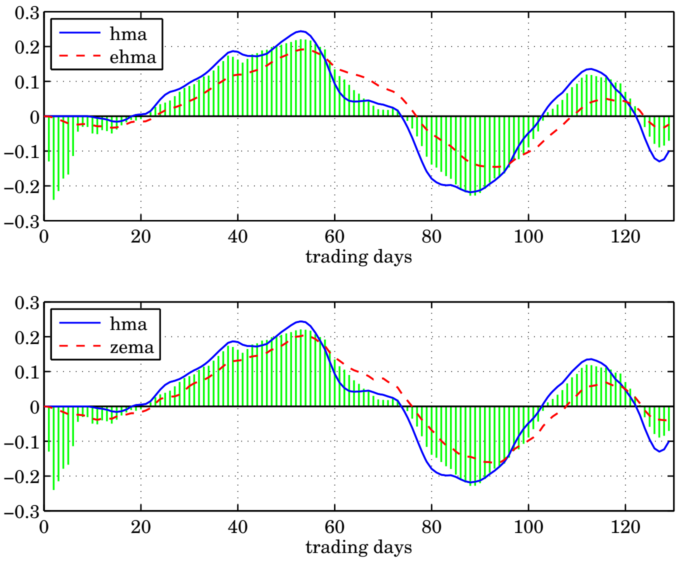


Fig. 25.8.2 Slope indicators, linear regression (stem) vs. HMA, EHMA, ZEMA.

25.9 Zigzag Indicator

The zigzag indicator [864,929–934] is a trend indicator constructed by joining by straight lines the so-called “swing-highs” and “swing-lows” of a price series.

The swing points are local extrema (local maxima or minima) determined by requiring that the price variations in the immediate neighborhoods of the extrema be greater than a specified percentage amount, for example, if $y(n)$ is a point to the left or to the right of a local maximum or minimum, $y_{\max}$ or $y_{\min}$, then, if $p < 1$ is the specified percentage amount (p = percentage/100), the following conditions must be satisfied by $y_{\max}, y_{\min}$,

$$y_{\max} - y(n) > p \cdot y_{\max} \qquad \text{(at local maximum)}$$

$$y(n) - y_{\min} > p \cdot y_{\min} \qquad \text{(at local minimum)}$$

Fig. 25.9.1 shows the IBM closing prices over the time period from 9/1/2000 to 5/29/2001. The zigzag indicator is plotted for three values of $p = 5, 10, 20$ percent.

Larger values p result in more filtering of small price variations, showing more clearly the broader features and major trends of the data.

It should be noted that the last leg of the zigzag is not a reliable indicator of future trends because the next future data point can alter that leg. See Refs. [930–934] that address this issue and try to improve the zigzag.

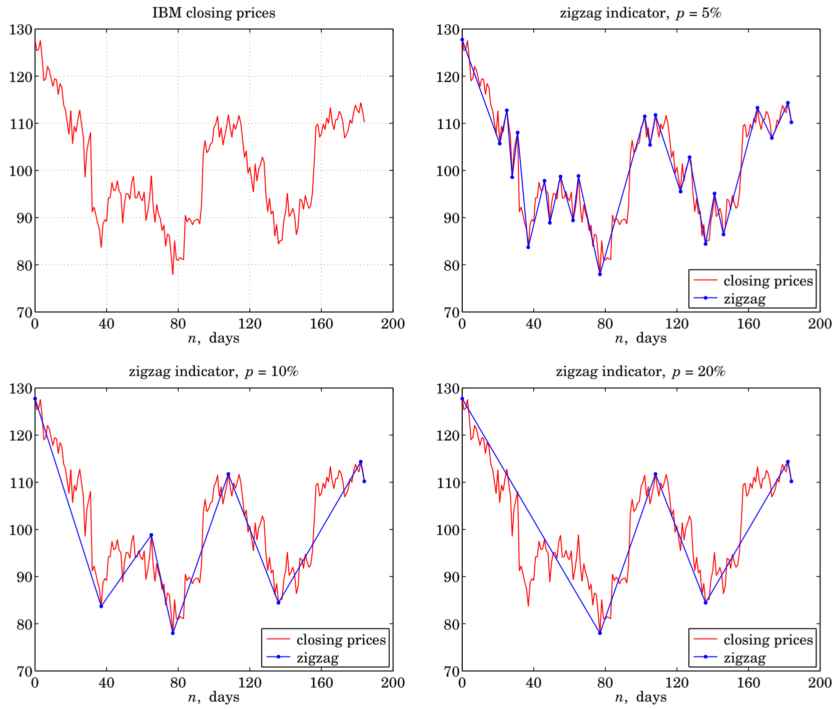


Fig. 25.9.1 Zigzag indicator examples.

The zigzag indicator can be implemented by the MATLAB function, **zigzag**, in the ISP2e toolbox, with usage,

```
[z,nz] = zigzag(y,p);        % zigzag indicator

y = closing values, or other indicator
p = percentage changes to be ignored, e.g., p = 8 for 8%
z  = zigzag points of swing-highs and swing-lows
nz = zigzag time instants, note (MATLAB index) = nz+1
```

The MATLAB code for generating the above graphs was as follows:

```
Y = xlsread('IBM-sep00-jun01.xlsx');   % data file in ISP2e toolbox
```

```
Y = Y(:,1:4);                         % extract O,H,L,C prices
y = Y(:,4);                           % closing prices
n = 0:length(y)-1;                    % trading days

p = 10;                               % 10 percent

[z,nz] = zigzag(y,p);                 % zigzag points and times

figure; plot(n,y,'r-', nz,z,'b.-');   % z-points connected by straight lines
```

25.10 L_0 Trend Indicator

A similar indicator to the zigzag, that also captures the broad variations and major trends of the data, can be constructed using the sparse versions of the Whittaker-Henderson smoothing problem, as we discuss in Sec. 26.7. In particular, the L_0 version provides the sparsest solution and is ideally suited for this purpose.

Given a length-N vector of observations, y_n, $0 \le n \le N-1$, the L_0 version minimizes the following L_0-regularized least-squares performance index, for determining a length-N smoothed signal x_n,

$$\mathcal{J} = \sum_{n=0}^{N-1} \left| y_n - x_n \right|^2 + \lambda \sum_{n=s}^{N-1} \left| \nabla^s x_n \right|^0 = \|\mathbf{y} - \mathbf{x}\|_2^2 + \lambda \|D_s \mathbf{x}\|_0 = \min \tag{25.10.1}$$

where the L_0 norm[†], $\|D_s\mathbf{x}\|_0$, is the cardinality of the vector $D_s\mathbf{x}$, that is, the number of its non-zero entries, and D_s is the $(N-s)\times N$ convolution matrix corresponding to the s-difference operator ∇^s. It can be constructed in MATLAB by,

```
Ds = diff(eye(N),s);    % or, sparsely, Ds = diff(speye(N),s);
```

The choice $s = 2$ results in an almost piecewise linear solution x_n, whereas, $s = 1$ results in a piecewise constant solution. The piecewise linear case is better suited if the price data exhibit upgoing or downgoing zigzag trends, while the piecewise constant case would be more suited for data trending sideways.

The criterion (25.10.1) forces the L_0 term to become sparse, i.e., $D_s\mathbf{x}$ is sparse, being almost zero essentially everywhere, except at the kink points. If $s = 2$, the kink points are where the slopes change — with $D_s\mathbf{x}$ being the 2nd difference of $\mathbf{x}$, the condition that $D_s\mathbf{x} \approx 0$ implies that $x(n)$ would be an approximately linear function of time n, resulting in almost linear segments between kink points. On the other hand, if $s = 1$, then $D_s\mathbf{x}$ is the first difference of $\mathbf{x}$, and the sparsity of $D_s\mathbf{x}$ would imply that $x(n)$ would be almost piecewise constant.

Larger values of the regularization parameter λ, result in sparser $D_s\mathbf{x}$, and capture more broadly the trends in the data. Smaller values of λ, follow the data more closely.

The minimization of Eq. (25.10.1) can be carried out using an iterative reweighted least-squares (IRLS) iteration as discussed in Sec. 26.7. The following MATLAB function, **l0trend**, in the ISP2e toolbox, implements this indicator with usage,

[†] L_0 is not strictly-speaking a proper norm.

```
[x,k] = l0trend(y,lambda);       % L0 trend indicator

y = closing values, or other indicator
lambda = L0-regularized least-squares lambda parameter
x = zigzag piecewise (almost) linear estimate of y
k = time instants of kink points, note (MATLAB index) = k+1
```

There are some additional input parameters that allow one to set the value of s (by default $s = 2$), as well the number of IRLS iterations, and the threshold for determining the kink points — see the help file for the function, `l0trend.m`, for details.

Fig. 25.10.1 plots the L_0 trend indicator for the same IBM data as in Fig. 25.9.1. As expected, the larger value of λ captures the major trends, whereas the smaller value tries to follow the data more closely. Fig. 25.10.2 shows the L_0 trend for piecewise constant segments.

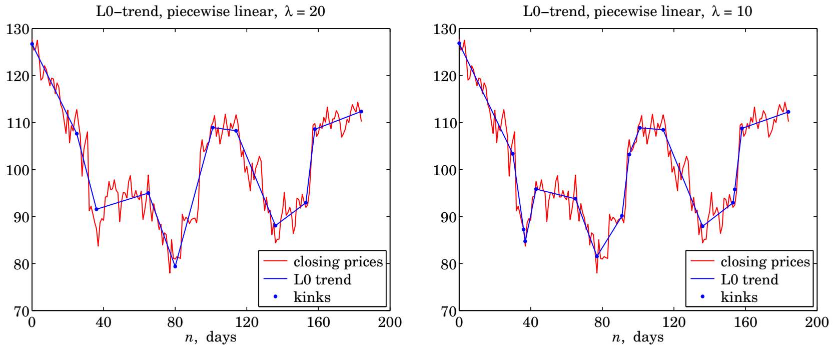


Fig. 25.10.1 L_0 trend indicator – almost piecewise linear ($s = 2$).

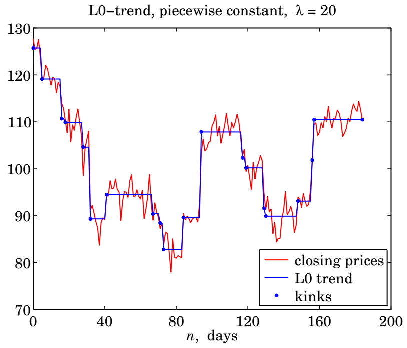


Fig. 25.10.2 L_0 trend indicator – almost piecewise constants ($s = 1$).

The following MATLAB code segment demonstrates how the above graphs were generated, including how to use the extra input parameters for the $s = 1$ case,

```
Y = xlsread('IBM-sep00-jun01.xlsx');   % data file in ISP2e toolbox
Y = Y(:,1:4);                          % extract O,H,L,C prices
y = Y(:,4);                            % closing prices
n = 0:length(y)-1;                     % trading days

la = 20;                               % lambda regularization parameter

[x,k] = l0trend(y,la);                 % s=2 case by default

figure; plot(n,y,'r-', n,x,'b-', k,x(k+1),'b.');

ek = 0.1;                              % kink threshold
K = 10;                                % number of IRLS iterations
s = 1;                                 % s can only be 1 or 2
[x,k] = l0trend(y,la,ek,K,s);          % s=1 case

figure; plot(n,y,'r-', n,x,'b-', k,x(k+1),'b.');
```

This indicator can also be used to make predictions of future values by simply extrapolating the last leg using its effective slope. This can be very risky, especially if the end-point becomes an actual kink point in the future of the data – however, it should be OK if the current trend continues.

Fig. 25.10.3 shows the previous examples, extrapolated 15 days into the future. The following MATLAB code demonstrates how to calculate the predicted values, d-steps ahead into the future, with the two cases $s = 2$ and $s = 1$ combined into one,

```
[x,k] = l0trend(y,la,ek,K,s);   % estimate x, and kink points k
d = ...                         % choose a value for d
k2 = k(end);                    % last kink point
k1 = k(end-1);                  % penultimate kink point
x2 = x(k2+1); x1 = x(k1+1);     % estimated values at the kink points

kpred = k2+1 : k2+d             % range of prediction times

ypred = x2 + (s-1) * (kpred-k2) * (x2-x1)/(k2-k1)   % predictions
```

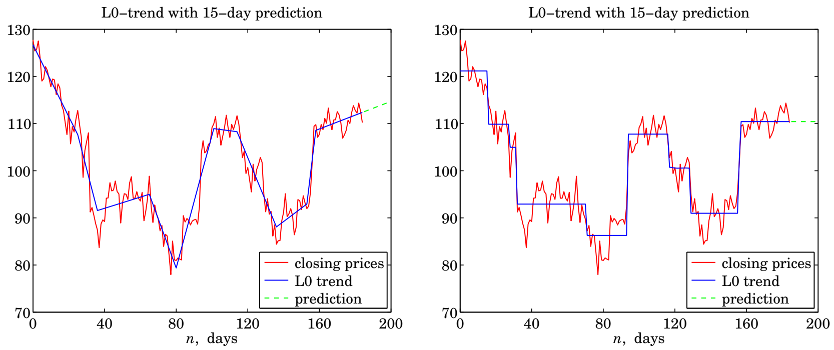


Fig. 25.10.3 L_0 trends with 15-day predictions.

25.11 Envelopes, Bands, and Channels

Moving averages help market traders discern trends by smoothing out variations in the data. However, such variations can provide additional useful information, such as gauging volatility, or, identifying extremes in the data that provide trading opportunities, or observing how prices settle into their trends.

Trading envelopes or bands or channels consist of two curves drawn above and below a moving average trendline. The two bounds define a zone of variation, or volatility, about the average, within which most of the price fluctuations are expected to lie.

The typical trading rule is that when a price closes near or above the upper bound, it signals that the stock is overbought and suggests trading in the opposite direction. Similarly, if a price moves below the lower bound it signals that the stock is oversold and suggests an opposite reaction.

In this section we discuss the following types of bands and their computation,

- Bollinger bands
- Projection bands
- Fixed-width bands
- Starc bands
- Standard-error bands
- Donchian channels
- Keltner bands
- Parabolic SAR

Examples of these are shown in Figs. 25.11.1 and 25.11.2 applied to the same Nicor data that we used previously. Below we give brief descriptions of how such bands are computed. We use our previously discussed MATLAB functions, such as SMA, LREG, etc., to summarize the computations. Further references are given in [907–918].

Bollinger Bands

Bollinger bands [907–911] are defined relative to an N-day SMA of the closing prices, where typically, $N = 14$. The two bands are taken to be two standard deviations above and below the SMA. The following MATLAB code clarifies the computation,

```
M = sma(y,N);                    % N-point SMA of closing prices y
S = stdev(y,N);                  % std-dev relative to M
L = M - 2*S;                     % lower bound
U = M + 2*S;                     % upper bound
```

where the function, **stdev**, uses the built-in function **std** to calculate the standard deviation over each length-N data window, and its essential code is,

```
for n=1:length(y),
   S(n) = std(y(max(1,n-N+1):n));
end
```

where the data window length is N for $n \geq N$, and n during the initial transients $n < N$.

Standard-Error Bands

Standard-error bands [913] use the PMA/linear-regression moving average as the middle trendline and shift it by two standard errors up and down. The calculation is summarized below with the help of the function **lreg**, in which the quantities, **y, a, se**, represent the closing prices, the local level, and the standard error,

```
[a,~,~,se] = lreg(y,N);    % N-point linear regression
L = a - 2*se;              % lower bound
U = a + 2*se;              % upper bound
```

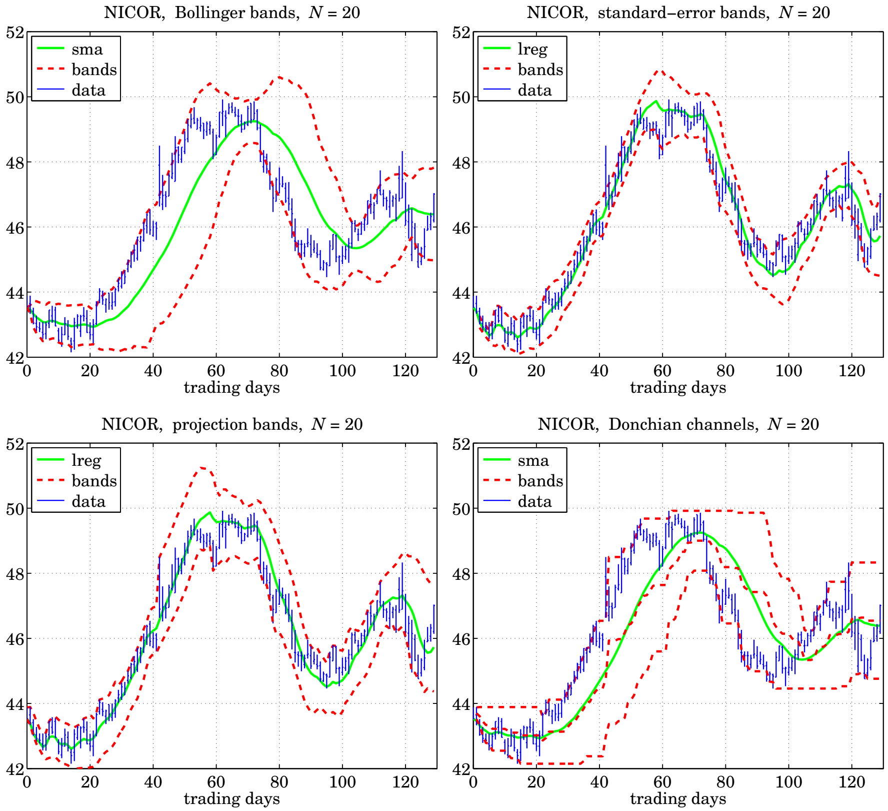


Fig. 25.11.1 Bollinger bands, standard-error bands, projection bands, and Donchian channels.

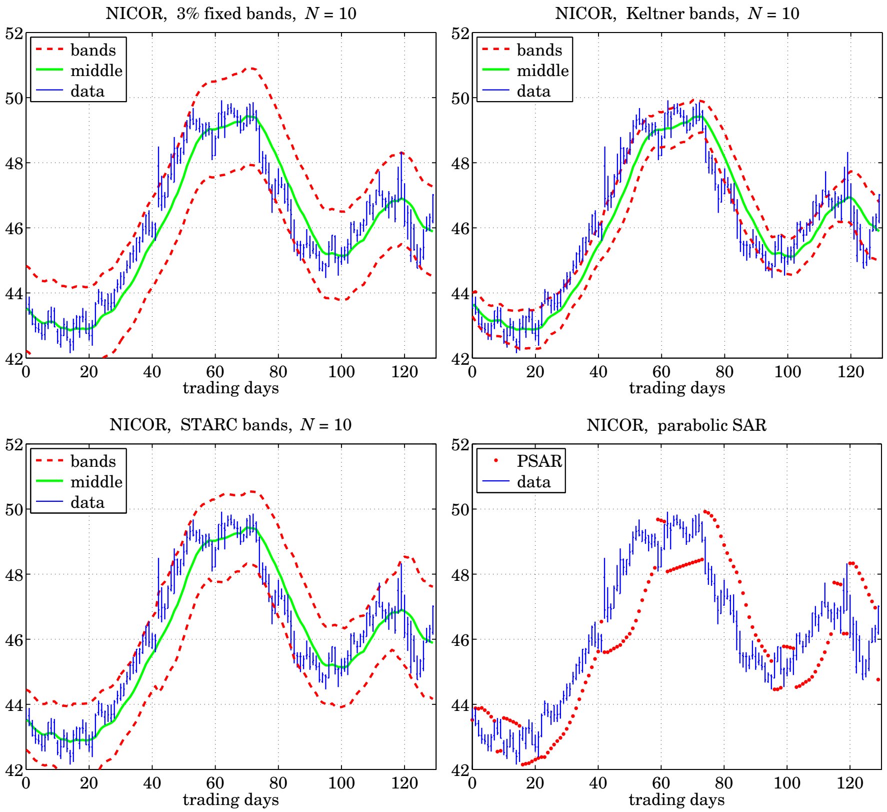


Fig. 25.11.2 Fixed-width bands, Keltner bands, STARC bands, and parabolic SAR.

Projection Bands

Projection bands [912] also use the linear regression function **lreg** to calculate the local slopes for the high and low prices, H, L. The value of the upper (lower) band at the n-th time instant is determined by considering the values of the highs H (lows L) over the look-back period, $n - N + 1 \le t \le n$, extrapolating each of them linearly according to their slope to the current time instant n, and taking the maximum (minimum) among them. The following MATLAB code implements the procedure,

```
[~,bL] = lreg(L,N);     % linear regression slope for Low
[~,bH] = lreg(H,N);     % linear regression slope for High

for n=0:length(H)-1,
   t = (max(0,n-N+1) : n)';          % look-back interval
   Lo(n+1) = min(L(t+1) + bL(n+1)*(n-t));    % lower band
   Up(n+1) = max(H(t+1) + bH(n+1)*(n-t));    % upper band
end
```

Donchian Channels

Donchian channels [915] are constructed by finding, at each time instant n, the highest high (resp. lowest low) over the past time interval, $n-N \le t \le n-1$, that is, the value of the upper bound at the n-th day, is the maximum of the highs over the previous N days, not including the current day, i.e., $\max[H_{n-1}, H_{n-2}, \dots, H_{n-N}]$. The code below describes the procedure,

```
for n = 2:length(H)         % n is MATLAB index
   t = max(1,n-N) : n-1;    % past N days
   Lo(n) = min(L(t));       % lower band
   Up(n) = max(H(t));       % upper band
end

Mid = (Up + Lo)/2;          % middle band
```

Fixed-Width Bands

Fixed-width bands or envelopes [914] shift an N-point SMA of the closing prices by a certain percentage, such as, typically, 3 percent,

```
M = sma(C,N);       % N-point SMA of closing prices C

L = M - p*M;        % lower band, e.g., p = 0.03
U = M + p*M;        % upper band
```

Keltner Bands

Keltner bands or channels [914], use as the middle trendline an N-point SMA of the average of the high, low, and closing prices, $(H+L+C)/3$, and use an N-point SMA of the difference $(H-L)$ as the bandwidth, representing a measure of volatility. The code below implements the operations,

```
M = sma((H+L+C)/3,N);    % SMA of (H+L+C)/3
D = sma(H-L,N);          % SMA of (H-L)

L = M - D;               % lower band
U = M + D;               % upper band
```

The typical value of N is 10, and the trading rule is that a “buy” signal is generated when the closing price C lies above the upper band, and a “sell” signal when C lies below the lower band.

Starc Bands

In Starc[†] bands [914] all three prices, high, low, and closing, H, L, C, are used. The middle band is an N-point SMA of the closing prices C, but the bandwidth is defined in terms of an N_a-point of the so-called “average true range” (ATR), which represents another measure of volatility. The code below describes the computation,

```
M = sma(C,N);            % SMA of closing prices
R = atr([H,L,C],Na);     % ATR = average true range

L = M - 2*R;             % lower and
U = M + 2*R;             % upper band
```

The ATR [865] is an N_a-point WEMA of the “true range”, defined as follows, at each time n,

$$\begin{aligned} T_n &= \max\big[H_n - L_n,\; H_n - C_{n-1},\; C_{n-1} - L_n\big] = \text{true range in } n\text{-th day} \\ R_n &= \text{wema}(T_n, N_a) = \text{ATR} \end{aligned} \tag{25.11.1}$$

and is implemented by the function **atr**, with the help of the **delay** function. Its essential MATLAB code is as follows, where `H,L,C` are column vectors,

```
T = max([H-L, H-delay(C,1), delay(C,1)-L], [], 2);      % row-wise max
R = wema(T,N);
```

MATLAB Functions

The following MATLAB functions implement the band indicators discussed above, where the various parameters are fully explained in the help files for these functions,

```
[L,U,M] = bbands(y,N,d);           % Bollinger bands
[L,U,a] = sebands(y,N,d,init);     % standard-error bands
[L,U,R,Rs] = pbands(Y,N,Ns);       % projection bands & oscillator
[L,U,M] = donch(Y,N);              % Donchian channels
[L,U,M] = fbands(Y,N,p);           % fixed-width bands
[L,U,M] = kbands(Y,N);             % Keltner bands
[L,U,M] = stbands(Y,N,Na);         % Starc bands

S = stdev(y,N,flag);               % standard deviation

[R,TR] = atr(Y,N);                 % average true range
```

The essential MATLAB code for generating Figs. 25.11.1 and 25.11.2 is as follows,

```
Y = xlsread('nicor.xls');   % load Nicor-Gas data
Y = Y(1:130,1:4);           % 130 trading days, and [O,H,L,C] prices
y = Y(:,4);                 % closing prices
t = 0:length(y)-1;          % trading days

N = 20;                                % used in Fig.6.22.1
```

[†] Stoller Average Range Channels

```
[L,U,M] = bbands(y,N);                   % Bollinger
figure; ohlc(t,Y); hold on;              % make OHLC bar chart
plot(t,M,’g-’, t,U,’r--’, t,L,’r--’);

[L,U,a] = sebands(y,N);                  % standard-error
figure; ohlc(t,Y); hold on;
plot(t,a,’g-’, t,U,’r--’, t,L,’r--’);

a = lreg(y,N);
[L,U] = pbands(Y,N);                     % projection
figure; ohlc(t,Y); hold on;
plot(t,a,’g-’, t,U,’r--’, t,L,’r--’);

[L,U,M] = donch(Y,N);                    % Donchian
figure; ohlc(t,Y); hold on;
plot(t,M,’r--’, t,L,’r--’, t,U,’r--’);
plot(t,sma(y,N),’g-’);

N=10;                                    % used in Fig.6.22.2

p=0.03;
[L,U,M] = fbands(Y,N,p);                 % fixed-width
figure; ohlc(t,Y); hold on;
plot(t,M,’g-’, t,U,’r--’, t,L,’r--’);

[L,U,M] = kbands(Y,N);                   % Keltner
figure; ohlc(t,Y); hold on;
plot(t,M,’g-’, t,U,’r--’, t,L,’r--’);

[L,U,M] = stbands(Y,N);                  % Starc
figure; ohlc(t,Y); hold on;
plot(t,M,’g-’, t,U,’r--’, t,L,’r--’);

H = Y(:,2); L = Y(:,3); ni=1; Ri=1;      % parabolic SAR
S = psar(H,L,Ri,ni);
figure; ohlc(t,Y); hold on;
plot(t,S,’r.’);
```

Parabolic SAR

Wilder’s parabolic stop & reverse (SAR) [865] is a trend-following indicator that helps a trader to switch positions from long to short or vice versa.

While holding a long position during a period of increasing prices, the SAR indicator lies *below* the prices and is also increasing. When the prices begin to fall and touch the SAR from above, then a “sell” signal is triggered with a reversal of position from long to short, and the SAR switches sides and begins to fall, lying *above* the falling prices. If subsequently, the prices begin to rise again and touch the SAR from below, then a “buy” signal is triggered and the position is reversed from short to long again, and so on.

The indicator is very useful as it keeps a trader constantly in the market and works well during trending markets with steady periods of increasing or decreasing prices, even though it tends to recommend buying relatively high and selling relatively low—the opposite of what is the ideal. It does not work as well during “sideways” or trading markets causing so-called “whipsaws.” It is usually used in conjunction with other indicators that confirm trend, such as the RSI or DMI. Some further references on the SAR are [919–924].

The SAR is computed in terms of the high and low price signals H_n, L_n and is defined as the exponential moving average of the *extreme price* reached within each trending period, but it uses a time-varying EMA parameter, $\lambda_n = 1 - \alpha_n$, as well as additional conditions that enable the reversals. Its basic EMA recursion from day n to day $n{+}1$ is,

$$S_{n+1} = \lambda_n S_n + \alpha_n E_n = (1-\alpha_n)S_n + \alpha_n E_n\,, \quad \text{or,}$$

$$\boxed{S_{n+1} = S_n + \alpha_n(E_n - S_n)} \quad \text{(SAR)} \tag{25.11.2}$$

where E_n is the extreme price reached during the current trending position, that is, the highest high reached up to day n during an up-trending period, or the lowest low up to day n during a down-trending period. At the beginning of each trending period, S_n is initialized to be the extreme price of the previous trending period.

The EMA factor α_n increases linearly with time, starting with an initial value, α_i, at the beginning of each trending period, and then increasing by a fixed increment $\Delta\alpha$, but only every time a *new* extreme value is reached, that is,

$$\alpha_{n+1} = \begin{cases} \alpha_n + \Delta\alpha\,, & \text{if} \quad E_{n+1} \neq E_n \\ \alpha_n\,, & \text{if} \quad E_{n+1} = E_n \end{cases} \tag{25.11.3}$$

where we note that $E_{n+1} \neq E_n$ happens when E_{n+1} is strictly greater than E_n during an up-trend, or, E_{n+1} is strictly less than E_n during a down-trend. Moreover, an additional constraint is that α_n is not allowed to exceed a certain maximum value, α_m. The values recommended by Wilder [865] are,

$$\alpha_i = 0.02\,, \quad \Delta\alpha = 0.02\,, \quad \alpha_m = 0.2$$

Because of the increasing α_n parameter, the EMA has a time-varying decreasing lag,[†] thus, tracking more quickly the extreme prices as time goes by. As a result, S_n has a particular curved shape that resembles a parabola, hence the name “parabolic” SAR.

The essential steps in the calculation of the SAR are summarized in the following MATLAB code, in which the inputs are the quantities, `H,L`, representing the high and low price signals, H_n, L_n, while the output quantities, `S,E,a,R`, represent, S_n, E_n, α_n, R_n, where R_n holds the current position and is equal to ±1 for long/short.

[†] The EMA equivalent length decreases from, $N_i = 2/\alpha_i - 1 = 99$, down to, $N_m = 2/\alpha_m - 1 = 9$.

```
Hi = max(H(1:ni));            % initial highest high, default
Li = min(L(1:ni));            % initial lowest low

R(ni) = Ri;                   % initialize outputs at starting time n=ni
a(ni) = ai;
S(ni) = Li*(Ri==1) + Hi*(Ri==-1);
E(ni) = Hi*(Ri==1) + Li*(Ri==-1);

for n = ni : length(H)-1

   S(n+1) = S(n) + a(n) * (E(n) - S(n));                % SAR update

   r = R(n);                                        % current position

   if (r==1 & L(n+1)<=S(n+1)) | (r==-1 & H(n+1)>=S(n+1))   % reversal
      r = -r;                                           % reverse r
      S(n+1) = E(n);                                    % reset new S
      E(n+1) = H(n+1)*(r==1) + L(n+1)*(r==-1);          % reset new E
      a(n+1) = ai;                                      % reset new a
   else                                                 % no reversal
      if n>2                                            % new S
        S(n+1) = min([S(n+1), L(n-1), L(n)])*(r==1) ...  % additional
               + max([S(n+1), H(n-1), H(n)])*(r==-1);    % conditions
      end

      E(n+1) = max(E(n),H(n+1))*(r==1) ...              % new E
             + min(E(n),L(n+1))*(r==-1);

      a(n+1) = min(a(n) + (E(n+1)~=E(n)) * Da, am);     % new a
   end

   R(n+1) = r;                                          % new R

end  % for-loop
```

If the current trading position is long ($r = 1$), corresponding to an up-trending market, then, a reversal of position to short ($r = -1$) will take place at time $n+1$ if the low price L_{n+1} touches or becomes less than the SAR, that is, if, $L_{n+1} \le S_{n+1}$. Similarly, if the current position is short, corresponding to a down-trending market, then, a reversal of position to long will take place at time $n+1$ if the high price H_{n+1} touches or becomes greater than the SAR, that is, if, $H_{n+1} \ge S_{n+1}$. At such reversal time points, the SAR is reset to be equal to the extreme price of the previous trend, that is, $S_{n+1} = E_n$, and the E_{n+1} is reset to be either L_{n+1} if reversing to short, or H_{n+1} if reversing to long, and the EMA parameter is reset to, $\alpha_{n+1} = \alpha_i$.

An additional condition is that during an up-trend, the SAR for tomorrow, S_{n+1}, is not allowed to become greater that either today's or yesterday's lows, L_n, L_{n-1}, and in such case it is reset to the minimum of the two lows. Similarly, during a down-trend, the S_{n+1} is not allowed to become less that either today's or yesterday's highs, H_n, H_{n-1}, and is reset to the maximum of the two highs. This is enforced by the code line,

$$S_{n+1} = \min\big([S_{n+1}, L_{n-1}, L_n]\big)\cdot(r{=}{=}1) + \max\big([S_{n+1}, H_{n-1}, H_n]\big)\cdot(r{=}{=}{-1})$$

The parabolic SAR is implemented with the MATLAB function **psar**, with usage,

```
[S,E,a,R] = psar(H,L,Ri,ni,af,Hi,Li);     % parabolic SAR
```

```
H  = vector of High prices, column
L  = vector of Low prices, column, same length as H
```

```
Ri = starting position, long Ri = 1, short Ri = -1
ni = starting time index, default ni = 1,  all outputs are NaNs for n<ni
af = [ai,da,am] = [initial EMA factor, increment, maximum factor]
     default, af = [0.02, 0.02, 0.2]
Hi,Li = initial high and low used to initialize S(n),E(n) at n=ni,
        default, Hi = max(H(1:ni)), Li = min(L(1:ni))

S = parabolic SAR, same size as H
E = extremal price, same size as H
a = vector of EMA factors, same size as H
R = vector of positions, R = +1/-1 for long/short, same size as H
```

The SAR signal S_n is usually plotted with dots, as shown for example, in the bottom right graph of Fig. 25.11.2. Fig. 25.11.3 shows two more examples.

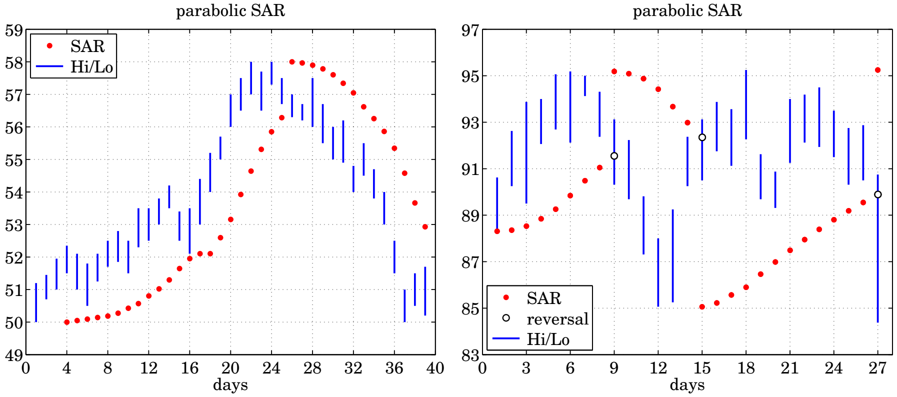


Fig. 25.11.3 Parabolic SAR examples from Wilder [865] and Achelis [864].

The left graph reproduces Wilder's original example [865] and was generated by the following MATLAB code,[†]

```
Y = xlsread('psarexa.xls');
t = Y(:,1); H = Y(:,2); L = Y(:,3);          % extract H,L signals

Ri = 1; ni = 4;                              % initialize SAR
[S,E,a,R] = psar(H,L,Ri,ni);

num2str([t, H, L, a, E, S, R], '%8.2f');

figure; ohlc(t,[H,L], t,S,'r.');             % make OHLC plot, including SAR
```

which reproduces table on p.13 of [865]. The right graph is from Achelis [864]. The SAR is plotted with filled dots, but at the end of each trending period and shown with open circles are the points that triggered the reversals. The MATLAB code for this example is similar to the above,[†]

[†] data are from Wilder [865]

[†] data are from Wilder [864]

```
Y = xlsread('psarexb.xls');
t = Y(:,1); H = Y(:,2); L = Y(:,3);

Ri = 1; ni = 1;                             % initialize
[S,E,a,R] = psar(H,L,Ri,ni);                % compute SAR

num2str([t ,H, L, a, E, S, R], '%9.4f');
figure; ohlc(t,[H,L], t,S,'r.');            % make OHLC plot, including SAR
```

which reproduces the same table from [864]. The first up-trending period ends at day $n = 9$ at which the would be SAR, shown as an opened-circle, lies above the low of that day, thus, causing a reversal to short and that SAR is then replaced with the filled-circle value that lies above the highs, and corresponds to the highest high of the previous period that had occurred on day $n = 6$.

The second down-trending period ends at $n = 15$ at which point the SAR, shown as an opened-circle, is breached by the high on that day, thus causing a reversal to long, and the SAR is reset to the filled-circle value on that day lying below the data, and corresponds to the lowest low during the previous period that had been reached on day $n = 12$. Finally, another such reversal takes place on day $n = 27$ and the SAR is reset to the highest high that had occurred on day $n = 18$. To clarify, we list below the values of the SAR at the reversal points before and after the reversal takes place,

n	$S_{\text{before}}(n)$	$S_{\text{after}}(n)$
9	91.5448	$95.1875 = H_6$
15	92.3492	$85.0625 = L_{12}$
27	89.8936	$95.2500 = H_{18}$

25.12 Momentum, Oscillators, and Other Indicators

There exist several other indicators that are used in technical analysis, many of them built on those we already discussed. The following MATLAB functions implement some of the more popular ones, several are also included in MATLAB's financial toolbox. Additional references can be found in the Achelis book [864] and in [865–928,935–937].

```
R = rsi(y,N,type);                 % relative strength index, RSI
R = cmo(y,N);                      % Chande momentum oscillator, CMO
R = vhfilt(y,N);                   % Vertical Horizontal Filter, VHF
[Dp,Dm,DX,ADX] = dirmov(Y,N);      % directional movement system, +-DI,DX,ADX
---------------------------------------------------------------------------
[y,yr,ypr] = mom(x,d,xin);         % momentum and price rate of change
[y,ys,ypr] = prosc(x,N1,N2,N3);    % price oscillator & MACD
[pK,pD] = stoch(Y,K,Ks,D,M);       % stochastic, %K, %D oscillators
---------------------------------------------------------------------------
R = accdist(Y);                    % accumulation/distribution line
R = chosc(Y,N1,N2);                % Chaikin oscillator
R = cmflow(Y,N);                   % Chaikin money flow
R = chvol(Y,N);                    % Chaikin volatility
---------------------------------------------------------------------------
[P,N] = pnvi(Y,P0);                % positive/negative volume indices, PVI/NVI
R = cci(Y,N);                      % commodity channel index, CCI
R = dpo(Y,N);                      % detrended price oscillator, DPO
[R,N] = dmi(y,Nr,Ns,Nm);           % dynamic momentum index, DMI
[R,Rs] = forosc(y,N,Ns);           % forecast oscillator
---------------------------------------------------------------------------
[R,Rs] = trix(y,N,Ns,yin);         % TRIX oscillator
a = vema(y,N,Nv);                  % variable-length EMA
```

Below we discuss briefly their construction. Examples of their use are included in their respective help files. Several online examples can be found in the Fidelity Guide [935] and in the TradingView Wiki [936].

Relative Strength Index, RSI

The relative strength index (RSI) was introduced by Wilder [865] to be used in conjunction with the parabolic SAR to confirm price trends. It is computed as follows, where **y** is the column vector of daily closing prices,

```
x = diff(y);                 % price differences
xu = +x.*(x>0);              % upward differences
xd = -x.*(x<=0);             % downward differences
su = wema(xu,N);             % smoothed differences
sd = wema(xd,N);
RSI = 100*su/(su+sd);        % RSI
```

Chande Momentum Oscillator, CMO

If the **wema** function is replaced by **sma**, one obtains the Chande momentum oscillator,

```
x = diff(y);                   % price differences
xu = +x.*(x>0);                % upward differences
xd = -x.*(x<=0);               % downward differences
su = sma(xu,N);                % smoothed differences
sd = sma(xd,N);

CMO = 100*(su-sd)/(su+sd);     % CMO
```

Thus, the SMA-based RSI is related to CMO via,

$$\text{CMO} = 2\,\text{RSI} - 100 \quad \Leftrightarrow \quad \text{RSI} = \frac{\text{CMO} + 100}{2}$$

Vertical Horizontal Filter, VHF

The vertical horizontal filter (VHF) is similar to the RSI or CMO and helps to confirm a trend in trending markets. It is computed as follows, where the first N outputs are `NaN`s,

```
x = [NaN; diff(y)];            % y = column of closing prices
                               % x = price differences
for n=N+1:length(y),
   yn = y(n-N+1:n);            % length-N look-back window
   xn = x(n-N+1:n);
   R(n) = abs(max(yn)-min(yn)) / sum(abs(xn));     % VHF
end
```

Directional Movement System

The directional movement system was also proposed by Wilder [865] and consists of several indicators, the plus/minus directional indicators, (±DI), the directional index (DX), and the average directional index (ADX). These are computed as follows,

```
R = atr(Y,N);                        % average true range

DH = [0;  diff(H)];                  % high price differences
DL = [0; -diff(L)];                  % low price differences

Dp = DH .* (DH>DL) .* (DH>0);        % daily directional movements
Dm = DL .* (DL>DH) .* (DL>0);        %

Dp = wema(Dp,N);                     % averaged directional movements
Dm = wema(Dm,N);                     %

Dp = 100 * Dp ./ R;                  % +DI,-DI directional indicators
Dm = 100 * Dm ./ R;
DX = 100*abs(Dp - Dm)./(Dp + Dm);    % directional index, DI
ADX = wema(DX,N);                    % average directional index, ADX
```

Momentum and Price Rate of Change

In its simplest form a momentum indicator is the difference between a price today, $x(n)$, and the price d days ago, $x(n-d)$, but it can also be expressed as a ratio, or as a percentage, referred to as *price rate of change*,

$$\begin{aligned} y(n) &= x(n) - x(n-d) = \text{momentum} \\ y_r(n) &= 100\cdot\frac{x(n)}{x(n-d)} = \text{momentum as ratio} \\ y_p(n) &= 100\cdot\frac{x(n) - x(n-d)}{x(n-d)} = \text{price rate of change} \end{aligned}$$

It can be implemented simply with the help of the function, **delay**,

```
y   = x - delay(x,d);
yr  = x./delay(x,d) * 100;
yp = (x-delay(x,d))./delay(x,d) * 100;
```

Price Oscillator and MACD

The standard moving average convergence/divergence (MACD) indicator is defined as the difference between two EMAs of the daily closing prices: a length-12 shorter/faster EMA and a length-26 longer/slower EMA. A length-9 EMA of the MACD difference is also computed as a trigger signal.

Typically, a buy (sell) signal is indicated when the MACD rises above (falls below) zero, or when it rises above (falls below) its smoothed signal line.

The MACD can also be represented as a percentage resulting into the price oscillator, and also, different EMA lengths can be used. The following code segment illustrates the computation, where **x** are the closing prices,

```
y1 = sema(x,N1);      % fast EMA, default N1=12
y2 = sema(x,N2);      % slow EMA, default N2=26

y = y1 - y2;          % MACD

ys = sema(y,N3);      % smoothed MACD signal, default N3=9

ypr = 100 * y./y2;    % price oscillator
```

Stochastic Oscillator

```
H = Y(:,1); L = Y(:,2); C = Y(:,3);          % extract H,L,C inputs

Lmin = NaN(size(C)); Hmax = NaN(size(C));    % NaNs for n<K

for n = K:length(C),                         % look-back period K
   Lmin(n) = min(L(n-K+1:n));                % begins at n=K
   Hmax(n) = max(H(n-K+1:n));
end

pK = 100 * sma(C-Lmin, Ks) ./ sma(Hmax-Lmin, Ks);   % percent-K

pD = sma(Pk, D);                                    % percent-D
```

Fast Stochastic has $K_s = 1$, i.e., no smoothing, and *Slow Stochastic* has, typically, $K_s = 3$.

Accumulation/Distribution

```
H=Y(:,1); L=Y(:,2); C=Y(:,3); V=Y(:,4);    % extract H,L,C,V
R = cumsum((2*C-H-L)./(H-L).*V);           % ACCDIST
```

Chaikin Oscillator

```
y = accdist(Y);                  % Y = [H,L,C,V] data matrix
R = sema(y,N1) - sema(y,N2);     % CHOSC, default, N1=3, N2=10
```

Chaikin Money Flow

```
H=Y(:,1); L=Y(:,2); C=Y(:,3); V=Y(:,4);        % extract H,L,C,V
R = sma((2*C-H-L)./(H-L).*V, N) ./ sma(V,N);   % CMFLOW
```

Chaikin Volatility

```
S = sema(H-L,N);                            % H,L given
R = (S - delay(S,N)) ./ delay(S,N) * 100;   % volatility
```

Positive/Negative Volume Indices, PNVI

These are defined recursively as follows, in MATLAB-like notation, where C_n, V_n are the closing prices and the volume,

$$P_n = P_{n-1} + (V_n > V_{n-1})\cdot\frac{C_n - C_{n-1}}{C_{n-1}}\cdot P_{n-1} = P_{n-1}\left(\frac{C_n}{C_{n-1}}\right)^{(V_n > V_{n-1})} = \text{PVI}$$

$$N_n = N_{n-1} + (V_n < V_{n-1})\cdot\frac{C_n - C_{n-1}}{C_{n-1}}\cdot N_{n-1} = N_{n-1}\left(\frac{C_n}{C_{n-1}}\right)^{(V_n < V_{n-1})} = \text{NVI}$$

and initialized to some arbitrary initial value, such as, $P_0 = N_0 = 1000$. The MATLAB implementation uses the function, **delay**,

```
P = P0 * cumprod( (C./delay(C,1)) .^ (V>delay(V,1)) );   % PNVI
N = P0 * cumprod( (C./delay(C,1)) .^ (V<delay(V,1)) );
```

Commodity Channel Index, CCI

```
T = (H+L+C)/3;                              % H,L,C given
M = sma(T,N);
for n=N+1:length(C),
   D(n) = mean(abs(T(n-N+1:n) - M(n)));     % mean deviation
end
R = (T-M)./D/0.015;                         % CCI
```

Detrended Price Oscillator, DPO

```
S = sma(y,N,’n’);           % y = column of closing prices
M = floor(N/2) + 1;         % advancing time
R = y - delay(S,-M,’n’);    % DPO, i.e., R(n) = y(n) - S(n+M)
```

Dynamic Momentum Index, DMI

```
x = [NaN; diff(y)];         % y = column of closing prices
xu =  x .* (x>0);           % upward differences
xd = -x .* (x<=0);          % downward differences
S = stdev(y,Ns);            % Ns-period stdev
V = S ./ sma(S,Nm);         % volatility measure
N = floor(Nr ./ V);         % effective length
N(N>Nmax) = Nmax;           % restrict max and min N
N(N<Nmin) = Nmin;
Nstart = Nr + Ns + Nm;
su1 = mean(xu(2:Nstart));       % initialize at start time
sd1 = mean(xd(2:Nstart));
switch lower(type)
  case ’wema’                   % WEMA type
    for n = Nstart+1:length(y),
       su(n) = su1 + (xu(n) - su1) / N(n); su1 = su(n);
       sd(n) = sd1 + (xd(n) - sd1) / N(n); sd1 = sd(n);
    end
  case ’sma’                    % SMA type
    for n = Nstart+1:length(y),
       su(n) = mean(xu(n-N(n)+1:n));
       sd(n) = mean(xd(n-N(n)+1:n));
    end
end
R = 100 * su./(su+sd);          % DMI
```

Forecast Oscillator

```
yp = pma(y,N,1);            % time series forecast
x = y - delay(yp,1);
R = 100 * x./y;             % forecast oscillator
Rs = sma(R,Ns);             % trigger signal
```

TRIX Oscillator

```
[~,~,~,~,~,a3] = tema(y,N,cin);     % triple EMA
R = 100*(a3 - delay(a3,1))./a3;     % TRIX
Rs = sma(R,Ns);                     % smoothed TRIX
```

Variable-Length EMA

```
la = (N-1)/(N+1); al = 1-la;          % EMA parameter
switch lower(type)
  case 'cmo'                          % CMO volatility
    V = abs(cmo(y,Nv))/100;
  case 'r2'                           % R^2 volatility
    [~,~,V] = lreg(y,Nv);
end
for n=Nv+1:length(y),                 % default si=y(Nv)
   s(n) = si + al*V(n)*(y(n)-si);     % EMA recursion
   si = s(n);
end
```

25.13 MATLAB Functions

We summarize the MATLAB functions discussed in this chapter:

```
% ------------------------------------
% Exponential Moving Average Functions
% ------------------------------------
% ema       - exponential moving average - exact version
% stema     - steady-state exponential moving average
% lpbasis   - fit order-d polynomial to first L inputs
% emap      - map equivalent lambdas's between d=0 and d=1 EMAs
% emaerr    - MSE, MAE, MAPE error criteria
% emat      - transformation matrix from polynomial to cascaded basis
% mema      - multiple exponential moving average
% holt      - Holt's exponential smoothing
% holterr   - MSE, MAE, MAPE error criteria for Holt
```

The technical analysis functions are:

```
% ----------------------------
% Technical Analysis Functions
% ----------------------------
% accdist   - accumulation/distribution line
% atr       - true range & average true range
% cci       - commodity channel index
% chosc     - Chaikin oscillator
% cmflow    - Chaikin money flow
% chvol     - Chaikin volatility
% cmo       - Chande momentum oscillator
% dirmov    - directional movement system, +-DI, DX, ADX
% dmi       - dynamic momentum index (DMI)
```

```
% dpo       - detrended price oscillator
% forosc    - forecast oscillator
% pnvi      - positive and negative volume indices, PVI, NVI
% prosc     - price oscillator & MACD
% psar      - Wilder's parabolic SAR
% rsi       - relative strength index, RSI
% stdev     - standard deviation index
% stoch     - stochastic oscillator, %K, %D oscillators
% trix      - TRIX oscillator
% vhfilt    - Vertical Horizontal Filter
%
% ------------ moving averages ---------------------------
%
% bma       - Butterworth moving average
% dema      - steady-state double exponential moving average
% ehma      - exponential Hull moving average
% gdema     - generalized dema
% hma       - Hull moving average
% ilrs      - integrated linear regression slope indicator
% delay     - delay or advance by d samples
% mom       - momentum and price rate of change
% lreg      - linear regression, slope, and R-squared indicators
% pma       - predictive moving average, linear fit
% pmaimp    - predictive moving average impulse response
% pma2      - predictive moving average, polynomial order d=1,2
% pmaimp2   - predictive moving average impulse response, d=1,2
% sema      - single exponential moving average
% shma      - SMA-based Hull moving average
% sma       - simple moving average
% t3        - Tillson's T3 indicator, triple gdema
% tema      - triple exponential moving average
% tma       - triangular moving average
% vema      - variable-length exponential moving average
% wema      - Wilder's exponential moving average
% wma       - weighted or linear moving average
% zema      - zero-lag EMA
%
% --------------- bands ----------------------------------
%
% bbands    - Bollinger bands
% donch     - Donchian channels
% fbands    - fixed-envelope bands
% kbands    - Keltner bands or channels
% pbands    - Projection bands and projection oscillator
% sebands   - standard-error bands
% stbands   - STARC bands
%
% --------------- misc ----------------------------------
%
% ohlc      - make Open-High-Low-Close bar chart
% ohlcyy    - OHLC with other indicators on the same graph
% yylim     - adjust left/right ylim
%
% r2crit    - R-squared critical values
% tcrit     - critical values of Student's t-distribution
% tdistr    - cumulative t-distribution
```

25.14 Computer Project – Markowitz Portfolio Theory

This project, divided into separate questions, deals with Markowitz's optimum mean-variance portfolio theory [940–951]. It represents a special case of a linearly-constrained quadratic optimization problem. The project develops the following topics from financial engineering:

- optimum mean-variance Markowitz portfolios
- efficient frontier between risk and return
- quantifying risk aversion
- two mutual fund theorem
- inequality-constrained portfolios without short selling
- risk-free assets and tangency portfolio
- capital asset line and the Sharp ratio
- market portfolios and capital market line
- stock's beta, security market line, risk premium
- capital asset pricing model (CAPM)
- minimum-variance with multiple constraints

1. ***Mean-Variance Portfolio Theory.*** The following constrained optimization problem finds application in investment analysis and optimum portfolio selection in which one tries to balance *return* versus *risk*.[†]

   Suppose one has identified M stocks or assets into which to invest,

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{bmatrix}$$

   where y_i is a random variable that represents the value of the ith asset. From historical data, the expected returns of the individual assets are assumed to be known,

$$E[\mathbf{y}] = \mathbf{m} = \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ m_M \end{bmatrix}$$

   as are the cross-correlations between the assets,

$$R_{ij} = E\big[(y_i - m_i)(y_j - m_j)\big]\,, \quad 1 \le i,j \le M$$

   or, matrix-wise,

$$R = E\big[(\mathbf{y} - \mathbf{m})(\mathbf{y} - \mathbf{m})^T\big]$$

[†] Harry Markowitz received the Nobel prize in economics for this work.

where R is assumed to have full rank. The variance $\sigma_i^2 = R_{ii}$ is a measure of the *volatility*, or *risk*, of the ith asset.

An investment portfolio is selected by choosing the percentage a_i to invest in the ith asset y_i. The value of the portfolio is defined by the random variable:

$$y = \sum_{i=1}^{M} a_i y_i = [a_1, a_2, \dots, a_M]\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{bmatrix} = \mathbf{a}^T\mathbf{y}$$

where the weights a_i must add up to unity (negative weights are allowed, describing so-called "short sells"):

$$\sum_{i=1}^{M} a_i = [a_1, a_2, \dots, a_M]\begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} = \mathbf{a}^T\mathbf{u} = 1\,, \quad \text{where} \quad \mathbf{u} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

The expected *return* of the portfolio and its variance or *risk* are given by:

$$\begin{aligned} \mu &= E[y] = E[\mathbf{a}^T\mathbf{y}] = \mathbf{a}^T\mathbf{m} = \text{return} \\ \sigma^2 &= E\big[(y-\mu)^2\big] = \mathbf{a}^T R\mathbf{a} = \text{risk} \end{aligned}$$

An *optimum portfolio* may be defined by finding the weights that minimize the risk σ^2 for a given value of the return μ, that is,

$$\sigma^2 = \mathbf{a}^T R\,\mathbf{a} = \min\,, \quad \text{subject to} \quad \begin{cases} \mathbf{a}^T\mathbf{m} = \mu \\ \mathbf{a}^T\mathbf{u} = 1 \end{cases} \tag{25.14.1}$$

(a) Incorporate the constraints by means of two Lagrange multipliers, λ_1, λ_2, and minimize the modified performance index:

$$J = \frac{1}{2}\mathbf{a}^T R\,\mathbf{a} + \lambda_1(\mu - \mathbf{a}^T\mathbf{m}) + \lambda_2(1 - \mathbf{a}^T\mathbf{u}) = \min$$

Show that the quantities $\{\mathbf{a}, \lambda_1, \lambda_2\}$ can be obtained from the solution of the $(M+2)\times(M+2)$ linear system of equations:

$$\begin{bmatrix} R & -\mathbf{m} & -\mathbf{u} \\ \mathbf{m}^T & 0 & 0 \\ \mathbf{u}^T & 0 & 0 \end{bmatrix}\begin{bmatrix} \mathbf{a} \\ \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mu \\ 1 \end{bmatrix} \tag{25.14.2}$$

where the invertibility of this matrix requires that the vectors $\mathbf{m}$ and $\mathbf{u}$ not be collinear.

(b) Show that the solution of this system for **a** takes the form:

$$\mathbf{a} = \lambda_1 R^{-1}\mathbf{m} + \lambda_2 R^{-1}\mathbf{u} \tag{25.14.3}$$

and that λ_1, λ_2 can be obtained from the reduced 2×2 linear system:

$$\begin{bmatrix} A & B \\ B & C \end{bmatrix}\begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} \mu \\ 1 \end{bmatrix} \quad \Rightarrow \quad \begin{aligned} \lambda_1 &= \frac{\mu C - B}{D} \\ \lambda_2 &= \frac{A - \mu B}{D} \end{aligned}$$

where, A, B, C, D, are defined in terms of $\mathbf{m}, R$ by

$$\begin{aligned} A &= \mathbf{m}^T R^{-1}\mathbf{m} \\ B &= \mathbf{m}^T R^{-1}\mathbf{u} \\ C &= \mathbf{u}^T R^{-1}\mathbf{u} \\ D &= AC - B^2 \end{aligned}$$

(c) Show that the quantities A, C, D are non-negative.

(d) Show that the minimized value of the risk $\sigma^2 = \mathbf{a}^T R\mathbf{a}$ can be written in the form:

$$\sigma^2 = \mu\lambda_1 + \lambda_2 = \frac{C\mu^2 - 2B\mu + A}{D} \tag{25.14.4}$$

Thus, the dependence of the variance σ^2 on the return μ has a parabolic shape, referred to as the *efficient frontier*.

(e) The apex of this parabola is obtained by minimizing Eq. (25.14.4) with respect to μ. Setting $\partial\sigma^2/\partial\mu = 0$, show that the absolute minimum is reached for the following values of the return, risk, and weights:

$$\mu_0 = \frac{B}{C}, \quad \sigma_0^2 = \frac{1}{C}, \quad \mathbf{a}_0 = \frac{R^{-1}\mathbf{u}}{\mathbf{u}^T R^{-1}\mathbf{u}} \tag{25.14.5}$$

(f) Show that Eq. (25.14.4) can be re-expressed as

$$\boxed{\sigma^2 = \sigma_0^2 + \frac{C}{D}(\mu - \mu_0)^2} \quad \text{(efficient frontier)} \tag{25.14.6}$$

which can be solved for μ in terms of σ^2, as is common in practice:

$$\mu = \mu_0 \pm \sqrt{\frac{D}{C}}\sqrt{\sigma^2 - \sigma_0^2} \tag{25.14.7}$$

Of course, only the upper sign corresponds to the efficient frontier because it yields higher return for the same risk. (See some example graphs below.)

(g) Show that the optimum portfolio of Eq. (25.14.3) can be written in the form:

$$\mathbf{a} = \mathbf{a}_0 + \frac{C}{D}(\mu - \mu_0)\, R^{-1}(\mathbf{m} - \mu_0\,\mathbf{u}) \tag{25.14.8}$$

where $\mathbf{a}_0$ was defined in Eq. (25.14.5). Thus, as expected, $\mathbf{a} = \mathbf{a}_0$, if $\mu = \mu_0$.

(h) Show that the optimum portfolio of Eq. (25.14.3) can be written in the form

$$\mathbf{a} = \mu\,\mathbf{g} + \mathbf{h}$$

where $\mathbf{g}, \mathbf{h}$ depend only on the asset statistics $\mathbf{m}, R$ and are independent of μ. Moreover, show that, $\mathbf{m}^T\mathbf{g} = 1$, and, $\mathbf{m}^T\mathbf{h} = 0$, and that, $\mathbf{u}^T\mathbf{g} = 0$, and, $\mathbf{u}^T\mathbf{h} = 1$. In particular, show that $\mathbf{g}, \mathbf{h}$ are given by,

$$\begin{aligned} \mathbf{g} &= \frac{C}{D}R^{-1}\mathbf{m} - \frac{B}{D}R^{-1}\mathbf{u} \\ \mathbf{h} &= \frac{A}{D}R^{-1}\mathbf{u} - \frac{B}{D}R^{-1}\mathbf{m} \end{aligned}$$

(i) Consider two optimal portfolios $\mathbf{a}_1$ and $\mathbf{a}_2$ having return-risk values that lie on the efficient frontier, μ_1, σ_1 and μ_2, σ_2, satisfying Eq. (25.14.6), and assume that $\mu_1 < \mu_2$. Using the results of the previous question, show that any other optimum portfolio with return-risk pair μ, σ, such that $\mu_1 < \mu < \mu_2$, can be constructed as a linear combination of $\mathbf{a}_1, \mathbf{a}_2$ as follows, with positive weights p_1, p_2, such that, $p_1 + p_2 = 1$,

$$\mathbf{a} = p_1\mathbf{a}_1 + p_2\mathbf{a}_2\,, \quad p_1 = \frac{\mu_2 - \mu}{\mu_2 - \mu_1}\,, \quad p_2 = \frac{\mu - \mu_1}{\mu_2 - \mu_1}$$

Thus, the investor need only invest in the two *standard portfolios* $\mathbf{a}_1$ and $\mathbf{a}_2$ in the proportions p_1 and p_2, respectively. This is known as the *two mutual fund theorem*. The restriction $\mu_1 < \mu < \mu_2$ can be relaxed if short selling of the funds is allowed.

(j) Consider a portfolio of four assets having return (given as annual rate) and covariance matrix:

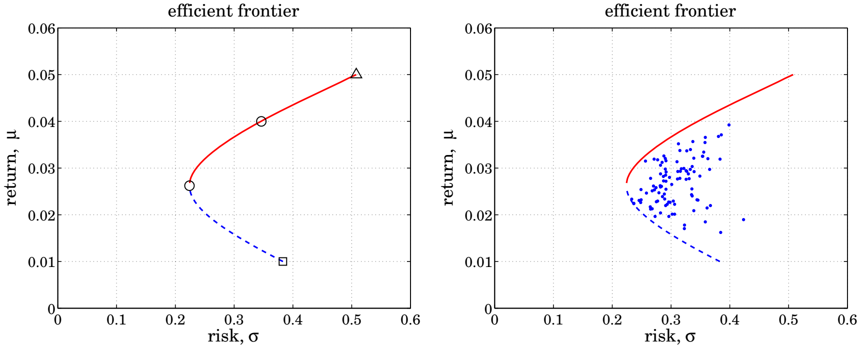
$$\mathbf{m} = \begin{bmatrix} 0.02 \\ 0.03 \\ 0.01 \\ 0.05 \end{bmatrix}, \quad R = \begin{bmatrix} 0.10 & -0.02 & 0.04 & -0.01 \\ -0.02 & 0.20 & 0.05 & 0.02 \\ 0.04 & 0.05 & 0.30 & 0.03 \\ -0.01 & 0.02 & 0.03 & 0.40 \end{bmatrix}$$

Make a plot of the efficient frontier of μ versus σ according to Eq. (25.14.7). To do so, choose 51 equally-spaced μs in the interval $[\min(\mathbf{m}), \max(\mathbf{m})]$ and calculate the corresponding σs.

Compute the risk σ and weight vector **a** that would achieve a return of $\mu = 0.04$ and indicate that point on the efficient frontier.

Why wouldn't an investor want to put all his/her money in stock y_4 since it has a higher return of $m_4 = 0.05$?

(k) Generate 100 weight vectors, $\mathbf{a}$, randomly (but such that $\mathbf{a}^T\mathbf{u} = 1$), compute the values of the quantities, $\mu = \mathbf{a}^T\mathbf{m}$, and, $\sigma^2 = \mathbf{a}^T R\mathbf{a}$, and make a scatterplot of the points (μ, σ) to see that they lie on or below the efficient frontier.



2. ***Risk aversion.*** A somewhat different minimization criterion is often chosen for the portfolio selection problem, that is,

$$J = \frac{1}{2}\mathbf{a}^T R\mathbf{a} - \gamma\mathbf{a}^T\mathbf{m} = \min\,, \quad \text{subject to } \mathbf{u}^T\mathbf{a} = 1 \tag{25.14.9}$$

where γ is a positive parameter that quantifies the investor's aversion for risk versus return—small γ emphasizes low risk, larger γ, high return.

For various values of γ, the optimum weights $\mathbf{a}$ are determined and then the corresponding return, $\mu = \mathbf{a}^T\mathbf{m}$, and risk, $\sigma^2 = \mathbf{a}^T R\mathbf{a}$, are calculated. The resulting plot of the pairs (μ, σ) is the efficient frontier in this case.

Given the parameters $\gamma, \mathbf{m}, R$, incorporate the constraint, $\mathbf{a}^T\mathbf{u} = 1$, with a Lagrange multiplier and work with the modified performance index:

$$J = \frac{1}{2}\mathbf{a}^T R\mathbf{a} - \gamma\mathbf{a}^T\mathbf{m} + \lambda_2(1 - \mathbf{u}^T\mathbf{a}) = \min$$

Show that one obtains exactly the same solution for $\mathbf{a}$ as in the previous problem and that the correspondence between the assumed γ and the realized return μ is given by

$$\mu = \mu_0 + \frac{D}{C}\gamma$$

3. ***Portfolio with inequality constraints.*** If short sales are not allowed, the portfolio weights a_i must be restricted to be in the interval, $0 \le a_i \le 1$. In this case, the following optimization problem must be solved:

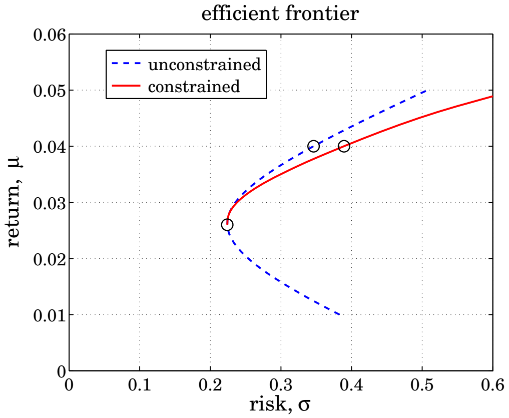
$$\sigma^2 = \mathbf{a}^T R\mathbf{a} = \min\,, \quad \text{subject to} \quad \begin{cases} \mathbf{a}^T\mathbf{m} = \mu \\ \mathbf{a}^T\mathbf{u} = 1 \\ 0 \le a_i \le 1\,, \quad i = 1, 2, \dots, M \end{cases} \tag{25.14.10}$$

This is a convex optimization problem that can be solved conveniently using the CVX package.[†] For example, given a desired value for μ, the following CVX code will solve for $\mathbf{a}$:

```
% define M, m, R, mu
cvx_begin
   variable a(M)
   minimize( a'*R*a );
   subject to
      a'*u == 1;
      a'*m == mu;
      a <= ones(M,1);
     -a <= zeros(M,1);
cvx_end
```

(a) For the numerical example of Question (1.j), choose 51 equally-spaced μs in the interval $[\min(\mathbf{m}), \max(\mathbf{m})]$ and calculate the corresponding $\mathbf{a}$ and σs, and plot the efficient frontier, that is the pairs (μ, σ). Superimpose on this graph the efficient frontier of the unconstrained case from Question 1.

(b) Compute the risk σ and weight vector $\mathbf{a}$ that would achieve a return of $\mu = 0.04$ and indicate that point on the efficient frontier of part (a). Place on the graph also the unconstrained solution for the same μ, and explain why the inequality-constrained case is slightly worse.

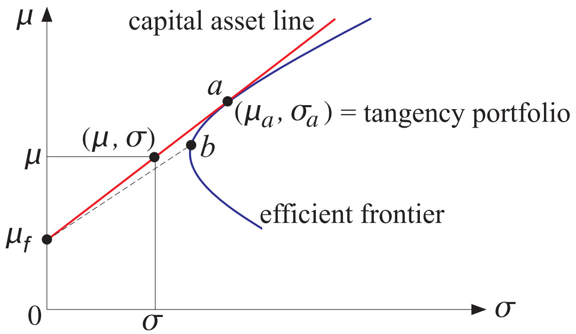


4. ***Capital Asset Line.*** A variation of the mean-variance portfolio theory was considered by W. F. Sharpe, who in addition to the collection of risky assets, allowed the presence of a *risk-free* asset, such as a Treasury T-bill, that has a fixed guaranteed return, say $\mu = \mu_f$, and no risk, $\sigma_f = 0$.

Let us assume again that we also have M risky assets $\mathbf{y} = [y_1, y_2, \dots, y_M]^T$ with expected returns $E[\mathbf{y}] = \mathbf{m} = [m_1, m_2, \dots, m_M]^T$, and known covariance matrix

[†] `http://cvxr.com/cvx`

$R = E\big[(\mathbf{y}-\mathbf{m})(\mathbf{y}-\mathbf{m})^T\big]$. Clearly, we must assume that $\mu_f < m_i$, $i = 1, 2, \dots, M$, otherwise, we would put all our money into the risk-free asset. We form an optimum portfolio of risky assets $y = \mathbf{a}^T\mathbf{y}$ by choosing a point on the efficient frontier, for example, the indicated point b on the figure below.



Then, we join with a straight line the point b to the risk-free point μ_f (the dashed line on the figure), and we allocate a fraction, w_f, of our total funds to the risk-free asset and hence a fraction, $(1-w_f)$, to the portfolio y, that is, the combined portfolio will be:

$$y_{\text{tot}} = w_f\mu_f + (1-w_f)y = w_f\mu_f + (1-w_f)\mathbf{a}^T\mathbf{y}$$

with mean and variance:

$$\begin{aligned} \mu &= E[y_{\text{tot}}] = w_f\mu_f + (1-w_f)\mathbf{a}^T\mathbf{m} \\ \sigma^2 &= E\big[(y_{\text{tot}}-\mu)^2\big] = (1-w_f)^2\,\mathbf{a}^T R\mathbf{a} \end{aligned} \tag{25.14.11}$$

The lowest location for b is at the apex of the frontier, for which we have $\mathbf{a} = \mathbf{a}_0$. It should be evident from the figure that if we move the point b upwards along the efficient frontier, increasing the slope of the dashed straight line, we would obtain a better portfolio having larger μ.

We may increase the slope until the dashed line becomes tangent to the efficient frontier, say at the point a, which would correspond to an optimum portfolio $\mathbf{a}$ with mean and variance μ_a, σ_a.

This portfolio is referred to as the *tangency portfolio* and the tangent line (red line) is referred to as the *capital asset line* and its maximum slope, say β, as the *Sharpe ratio*. Eqs. (25.14.11) become now:

$$\begin{aligned} \mu &= w_f\mu_f + (1-w_f)\mu_a \\ \sigma &= (1-w_f)\sigma_a \end{aligned} \tag{25.14.12}$$

where $\mu_a = \mathbf{a}^T\mathbf{m}$ and $\sigma_a^2 = \mathbf{a}^T R\mathbf{a}$. Solving the second of Eq. (25.14.12) for w_f, we find $1 - w_f = \sigma/\sigma_a$, and substituting in the first, we obtain the equation for the straight line on the (μ, σ) plane:

$$\mu = \mu_f + \beta\sigma\,, \quad \beta = \frac{\mu_a - \mu_f}{\sigma_a} = \text{slope, Sharpe ratio} \tag{25.14.13}$$

This line is the *efficient frontier* for the combined portfolio. Next we impose the condition that the point a be a tangency point on the efficient frontier. This will fix μ_a, σ_a. We saw that the frontier is characterized by the parabolic curve of Eq. (25.14.6) with the optimum weight vector given by (25.14.8). Applying these to the pair (μ_a, σ_a), we have:

$$\begin{aligned} \sigma_a^2 &= \sigma_0^2 + \frac{C}{D}(\mu_a - \mu_0)^2 \\ \mathbf{a} &= \mathbf{a}_0 + \frac{C}{D}(\mu_a - \mu_0)\,R^{-1}(\mathbf{m} - \mu_0\mathbf{u}) \end{aligned} \tag{25.14.14}$$

The slope of the tangent at the point a is given by the derivative $d\mu_a/d\sigma_a$, and it must agree with the slope β of the line (25.14.13):

$$\frac{d\mu_a}{d\sigma_a} = \beta = \frac{\mu_a - \mu_f}{\sigma_a} \tag{25.14.15}$$

(a) Using condition (25.14.15) and Eq. (25.14.14), show the following relationships:

$$\begin{aligned} &\frac{C}{D}(\mu_a - \mu_0)(\mu_0 - \mu_f) = \frac{1}{C} \\ &\sigma_a^2 = \frac{C}{D}(\mu_a - \mu_0)(\mu_a - \mu_f) \\ &\beta = \sigma_a C(\mu_0 - \mu_f) \end{aligned} \tag{25.14.16}$$

The first can be solved for μ_a, and show that it can be expressed as:

$$\mu_a = \frac{A - \mu_f B}{C(\mu_0 - \mu_f)} \tag{25.14.17}$$

(b) Using Eqs. (25.14.14) and (25.14.16), show that the optimum weights are given by

$$\mathbf{a} = \frac{1}{C(\mu_0 - \mu_f)}\,R^{-1}(\mathbf{m} - \mu_f\mathbf{u}) \tag{25.14.18}$$

and verify that they satisfy the constraints $\mathbf{a}^T\mathbf{m} = \mu_a$ and $\mathbf{a}^T\mathbf{u} = 1$.

(c) Show that the slope β can also be expressed by:

$$\beta^2 = \left(\frac{\mu_a - \mu_f}{\sigma_a}\right)^2 = (\mathbf{m} - \mu_f\mathbf{u})^T R^{-1}(\mathbf{m} - \mu_f\mathbf{u}) \tag{25.14.19}$$

(d) Define $\mathbf{w} = (1 - w_f)\mathbf{a}$ to be the effective weight for the total portfolio:

$$y_{\text{tot}} = w_f\mu_f + (1-w_f)\mathbf{a}^T\mathbf{y} = w_f\mu_f + \mathbf{w}^T\mathbf{y} \tag{25.14.20}$$

Show that $\mathbf{w}$ is given in terms of the return $\mu = w_f\mu_f + (1-w_f)\mu_a$ as follows:

$$\mathbf{w} = (1-w_f)\mathbf{a} = \frac{\mu - \mu_f}{\beta^2}\,R^{-1}(\mathbf{m} - \mu_f\mathbf{u}) \tag{25.14.21}$$

(e) The optimality of the capital asset line and the tangency portfolio can also be derived directly by considering the following optimization problem. Let w_f and $\mathbf{w}$ be the weights to be assigned to the risk-free asset μ_f and the risky assets $\mathbf{y}$. Then the total portfolio, its mean μ, and variance σ^2 are given by:

$$\begin{aligned} y_{\text{tot}} &= w_f\mu_f + \mathbf{w}^T\mathbf{y} \\ \mu &= w_f\mu_f + \mathbf{w}^T\mathbf{m} \\ \sigma^2 &= \mathbf{w}^T R\mathbf{w} \end{aligned} \tag{25.14.22}$$

where the weights must add up to unity: $w_f + \mathbf{w}^T\mathbf{u} = 1$. Given μ, we wish to determine the weights $w_f, \mathbf{w}$ to minimize σ^2. Incorporating the constraints by two Lagrange multipliers, we obtain the performance index:

$$J = \frac{1}{2}\mathbf{w}^T R\mathbf{w} + \lambda_1(\mu - w_f\mu_f - \mathbf{w}^T\mathbf{m}) + \lambda_2(1 - w_f - \mathbf{w}^T\mathbf{u}) = \min$$

Show that the minimization of J with respect to $w_f, \mathbf{w}$ results in:

$$\mathbf{w} = \lambda_1 R^{-1}(\mathbf{m} - \mu_f\mathbf{u})\,, \quad \lambda_2 = -\lambda_1\mu_f$$

(f) Imposing the constraints show that,

$$\mathbf{w}^T(\mathbf{m} - \mu_f\mathbf{u}) = \mu - \mu_f\,, \quad \lambda_1 = \frac{\mu - \mu_f}{\beta^2}$$

where β^2 is given as in Eq. (25.14.19),

$$\beta^2 = (\mathbf{m} - \mu_f\mathbf{u})^T R^{-1}(\mathbf{m} - \mu_f\mathbf{u})$$

and hence, show that $\mathbf{w}$ is given by Eq. (25.14.21)

$$\mathbf{w} = \frac{\mu - \mu_f}{\beta^2} R^{-1}(\mathbf{m} - \mu_f\mathbf{u})$$

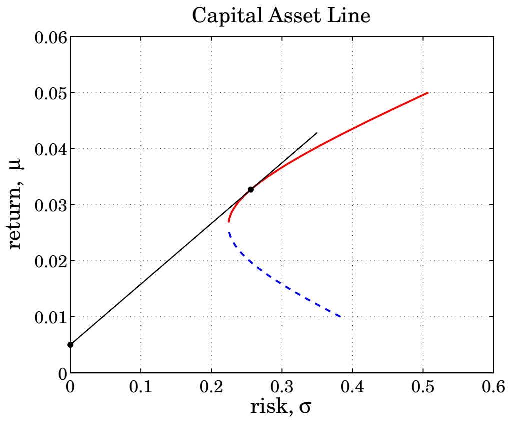
(g) Show that $\sigma^2 = \mathbf{w}^T R\mathbf{w}$ is given by,

$$\sigma^2 = \left(\frac{\mu - \mu_f}{\beta}\right)^2$$

which corresponds to the straight-line frontier on the μ, σ plane:

$$\mu = \mu_f + \beta\sigma$$

(h) For the numerical example of Question (1.j) and for a fixed-asset return of $\mu_f = 0.005$, connect the points (μ_a, σ_a) and $(\mu_f, 0)$ by a straight line and plot it together with the parabolic efficient frontier of the risky assets, and verify the tangency of the line.



5. ***Security market line and CAPM.*** When the collection of risky stocks of the previous problem are taken to be representative of the market, such as for example, the stocks in the S&P 500 index, the tangency portfolio at point a is referred to as the *market portfolio*, and the capital asset line, as the *capital market line*.

Let $y_a = \mathbf{a}^T\mathbf{y}$ be the linear combination of stocks for the market portfolio, and consider another stock y_i with return and risk μ_i, σ_i. The stock y_i is arbitrary and not necessarily belonging to those that make up the representative market. Define the *beta* for the stock as the ratio of the following covariances relative to the market portfolio:

$$\beta_i = \frac{R_{ia}}{R_{aa}}$$

where $R_{ia} = E[(y_i - \mu_i)(y_a - \mu_a)]$ and $R_{aa} = \sigma_a^2 = E[(y_a - \mu_a)^2]$.

Let us now make a portfolio y consisting of a percentage w of the stock y_i and a percentage $(1 - w)$ of the market y_a. Then, y and its mean and variance will be given as follows, where $R_{ii} = \sigma_i^2$.

$$\begin{aligned} y &= wy_i + (1 - w)y_a \\ \mu &= w\mu_i + (1 - w)\mu_a \\ \sigma^2 &= w^2R_{ii} + 2w(1 - w)R_{ia} + (1 - w)^2R_{aa} \end{aligned} \tag{25.14.23}$$

As w varies, the pair (μ, σ) varies over its own parabolic efficient frontier.

The *capital asset pricing model* (CAPM) asserts that the tangent line at the market point $y = y_a$ on this frontier obtained when $w = 0$, coincides with the capital market line between the risk-free asset μ_f and the market portfolio. This establishes the following relationship to be proved below:

$$\mu_i - \mu_f = \beta_i(\mu_a - \mu_f) \tag{25.14.24}$$

so that the excess return above μ_f, called the *risk premium*, is proportional to the stock's beta. The straight line of μ_i vs. β_i is referred to as the *security market line*.

(a) Using the differentiation rule,

$$\frac{d\sigma}{d\mu} = \frac{d\sigma}{dw}\cdot\frac{dw}{d\mu}$$

show that the slope at the market point, i.e., at $w = 0$, is given by

$$\frac{d\mu}{d\sigma}\bigg|_{w=0} = \frac{\mu_i - \mu_a}{(\beta_i - 1)\,\sigma_a} \tag{25.14.25}$$

(b) Then, derive Eq. (25.14.24) by equating (25.14.25) to the slope (25.14.13).

6. ***Minimum-variance with multiple constraints.*** A generalization of the portfolio constrained minimization problem involves more that two constraints:

$$J = \frac{1}{2}\,\mathbf{a}^T R\,\mathbf{a} - \mathbf{b}^T\mathbf{a} = \min$$

subject to K linear constraints:

$$\mathbf{c}_i^T\mathbf{a} = \mu_i\,,\quad i = 1,2,\dots,K$$

where R is an $M\times M$ positive definite symmetric matrix, $\mathbf{b}$ is a given $M\times 1$ vector, and we assume that $K < M$ and that the $M\times 1$ vectors $\mathbf{c}_i$ are linearly independent. Defining the $M\times K$ matrix of constraints C and the K-dimensional vector of "returns" $\boldsymbol{\mu}$,

$$C = [\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_K]\,,\quad \boldsymbol{\mu} = \begin{bmatrix}\mu_1\\ \mu_2\\ \vdots\\ \mu_K\end{bmatrix}$$

the above minimization problem can be cast compactly in the form:

$$J = \frac{1}{2}\mathbf{a}^T R\mathbf{a} - \mathbf{b}^T\mathbf{a} = \min\,,\quad \text{subject to}\quad C^T\mathbf{a} = \boldsymbol{\mu}$$

(a) Introduce a K-dimensional vector of Lagrange multipliers $\boldsymbol{\lambda}$ and replace the performance index by:

$$J = \frac{1}{2}\mathbf{a}^T R\mathbf{a} - \mathbf{b}^T\mathbf{a} + \boldsymbol{\lambda}^T(\boldsymbol{\mu} - C^T\mathbf{a}) = \min$$

Show that the quantities $\mathbf{a}, \boldsymbol{\lambda}$ may be obtained as the solution of the $(M{+}K)\times(M{+}K)$ linear system of equations:

$$\begin{bmatrix} R & -C\\ C^T & \mathbf{0}\end{bmatrix}\begin{bmatrix}\mathbf{a}\\ \boldsymbol{\lambda}\end{bmatrix} = \begin{bmatrix}\mathbf{b}\\ \boldsymbol{\mu}\end{bmatrix}$$

(b) Show that the above matrix has the following inverse:

$$\left[\begin{array}{c|c} R^{-1} - R^{-1}C(C^TR^{-1}C)^{-1}C^TR^{-1} & R^{-1}C(C^TR^{-1}C)^{-1}\\ \hline -(C^TR^{-1}C)^{-1}C^TR^{-1} & (C^TR^{-1}C)^{-1}\end{array}\right]$$

and explain why the assumed full rank of C guarantees the existence of the matrix inverse $(C^TR^{-1}C)^{-1}$.

(c) Show that the solution for $\mathbf{a}$ and $\boldsymbol{\lambda}$ can be obtained by:

$$\begin{aligned}\boldsymbol{\lambda} &= (C^TR^{-1}C)^{-1}[\boldsymbol{\mu} - C^TR^{-1}\mathbf{b}]\\ \mathbf{a} &= R^{-1}[\mathbf{b} + C\boldsymbol{\lambda}]\end{aligned}$$

(d) Show that the "variance" $\sigma^2 = \mathbf{a}^T R\mathbf{a}$ is parabolic in the "returns", like Eq. (25.14.4), thus defining a generalized "efficient frontier":

$$\sigma^2 = \sigma_0^2 + \boldsymbol{\mu}^T(C^TR^{-1}C)^{-1}\boldsymbol{\mu}$$

where the constant σ_0^2 is defined by:

$$\sigma_0^2 = \mathbf{b}^T[R^{-1} - R^{-1}C(C^TR^{-1}C)^{-1}C^TR^{-1}]\mathbf{b}$$

Note that if additional inequality constraints are included, such as for example $a_i > 0$ for the weights, then this becomes a much harder problem that must be solved with *quadratic programming* techniques. The CVX package or MATLAB's function `quadprog` from the optimization toolbox solves such problems. The antenna or sensor array version of this problem is know as linearly-constrained minimum-variance (LCMV) beamforming [45].

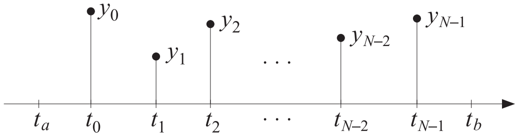
26

Whittaker-Henderson Smoothing

26.1 Smoothing Splines

We recall from Sec. 23.12 that the minimum-R_s filters had the property of maximizing the smoothness of the filtered output signal by minimizing the mean-square value of the s-differenced output, that is, the quantity $E\big[(\nabla^s\hat{x}_n)^2\big]$ in the notation of Eq. (23.12.11). Because of their finite span, minimum-R_s filters belong to the class of *local* smoothing methods. Smoothing splines are *global* methods in the sense that their design criterion involves the entire data signal to be smoothed, but their objective is similar, that is, to maximize smoothness.

Splines assume an observation model of the form, $y(t)= x(t)+v(t)$, where $x(t)$ is a smooth trend to be estimated on the basis of N noisy observations, $y_n = y(t_n)$, measured at N time instants t_n, for $n = 0, 1, \dots, N-1$, as shown below.



The times t_n, called the *knots*, are not necessarily equally-spaced, but are in increasing order and are assumed to lie within a slightly larger interval $[t_a, t_b]$, that is,

$$t_a < t_0 < t_1 < t_2 < \cdots < t_{N-1} < t_b$$

A smoothing spline fits a continuous function $x(t)$, taken to be the estimate of the underlying smooth trend, by solving the optimization problem:

$$\mathcal{J} = \sum_{n=0}^{N-1} w_n\big(y_n - x(t_n)\big)^2 + \lambda\int_{t_a}^{t_b}\big[x^{(s)}(t)\big]^2 dt = \min \tag{26.1.1}$$

where $x^{(s)}(t)$ denotes the s-th derivative of $x(t)$, λ is a positive "smoothing parameter," and w_n are given non-negative weights.

The performance index strikes a balance between interpolation and smoothing. The first term attempts to interpolate the data by $x(t)$, while the second attempts to minimize the roughness or maximize the smoothness of $x(t)$. The balance between the

two terms is controlled by the parameter λ; larger λ increases smoothing, smaller λ interpolates the data more closely.

Schoenberg [959] has shown that the solution to the problem (26.1.1) is a so-called *natural smoothing spline* of polynomial order $2s-1$, that is, $x(t)$ has $2s-2$ continuous derivatives, it is a polynomial of degree $2s-1$ within each subinterval (t_n, t_{n+1}), for $n = 0, 1, \dots, N-2$, and it is a polynomial of order $s-1$ within the end subintervals $[t_a, t_0)$ and $(t_{N-1}, t_b]$. See Ref. [45] for a detailed discussion of splines, especially cubic splines corresponding to the case $s = 2$, including several computational examples.

For discrete-time sampled data, the problem was originally posed and solved for special cases of s by Thiele, Bohlmann, Whittaker, and Henderson [1007–1014], and is referred to as *Whittaker-Henderson* smoothing — we consider it in this chapter.

Besides their extensive use in drafting and computer graphics, splines have many other applications. A large online bibliography can be found in [952]. A small subset of references on interpolating and smoothing splines and their applications is [953–1006].

26.2 Whittaker-Henderson Smoothing Methods

In this chapter,[†] we discuss Whittaker-Henderson smoothing methods and some generalizations. Whittaker-Henderson smoothing is a discrete-time version of spline smoothing for equally spaced data. Some of the original papers by Thiele, Bohlmann, Whittaker, Henderson and others,[‡] and their applications to trend extraction in the actuarial sciences, physical sciences, and business and finance, are given in [1007–1040], including Hodrick-Prescott filters in finance [1041–1069], and recent sparse realizations in terms of the L_1 norm [1070–1080], as well as extensions to seasonal data [1214–1217,1228,1230].

The Whittaker-Henderson smoothing method, based on L_2 regularization, is one of the most effective methods of trend extraction, trying to balance fidelity to the observations, yet resulting in a smooth trend having a prescribed degree of smoothness. It has been termed the "perfect smoother" [1030].

It is defined by the minimization of the following performance index, where y_n, $n = 0, 1, \dots, N-1$, are the observations, and x_n the resulting smoothed estimate,

$$\boxed{\mathcal{J} = \sum_{n=0}^{N-1} w_n\big|y_n - x_n\big|^2 + \lambda\sum_{n=s}^{N-1}\big|\nabla^s x_n\big|^2 = \min} \tag{26.2.1}$$

where $\nabla^s x_n$ represents the backward-difference operator $\nabla x_n = x_n - x_{n-1}$ applied s times. We encountered this operation in Sec. 23.12 on minimum-R_s Henderson filters. The corresponding difference filter and its impulse response are

$$\begin{aligned} D_s(z) &= (1 - z^{-1})^s \\ d_s(k) &= (-1)^k\binom{s}{k}, \quad 0 \le k \le s \end{aligned} \tag{26.2.2}$$

[†] adapted from the author's book on *Applied Optimum Signal Processing* [45]

[‡] Bohlmann considered the case $s = 1$, Whittaker and Henderson, $s = 3$, and Hodrick-Prescott, $s = 2$.

For example, we have for $s = 1, 2, 3$,

$$\mathbf{d}_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \quad \mathbf{d}_2 = \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}, \quad \mathbf{d}_3 = \begin{bmatrix} 1 \\ -3 \\ 3 \\ -1 \end{bmatrix}$$

Because $D_s(z)$ is a highpass filter, the performance index attempts, in its second term, to minimize the spectral energy of x_n at the high frequency end, while attempting to interpolate the noisy observations with the first term. The result is a lowpass, smoothing, operation. In fact, the filter $D_s(z)$ may be replaced by any other (causal) FIR highpass filter $D(z)$, or d_n in the time domain, with a similar result. Thus, a more general version of (26.2.1) would be:

$$\mathcal{J} = \sum_{n=0}^{N-1} w_n \big|y_n - x_n\big|^2 + \lambda \sum_{n=s}^{N-1} \big|d_n * x_n\big|^2 = \min \tag{26.2.3}$$

where $d_n * x_n$ denotes convolution and s is the filter order, that is, we assume that the impulse response is $d_n = [d_0, d_1, \dots, d_s]$. The criteria (26.2.1) and (26.2.3) are examples of the method of *regularization* for ill-conditioned linear systems.

The summation limits of the second terms in Eqs. (26.2.1) and (26.2.3) restrict the convolutional operations to their steady-state range. For example, for a length-N causal input $\{x_0, x_1, \dots, x_{N-1}\}$, the s-difference filter has the full convolutional output:

$$g_n = \nabla^s x_n = \sum_{k=\max(0,n-N+1)}^{\min(s,n)} d_s(k) x_{n-k}\,, \quad 0 \le n \le N-1+s \tag{26.2.4}$$

and the steady-state output (assuming $N > s$):

$$g_n = \nabla^s x_n = \sum_{k=0}^{s} d_s(k) x_{n-k}\,, \quad s \le n \le N-1 \tag{26.2.5}$$

Similarly, we have in the more general case,

$$\begin{aligned} g_n &= d_n * x_n = \sum_{k=\max(0,n-N+1)}^{\min(s,n)} d_k x_{n-k}\,, \quad 0 \le n \le N-1+s \\ g_n &= d_n * x_n = \sum_{k=0}^{s} d_k x_{n-k}\,, \quad s \le n \le N-1 \end{aligned} \tag{26.2.6}$$

In Sec. 23.12 we worked with the full convolutional form (26.2.4) and implemented it in a matrix form using the convolution matrix. We recall that the MATLAB functions `binom` and `diffmat` can be used to compute the impulse response $d_s(k)$ and the corresponding $(N+s)\times N$ full convolutional matrix D_s.

The filtering operation $g_n = \nabla^s x_n$, $0 \le n \le N-1+s$, can be expressed vectorially as $\mathbf{g} = D_s \mathbf{x}$, where $\mathbf{x}$ is the N-dimensional input vector $\mathbf{x} = [x_0, x_1, \dots, x_{N-1}]^T$, and $\mathbf{g} = [g_0, g_1, \dots, g_{N-1+s}]^T$, the $(N+s)$-dimensional output vector. Similarly, the operation $g_n = d_n * x_n$ can be expressed as $\mathbf{g} = D_{\text{full}}\mathbf{x}$, where the $(N+s)\times N$ full convolution matrix can be constructed using `convmat`—the sparse version of `convmtx`,

```
Dfull = convmat(d,N);     % sparse full convolution matrix
```

where $\mathbf{d} = [d_0, d_1, \dots, d_s]^T$. The steady-state versions of the full convolution matrices are obtained by extracting their middle $N-s$ rows, and therefore, they have dimension $(N-s)\times N$. For example, we have for $N = 5$ and $s = 2$, with $\mathbf{d} = [d_0, d_1, d_2]^T$,

$$D_{\text{full}} = \left[\begin{array}{ccccc} d_0 & 0 & 0 & 0 & 0 \\ d_1 & d_0 & 0 & 0 & 0 \\ \hline d_2 & d_1 & d_0 & 0 & 0 \\ 0 & d_2 & d_1 & d_0 & 0 \\ 0 & 0 & d_2 & d_1 & d_0 \\ \hline 0 & 0 & 0 & d_2 & d_1 \\ 0 & 0 & 0 & 0 & d_2 \end{array}\right] \quad \Rightarrow \quad D = \begin{bmatrix} d_2 & d_1 & d_0 & 0 & 0 \\ 0 & d_2 & d_1 & d_0 & 0 \\ 0 & 0 & d_2 & d_1 & d_0 \end{bmatrix} = \begin{bmatrix} d_2 & 0 & 0 \\ d_1 & d_2 & 0 \\ d_0 & d_1 & d_2 \\ 0 & d_0 & d_1 \\ 0 & 0 & d_0 \end{bmatrix}^T$$

The last expression shows that the steady matrix can also be viewed as the transposed of the convolution matrix of the reversed filter with $N-s$ columns. Thus, in MATLAB two possible ways of constructing D are:

$$\boxed{\begin{array}{ll} 1) & D_{\text{full}} = \texttt{convmat}(d, N); \quad D = D_{\text{full}}(s+1:N, :\,) \\ 2) & D = \texttt{convmat}(\texttt{flip}(d), N-s)'; \end{array}} \tag{26.2.7}$$

For the s-difference filter, we can use the equivalent (sparse) constructions:

$$\boxed{\begin{array}{ll} 1) & D_{\text{full}} = \texttt{diffmat}(s, N); \quad D = D_{\text{full}}(s+1:N, :\,) \\ 2) & D = (-1)^s * \texttt{diffmat}(s, N-s)'; \\ 3) & D = \texttt{diff}(\texttt{speye}(N), s); \end{array}} \tag{26.2.8}$$

where the second method is valid because the reversed binomial filter is $(-1)^s$ times the unreversed one. The third method is the fastest [1030], but does not generalize to an arbitrary filter $\mathbf{d}$. As an example, we have for $N = 7$, $s = 2$, and $\mathbf{d} = [1, -2, 1]^T$:

$$D = \begin{bmatrix} 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 & 1 \end{bmatrix}$$

The corresponding steady-state output vector $\mathbf{g} = [g_s, g_{s+1}, \dots, g_{N-1}]^T$ is given by $\mathbf{g} = D\mathbf{x}$, with squared norm,

$$\sum_{n=s}^{N-1} [d_n * x_n]^2 = \sum_{n=s}^{N-1} g_n^2 = \mathbf{g}^T\mathbf{g} = \mathbf{x}^T(D^T D)\mathbf{x}$$

Therefore, the performance index (26.2.1) or (26.2.3) can be written compactly as:

$$\mathcal{J} = (\mathbf{y} - \mathbf{x})^T W (\mathbf{y} - \mathbf{x}) + \lambda \mathbf{x}^T (D^T D)\mathbf{x} = \min \tag{26.2.9}$$

where W is the diagonal matrix of the weights, $W = \text{diag}([w_0, w_1, \dots, w_{N-1}])$. The optimum solution is obtained by setting the gradient with respect to $\mathbf{x}$ to zero,

$$\frac{\partial \mathcal{J}}{\partial \mathbf{x}} = -2W(\mathbf{y}-\mathbf{x}) + 2\lambda(D^T D)\mathbf{x} = 0 \quad \Rightarrow \quad (W + \lambda D^T D)\mathbf{x} = W\mathbf{y}$$

with solution, which may be regarded as the estimate of $\mathbf{x}$ in the signal model $\mathbf{y} = \mathbf{x} + \mathbf{v}$,

$$\boxed{\hat{\mathbf{x}} = (W + \lambda D^T D)^{-1} W\mathbf{y}} \tag{26.2.10}$$

The matrix D plays the same role as the matrix Q^T in the spline smoothing case, but for equally-spaced data. As was the case in Sec. 23.12, the matrix $D^T D$ is essentially equivalent to the $(2s)$-differencing operator ∇^{2s}, after ignoring the first s and last s rows. For example, we have for $N = 7$ and $s = 2$,

$$D^T D = \left[\begin{array}{rrrrrrr} 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ -2 & 5 & -4 & 1 & 0 & 0 & 0 \\ \hline 1 & -4 & 6 & -4 & 1 & 0 & 0 \\ 0 & 1 & -4 & 6 & -4 & 1 & 0 \\ 0 & 0 & 1 & -4 & 6 & -4 & 1 \\ \hline 0 & 0 & 0 & 1 & -4 & 5 & -2 \\ 0 & 0 & 0 & 0 & 1 & -2 & 1 \end{array}\right]$$

where we recognize the expansion coefficients $(1-z^{-1})^4 = 1 - 4z^{-1} + 6z^{-2} - 4z^{-3} + z^{-4}$.

The $N \times N$ matrix $(W + \lambda D^T D)$ is sparse and banded with bandwidth $2s + 1$, and therefore, MATLAB solves Eq. (26.2.10) very efficiently by default (as long as it is implemented by the backslash operator). The function `whsm` implements Eq. (26.2.10):

```
x = whsm(y,lambda,s,w);        % Whittaker-Henderson smoothing
```

where method (2) is used internally to compute D, and `w` is the vector of weights, which defaults to unity. The function `whgen` is the generalized version that uses an arbitrary highpass filter $\mathbf{d}$, whose steady convolution matrix D is also computed by method (2):

```
x = whgen(y,lambda,d,w);        % generalized Whittaker-Henderson smoothing
```

Denoting the "hat" filtering matrix $H = (W + \lambda D^T D)^{-1} W$, and defining the error $\mathbf{e} = \mathbf{y} - \hat{\mathbf{x}} = (I - H)\mathbf{y}$, we may define a generalized cross-validation criterion for determining the smoothing parameter λ,

$$\text{GCV}(\lambda) = \frac{\mathbf{e}^T W \mathbf{e}}{[\text{tr}(I - H)]^2} = \min \tag{26.2.11}$$

The function `whgcv` calculates it at any vector of λ's and finds the corresponding optimum:

```
[gcv,lopt] = whgcv(y,la,s,w);        % Whittaker-Henderson GCV evaluation
```

The GCV criterion should be used with some caution because it suffers from the same problem, as in the spline case, of typically underestimating the proper value of λ.

The Whittaker-Henderson method was compared to the local polynomial and minimum roughness filters in Examples 23.10.2 and 23.12.1. Some additional examples are discussed below.

Example 26.2.1: *NIST ENSO data.* We apply the Whittaker-Henderson (WH) smoothing method to the ENSO data which are another benchmark example in the NIST Statistical Reference Dataset Archives, and original reference [37]. The data file `ENSO.dat` is available online from the NIST web sites:

```
http://www.itl.nist.gov/div898/strd/nls/nls_main.shtml
http://www.itl.nist.gov/div898/strd/nls/data/enso.shtml
```

The data represent the monthly averaged atmospheric pressure differences between Easter Island and Darwin, Australia. In the nonlinear NIST fit, the data are fitted to three sinusoids of unknown amplitudes and frequencies, except that one of the sinusoids is kept at the annual frequency. There are three significant cycles at 12, 26, and 44 months.

The upper-left graph of Fig. 26.2.1 compares the NIST fit with the Whittaker-Henderson method. We used $s = 3$ and smoothing parameter $\lambda_{\text{opt}} = 6.6$. which was determined by the GCV function `whgcv`.

The lower-left graph shows the corresponding periodogram spectra plotted versus period in units of months/cycle. The three dominant peaks are evident. In the spectrum graphs, the digital frequency is $\omega = 2\pi f$ rads/month, with f measured in cycles/month, and with the corresponding period $p = 1/f$ measured in months/cycle.

The time-domain WH signal agrees fairly well with the NIST fit. We note that in the places where the two disagree, the WH fit appears to be a better representation of the noisy data.

The upper-right graph shows the application of the SVD enhancement method, which typically works well for sinusoids in noise. The embedding dimension was $M = 20$ and the assumed rank $r = 6$ (three real sinusoids are equivalent to six complex ones.) The lower-right graph shows the corresponding spectral peaks. The following MATLAB code illustrates the generation of the four graphs:

```
Y = loadfile('ENSO.dat');                          % data file in AOSP toolbox
y = Y(:,1); t = Y(:,2);                            % extract data signal

b1 = 1.0510749193E+01; b2 = 3.0762128085E+00; b3 = 5.3280138227E-01;
b4 = 4.4311088700E+01; b5 =-1.6231428586E+00; b6 = 5.2554493756E-01;
b7 = 2.6887614440E+01; b8 = 2.1232288488E-01; b9 = 1.4966870418E+00;

yf = b1 + b2*cos(2*pi*t/12) + b3*sin(2*pi*t/12) + b5*cos(2*pi*t/b4) ...
    + b6*sin(2*pi*t/b4) + b8*cos(2*pi*t/b7) + b9*sin(2*pi*t/b7);

s=3; la = linspace(2,10,100);                      % search range for $\lambda$
[gcv,lopt]=whgcv(y,la,s);                          % $\lambda_{\text{opt}} = 6.6$

yw = whsm(y,lopt,s);                               % WH smoothing method
M=20; r=6; ye = svdenh(y,M,r);                     % SVD enhancement method

figure; plot(t,y,'.', t,yw,'-', t,yf,':');         % upper-left graph
figure; plot(t,y,'.', t,ye,'-', t,yf,':');         % upper-right graph

p = linspace(6,54, 481); w = 2*pi./p;              % period in months/cycle
```

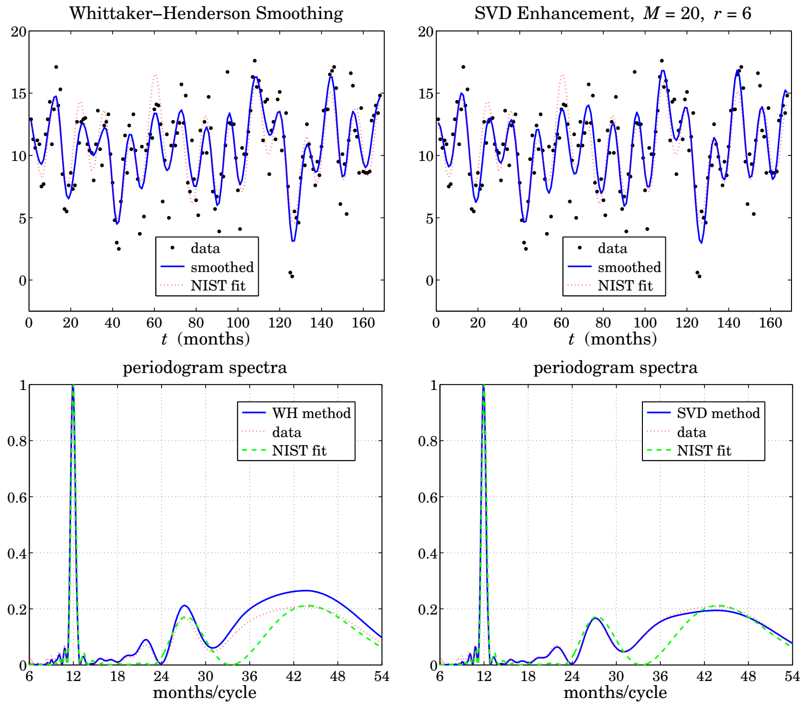


Fig. 26.2.1 Smoothed ENSO signal and spectra.

```
Sy = abs(freqz(zmean(y), 1, w)).^2;     Sy = Sy/max(Sy);        % spectra
Sf = abs(freqz(zmean(yf), 1, w)).^2;    Sf = Sf/max(Sf);
Sw = abs(freqz(zmean(yw), 1, w)).^2;    Sw = Sw/max(Sw);
Se = abs(freqz(zmean(ye), 1, w)).^2;    Se = Se/max(Se);

figure; plot(p,Sw, p,Sy,':', p,Sf,'--');          % lower-left graph
figure; plot(p,Se, p,Sy,':', p,Sf,'--');          % lower-right graph
```

The b_i parameters and the signal `yf` represent the NIST fit. Anticipating the three relevant peaks, the spectra were computed only over the period range $6 \le p \le 54$ months. The function `zmean` removes the mean of the signal so that the spectrum is not masked by the DC component. □

26.3 Regularization Filters

Most of the results of the spline smoothing case carry over to the discrete case. For example, we may obtain an equivalent digital filter by taking the signals to be double-

sided and infinite. Using the Parseval identity, the performance index (26.2.3) becomes:

$$\begin{aligned}\mathcal{J} &= \sum_{n=-\infty}^{\infty}|y_n - x_n|^2 + \lambda\sum_{n=-\infty}^{\infty}|d_n * x_n|^2 = \\ &= \int_{-\pi}^{\pi}\big|Y(\omega)-X(\omega)\big|^2\frac{d\omega}{2\pi} + \lambda\int_{-\pi}^{\pi}\big|D(\omega)X(\omega)\big|^2\frac{d\omega}{2\pi} = \min\end{aligned} \tag{26.3.1}$$

where $D(\omega)$ is the frequency response[†] of the filter d_n, and we assumed unity weights, $w_n = 1$. The vanishing of the functional derivative of $\mathcal{J}$ with respect to $X^*(\omega)$,

$$\frac{\delta\mathcal{J}}{\delta X^*(\omega)} = X(\omega)-Y(\omega)+\lambda|D(\omega)|^2X(\omega) = 0 \tag{26.3.2}$$

gives the effective equivalent smoothing filter $H(\omega) = X(\omega)/Y(\omega)$:

$$H(\omega) = \frac{1}{1+\lambda|D(\omega)|^2} \tag{26.3.3}$$

The corresponding z-domain transfer function is obtained by noting that for real-valued d_n, we have $|D(\omega)|^2 = D(z)D(z^{-1})$, where $z = e^{j\omega}$, so that,

$$H(z) = \frac{1}{1+\lambda D(z)D(z^{-1})} \tag{26.3.4}$$

Such “recursive regularization filters” have been considered in [1042]. In particular, for the s-difference filter $D_s(z) = (1-z^{-1})^s$, we have:

$$\boxed{H(z) = \frac{1}{1+\lambda(1-z^{-1})^s(1-z)^s}} \quad \text{(Whittaker-Henderson filter)} \tag{26.3.5}$$

Similarly, Eq. (26.3.2) can be written in the z-domain and converted back to the time domain. Noting that $D_s(z)D_s(z^{-1}) = (1-z^{-1})^s(1-z)^s = (-1)^s z^s(1-z^{-1})^{2s} = (-1)^s z^s D_{2s}(z)$, we have:

$$\begin{aligned}&X(z)-Y(z)+\lambda(-1)^s z^s(1-z^{-1})^{2s}X(z) = 0\,, \qquad \text{or,}\\ &(-1)^s z^s D_{2s}(z)X(z) = \lambda^{-1}\big(Y(z)-X(z)\big),\end{aligned}$$

resulting in the time-domain $(2s)$-difference equation:

$$(-1)^s\nabla^{2s}x_{n+s} = \lambda^{-1}(y_n - x_n) \tag{26.3.6}$$

In the early years, some ingenious methods were developed for solving this type of equation [1009–1018]. Noting that $|D_s(\omega)| = |1-e^{-j\omega}|^s = 2^s\big|\sin(\omega/2)\big|^s$, we obtain the frequency response of (26.3.5):

$$H(\omega) = \frac{1}{1+\lambda\left|2\sin\dfrac{\omega}{2}\right|^{2s}} \tag{26.3.7}$$

[†] Here, ω is the digital frequency in units of radians per sample.

The complementary highpass filter $H_c(z)= 1 - H(z)$ extracts the error residual component from the observations y_n, that is, $e_n = y_n - x_n$, or in the z-domain, $E(z)= Y(z)-X(z)= Y(z)-H(z)Y(z)= H_c(z)Y(z)$. Its transfer function and frequency response are given by:

$$H_c(z)= \frac{\lambda(1-z^{-1})^s(1-z)^s}{1+\lambda(1-z^{-1})^s(1-z)^s}\,,\quad H_c(\omega)= \frac{\lambda\left|2\sin\dfrac{\omega}{2}\right|^{2s}}{1+\lambda\left|2\sin\dfrac{\omega}{2}\right|^{2s}} \tag{26.3.8}$$

Fig. 26.3.1 shows a plot of $H(\omega)$ and $H_c(\omega)$ for $s = 1,2,3$ and the two values of the smoothing parameter $\lambda = 5$ and $\lambda = 50$. Increasing λ narrows the response of the lowpass filter and widens the response of the highpass one.

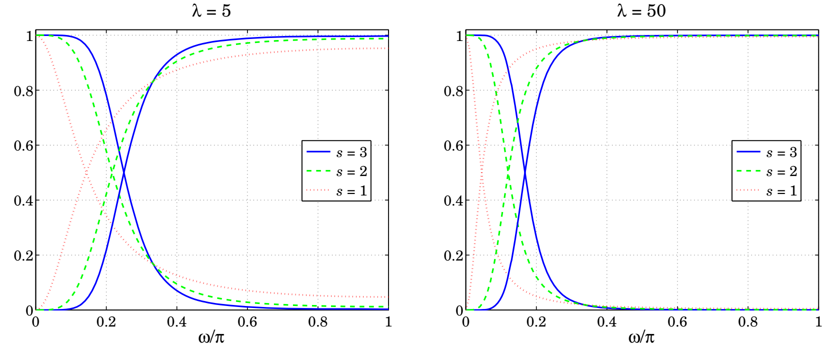


Fig. 26.3.1 Frequency responses of Whittaker-Henderson filters.

26.4 Hodrick-Prescott Filters

In macroeconomic applications such as extracting business cycles from GDP data, the standard signal model $y_n = x_n + v_n$ is interpreted to consist of a long-term trend represented by x_n and a shorter-term cyclical component v_n. The filters $H(z)$ and $H_c(z)$ extract the trend and cyclical components, respectively.

The use of Whittaker-Henderson smoothing with $s = 2$ has been advocated by Hodrick and Prescott [1041] and has become standard in such applications. The Whittaker-Henderson filters are referred to as *Hodrick-Prescott filters* and there is a very large literature on the subject and on the use of other types of bandpass filters for extracting business cycles, a subset of which is [1041–1067].

As is the case in typical filter design, the filter parameter λ can be fixed by specifying a desired value for the filter's cutoff frequency ω_c corresponding to some standardized value of the gain. For the lowpass filter we have for general s, the condition:

$$\frac{1}{1+\lambda\left|2\sin\dfrac{\omega_c}{2}\right|^{2s}} = G_c \tag{26.4.1}$$

where G_c is desired value of the gain. The 3-dB cutoff frequency ω_c corresponds to $G_c = 1/\sqrt{2}$. In macroeconomic applications, the 6-dB frequency is often used, corresponding to the choice $G_c = 1/2$. For the highpass case, measuring the gain G_c relative to that at the Nyquist frequency $\omega = \pi$, we have the condition:

$$\frac{\lambda\left|2\sin\dfrac{\omega_c}{2}\right|^{2s}}{1+\lambda\left|2\sin\dfrac{\omega_c}{2}\right|^{2s}} = G_c\,\frac{2^{2s}\lambda}{1+2^{2s}\lambda} \tag{26.4.2}$$

Typically, *business cycles* are defined [1048] as having frequency components with periods between 6 and 32 quarters (1.5 to 8 years). A bandpass filter with a passband $[\omega_1,\omega_2]= [2\pi/32,\, 2\pi/6]$ radians/quarter would extract such cycles.

The Hodrick-Prescott highpass filter $H_c(\omega)$ must therefore have a cutoff frequency of about $\omega_c = \omega_1 = 2\pi/32$. Hodrick-Prescott advocate the use of $\lambda = 1600$ for quarterly data. Interestingly, the values of $\lambda = 1600$ and $\omega_c = 2\pi/32$ rads/quarter, correspond to almost a 3-dB gain. Indeed, the gain calculated from Eq. (26.4.2) with $s = 2$ turns out to be $G_c = 0.702667 \equiv -3.065$ dB.

Using the same ω_c and G_c, but different values of s requires adjusting the value of λ. For example, solving Eq. (26.4.2) for λ with $s = 1,2,3$ gives:

$$\lambda = 1600\ (s=2)\,,\quad \lambda = 60.654\ (s=1)\,,\quad \lambda = 41640\ (s=3) \tag{26.4.3}$$

Similarly, the value of λ must be adjusted if the sampling frequency is changed. For example, the same cutoff frequency expressed in different units is:

$$\omega_c = \frac{2\pi}{32}\,\frac{\text{radians}}{\text{quarter}} = \frac{2\pi}{8}\,\frac{\text{radians}}{\text{year}} = \frac{2\pi}{96}\,\frac{\text{radians}}{\text{month}} \tag{26.4.4}$$

The above value $G_c = 0.702667$ used in (26.4.2) with $s = 2$ then gives the following values of λ for quarterly, yearly, and monthly sampled data:

$$\lambda = 1600\ \text{(quarterly)}\,,\quad \lambda = 6.677\ \text{(yearly)}\,,\quad \lambda = 128878\ \text{(monthly)} \tag{26.4.5}$$

Similarly, using the slightly more exact value $G_c = 1/\sqrt{2}$ and $s = 2$ gives:

$$\lambda = 1634.5\ \text{(quarterly)}\,,\quad \lambda = 6.822\ \text{(yearly)}\,,\quad \lambda = 131659\ \text{(monthly)} \tag{26.4.6}$$

There is not much agreement as to the values of λ to be used for annual and monthly data. Two other sets of values are as follows, with the first being used by the European Central Bank and the second recommended by [1058,1065],

$$\begin{array}{ll} \lambda = 1600/4^2 = 100\ \text{(yearly)}\,, & \lambda = 1600\times 3^2 = 14400\ \text{(monthly)} \\ \lambda = 1600/4^4 = 6.25\ \text{(yearly)}\,, & \lambda = 1600\times 3^4 = 129600\ \text{(monthly)} \end{array} \tag{26.4.7}$$

The latter choice is essentially the same as that of Eq. (26.4.5) based on the criterion (26.4.2). Indeed, for small ω_c, we may make the approximation $2\sin(\omega_c/2)\approx\ \omega_c$. Since $2^{2s}\lambda$ is typically much larger than unity, the right-hand side of Eq. (26.4.2) can be replaced by G_c, resulting in the following approximate solution, which turns out to be valid up to about $\omega_c \le 0.3\pi$,

$$\frac{\lambda\omega_c^{2s}}{1+\lambda\omega_c^{2s}} = G_c \quad\Rightarrow\quad \lambda = \frac{G_c}{(1-G_c)\,\omega_c^{2s}} \tag{26.4.8}$$

If in this formula, we adjust G_c to get $\lambda = 1600$ at $\omega_c = 2\pi/32$, we find $G_c = 0.70398 \equiv -3.049$ dB, which in turn generates the second set of values in Eq. (26.4.7).

Example 26.4.1: *US GDP for investment.* A protypical example is the application of the Hodrick-Prescott filter to the US GDP. Fig. 26.4.1 shows the real gross domestic product in chained (2000) dollars from private domestic investment, seasonally adjusted at annual rates. The data can be retrieved (as Table 1.1.6) from the BEA web sites:

```
http://www.bea.gov/
http://www.bea.gov/national/nipaweb/Index.asp
```

The signal to be smoothed is the log of the GDP, that is, $y = \log_{10}(\text{GDP})$, and the ordinate units are such that $y = 12$ corresponds to GDP $= 10^{12}$, or, one trillion dollars. The data are quarterly and span the years 1947–2008.

The upper-left graph shows the raw data and the WH-smoothed signal computed with $s = 2$ and $\lambda = 1600$, which as we mentioned above correspond to an approximate 3-dB cutoff frequency of 32 quarters. The upper-right graph shows the residual cyclical component. Its deviations above or below zero indicate the business cycles.

For comparison, the left-bottom graph shows the WH-smoothed signal with $s = 3$ and $\lambda = 41640$ adjusted to match the same 3-dB cutoff frequency as the $s = 2$ case, see Eq. (26.4.3). The lower-right graph shows the smoothed trend from the SVD enhancement method applied with embedding dimension $M = 9$ and rank $r = 1$. The following MATLAB code illustrates the generation of the four graphs:

```
Y = loadfile('USGDP_Inv.dat');                      % data file in AOSP toolbox
y = log10(Y(:,2) * 1e9);                            % Y was in billions
t = taxis(y,4,1947);                                % t-axis in quarters since 1947

s = 2; la = 1600; yt = whsm(y,la,s);                % WH smoothing with s = 2
figure; plot(t,yt,'-', t,y,'--');                   % upper-left graph

yc = y-yt;                                          % cyclical component
figure; plot(t,yc, '-');                            % upper-right graph

s = 3; la = 41640.16; yt = whsm(y,la,s);            % WH smoothing with s = 3
figure; plot(t,yt,'-', t,y,'--');                   % bottom-left graph

M=9; r=1; ye = svdenh(y,M,r);                       % SVD enhancement method
figure; plot(t,ye,'-', t,y,'--');                   % bottom-right graph
```

Except near the end-points, the smoothed trend for $s = 3$ is virtually indistinguishable from the $s = 2$ case, and therefore, it would lead to the same prediction of business cycles. The SVD trend is also very comparable. □

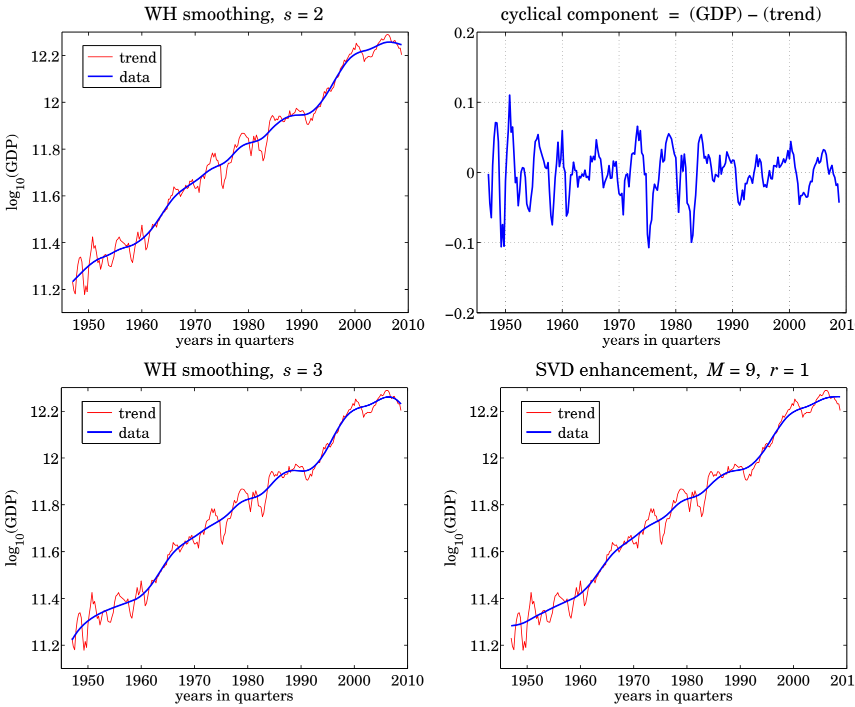


Fig. 26.4.1 U.S. quarterly GDP in private investment, 1947–2008.

26.5 Poles and Impulse Response

Because of the invariance under the substitution $z \to z^{-1}$, the $2s$ poles of the filter $H(z)$ of Eq. (26.3.5) come in two groups with s poles inside the unit circle and their reciprocals outside. It follows that $H(z)$ can be expressed in the factored form:

$$H(z) = \frac{1}{1+\lambda(1-z^{-1})^s(1-z)^s} = \prod_{k=1}^{s}\left[\frac{(1-z_k)^2}{(1-z_k z^{-1})(1-z_k z)}\right] \tag{26.5.1}$$

where z_k, $k = 1, 2, \dots, s$, denote the s poles inside the unit circle. The numerator factors $(1-z_k)^2$ ensure that the right-hand side has unity gain at DC $(z = 1)$, as does the left-hand side. The stable impulse response is double-sided and can be obtained by performing an inverse z-transform with the unit circle as the inversion contour [3]:

$$h_n = \oint_{\text{u.c.}} H(z)\,z^n\,\frac{dz}{2\pi jz}\,, \qquad -\infty < n < \infty \tag{26.5.2}$$

Inserting the factored form (26.5.1) into (26.5.2), we find

$$h_n = \sum_{k=1}^{s} A_k z_k^{|n|}\,, \qquad -\infty < n < \infty \tag{26.5.3}$$

where the coefficients A_k are given by

$$A_k = \left(\frac{1-z_k}{1+z_k}\right)\prod_{\substack{i=1\\ i\neq k}}^{s}\left[\frac{(1-z_i)^2}{(1-z_i z_k^{-1})(1-z_i z_k)}\right], \quad k = 1,2,\dots,s \tag{26.5.4}$$

The poles can be obtained in the form $z_k = e^{j\omega_k}$, where ω_k are the complex frequencies of the denominator, that is, the frequencies that are solutions of the equation:

$$1 + \lambda\left[2\sin\frac{\omega}{2}\right]^{2s} = 0 \tag{26.5.5}$$

where we note that even though ω is complex, we still have $(1-z^{-1})(1-z) = 4\sin^2(\omega/2)$ for $z = e^{j\omega}$. The solution of Eq. (26.5.5) is straightforward. The s frequencies ω_k that lead to poles z_k that are inside the unit circle can be parametrized as follows:

$$\left[2\sin\frac{\omega}{2}\right]^{2s} = \frac{-1}{\lambda} = \frac{e^{j\pi(2k-1)}}{\lambda} \quad\Rightarrow\quad \sin\frac{\omega_k}{2} = \frac{e^{j\pi(2k-1)/(2s)}}{2\lambda^{1/(2s)}}, \quad k = 1,2,\dots,s$$

Thus, the desired set of poles are:

$$z_k = e^{j\omega_k}, \quad \omega_k = 2\arcsin\left[\frac{e^{j\theta_k}}{2\lambda^{1/(2s)}}\right], \quad \theta_k = \frac{\pi(2k-1)}{2s}, \quad k = 1,2,\dots,s \tag{26.5.6}$$

If s is even, then the z_k (and the coefficients A_k) come in conjugate pairs. If s is odd, then the zero at $k = (s+1)/2$ is real and the rest come in conjugate pairs. In either case, h_n given by (26.5.3) is real-valued and decays exponentially from either side of the time axis. The MATLAB function `whimp` calculates h_n at any vector of ns and also produces the poles and residues z_k, A_k, $k = 1,2,\dots,s$,

```
[h,z,A] = whimp(lambda,s,n);      % Whittaker-Henderson impulse response and poles
```

Fig. 26.5.1 shows the impulse responses for $s = 1,2,3$ with λs chosen as in (26.4.3) so that the (complementary) filters have the same 3-dB cutoff frequency. The impulse response of the complementary filter is $h_c(n) = \delta(n) - h(n)$. Therefore, the three responses will have roughly the same time width.

26.6 Regularization and Kernel Machines

Regularization was initially invented as a method for solving ill-posed, inconsistent, overdetermined, and ill-conditioned inverse problems. Recently it has been applied also to support-vector machines and kernel methods for machine learning. There is a huge literature on the subject, a small subset of which is [1085–1125]. Here, we present a short discussion with particular emphasis on deconvolution and kernel regression methods.

Both spline and Whittaker-Henderson smoothing are examples of regularization. The performance index Eq. (26.2.3) can be generalized further to cover the case of deconvolution, or inverse filtering,

$$\mathcal{J} = \sum_n |y_n - f_n * x_n|^2 + \lambda\sum_n |d_n * x_n|^2 = \min \tag{26.6.1}$$

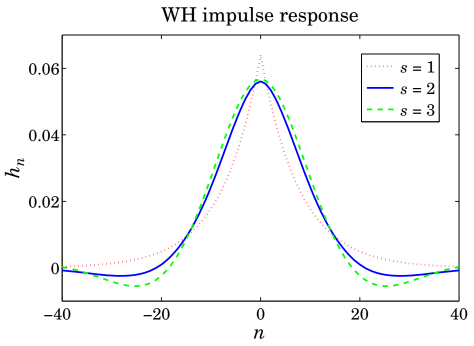


Fig. 26.5.1 Impulse responses of Whittaker-Henderson filters.

where f_n and d_n are FIR filters. This attempts to solve, $y_n = f_n * x_n$, for x_n by deconvolving the effect of f_n. We may write (26.6.1) in a compact matrix form using the convolution matrices F, D of the two filters:

$$\mathcal{J} = \|\mathbf{y} - F\mathbf{x}\|^2 + \lambda\|D\mathbf{x}\|^2 = (\mathbf{y} - F\mathbf{x})^T(\mathbf{y} - F\mathbf{x}) + \lambda\mathbf{x}^T(D^T D)\mathbf{x} = \min \tag{26.6.2}$$

The solution is obtained from the gradient,

$$\frac{\partial\mathcal{J}}{\partial\mathbf{x}} = -2F^T(\mathbf{y} - F\mathbf{x}) + 2\lambda D^T D\mathbf{x} = 0, \quad \text{or,}$$

$$\hat{\mathbf{x}} = (F^T F + \lambda D^T D)^{-1}F^T\mathbf{y} \tag{26.6.3}$$

The problem (26.6.2) is of course much more general than inverse filtering. The method is known as *Tikhonov regularization* and as *ridge regression*. The linear system, $\mathbf{y} = F\mathbf{x}$, may in general be overdetermined, or underdetermined, or rank defective. To simplify the discussion, we assume here that the linear system is either square and invertible or overdetermined but F having full rank. For $\lambda = 0$, we obtain, $\hat{\mathbf{x}} = (F^T F)^{-1}F^T\mathbf{y}$, which is recognized as the unique pseudoinverse least-squares solution. In the square case, we have, $\hat{\mathbf{x}} = F^{-1}\mathbf{y}$. We are envisioning a signal model of the form, $\mathbf{y} = F\mathbf{x} + \mathbf{v}$, and the objective is to determine an estimate of $\mathbf{x}$. We have then,

$$\hat{\mathbf{x}} = F^{-1}\mathbf{y} = F^{-1}(F\mathbf{x} + \mathbf{v}) = \mathbf{x} + F^{-1}\mathbf{v} \tag{26.6.4}$$

A potential problem with this estimate is that if F is ill-conditioned with a large condition number—a common occurrence in practice—the resulting inverse-filtered noise component $\mathbf{u} = F^{-1}\mathbf{v}$ may be magnified to such an extent that it will mask the desired term $\mathbf{x}$, rendering the estimate $\hat{\mathbf{x}}$ useless. The same can happen in the overdetermined case. The presence of the regularization term helps in this regard by providing a more well-conditioned inverse. For the deconvolution problem, one typically selects D to be the unit matrix, $D = I$, leading to the solution,

$$\hat{\mathbf{x}} = (F^T F + \lambda I)^{-1}F^T\mathbf{y} \tag{26.6.5}$$

To see how regularization improves the condition number, let $\lambda_{\max}, \lambda_{\min}$ be the maximum and minimum eigenvalues of F^TF. Then, the condition numbers of F^TF and $F^TF + \lambda I$ are $\lambda_{\max}/\lambda_{\min}$ and $(\lambda_{\max} + \lambda)/(\lambda_{\min} + \lambda)$. A highly ill-conditioned problem would have $\lambda_{\min} \ll \lambda_{\max}$. It is straightforward to verify that the larger the λ, the more the condition number of the regularized matrix is reduced:

$$\lambda \gg 1 \quad \Rightarrow \quad \frac{\lambda_{\max} + \lambda}{\lambda_{\min} + \lambda} \ll \frac{\lambda_{\max}}{\lambda_{\min}}$$

For example, if $\lambda_{\min} = 10^{-3}$ and $\lambda_{\max} = 10^3$, we have $\lambda_{\max}/\lambda_{\min} = 10^6$, but the regularized version $(\lambda_{\max} + \lambda)/(\lambda_{\min} + \lambda)$ takes approximately the values 11, 2, 1.1, for $\lambda = 10^2, 10^3, 10^4$, respectively.

Regularization is not without problems. For noisy data the basic tradeoff is that improving the condition number by increasing λ causes more distortion and smoothing of the desired signal component $\mathbf{x}$. As usual, choosing the proper value of λ is more of an art than science and requires some trial-and-error experimentation. The method of cross-validation can also be applied [1125] as a guide.

Other choices for D, for example differencing matrices, are used in applications such as edge-preserving deblurring of images.

Next, we consider briefly the connection of regularization to machine learning and reproducing kernel Hilbert spaces. Spline smoothing [45] is a regularization and learning example for continuous-time functions, in which the following performance index attempts to "learn" the unknown function $f(t)$ from a finite subset of N noisy observations, $y_n = f(t_n) + v_n$, $n = 0, 1, \dots, N-1$.

$$\mathcal{J} = \sum_{n=0}^{N-1}\big[y_n - f(t_n)\big]^2 + \lambda\int_{t_a}^{t_b}\big[\ddot{f}(t)\big]^2 dt = \min \tag{26.6.6}$$

The concept can be generalized from functions of time to multivariable functions of some independent variable, say $\mathbf{x}$, such as three-dimensional space. The observed data samples are of the form, $y_n = f(\mathbf{x}_n) + v_n$, and the objective is to learn the unknown function $f(\mathbf{x})$. The performance index (26.6.6) is replaced by,

$$\mathcal{J} = \sum_{n=0}^{N-1}\big[y_n - f(\mathbf{x}_n)\big]^2 + \lambda\|f\|^2 = \min \tag{26.6.7}$$

where $\|f\|$ is an appropriate norm that depends on the approach one takes to the minimization problem. One possible and very successful approach is to use a *neural network* to model the unknown function $f(\mathbf{x})$. In this case the regularization norm depends on the parameters of the neural network and its assumed structure (typically a single-hidden layer is sufficient.)

The *reproducing kernel* approach that we discuss here is to assume that $f(\mathbf{x})$ can be represented as a linear combination of a finite or infinite set of nonlinear basis functions $\phi_i(\mathbf{x})$, $i = 1, 2, \dots, M$, where for now we will assume that M is finite,

$$f(\mathbf{x}) = \sum_{i=1}^{M}\phi_i(\mathbf{x})c_i = \boldsymbol{\phi}^T(\mathbf{x})\mathbf{c}\,, \qquad \boldsymbol{\phi}(\mathbf{x}) = \begin{bmatrix}\phi_1(\mathbf{x})\\ \phi_2(\mathbf{x})\\ \vdots\\ \phi_M(\mathbf{x})\end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix}c_1\\ c_2\\ \vdots\\ c_M\end{bmatrix} \tag{26.6.8}$$

This is analogous to the approach of Chap. 23.1 where $f(t)$ was modeled as a polynomial in t and expanded in the monomial basis functions $s_i(t) = t^i$. Here, we define the regularization norm in terms of the coefficients c_i as follows:

$$\|f\|^2 = \sum_{i=1}^{M}\frac{c_i^2}{\lambda_i} = \mathbf{c}^T\Lambda^{-1}\mathbf{c} \tag{26.6.9}$$

where λ_i is a set of some given positive coefficients, and $\Lambda = \text{diag}\big([\lambda_1, \lambda_2, \dots, \lambda_M]\big)$. The function values at the N observation points are $f(\mathbf{x}_n) = \boldsymbol{\phi}^T(\mathbf{x}_n)\mathbf{c}$, and can be arranged into an N-dimensional column vector:

$$\mathbf{f} = \begin{bmatrix}f(\mathbf{x}_0)\\ \vdots\\ f(\mathbf{x}_n)\\ \vdots\\ f(\mathbf{x}_{N-1})\end{bmatrix} = \Phi\mathbf{c}\,, \qquad \Phi = \begin{bmatrix}\boldsymbol{\phi}^T(\mathbf{x}_0)\\ \vdots\\ \boldsymbol{\phi}^T(\mathbf{x}_n)\\ \vdots\\ \boldsymbol{\phi}^T(\mathbf{x}_{N-1})\end{bmatrix} \tag{26.6.10}$$

where Φ has dimension $N\times M$. Thus, the performance index can be written compactly,

$$\boxed{\mathcal{J} = (\mathbf{y} - \Phi\mathbf{c})^T(\mathbf{y} - \Phi\mathbf{c}) + \lambda\mathbf{c}^T\Lambda^{-1}\mathbf{c} = \min} \tag{26.6.11}$$

The solution for the optimum coefficients $\mathbf{c}$ is obtained by setting the gradient to zero:

$$\frac{\partial\mathcal{J}}{\partial\mathbf{c}} = -2\Phi^T(\mathbf{y} - \Phi\mathbf{c}) + 2\lambda\,\Lambda^{-1}\mathbf{c} = 0 \quad \Rightarrow \quad (\lambda\Lambda^{-1} + \Phi^T\Phi)\mathbf{c} = \Phi^T\mathbf{y} \tag{26.6.12}$$

$$\mathbf{c} = (\lambda\Lambda^{-1} + \Phi^T\Phi)^{-1}\Phi^T\mathbf{y} \tag{26.6.13}$$

Using the matrix-inversion lemma, we have:

$$(\lambda\Lambda^{-1} + \Phi^T\Phi)^{-1} = \frac{1}{\lambda}\big[\Lambda - \Lambda\Phi^T(\lambda I + \Phi\Lambda\Phi^T)^{-1}\Phi\Lambda\big] \tag{26.6.14}$$

from which it follows that:

$$(\lambda\Lambda^{-1} + \Phi^T\Phi)^{-1}\Phi^T = \Lambda\Phi^T(\lambda I + \Phi\Lambda\Phi^T)^{-1} \tag{26.6.15}$$

where I is the $N\times N$ identity matrix. Thus, the optimal coefficients are given by

$$\mathbf{c} = \Lambda\Phi^T(\lambda I + \Phi\Lambda\Phi^T)^{-1}\mathbf{y} \tag{26.6.16}$$

The observation vector $\mathbf{f} = \Phi\mathbf{c}$ and estimated function value $f(\mathbf{x}) = \boldsymbol{\phi}^T(\mathbf{x})\mathbf{c}$ are then,

$$\boxed{\begin{aligned}&\mathbf{f} = \Phi\Lambda\Phi^T(\lambda I + \Phi\Lambda\Phi^T)^{-1}\mathbf{y}\\ &f(\mathbf{x}) = \boldsymbol{\phi}^T(\mathbf{x})\Lambda\Phi^T(\lambda I + \Phi\Lambda\Phi^T)^{-1}\mathbf{y}\end{aligned}} \tag{26.6.17}$$

The appearance of the bilinear products of the basis functions suggests that we define the *kernel* function:

$$\boxed{K(\mathbf{x},\mathbf{x}') = \boldsymbol{\phi}^T(\mathbf{x})\Lambda\boldsymbol{\phi}(\mathbf{x}') = \sum_{i=1}^{M}\lambda_i\phi_i(\mathbf{x})\phi_i(\mathbf{x}')} \tag{26.6.18}$$

Let us also define the $N\times N$ symmetric positive-definite kernel matrix K and N-dimensional coefficient vector $\mathbf{a} = [a_0, a_1, \dots, a_{N-1}]^T$ by

$$\boxed{\begin{aligned} K &= \Phi\Lambda\Phi^T \\ \mathbf{a} &= (\lambda I + K)^{-1}\mathbf{y} \end{aligned}} \tag{26.6.19}$$

so that $\mathbf{c} = \Lambda\Phi^T\mathbf{a}$ and $\mathbf{f} = \Phi\mathbf{c} = \Phi\Lambda\Phi^T\mathbf{a} = K\mathbf{a}$ and $f(\mathbf{x}) = \boldsymbol{\phi}^T(\mathbf{x})\Lambda\Phi^T\mathbf{a}$. The matrix elements of K can be expressed in terms of the kernel function:

$$K_{nm} = (\Phi\Lambda\Phi^T)_{nm} = \boldsymbol{\phi}^T(\mathbf{x}_n)\Lambda\boldsymbol{\phi}(\mathbf{x}_m) = K(\mathbf{x}_n, \mathbf{x}_m) \tag{26.6.20}$$

for $n, m = 0, 1, \dots, N-1$. Similarly, we have for $f(\mathbf{x})$,

$$\begin{aligned} f(\mathbf{x}) &= \boldsymbol{\phi}^T(\mathbf{x})\Lambda\Phi^T\mathbf{a} = \boldsymbol{\phi}^T(\mathbf{x})\Lambda\big[\boldsymbol{\phi}(x_0), \dots, \boldsymbol{\phi}(\mathbf{x}_n), \dots, \boldsymbol{\phi}(x_{N-1})\big]\mathbf{a} \\ &= \big[K(\mathbf{x},\mathbf{x}_1), \dots, K(\mathbf{x},\mathbf{x}_n), \dots, K(\mathbf{x},\mathbf{x}_{N-1})\big]\mathbf{a} = \sum_{n=0}^{N-1} K(\mathbf{x},\mathbf{x}_n)\,a_n \end{aligned} \tag{26.6.21}$$

Thus, we may express (26.6.17) directly in terms of the kernel function and the coefficient vector $\mathbf{a}$,

$$\boxed{\mathbf{f} = K\mathbf{a}\,, \quad f(\mathbf{x}) = \sum_{n=0}^{N-1} K(\mathbf{x},\mathbf{x}_n)\,a_n} \tag{26.6.22}$$

Moreover, since $\mathbf{c} = \Lambda\Phi^T\mathbf{a}$, the norm $\|f\|^2$ can also be expressed in terms of K and the vector $\mathbf{a}$ as follows, $\|f\|^2 = \mathbf{c}^T\Lambda^{-1}\mathbf{c} = (\mathbf{a}^T\Phi\Lambda)\Lambda^{-1}(\Lambda\Phi^T\mathbf{a}) = \mathbf{a}^T(\Phi\Lambda\Phi^T)\mathbf{a}$, or,

$$\|f\|^2 = \mathbf{a}^T K\mathbf{a} \tag{26.6.23}$$

Thus, the knowledge of the kernel function $K(\mathbf{x},\mathbf{x}')$—rather than the knowledge of the possibly infinite set of basis functions $\phi_i(\mathbf{x})$—is sufficient to formulate and solve the regularization problem. Indeed, an equivalent optimization problem to (26.6.11) is the following, with the performance index to be minimized with respect to $\mathbf{a}$:

$$\boxed{\mathcal{J} = (\mathbf{y} - K\mathbf{a})^T(\mathbf{y} - K\mathbf{a}) + \lambda\mathbf{a}^T K\mathbf{a} = \min} \tag{26.6.24}$$

The vanishing of gradient with respect to $\mathbf{a}$ leads to the same solution as (26.6.19),

$$\frac{\partial\mathcal{J}}{\partial\mathbf{a}} = -2K^T(\mathbf{y} - K\mathbf{a}) + 2\lambda K\mathbf{a} = 0 \quad\Rightarrow\quad (\lambda K + K^T K)\mathbf{a} = K^T\mathbf{y} \quad\Rightarrow\quad \mathbf{a} = (\lambda I + K)^{-1}\mathbf{y}$$

where we used the symmetry property $K^T = K$ and assumed that K was invertible.

The linear vector space of functions of the form $f(\mathbf{x}) = \boldsymbol{\phi}^T(\mathbf{x})\mathbf{c}$, spanned by the set of basis functions $\{\phi_i(\mathbf{x}),\ i = 1, 2, \dots, M\}$, can be turned into an inner-product space (a Hilbert space if $M = \infty$) by endowing it with the inner product induced by the norm (26.6.9). That is, for any two functions $f_1(\mathbf{x}) = \boldsymbol{\phi}^T(\mathbf{x})\mathbf{c}_1$ and $f_2(\mathbf{x}) = \boldsymbol{\phi}^T(\mathbf{x})\mathbf{c}_2$, we define the inner product:

$$\langle f_1, f_2\rangle = \mathbf{c}_1^T\Lambda^{-1}\mathbf{c}_2 \tag{26.6.25}$$

The resulting vector space, say $\mathcal{H}$, is referred to as a *reproducing kernel Hilbert space*. By writing the kernel function in the form, $K(\mathbf{x},\mathbf{x}') = \boldsymbol{\phi}^T(\mathbf{x})\Lambda\boldsymbol{\phi}(\mathbf{x}') \equiv \boldsymbol{\phi}^T(\mathbf{x})\mathbf{c}(\mathbf{x}')$, with $\mathbf{c}(\mathbf{x}') = \Lambda\boldsymbol{\phi}(\mathbf{x}')$, we see that, as a function of $\mathbf{x}$ for each fixed $\mathbf{x}'$, it lies in the space $\mathcal{H}$, and satisfies the two reproducing-kernel properties:

$$f(\mathbf{x}') = \big\langle f(\cdot), K(\cdot,\mathbf{x}')\big\rangle\,, \qquad K(\mathbf{x},\mathbf{x}') = \big\langle K(\cdot,\mathbf{x}), K(\cdot,\mathbf{x}')\big\rangle \tag{26.6.26}$$

These follow from the definition (26.6.25). Indeed, given $f(\mathbf{x}) = \boldsymbol{\phi}^T(\mathbf{x})\mathbf{c}$, we have,

$$\big\langle f(\cdot), K(\cdot,\mathbf{x}')\big\rangle = \mathbf{c}^T\Lambda^{-1}\mathbf{c}(\mathbf{x}') = \mathbf{c}^T\Lambda^{-1}\Lambda\,\boldsymbol{\phi}(\mathbf{x}') = \mathbf{c}^T\boldsymbol{\phi}(\mathbf{x}') = f(\mathbf{x}')$$

$$\big\langle K(\cdot,\mathbf{x}), K(\cdot,\mathbf{x}')\big\rangle = \mathbf{c}(\mathbf{x})^T\Lambda^{-1}\mathbf{c}(\mathbf{x}') = \boldsymbol{\phi}^T(\mathbf{x})\Lambda\Lambda^{-1}\Lambda\,\boldsymbol{\phi}(\mathbf{x}') = \boldsymbol{\phi}^T(\mathbf{x})\Lambda\,\boldsymbol{\phi}(\mathbf{x}') = K(\mathbf{x},\mathbf{x}')$$

One can re-normalize the basis functions by defining $\bar{\phi}_i(\mathbf{x}) = \lambda_i^{-1/2}\phi_i(\mathbf{x})$, or, vectorially $\bar{\boldsymbol{\phi}}(\mathbf{x}) = \Lambda^{-1/2}\boldsymbol{\phi}(\mathbf{x})$, which imply the renormalized basis matrix $\bar{\Phi} = \Phi\Lambda^{1/2}$ and coefficient vector $\bar{\mathbf{c}} = \Lambda^{-1/2}\mathbf{c}$. We obtain then the alternative expressions:

$$\begin{aligned} \bar{\mathbf{c}} &= \bar{\Phi}^T\mathbf{a} \\ \mathbf{f} &= \Phi\mathbf{c} = \bar{\Phi}\bar{\mathbf{c}} \\ K &= \Phi\Lambda\Phi^T = \bar{\Phi}\bar{\Phi}^T \\ \|f\|^2 &= \mathbf{c}^T\Lambda^{-1}\mathbf{c} = \bar{\mathbf{c}}^T\bar{\mathbf{c}} \end{aligned} \tag{26.6.27}$$

and kernel function,

$$K(\mathbf{x},\mathbf{x}') = \boldsymbol{\phi}^T(\mathbf{x})\Lambda\boldsymbol{\phi}(\mathbf{x}') = \bar{\boldsymbol{\phi}}^T(\mathbf{x})\bar{\boldsymbol{\phi}}(\mathbf{x}') \tag{26.6.28}$$

Eq. (26.6.28) expresses the kernel function as the dot product of two vectors and is known as the *kernel trick*. Given a kernel function $K(\mathbf{x},\mathbf{x}')$ that satisfies certain positive-definiteness conditions, the existence of basis functions satisfying Eq. (26.6.28) is guaranteed by Mercer's theorem [1117]. The remarkable property of the kernel regularization approach is Eq. (26.6.22), which is known as the *representer theorem* [1117],

$$f(\mathbf{x}) = \sum_{n=0}^{N-1} K(\mathbf{x},\mathbf{x}_n)\,a_n \tag{26.6.29}$$

It states that even though the original least-squares problem (26.6.11) was formulated in a possibly infinite-dimensional Hilbert space, the resulting solution is represented by a finite number of terms in Eq. (26.6.29). This property is more general than the above case and it applies to a performance index of the form:

$$\mathcal{J} = L(\mathbf{y} - \Phi\mathbf{c}) + \lambda\mathbf{c}^T\Lambda^{-1}\mathbf{c} = \min \tag{26.6.30}$$

where $L(\mathbf{z})$ is an arbitrary (convex, increasing, and differentiable) scalar function that replaces the quadratic norm $L(\mathbf{z}) = \mathbf{z}^T\mathbf{z}$. Indeed, the vanishing of the gradient gives,

$$\frac{\partial \mathcal{J}}{\partial \mathbf{c}} = -\Phi^T \frac{\partial L}{\partial \mathbf{y}} + 2\lambda \Lambda^{-1}\mathbf{c} = 0 \quad \Rightarrow \quad \mathbf{c} = \Lambda \Phi^T \frac{1}{2\lambda}\frac{\partial L}{\partial \mathbf{y}}$$

which implies for $\mathbf{f} = \Phi\mathbf{c}$ and $f(\mathbf{x}) = \boldsymbol{\phi}^T(\mathbf{x})\mathbf{c}$,

$$\mathbf{f} = K\mathbf{a}\,, \quad f(\mathbf{x}) = \sum_{n=0}^{N-1} K(\mathbf{x}, \mathbf{x}_n)\, a_n\,, \quad \text{with} \quad \mathbf{a} = \frac{1}{2\lambda}\frac{\partial L(\mathbf{y} - K\mathbf{a})}{\partial \mathbf{y}} \tag{26.6.31}$$

where the last equation is a nonlinear equation for the N-vector $\mathbf{a}$. Of course, in the quadratic-norm case, $L(\mathbf{z}) = \mathbf{z}^T\mathbf{z}$, we obtain the equivalent of (26.6.19),

$$\mathbf{a} = \frac{1}{\lambda}\,(\mathbf{y} - K\mathbf{a})$$

Kernels and the above representation property are used widely in machine learning applications, such as support vector machines [1104]. Some typical kernels that satisfy the representation property (26.6.28) are polynomial and gaussian of the type:

$$\begin{aligned} K(\mathbf{x}, \mathbf{x}') &= (c + \mathbf{x} \cdot \mathbf{x}')^p \\ K(\mathbf{x}, \mathbf{x}') &= \exp\Big(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\Big) \end{aligned} \tag{26.6.32}$$

By mapping nonlinear problems into linear ones, kernel methods offer a new paradigm for solving many of the classical problems of estimation and classification, including the “kernelization” of methods such as principal component analysis [1118], canonical correlation analysis, array processing [1121], and adaptive filtering [1124]. Some accessible overviews of kernel methods with emphasis on regularization are [1109,1115,1120]. For more details, the reader may consult the references [1085–1124].

26.7 Sparse Whittaker-Henderson Methods

Several variations of the Whittaker-Henderson method have been proposed in the literature that use different norms for the two terms of Eq. (26.2.1), such as the following criterion based on the L_r and the L_p norms, and using unity weights w_n for simplicity,

$$\mathcal{J}_{rp} = \sum_{n=0}^{N-1} |y_n - x_n|^r + \lambda \sum_{n=s}^{N-1} |\nabla^s x_n|^p = \min \tag{26.7.1}$$

Such criteria are capable of handling outliers in the data more effectively. Eq. (26.7.1) can be written vectorially with the help of the s-differencing matrix D of Eq. (26.2.8),

$$\mathcal{J}_{rp} = ||\mathbf{y} - \mathbf{x}||_r^r + \lambda ||D\mathbf{x}||_p^p = \min \tag{26.7.2}$$

where $\|\mathbf{x}\|_p$ denotes the L_p norm of the vector $\mathbf{x} = [x_0, x_1, \cdots, x_{N-1}]^T$,

$$\|\mathbf{x}\|_p = \Bigg[\sum_{n=0}^{N-1} |x_n|^p\Bigg]^{\frac{1}{p}} \quad \Rightarrow \quad ||\mathbf{x}||_p^p = \sum_{n=0}^{N-1} |x_n|^p$$

and similarly for $\|\mathbf{x}\|_r$. For $p = \infty$, we have instead,

$$\|\mathbf{x}\|_\infty = \max_{0 \le n \le N-1} |x_n|$$

For $p = 0$, we define $\|\mathbf{x}\|_0$ as the *cardinality* of the vector $\mathbf{x}$, that is, the number of *nonzero* elements of $\mathbf{x}$. We note that $\|\mathbf{x}\|_p$ is a proper norm only for $p \ge 1$, however, the cases $0 \le p < 1$ have also been considered.

The case $\mathcal{J}_{11}$ was studied in [1024,1028] and formulated as a linear programming problem, the case $\mathcal{J}_{pp}$, including the L_∞ norm case, $p = \infty$, was studied in [1026], and the more general case, $\mathcal{J}_{rp}$, in [1027]. More recently, the case $\mathcal{J}_{21}$, called L_1 *trend filtering*, has been considered in [1070] and has received a lot of attention [1071–1080].

Generally, the cases $\mathcal{J}_{2p}$ are examples of so-called *L_p-regularized least-squares* problems, which have been studied very extensively in inverse problems, with renewed interest in sparse modeling, statistical learning, compressive sensing applications—a small and very incomplete set of references on regularization and sparse regularization methods is [1084–1196].

Next, we concentrate on the original $\mathcal{J}_{22}$ criterion, and the $\mathcal{J}_{21}$ and $\mathcal{J}_{20}$ criteria,

$$\begin{aligned} \mathcal{J}_{22} &= ||\mathbf{y} - \mathbf{x}||_2^2 + \lambda\, ||D\mathbf{x}||_2^2 = \min \\ \mathcal{J}_{21} &= ||\mathbf{y} - \mathbf{x}||_2^2 + \lambda\, ||D\mathbf{x}||_1 = \min \\ \mathcal{J}_{20} &= ||\mathbf{y} - \mathbf{x}||_2^2 + \lambda\, ||D\mathbf{x}||_0 = \min \end{aligned} \tag{26.7.3}$$

The $\mathcal{J}_{21}$ and $\mathcal{J}_{20}$ criteria tend to promote the *sparsity* of the regularizing term $D\mathbf{x}$, that is, $D\mathbf{x}$ will be a *sparse* vector consisting mostly of zeros with a few nonzero entries. Since $D\mathbf{x}$ represents the s-differenced signal, $\nabla^s x_n$, its piecewise vanishing implies that the trend x_n will be a piecewise polynomial of order $s-1$, with the polynomial pieces joining continuously at few break (or, kink) points where $\nabla^s x_n$ is nonzero.

This is similar to the spline smoothing case, except here the locations of the break points are determined dynamically by the solution of the optimization problem, whereas in the spline case they are at prescribed locations.

For differencing order $s = 2$, used in Hodrick-Prescott and L_1-trend-filtering cases, the trend signal x_n will be a piecewise linear function of n, with a sparse number of slope changes. The case $s = 3$, used originally by Whittaker and Henderson, would correspond to piecewise parabolic segments in n. The case $s = 1$, corresponding to the original Bohlmann choice, results in a piecewise constant trend signal x_n. This case is known also as *total variation* minimization method[1135] and has been applied widely in image processing.

The $\mathcal{J}_{21}$ problem can be implemented easily in MATLAB with the CVX package.[†] The $\mathcal{J}_{20}$ problem, which produces the sparsest solution, can be solved by an *iterative reweighted L_1-regularized* method [1070], or alternatively, by an *iterative reweighted least-squares* method, which can also be used to solve the $\mathcal{J}_{21}$ and the $\mathcal{J}_{2p}$ problems.

There are several variants of the iterative reweighted least-squares (IRLS) method, [1126–1134,1138,1159,1163,1166,1171,1172], but the basic idea is to replace the L_p

[†] `http://cvxr.com/cvx/`

norm with a weighted L_2 norm, which can be solved iteratively. Given any real number $0 \le p \le 2$, let $q = 2 - p$, and note that for any real number $x \neq 0$, we can write,

$$|x|^p = \frac{|x|^2}{|x|^q} \approx \frac{|x|^2}{|x|^q + \varepsilon}$$

where ε is a sufficiently small positive number needed to also allow the case $x = 0$. Similarly, we can write for the L_p-norm of a vector $\mathbf{x} \in \mathbb{R}^N$,

$$\begin{aligned} \|\mathbf{x}\|_p^p &= \sum_{i=0}^{N-1} |x_i|^p \approx \sum_{i=0}^{N-1} \frac{|x_i|^2}{|x_i|^q + \varepsilon} = \mathbf{x}^T W(\mathbf{x})\mathbf{x} \\ W(\mathbf{x}) &= \text{diag}\left[\frac{1}{|\mathbf{x}|^q + \varepsilon}\right] = \text{diag}\left[\frac{1}{|x_0|^q + \varepsilon},\, \frac{1}{|x_1|^q + \varepsilon},\, \dots,\, \frac{1}{|x_{N-1}|^q + \varepsilon}\right] \end{aligned} \tag{26.7.4}$$

Then, the L_p-regularized problem $\mathcal{J}_{2p}$ can be written in the form,

$$\mathcal{J} = \|\mathbf{y} - \mathbf{x}\|_2^2 + \lambda \|D\mathbf{x}\|_p^p = \|\mathbf{y} - \mathbf{x}\|_2^2 + \lambda\, \mathbf{x}^T D^T W(D\mathbf{x})\, D\mathbf{x} = \min \tag{26.7.5}$$

which leads to to the following iterative algorithm,

$$\boxed{\begin{aligned} &\text{for } k = 1, 2, \dots, K, \text{ do:} \\ &\qquad W_{k-1} = W\big(D\mathbf{x}^{(k-1)}\big) \\ &\qquad \mathbf{x}^{(k)} = \arg\min_{\mathbf{x}} \|\mathbf{y} - \mathbf{x}\|_2^2 + \lambda\, \mathbf{x}^T D^T W_{k-1} D\mathbf{x} \end{aligned}} \qquad \text{(IRLS)} \tag{26.7.6}$$

with the algorithm initialized to the ordinary least-squares solution of criterion $\mathcal{J}_{22}$,

$$\mathbf{x}^{(0)} = \big(I + \lambda D^T D\big)^{-1} \mathbf{y}$$

The solution of the optimization problem in (26.7.6) at the kth step is:

$$\mathbf{x}^{(k)} = \big(I + \lambda D^T W_{k-1} D\big)^{-1} \mathbf{y}$$

Thus, the choices $p = 0$ and $p = 1$ should resemble the solutions of the L_0 and L_1 regularized problems.

26.8 Computer Experiments

Next, we consider a number of computer experiment examples that illustrate and compare the performaces of the various approaches, standard Whittaker-Henderson least-squares, regularized least-squares, sparse methods, as well as local polynomial smoothing methods and SVD enhancement [45] methods.

We summarize the criteria to be implemented in the computer experiments. For a length-N signal of observations, y_n, $0 \le n \le N-1$, the L_2, L_1, and L_0 optimization criteria for determining a length-N smoothed signal x_n are:

$$\boxed{\begin{aligned} (L_2):&\quad \mathcal{J} = \sum_{n=0}^{N-1} \big|y_n - x_n\big|^2 + \lambda \sum_{n=s}^{N-1} \big|\nabla^s x_n\big|^2 = \|\mathbf{y} - \mathbf{x}\|_2^2 + \lambda\, \|D_s\mathbf{x}\|_2^2 = \min \\ (L_1):&\quad \mathcal{J} = \sum_{n=0}^{N-1} \big|y_n - x_n\big|^2 + \lambda \sum_{n=s}^{N-1} \big|\nabla^s x_n\big|^1 = \|\mathbf{y} - \mathbf{x}\|_2^2 + \lambda\, \|D_s\mathbf{x}\|_1 = \min \\ (L_0):&\quad \mathcal{J} = \sum_{n=0}^{N-1} \big|y_n - x_n\big|^2 + \lambda \sum_{n=s}^{N-1} \big|\nabla^s x_n\big|^0 = \|\mathbf{y} - \mathbf{x}\|_2^2 + \lambda\, \|D_s\mathbf{x}\|_0 = \min \end{aligned}} \tag{26.8.1}$$

where the L_0 norm[†], $\|\mathbf{x}\|_0$, is the cardinality of the vector $\mathbf{x}$, that is, the number of its non-zero entries, and D_s is the $(N-s)\times N$ convolution matrix corresponding to the s-difference operator ∇^s. It can be constructed in MATLAB by

```
Ds = diff(eye(N),s);     % or, sparsely, Ds = diff(speye(N),s);
```

The solution of problem (L_2) is straightforward:

$$\mathbf{x} = (I + \lambda D_s^T D_s)^{-1} \mathbf{y} \tag{26.8.2}$$

The solution of problem (L_1) can be obtained with the CVX package[‡] as follows:

```
cvx_begin
   variable x(N)
   minimize( sum_square(x-y) + la * norm(Ds*x,1) );
cvx_end
```

Alternatively, the (L_1) problem, as well as the (L_0) problem, can be solved with the iteratively re-weighted least-squares (IRLS) method.

The purpose of the regularizing, λ-term, in Eq. (26.8.1) is to enforce a certain degree of smoothness on the solution, that is, since the minimization tries to make the term $\nabla^s x_n$ as small as possible and since $\nabla^s x_n$ acts as the s-th derivative of the signal, this would imply a smoother solution.

Furthermore, the most essential property of the L_1 and L_0 criteria is that they enforce the sparsity of the term,[‡] $\nabla^s x_n$, that is, $\nabla^s x_n = 0$ except for a few values of n, so that the signal x_n will be characterized of piecewise polynomials of degree $s - 1$.

Thus, the case $s = 1$ would be appropriate for modeling piecewise constant signals, i.e., polynomials of degree $s - 1 = 0$. This case has wide application in image denoising applications and is referred to as the “total-variation minimization” method [1135]. This makes sense for images since they often consist of large areas or patches of constant intensity.

The case $s = 2$ would be appropriate for modeling piecewise linear trends, i.e., polynomials of degree $s - 1 = 1$, and is the basic choice for s in the Hodrick-Prescott filter.

[†] L_0 is not strictly-speaking a norm.

[‡] `http://cvxr.com/cvx/` – please download and install this package on your computer.

[‡] in general, they enforce the sparsity of whatever quantity lies inside the norms $\|\cdot\|_1$ and $\|\cdot\|_0$.

Higher values of s, such as $s = 3$ or $s = 4$ can also be used, but they are not as common. Whittaker and Henderson used $s = 3$ in the actuarial context.

Regarding the choice of λ, the best practice is by trial-and-error. There exist criteria for choosing λ, however, they are not particularly good. The higher the λ, the more emphasis is placed on minimizing $\nabla^s x_n$, and the smoother the solution. Too high a value of λ might result in too smooth a signal. In practice, since we do not know the ground truth, we must resort to the trial-end-error method of trying several λ's until the result seems acceptable.

26.8.1 Total Variation Minimization

Consider a piece-wise constant, flat-top, signal x_n and its observed noisy version, y_n, defined by

$$y_n = x_n + v_n\,, \quad 0 \le n \le 600$$

where v_n is zero-mean white noise. The signals x_n and y_n have been saved into the attached MAT file, **yflat.mat**, and can be loaded with the command,

```
load yflat;            % loads signals in the variables x,y
```

a. Load and plot the two signals x_n, y_n in two separate graphs. The signal x_n will serve as the "ground truth" for evaluating its various estimated versions.

b. Let $s = 1$ and $\lambda = 5$. Using the L_2 criterion with solution given by Eq. (26.8.2), calculate and plot the estimated signal x_n, and on a separate graph, plot the s-differenced signal, $\nabla^s x_n$, which is expected to be small but not necessarily sparse by the minimization condition.

c. For the same values of s, λ, solve the L_1 problem using the CVX package, and then using the IRLS algorithm, and for both cases, plot the estimated signal x_n, as well as the s-differenced signal, $\nabla^s x_n$, which is expected to be very sparse by the minimization condition.

   Moreover, solve the L_0 problem using the IRLS algorithm, and plot the estimated signal x_n, and the s-differenced signal, $\nabla^s x_n$, which is also supposed to be very sparse.

   For the L_1 and L_0 IRLS cases, compute and plot also the iteration percentage error $P(k)$ versus iteration number k.

   Discuss the effectiveness of the L_2, L_1, and L_0 methods in reducing noise while not affecting the desired signal.

d. For comparison purposes, compute also the smoothed output using a single EMA filter and an equivalent SMA filter, choosing the duration of these filters to be less than the duration of the flat-top portions, e.g., $N = 39$, so that you can see both the transient and the steady-state parts. Plot the two smoothed filtered outputs on two separate graphs.

Some example graphs are shown below in Figs. 26.8.1 – 26.8.7.

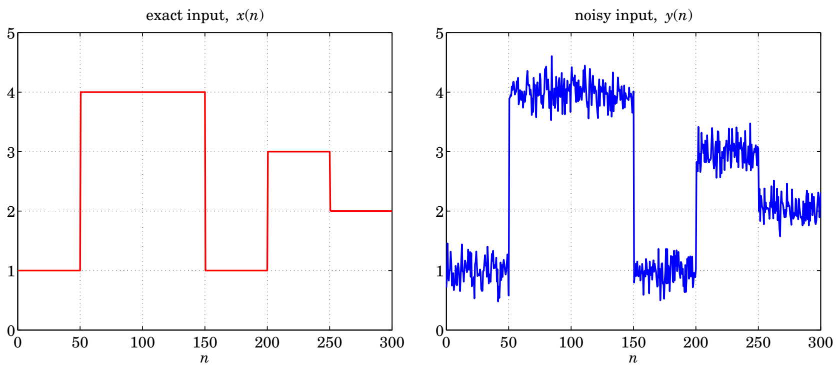


Fig. 26.8.1 Total variation minimization example of Sec. 26.8.1.

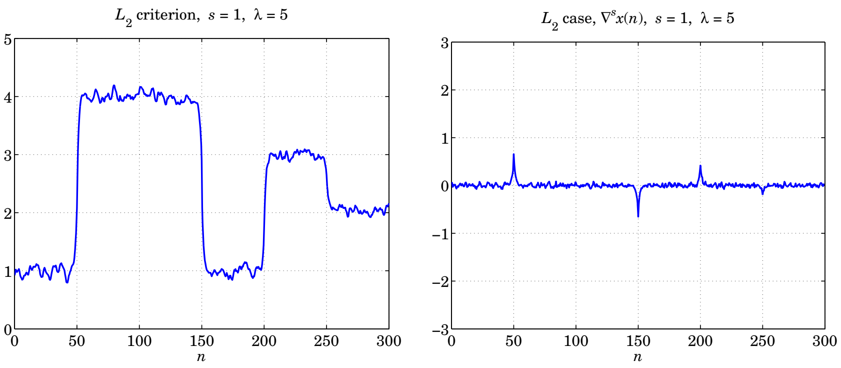


Fig. 26.8.2 Total variation minimization example of Sec. 26.8.1.

26.8.2 Local Linear Trends

Consider a piecewise linear signal x_n and its observed noisy version, y_n, defined by

$$y_n = x_n + v_n\,, \quad 0 \le n \le 600$$

where v_n is zero-mean white noise. The signals x_n and y_n have been saved into the attached MAT file, **ylin.mat**, and can be loaded with the command,

```
load ylin;             % loads signals in the variables x,y
```

Repeat questions (a–d) of the previous section, with the following changes: use $s = 2$, and $\lambda = 100$ for the L_2 and L_1 parts, and $\lambda = 1$ for the L_0 part. Also, use a DEMA filter instead of a single EMA to avoid any lag in the output. For the SMA case, the lag will be visible.

Some example graphs are shown below in Figs. 26.8.8 – 26.8.14.

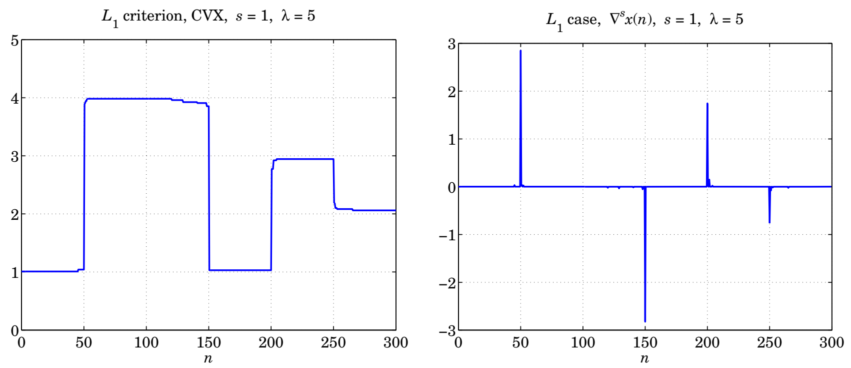


Fig. 26.8.3 Total variation minimization example of Sec. 26.8.1.

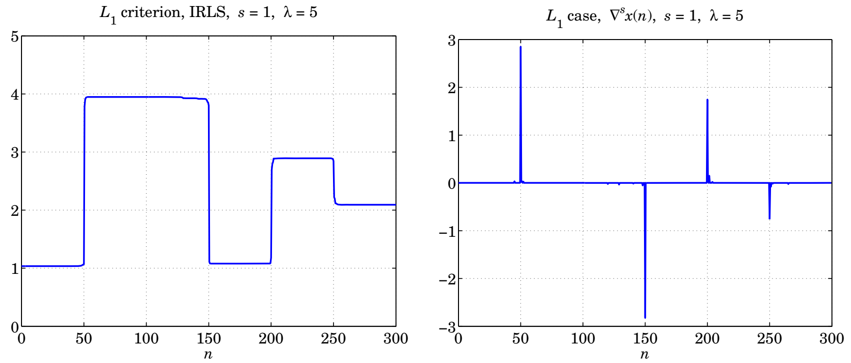


Fig. 26.8.4 Total variation minimization example of Sec. 26.8.1.

26.8.3 Global Warming Trends

Global Warming Trends. This is a continuation of Example 23.10.2 in which we compared several smoothing methods. Figs. 26.8.15 & 26.8.16 compare the Whittaker-Henderson trends for the L_2, L_1, and L_0 cases, with $s = 2$, as well as the corresponding regularizing differenced signals, $\nabla^s x_n$.

The L_1 case was computed with the CVX package. The corresponding IRLS implementation is not shown since it produces virtually indistinguishable graphs from CVX.

The L_0 case was implemented with the IRLS method and produced slightly sparser differenced signals as can be observed in the graphs. The MATLAB code used to generate these graphs is summarized below.

```
Y = loadfile('tavenh2v.dat');              % load temperature data file

n = Y(:,1); y = Y(:,14); N = length(y);    % extract dates and data
```

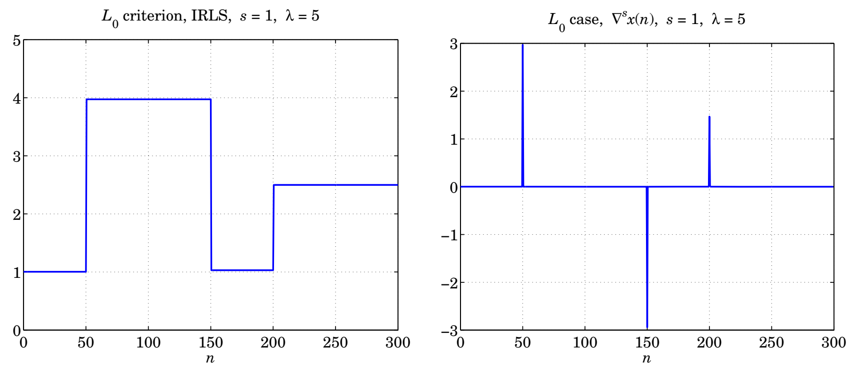


Fig. 26.8.5 Total variation minimization example of Sec. 26.8.1.

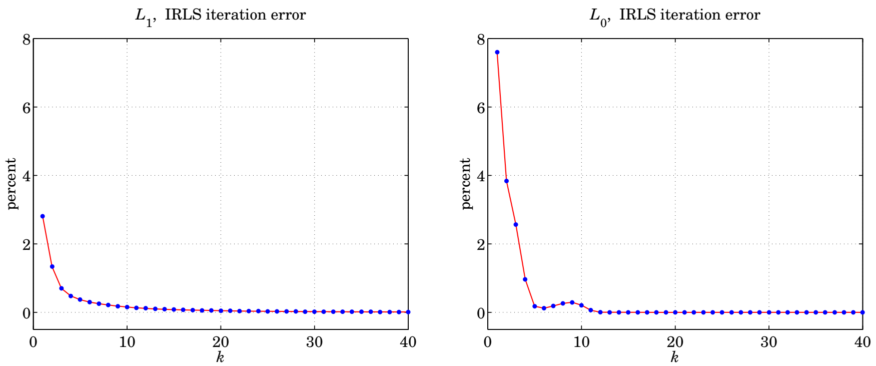


Fig. 26.8.6 Total variation minimization example of Sec. 26.8.1.

```
s = 2; Ds = diff(speye(N),s);              % (N-s)xN differencing matrix
ns = n(s:end-1);

la = 10000; x = whsm(y,la,s);              % Whittaker-Henderson with L2 norm

figure; plot(n,y,'r:', n,x,'b-');          % plot trend
figure; plot(ns, Ds*x,'b-');               % plot differenced trend

la = 10;                                   % Whittaker-Henderson with L1 norm
cvx_quiet(true);                           % CVX package, http:/cvxr.com/cvx/
cvx_begin
    variable x(N)
    minimize( sum_square(y-x) + la * norm(Ds*x,1) )
cvx_end

figure; plot(n,y,'r:', n,x,'b-');          % plot trend
figure; plot(ns, Ds*x,'b-');               % plot differenced trend
```

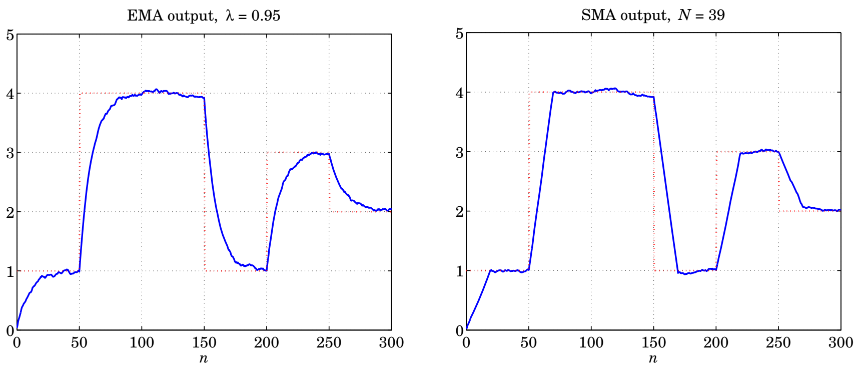


Fig. 26.8.7 Total variation minimization example of Sec. 26.8.1.

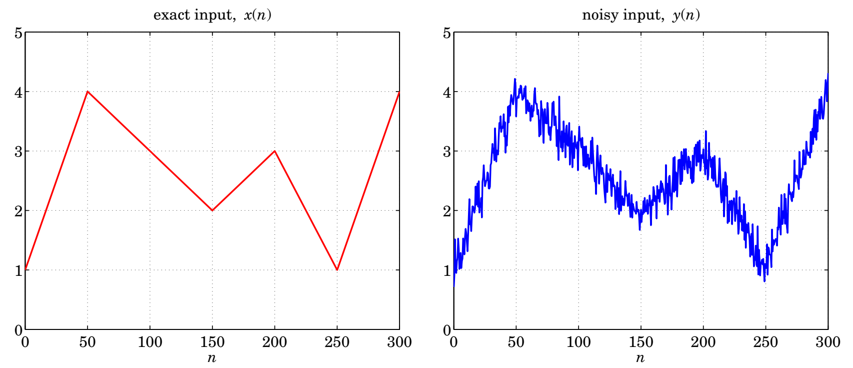


Fig. 26.8.8 Local linear trends – Sec. 26.8.2.

```
p = 0; q = 2 - p; epsilon = 1e-8;          % Whittaker-Henderson with L0 norm
I = speye(N); K = 10;                      % using K=10 IRLS iterations
la = 0.05;

x  = (I + la*Ds'*Ds) \ y;                  % initialize iteration

for k=1:K,                                 % IRLS iteration
   W = diag(1./(abs(Ds*x).^q + epsilon));
   xk = (I + la*Ds'*W*Ds) \ y;
   x = xk;
end

figure; plot(n,y,'r:', n,x,'b-');          % plot trend
figure; plot(ns, Ds*x,'b-');               % plot differenced trend
```

Figs. 26.8.17 & 26.8.18 compare the L_2, L_1, L_0 cases for $s = 3$, which fits piecewise quadratic polynomials to the data. The L_0 case is again the sparsest. (Color graphs online).

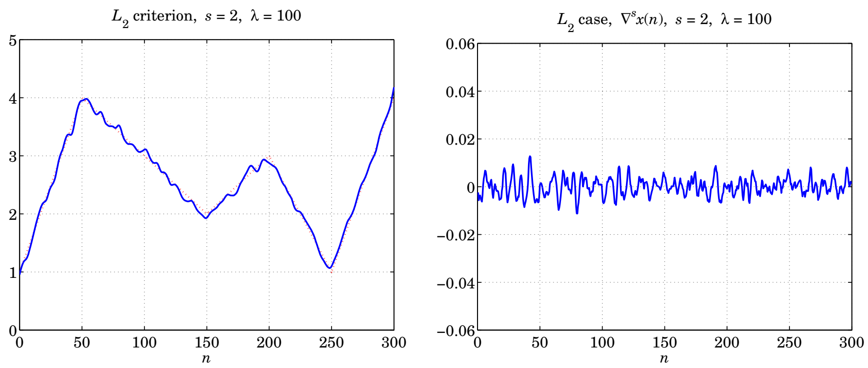


Fig. 26.8.9 Local linear trends – Sec. 26.8.2.

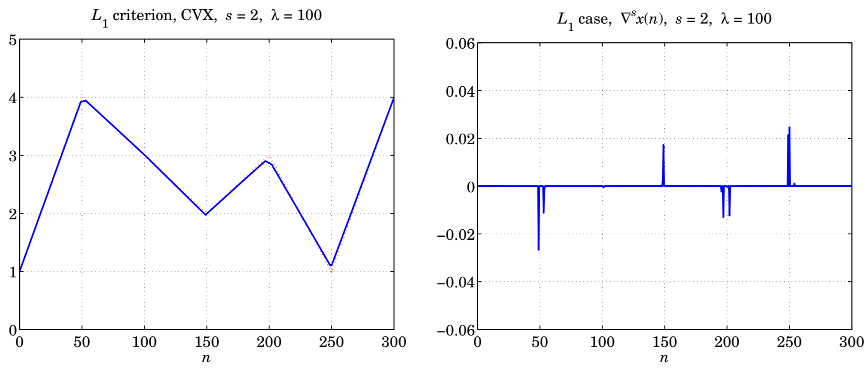


Fig. 26.8.10 Local linear trends – Sec. 26.8.2.

26.8.4 US GDP Macroeconomic Data

In this computer experiment, we study the *Whittaker-Henderson smoothing method* formulated with the L_2 and L_1 norms, and apply it to the US GDP macroeconomic data. Again, for a length-N signal, y_n, $0 \le n \le N-1$, the optimization criteria for determining a length-N smoothed signal x_n are,

$$\begin{aligned}
(L_2):\quad & \mathcal{J} = \sum_{n=0}^{N-1}\big|y_n - x_n\big|^2 + \lambda\sum_{n=s}^{N-1}\big|\nabla^s x_n\big|^2 = ||\mathbf{y}-\mathbf{x}||_2^2 + \lambda\,||D_s\mathbf{x}||_2^2 = \min \\
(L_1):\quad & \mathcal{J} = \sum_{n=0}^{N-1}\big|y_n - x_n\big|^2 + \lambda\sum_{n=s}^{N-1}\big|\nabla^s x_n\big| \;\;= ||\mathbf{y}-\mathbf{x}||_2^2 + \lambda\,||D_s\mathbf{x}||_1 = \min
\end{aligned}\tag{26.8.3}$$

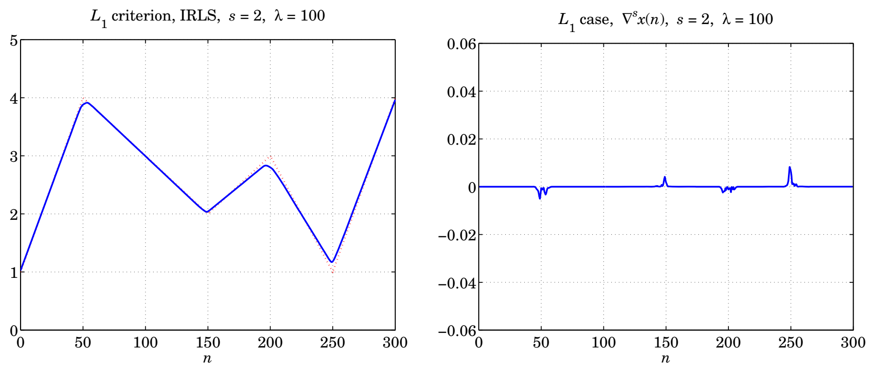


Fig. 26.8.11 Local linear trends – Sec. 26.8.2.

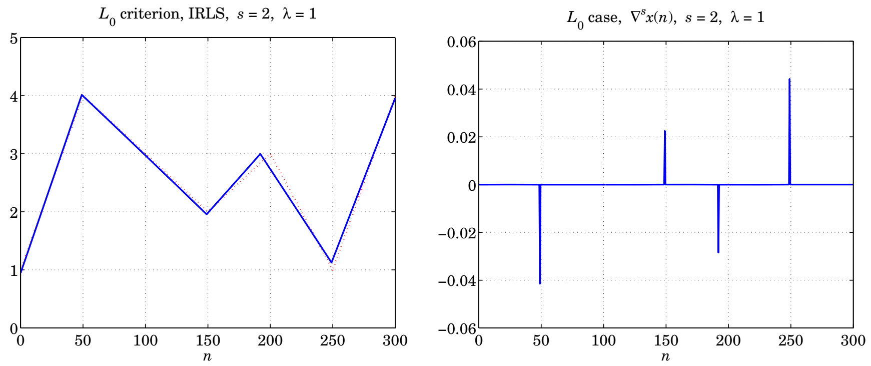


Fig. 26.8.12 Local linear trends – Sec. 26.8.2.

where D_s is the $(N-s)\times N$ convolution matrix corresponding the s-difference operator ∇^s. It can be constructed in MATLAB by,

```
Ds = diff(eye(N),s);        % or, in sparse form, Ds = diff(speye(N),s);
```

The solution of problem (L_2) is straightforward:

$$\mathbf{x} = (I + \lambda D_s^T D_s)^{-1}\mathbf{y} \tag{26.8.4}$$

The solution of problem (L_1) can be obtained with the CVX package as follows:

```
cvx_begin
   variable x(N)
   minimize( sum_square(x-y) + lambda * norm(Ds*x,1) );
cvx_end
```

It can also be solved with the *iterative reweighted least-squares* (IRLS) algorithm, as discussed in Sec. 26.7.

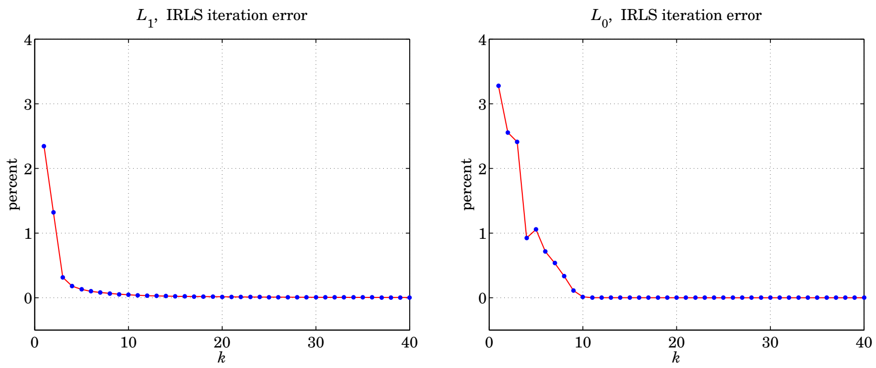


Fig. 26.8.13 Local linear trends – Sec. 26.8.2.

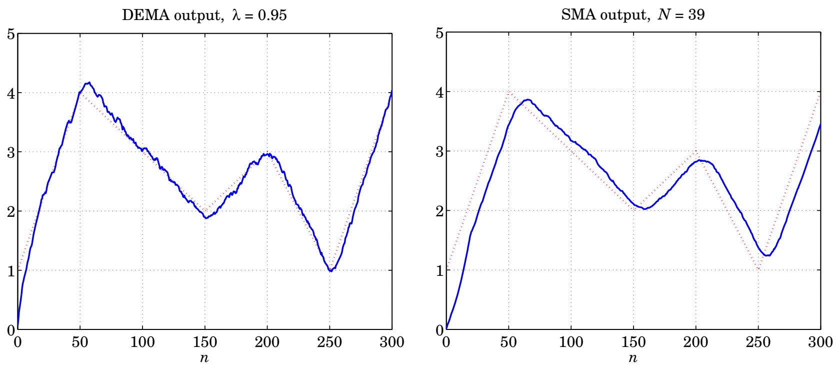


Fig. 26.8.14 Local linear trends – Sec. 26.8.2.

The second column of the AOSP data file, `USGDP_Inv.dat`, represents the quarterly US GDP for private investment in billions of dollars. Read this column with the help of the function `loadfile` and then take its log:

```
Y = loadfile('USGDP_Inv.dat');
y = log10(Y(:,2) * 1e9);
```

These data represent a prototypical example for the application of Whittaker-Henderson filters, referred to in this context as Hodrick-Prescott filters.

Questions

a. Choose difference order $s = 2$ and regularization parameter $\lambda = 1600$. Solve the Whittaker-Henderson problem (L_2) and plot the solution $\mathbf{x}$ together with the actual data $\mathbf{y}$.

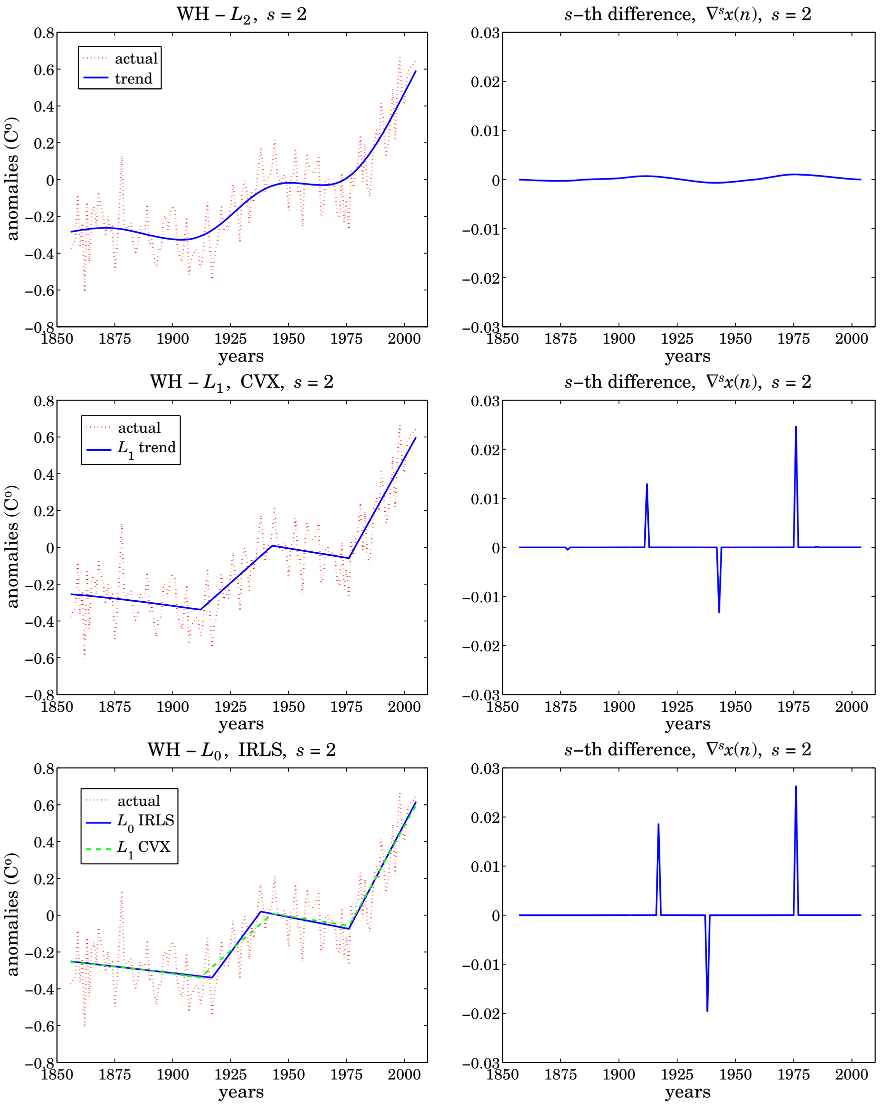


Fig. 26.8.15 Gloval warming trends, $s = 2$ cases.

b. Calculate the SVD enhanced version of $\mathbf{y}$, by the following steps: (i) remove and save the mean from $\mathbf{y}$, (ii) form its forward/backward data matrix using an embedding order of $M = 9$, (iii) subject it to $K = 8$ SVD enhancement iterations using rank $r = 1$, (iv) extract the enhanced signal from the enhanced data matrix, and (v) add the mean that was removed. Plot the resulting enhanced signal together with $\mathbf{y}$. SVD enhancement is discussed in detail in [45].

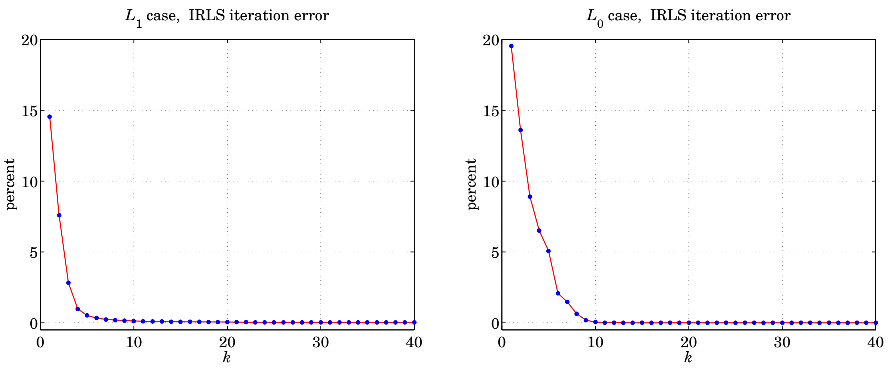


Fig. 26.8.16 Global Warming Trends – iteration errors for $s = 2$.

c. For the value $\lambda_1 = \lambda/480 = 1600/480$ and difference order $s = 2$, solve problem (L_1), and plot the solution $\mathbf{x}$ together with the actual data $\mathbf{y}$. Moreover, on a separate graph, plot the differenced signal $D_s\mathbf{x}$ using a stem plot and observe its sparseness, which means that $\mathbf{x}$ is piece-wise linear. The particular choice for λ_1 was made in order for the (L_2) and (L_1) problems to have comparable RMS errors.

d. Repeat parts (a) and (c) for $s = 1$ and regularization parameter $\lambda = 60.65$ for the (L_2) problem (justified in Sec. 26.4), and $\lambda_1 = 1$ for the (L_1) problem, chosen to achieve comparable RMS errors. Notice how the (L_1) problem results in a piece-wise constant fit. But $D_s\mathbf{x}$ is not as sparse because $s = 1$ is not really a good choice.

   The $s = 1$ case is an example of the so-called *total-variation minimization* method, used widely in image processing.

e. Repeat parts (a) and (c) for $s = 3$ and regularization parameter $\lambda = 41640.16$ for the (L_2) problem (justified in Sec. 26.4), and $\lambda_1 = \lambda/1000$ for the (L_1) problem, chosen to achieve comparable RMS errors. Here, the (L_1) problem will result in piece-wise quadratic polynomial fits. Some example graphs are shown below in Figs. 26.8.19 – 26.8.23.

26.9 Sparse Modeling – LASSO and BPDN

Regularization is a method for solving linear equations of the form, $A\mathbf{x} = \mathbf{b}$, that may be ill-posed, inconsistent, overdetermined, underdetermined, or ill-conditioned – some references on the topic are [1085–1125]. The regularized deconvolution problem of Eq. (26.6.2) was just such an example.

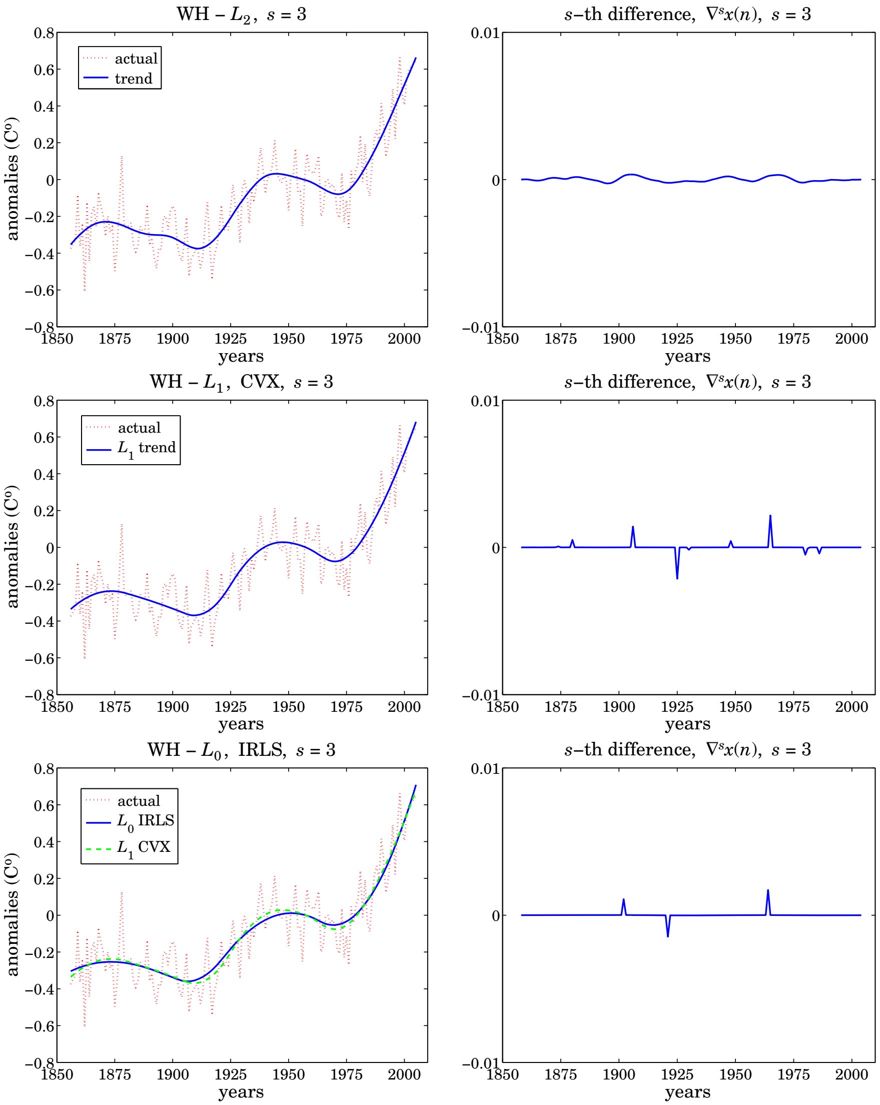


Fig. 26.8.17 Gloval warming trends, $s = 3$ cases.

In particular, *Tikhonov regularization* is defined by the following modified least-squares criterion, also known as *ridge regression*,

$$\mathcal{J} = \|\mathbf{b} - A\mathbf{x}\|^2 + \lambda\|\mathbf{x}\|^2 = \min \qquad (26.9.1)$$

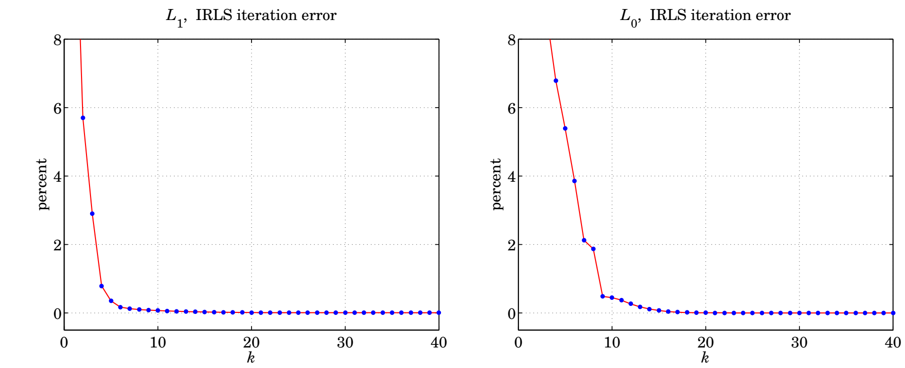


Fig. 26.8.18 Global Warming Trends – iteration errors for $s = 3$.

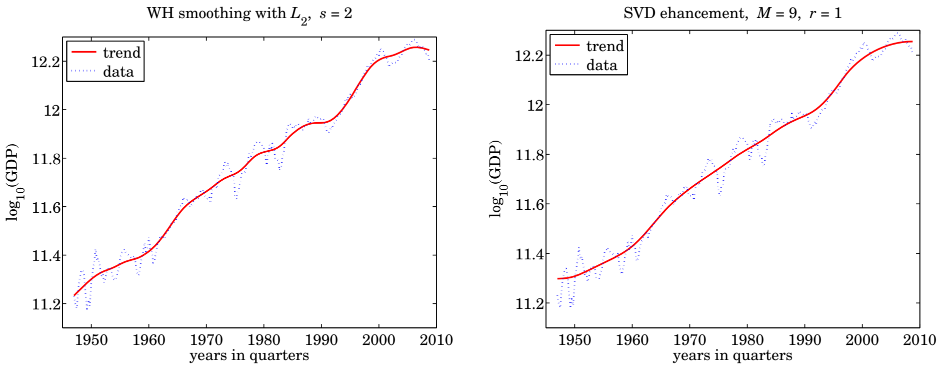


Fig. 26.8.19 US GDP example – Sec. 26.8.4.

Setting the gradient of $\mathcal{J}$ to zero, we find,

$$\frac{\partial\mathcal{J}}{\partial\mathbf{x}} = 2A^T(A\mathbf{x} - \mathbf{b}) + 2\lambda\mathbf{x} = 0 \quad \Rightarrow \quad (A^TA + \lambda I)\mathbf{x} = A^T\mathbf{b}$$

where I is the identity matrix, with solution,

$$\boxed{\mathbf{x} = (A^TA + \lambda I)^{-1}A^T\mathbf{b}}$$

Regularization is used in many practical inverse problems, such as the deblurring of images or tomography. The second term in the performance index (26.9.1) guards both against ill-conditioning and against noise in the data. If the parameter λ is chosen to be too large, it is possible that noise is removed too much at the expense of getting an accurate inverse. In large-scale inverse problems (e.g., a 512×512 image is represented by a vector $\mathbf{x}$ of dimension $512^2 = 2.6\times10^5$), the solution can be obtained iteratively, for example, using conjugate-gradients [1081–1083].

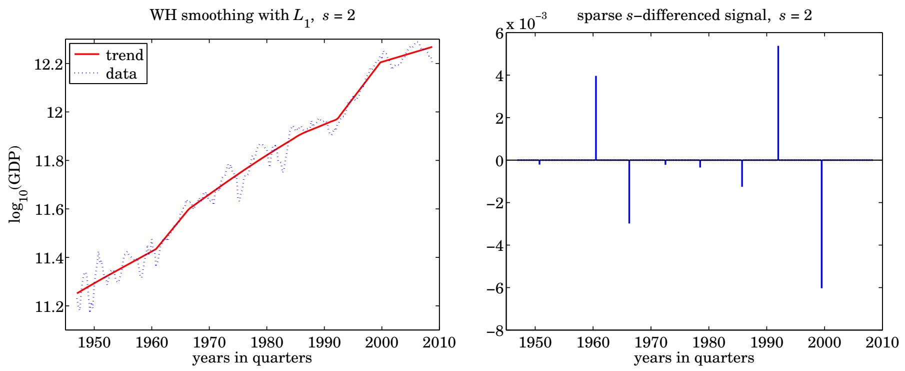


Fig. 26.8.20 US GDP example – Sec. 26.8.4.

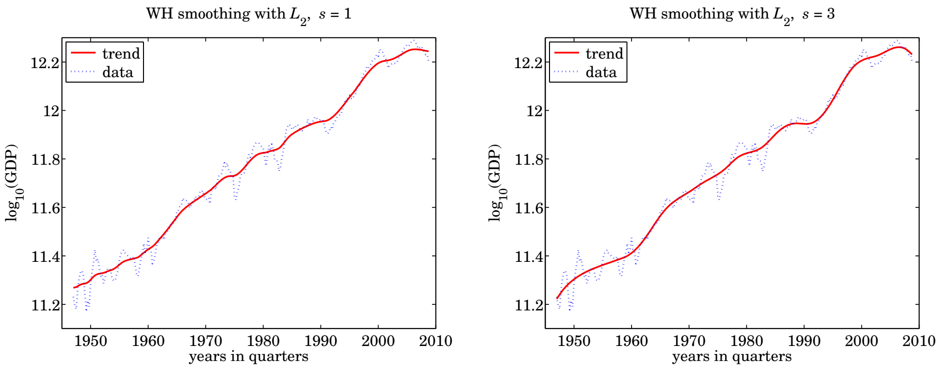


Fig. 26.8.21 US GDP example – Sec. 26.8.4.

Often, the second term, $\|\mathbf{x}\|^2$, in (26.9.1) is replaced by the more general term, $\|D\mathbf{x}\|^2 = \mathbf{x}^T D^T D\mathbf{x}$, where D is an appropriate matrix. For example, in an image restoration application, D could be chosen to be a differentiation matrix so that the performance index would attempt to preserve the sharpness of the image. The more general ridge regression performance index and its solution are:

$$\mathcal{J} = \|\mathbf{b} - A\mathbf{x}\|^2 + \lambda\|D\mathbf{x}\|^2 = \min \quad\Rightarrow\quad \boxed{\mathbf{x} = (A^TA + \lambda D^TD)^{-1}A^T\mathbf{b}} \tag{26.9.2}$$

For example, the Whittaker-Henderson case A is the identity matrix, $A = I$, and D is the s-differencing matrix D_s. Another variation of regularization is to assume a decomposition into multiple components of the form,

$$\mathbf{b} = A_1\mathbf{x}_1 + A_2\mathbf{x}_2$$

and impose different regularization constraints on each part, for example, with positive

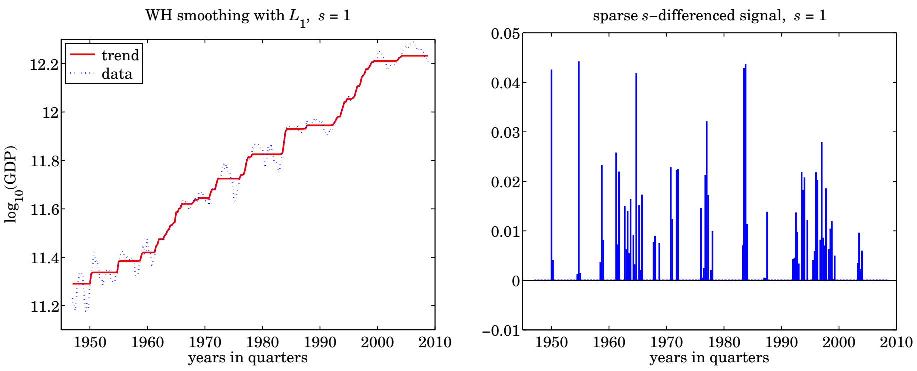


Fig. 26.8.22 US GDP example – Sec. 26.8.4.

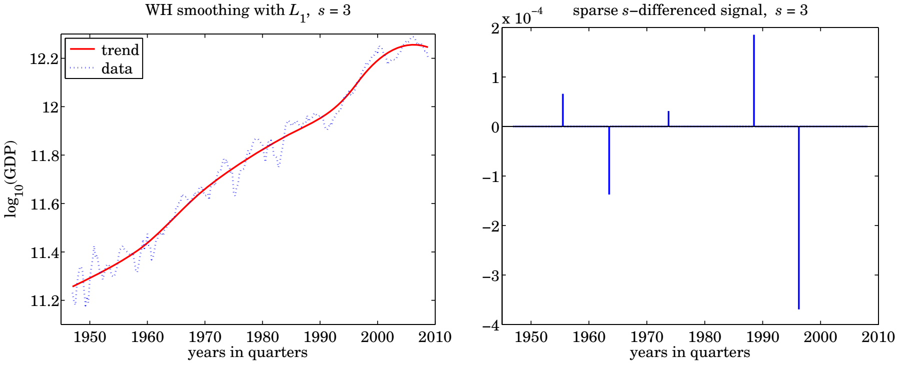


Fig. 26.8.23 US GDP example – Sec. 26.8.4.

parameters, λ_1, λ_2,

$$\mathcal{J} = \|\mathbf{b} - A_1\mathbf{x}_1 - A_2\mathbf{x}_2\|^2 + \lambda_1\|D_1\mathbf{x}_1\|^2 + \lambda_2\|D_2\mathbf{x}_2\|^2 = \min \tag{26.9.3}$$

whose minimization with respect to $\mathbf{x}_1, \mathbf{x}_2$, leads to the solution,

$$\begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} = \begin{bmatrix} A_1^TA_1 + \lambda_1 D_1^T D_1 & A_1^T A_2 \\ A_2^T A_1 & A_2^T A + \lambda_2 D_2^T D_2 \end{bmatrix}^{-1} \begin{bmatrix} A_1^T\mathbf{b} \\ A_2^T\mathbf{b} \end{bmatrix} \tag{26.9.4}$$

An example of such decomposition was the seasonal Whittaker-Henderson case discussed in Sec. 27.11 in which $\mathbf{x}_1$ represented the seasonal component, and $\mathbf{x}_2$, the trend. Another variation, similar to the *elastic net* [1146], is one in which one or both of the L_2 regularizing terms are replaced by their L_1 sparse versions, for example, $\|D_1\mathbf{x}_1\|_1$.

Sparse versions of the conventional regularized least-squares criteria are obtained by replacing the L_2-norm in the regularization term of Eq. (26.9.2) by the L_p norm, resulting in the alternative minimization criterion, referred to as L_p*-regularized least-squares*,

$$\boxed{\mathcal{J} = ||\mathbf{b} - A\mathbf{x}||_2^2 + \lambda||D\mathbf{x}||_p^p = \min} \qquad (L_p\text{-regularized least-squares}) \qquad (26.9.5)$$

where the first term in (26.9.5) is still the L_2 norm of the modeling error, $\mathbf{b} - A\mathbf{x}$, and $\|\mathbf{x}\|_p$ denotes the L_p norm of the vector $\mathbf{x} = [x_1, x_2, \cdots, x_M]^T$,

$$\|\mathbf{x}\|_p = \left[\sum_{n=1}^{M} |x_n|^p\right]^{\frac{1}{p}} \quad \Rightarrow \quad \|\mathbf{x}\|_p^p = \sum_{n=1}^{M} |x_n|^p$$

As mentioned before, even though $\|\mathbf{x}\|_p$ is a proper norm only for $p \geq 1$, the cases $0 \leq p \leq 1$ have also been considered widely because they promote the sparsity of the resulting solution vector $\mathbf{x}$, or rather, the sparsity of the vector, $D\mathbf{x}$, that appears inside the L_p norm in (26.9.5). In particular, the case $p = 1$ is special for the following reasons:

(a) it corresponds to the smallest possible proper norm,
(b) it typically results in a sparse solution, which under many circumstances is close to, or coincides with, the sparsest solution (based on the L_0 norm), and,
(c) the minimization problem (26.9.5) is a convex optimization problem for which there are efficient numerical methods.

We concentrate below on the three cases $p = 0, 1, 2$, and also set $D = I$ for now, and consider the following three optimization criteria for solving the linear system $A\mathbf{x} = \mathbf{b}$, with $A \in \mathbb{R}^{N\times M}$, $\mathbf{b} \in \mathbb{R}^N$, and, $\mathbf{x} \in \mathbb{R}^M$,

$$\begin{aligned}
(L_0):&\quad \mathcal{J} = \|\mathbf{b} - A\mathbf{x}\|_2^2 + \lambda\, \|\mathbf{x}\|_0 = \min \\
(L_1):&\quad \mathcal{J} = \|\mathbf{b} - A\mathbf{x}\|_2^2 + \lambda\, \|\mathbf{x}\|_1 = \min \qquad (26.9.6) \\
(L_2):&\quad \mathcal{J} = \|\mathbf{b} - A\mathbf{x}\|_2^2 + \lambda\, \|\mathbf{x}\|_2^2 = \min
\end{aligned}$$

where the L_0 norm, $\|\mathbf{x}\|_0$, is the cardinality of the vector $\mathbf{x}$, that is, the number of its non-zero entries. The criteria try to minimize the corresponding norm of $\mathbf{x}$ while being consistent with the given linear system. Criterion (L_0) results in the sparsest solution but is essentially intractable. Criterion (L_1) is used as an alternative to (L_0) and results also in a sparse solution. It is known as the *LASSO*[†] [1137], or as *basis pursuit denoising* (BPDN) [1140], or as L_1*-regularized least squares.*

There is a vast literature on the properties, applications, and numerical methods of the above criteria. A small and incomplete set of references is [1084–1124,1126–1195]. A comprehensive review is [1163]. Several MATLAB-based packages are also available [1196].

Below we discuss two examples that illustrate the sparsity of the resulting solutions: (i) an overdetermined sparse spike deconvolution problem, and (ii) an underdetermined sparse signal recovery example. In these examples, the (L_0) problem is solved with an iteratively re-weighted least-squares (IRLS) method, and the (L_1) problem, with the CVX package[‡] as well as with the IRLS method for comparison.

[†] Least Absolute Shrinkage and Selection Operator

[‡] `http://cvxr.com/cvx/`

We introduced the IRLS method in the context of sparse Whittaker-Henderson smoothing, or, L_1-trend filtering, in Sec. 26.7. There are several variants of this method, [1126–1134,1138,1159,1163,1166,1171,1172], but the basic idea is to replace the L_p norm with a weighted L_2 norm, which can be solved iteratively. We recall from Sec. 26.7 that given a real number $0 \leq p \leq 2$, set $q = 2 - p$, and write for any real number $x \neq 0$,

$$|x|^p = \frac{|x|^2}{|x|^q} \approx \frac{|x|^2}{|x|^q + \varepsilon}$$

where ε is a sufficiently small positive number needed to also allow the case $x = 0$. Similarly, we can write for the L_p-norm of a vector $\mathbf{x} \in \mathbb{R}^M$,

$$\begin{aligned}
\|\mathbf{x}\|_p^p &= \sum_{i=1}^{M} |x_i|^p \approx \sum_{i=1}^{M} \frac{|x_i|^2}{|x_i|^q + \varepsilon} = \mathbf{x}^T W(\mathbf{x})\mathbf{x} \\
W(\mathbf{x}) &= \text{diag}\left[\frac{1}{|\mathbf{x}|^q + \varepsilon}\right] = \text{diag}\left[\frac{1}{|x_1|^q + \varepsilon}, \frac{1}{|x_2|^q + \varepsilon}, \ldots, \frac{1}{|x_M|^q + \varepsilon}\right]
\end{aligned} \qquad (26.9.7)$$

Alternatively, one can define $W(\mathbf{x})$ as the pseudo-inverse of the diagonal matrix of the powers $|x_i|^q$, $i = 1, 2, \ldots, M$, that is, in MATLAB language,[*]

$$W(\mathbf{x}) = \text{pinv}\Big(\text{diag}\big[|x_1|^q,\ |x_2|^q,\ \ldots,\ |x_M|^q\big]\Big) \qquad (26.9.8)$$

Then, the L_p-regularized least-squares problem can be written in the form,

$$\mathcal{J} = \|\mathbf{b} - A\mathbf{x}\|_2^2 + \lambda\|\mathbf{x}\|_p^p = \|\mathbf{b} - A\mathbf{x}\|_2^2 + \lambda\, \mathbf{x}^T W(\mathbf{x})\, \mathbf{x} = \min \qquad (26.9.9)$$

This approximation leads to the following iterative solution in which the diagonal weighting matrix W to be used in the next iteration is replaced by its value from the previous iteration,

$$\boxed{\begin{aligned}
&\text{for } k = 1, 2, \ldots, K, \text{ do:} \\
&\quad W_{k-1} = W(\mathbf{x}^{(k-1)}) \\
&\quad \mathbf{x}^{(k)} = \arg\min_{\mathbf{x}} \|\mathbf{b} - A\mathbf{x}\|_2^2 + \lambda\, \mathbf{x}^T W_{k-1}\mathbf{x}
\end{aligned}} \qquad (\text{IRLS}) \qquad (26.9.10)$$

with the algorithm initialized to the ordinary least-squares solution of criterion (L_2):

$$\mathbf{x}^{(0)} = (\lambda\, I + A^T A)^{-1} A^T \mathbf{b}$$

The solution of the optimization problem in (26.9.10) at the kth step is:

$$\mathbf{x}^{(k)} = (\lambda\, W_{k-1} + A^T A)^{-1} A^T \mathbf{b}$$

Thus, the choices $p = 0$ and $p = 1$ should resemble the solutions of the L_0 and L_1 regularized problems. The IRLS algorithm (26.9.10) works well for moderate-sized problems ($N, M < 1000$). For large-scale problems ($N, M > 10^6$), the successive least-squares problems could be solved with more efficient methods, such as conjugate gradients.

[*] e.g., `pinv(diag([2, 0, 4]))` produces the diagonal matrix, `diag([0.50, 0, 0.25])`.

The general case of (26.9.5) that includes the smoothness-constraining matrix D can also be handled in the same way. Following the discussion of Sec. 26.7, we can write,

$$\mathcal{J} = ||\mathbf{b} - A\mathbf{x}||_2^2 + \lambda||D\mathbf{x}||_p^p = ||\mathbf{b} - A\mathbf{x}||_2^2 + \lambda\,\mathbf{x}^T D^T W(D\mathbf{x})\,D\mathbf{x} = \min \qquad (26.9.11)$$

which leads to to the following iterative algorithm,

$$\boxed{\begin{aligned} &\text{for } k = 1, 2, \dots, K, \text{ do:} \\ &\quad W_{k-1} = W(D\mathbf{x}^{(k-1)}) \\ &\quad \mathbf{x}^{(k)} = \arg\min_{\mathbf{x}} ||\mathbf{b} - A\mathbf{x}||_2^2 + \lambda\,\mathbf{x}^T D^T W_{k-1} D\mathbf{x} \end{aligned}} \qquad \text{(IRLS)} \qquad (26.9.12)$$

with the algorithm initialized to the ordinary least-squares solution:

$$\mathbf{x}^{(0)} = (A^T A + \lambda\,D^T D)^{-1} A^T \mathbf{b}$$

The solution of the optimization problem at the kth step is:

$$\mathbf{x}^{(k)} = (A^T A + \lambda\,D^T W_{k-1} D)^{-1} A^T \mathbf{b}$$

26.9.1 Sparse Spike Deconvolution Example

Consider a deconvolution problem in which the observed signal y_n is the noisy convolution, $y_n = h_n * s_n + v_n$, where v_n is zero-mean white noise of variance σ_v^2. The objective is to recover the signal s_n assuming knowledge of the filter h_n. For an FIR filter of order M and input of length L, the output will have length $N = L + M$, and we may cast the above convolutional filtering equation in the matrix form:

$$\mathbf{y} = H\mathbf{s} + \mathbf{v}$$

where $\mathbf{y}, \mathbf{v} \in \mathbb{R}^N$, $\mathbf{s} \in \mathbb{R}^L$, and H is the $N{\times}L$ convolution matrix corresponding to the filter. It can be constructed as a sparse matrix by the function:

```
H = convmat(h,L);    % H = convmtx(h,N) = non-sparse version
```

The filter is taken to be:

$$h_n = \cos\big(0.15(n - n_0)\big)\exp\big(-0.004(n - n_0)^2\big), \qquad n = 0, 1, \dots, M$$

where $M = 53$ and $n_0 = 25$. The input is a sparse spike train consisting of S spikes:

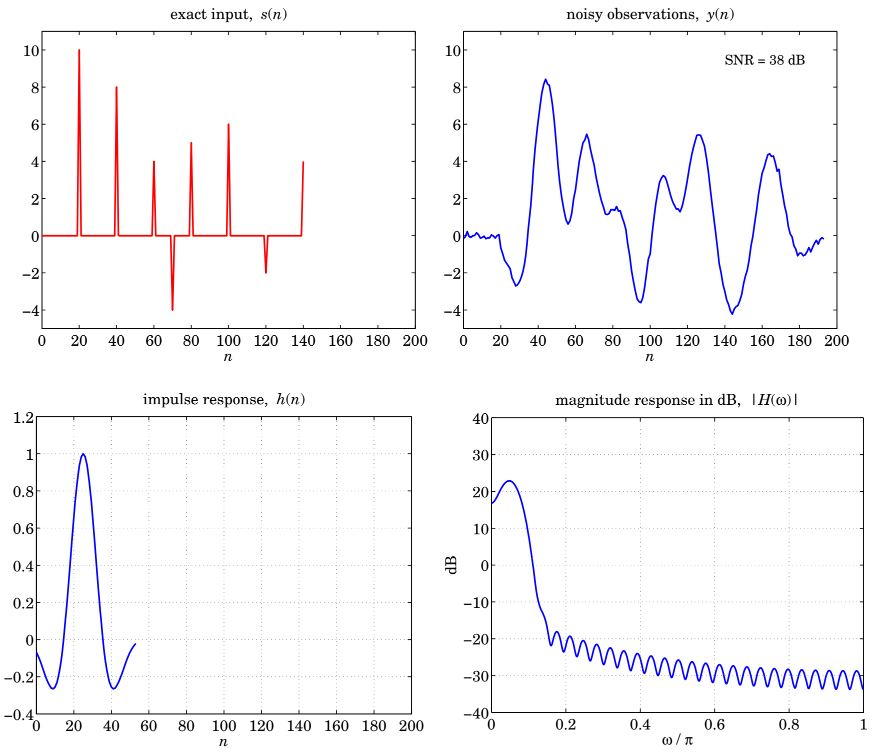
$$s_n = \sum_{i=1}^{S} a_i \delta(n - n_i), \qquad n = 0, 1, \dots, L - 1 \qquad (26.9.13)$$

where $S = 8$ and the spike locations and amplitudes are given as follows:

$$n_i = [20,\ 40,\ 60,\ 70,\ 80,\ 100,\ 120,\ 140], \qquad a_i = [10,\ 8,\ 4,\ -4,\ 5,\ 6,\ -2,\ 4]$$

The input signal length is defined from the last spike location to be $L = n_8 + 1 = 141$. The noise standard deviation is chosen to be $\sigma_v = 0.1$, which corresponds to approximately 38 dB signal-to-noise ratio, that is, $SNR = 20\log_{10}(\max|H\mathbf{s}|/\sigma_v) = 38$.

The input signal s_n and the convolved noisy signal y_n are shown below. Also shown are the impulse response h_n and the corresponding magnitude response $|H(\omega)|$ plotted in dB versus $0 \le \omega \le \pi$ rads/sample.


exact input, $s(n)$

noisy observations, $y(n)$

impulse response, $h(n)$

magnitude response in dB, $|H(\omega)|$

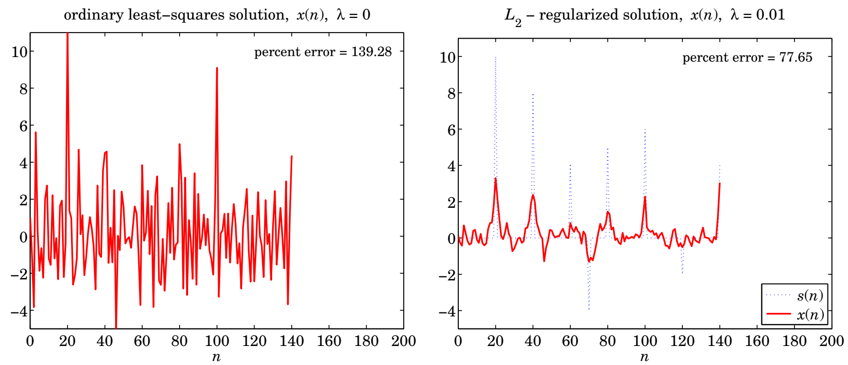
We note that $H(\omega)$ occupies a low frequency band, thus, we expect the effective deconvolution inverse filtering operation by $1/H(\omega)$ to be very sensitive to even small amounts of noise in y_n, even though the noise is barely visible in y_n itself. The three criteria of Eq. (26.9.6) to be implemented are,

$$\begin{aligned} \mathcal{J} &= \|\mathbf{y} - H\mathbf{x}\|_2^2 + \lambda\,\|\mathbf{x}\|_0 = \min \\ \mathcal{J} &= \|\mathbf{y} - H\mathbf{x}\|_2^2 + \lambda\,\|\mathbf{x}\|_1 = \min \\ \mathcal{J} &= \|\mathbf{y} - H\mathbf{x}\|_2^2 + \lambda\,\|\mathbf{x}\|_2^2 = \min \end{aligned} \qquad (26.9.14)$$

The L_2 case with $\lambda = 0$ corresponds to the ordinary (full-rank overdetermined) least-squares solution of the linear system, $H\mathbf{x} = \mathbf{y}$, that is, $\mathbf{x}_{\text{ord}} = (H^T H)^{-1} H^T \mathbf{y}$, or, $\mathbf{x}_{\text{ord}} = H\backslash\mathbf{y}$, in MATLAB.

Similarly, the L_2-regularized solution with non-zero λ is, $\mathbf{x}_2 = (\lambda I + H^T H)^{-1} H^T \mathbf{y}$.

These two solutions are depicted below, displaying also the percent error of recovering the desired signal $\mathbf{s}$, defined in terms of the L_2 norms by, $P_{\text{error}} = 100 \cdot \|\mathbf{x}-\mathbf{s}\|_2 \,/\, \|\mathbf{s}\|_2$.



The MATLAB code for generating the above six graphs was as follows:

```
g = @(x) cos(0.15*x).*exp(-0.004*x.^2);     % filter function
delta = @(x) (x==0);

M = 53; n0 = 25; k = (0:M)'; h = g(k-n0);   % filter h(n)

ni = [20  40  60  70  80  100  120  140];   % spike locations & amplitudes
ai = [10  8   4   -4  5    6    -2   4 ];

L = ni(end)+1; N = M+L;                 % L = 141, N = 194
n = (0:L-1)'; t = (0:N-1)';             % time indices for s(n) and y(n)

s = 0;
for i=1:length(ni),                     % exact input s(n)
  s = s + ai(i) * delta(n-ni(i));
end

H = convmat(h,L);                       % NxL=194x141 convolution matrix

sigma = 0.1;
seed = 2017; randn('state',seed);       % initialize generator

y = H*s + sigma * randn(N,1);           % noisy observations y(n)

w = linspace(0,1,1001)*pi;              % frequencies in rads/sample
Hmag = 20*log10(abs(dtft(h,w)));        % can also use freqz(h,1,w)

xord = H\y;                             % ordinary least-squares
Perr = 100 * norm(s-xord)/norm(s);

la = 0.01;
x2 = (la * eye(L) + H'*H) \ (H'*y);     % L2-regularized
Perr = 100 * norm(s-x2)/norm(s);

figure; plot(n,s);    figure; plot(t,y);         % plot s(n) and y(n)
figure; plot(k,h);    figure; plot(w/pi,Hmag);   % plot h(n) and H(w)
figure; plot(n,xord); figure; plot(n,x2);        % plot xord(n) and x2(n)
```

As expected from the lowpass nature of $H(\omega)$, the ordinary least-squares solution is too noisy to be useful, while the regularized one is only slightly better. The effect of increasing λ is to smooth the noise further, but at the expense of flattening and broadening the true spikes (for example, try the value, $\lambda = 0.1$).

To understand this behavior from the frequency point of view, let us pretend that the signals y_n, x_n are infinitely long. Following the approach of Sec. 26.3, we may replace the (L_2) criterion in Eq. (26.9.14) by the following,

$$\begin{aligned}\mathcal{J} &= \sum_{n=-\infty}^{\infty} \big|y_n - h_n * x_n\big|^2 + \lambda \sum_{n=-\infty}^{\infty} \big|x_n\big|^2 = \\ &= \int_{-\pi}^{\pi} \big|Y(\omega) - H(\omega)X(\omega)\big|^2 \frac{d\omega}{2\pi} + \lambda \int_{-\pi}^{\pi} \big|X(\omega)\big|^2 \frac{d\omega}{2\pi} = \min\end{aligned} \tag{26.9.15}$$

where we used Parseval's identity. The vanishing of the functional derivative of $\mathcal{J}$ with respect to $X^*(\omega)$, then leads to the following regularized inverse filtering solution,

$$\frac{\delta\mathcal{J}}{\delta X^*(\omega)} = \big|H(\omega)\big|^2 X(\omega) - H^*(\omega)Y(\omega) + \lambda X(\omega) = 0\,, \quad \text{or,} \tag{26.9.16}$$

$$\boxed{X(\omega) = \frac{H^*(\omega)}{\lambda + \big|H(\omega)\big|^2}\,Y(\omega)} \quad \text{(regularized inverse filter)} \tag{26.9.17}$$

If we express $Y(\omega)$ in terms of the spectrum $S(\omega)$ of the desired signal and the spectrum $V(\omega)$ of the added noise, then, Eq. (26.9.17) leads to,

$$Y(\omega) = H(\omega)S(\omega) + V(\omega) \quad \Rightarrow \quad X(\omega) = \frac{\big|H(\omega)\big|^2}{\lambda + \big|H(\omega)\big|^2}\,S(\omega) + \frac{H^*(\omega)}{\lambda + \big|H(\omega)\big|^2}\,V(\omega)$$

For $\lambda = 0$, this becomes the ordinary inverse filter,

$$X(\omega) = \frac{1}{H(\omega)}\,Y(\omega) = S(\omega) + \frac{1}{H(\omega)}\,V(\omega)$$

which, although it recovers the $S(\omega)$ term, it greatly amplifies the portions of the white-noise spectrum that lie in the stopband of the filter, that is where, $H(\omega) \approx 0$. For $\lambda \neq 0$ on the other hand, the regularization filter acts as a lowpass filter, becoming vanishingly small over the stopband, and hence removing some of the noise, but also smoothing and broadening the spikes for the same reason, that is, removing some of the high-frequencies in $S(\omega)$.

By contrast, the L_0 and L_1 regularized criteria of Eq. (26.9.14) behave dramatically differently and are capable of accurately extracting the input spikes, as seen in the graphs of Fig. 26.9.1.

The L_1 case was computed with the CVX package, as well as with the IRLS algorithm of Eq. (26.9.10), with the parameter values, $\lambda = 0.1$, $p = 1$, $q = 1$, $\epsilon = 10^{-5}$, and $K = 100$ iterations.

The L_0 case was computed with the IRLS algorithm using parameters, $\lambda = 0.1$, $p = 0$, $q = 2$, $\epsilon = 10^{-5}$, and $K = 100$ iterations—however, it actually converges within about

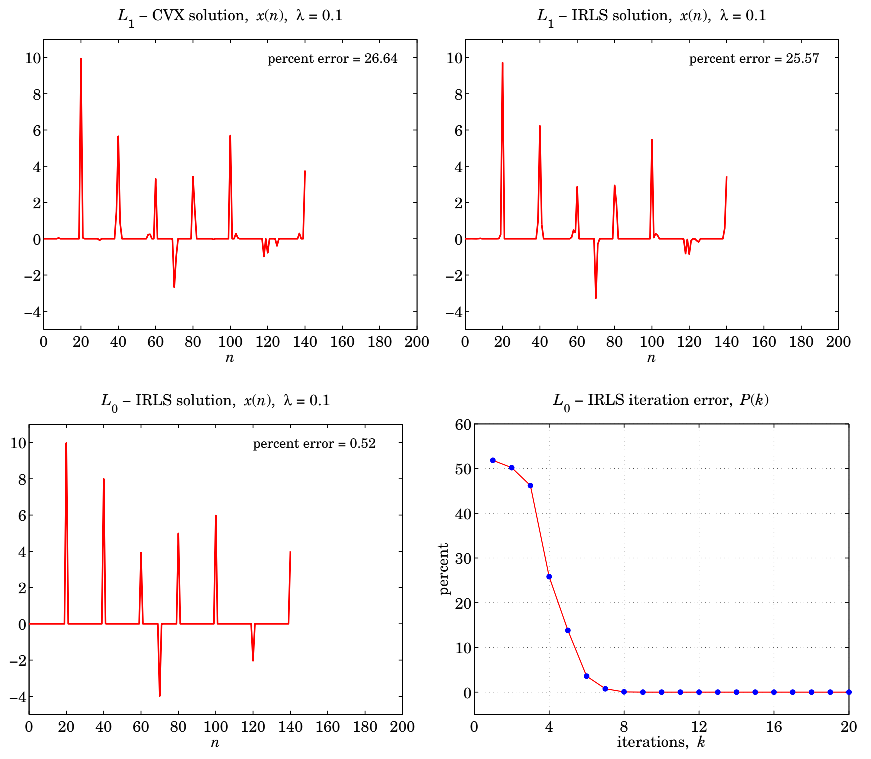


Fig. 26.9.1 Deconvolved signals based on the L_1 and L_0 criteria.

10 iterations as seen in the bottom-right graph that plots the iteration percentage error defined at the kth iteration by, $P(k) = 100 \cdot \|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\|_2 \,/\, \|\mathbf{x}^{(k-1)}\|_2$.

The recovered signal in the L_0 case is slightly sparser than that of the L_1 case, as is seen in the figures, or by evaluating the reconstruction error, $P_{\text{error}} = 100 \cdot \|\mathbf{x} - \mathbf{s}\|_2 \,/\, \|\mathbf{s}\|_2$, but both versions fairly accurately extract the spike amplitudes and locations. The MATLAB code used to produce these four graphs was as follows.

```
la = 0.1;

cvx_begin                                   % L1 case - CVX solution
   variable x(L)
   minimize( sum_square(H*x-y) + la * norm(x,1) )
cvx_end

Perr = 100*norm(x-s)/norm(s);               % reconstruction error

figure; plot(n,x,'r-');                     % plot L1 - CVX version

% -------------------------------------------------------------------
```

```
p=1; q=2-p; epsilon=1e-5; K=100;            % L1 case - IRLS solution
W = speye(L);

x0 = (la * W + H'*H) \ (H'*y);

for k=1:K,
   W = diag(1./(abs(x0).^q + epsilon));
   x = (la * W + H'*H) \ (H'*y);
   P(k) = norm(x-x0)*100/norm(x0);
   x0 = x;
end

Perr = 100*norm(x-s)/norm(s);               % reconstruction error

figure; plot(n,x,'r-');                     % plot L1 - IRLS version

% -------------------------------------------------------------------

p=0; q=2-p; epsilon=1e-5; K=100;            % L0 case - IRLS solution
W = speye(L);

x0 = (la * W + H'*H) \ (H'*y);              % initialize iteration

for k=1:K,                                  % IRLS iteration
   W = diag(1./(abs(x0).^q + epsilon));
   x = (la * W + H'*H) \ (H'*y);
   P(k) = 100*norm(x-x0)/norm(x0);          % iteration error
   x0 = x;
end

Perr = 100*norm(x-s)/norm(s);               % reconstruction error

figure; plot(n,x,'r-');                     % plot L0 - IRLS version
k = 1:K;
figure; plot(k,P,'r-', k,P,'b.');           % plot iteration error P(k)
```

26.9.2 Sparse Signal Recovery Example

In this example, based on [1150], we consider the underdetermined noisy linear system:

$$\mathbf{y} = A\mathbf{s} + \mathbf{v}$$

where $A \in \mathbb{R}^{1000\times 2000}$, $\mathbf{s} \in \mathbb{R}^{2000}$, and $\mathbf{y}, \mathbf{v} \in \mathbb{R}^{1000}$. The matrix A has full rank and consists of zero-mean, unit-variance, gaussian, independent random entries, and the 2000-long input signal $\mathbf{s}$ is sparse with only $L = 100$ non-zero entries taken to be randomly positioned within its length, and constructed to have amplitudes ±1 with random signs and then weighted by a triangular window in order to get a variety of values.

The noise $\mathbf{v}$ is zero-mean gaussian white noise with standard deviation $\sigma_v = 0.1$. The recovery criteria are as in Eq. (26.9.6),

$$\begin{aligned}
\mathcal{J} &= \|\mathbf{y} - A\mathbf{x}\|_2^2 + \lambda\, \|\mathbf{x}\|_0 = \min \\
\mathcal{J} &= \|\mathbf{y} - A\mathbf{x}\|_2^2 + \lambda\, \|\mathbf{x}\|_1 = \min \qquad (26.9.18) \\
\mathcal{J} &= \|\mathbf{y} - A\mathbf{x}\|_2^2 + \lambda\, \|\mathbf{x}\|_2^2 = \min
\end{aligned}$$

Fig. 26.9.2 shows the signal $s(n)$ and the observations $y(n)$, as well as the recovered signals $x(n)$ based on the above criteria. The L_1 solution was computed with the CVX package and the IRLS algorithm, and the L_0 solution, with the IRLS algorithm. The parameter λ was chosen to be $\lambda = 0.1$ in the L_1 and L_0 cases, and $\lambda = 0$ for the L_2 case, which corresponds to the usual minimum-norm solution of the underdetermined linear system $A\mathbf{x} = \mathbf{y}$, that is, $\mathbf{x} = A^+\mathbf{y} = A^T(AA^T)^{-1}\mathbf{y}$, in terms of the pseudo-inverse of A. Note that using $\lambda = 0.1$ in the L_2 case is virtually indistinguishable from the $\lambda = 0$ case.

The L_2 criterion does not produce an acceptable solution. But both the L_1 and the L_0 criteria accurately recover the sparse signal $s(n)$, with the L_0 solution being somewhat sparser and resulting in smaller recovery error, $P_{\text{error}} = 100\cdot\|\mathbf{x}-\mathbf{s}\|_2\,/\,\|\mathbf{s}\|_2$.

The IRLS algorithms were run with parameters $\lambda = 0.1$, $\epsilon = 10^{-6}$, and $K = 20$ iterations. The successive iteration percentage errors, $P(k) = 100\cdot\|\mathbf{x}^{(k)}-\mathbf{x}^{(k-1)}\|_2\,/\,\|\mathbf{x}^{(k-1)}\|_2$, are plotted versus k in Fig. 26.9.3 for the L_1 and L_0 cases. The MATLAB code used to produce the solutions and graphs is given below.

```
N = 1000; M = 2000; L = 100;               % L-sparse

seed = 1000;                               % initialize generators
randn('state',seed);
rand('state',seed);

A = randn(N,M);                            % random NxM matrix
s = zeros(M,1);

I = randperm(M); I = I(1:L);               % L random indices in 1:M
s(I) = sign(randn(L,1));                   % L random signs at locations I

t = (0:N-1)'; n = (0:M-1)';
w = 1 - abs(2*n-M+1)/(M-1);                % triangular window
s = s .* w;                                % L-sparse windowed input

sigma = 0.1;
v = sigma * randn(N,1);
y = A*s + v;                               % noisy observations

SNR = 20*log10(norm(A*s,Inf)/sigma);       % SNR = 45 dB

figure; stem(n,s); figure; stem(t,y);      % plot s(n) and y(n)

% -----------------------------------

rank(A);                                   % verify full rank = 1000

x = pinv(A)*y;                             % L2 - minimum-norm solution

Perr = 100 * norm(x-s)/norm(s);            % reconstruction error = 71.21 %

figure; stem(n,x,'r-');

% -----------------------------------

la = 0.1;

cvx_begin                                  % L1 - CVX solution
   variable x(M)
```

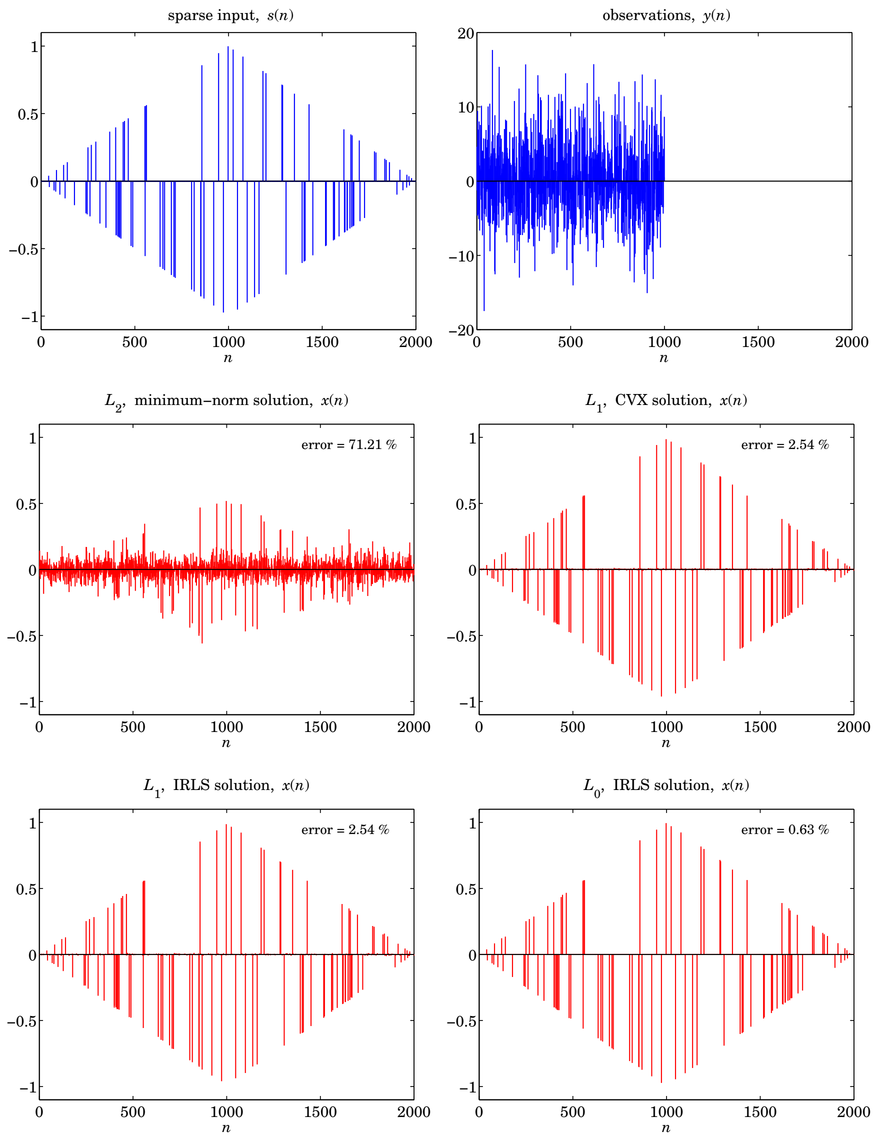


Fig. 26.9.2 Recovered signals based on the L_2, L_1, and L_0 criteria.

```
   minimize( sum_square(A*x-y) + la * norm(x,1) )
cvx_end

Perr = 100 * norm(x-s)/norm(s);            % reconstruction error = 2.54 %
```

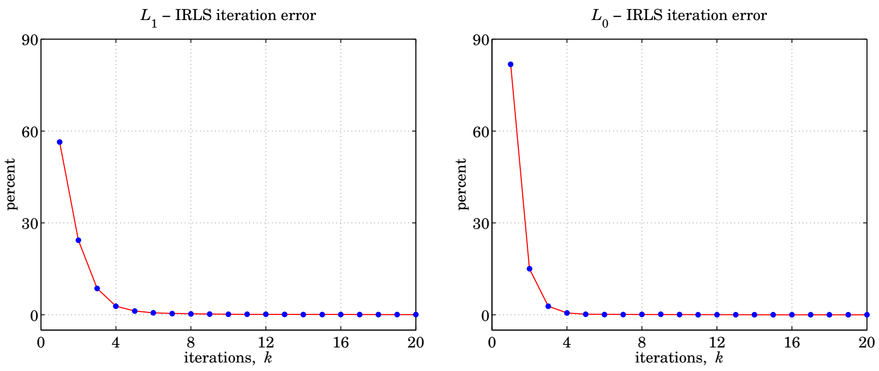


Fig. 26.9.3 IRLS iteration error based on the L_1, and L_0 criteria.

```
figure; stem(n,x,'r-');

% ----------------------------------

p = 1; q = 2-p; epsilon = 1e-6; K = 20;
W = speye(M);

ATA = A'*A;
ATy = A'*y;

x0 = (la*W + ATA) \ ATy;

for k=1:K,                                  % L1 - IRLS solution
   W = diag(1./(abs(x0).^q + epsilon));
   x = (la*W + ATA) \ ATy;
   P(k) = norm(x-x0)*100/norm(x0);
   x0 = x;
end

Perr = 100 * norm(x-s)/norm(s);             % reconstruction error = 2.54 %

figure; stem(n,x,'r-');

k = 1:K;
figure; plot(k,P,'r-', k,P,'b.');           % iteration error

% ----------------------------------

p = 0; q = 2-p; epsilon = 1e-6; K = 20;
W = speye(M);

x0 = (la*W + ATA) \ ATy;

for k=1:K,                                  % L0 - IRLS solution
   W = diag(1./(abs(x0).^q + epsilon));
   x = (la*W + ATA) \ ATy;
```

```
   P(k) = norm(x-x0)*100/norm(x0);
   x0 = x;
end

Perr = 100 * norm(x-s)/norm(s);             % reconstruction error = 0.63 %

figure; stem(n,x,'r-');

k = 1:K;
figure; plot(k,P,'r-', k,P,'b.');           % iteration error
```

26.10 Problems

26.1 For the case $s = 1$, show that the Whittaker-Henderson filter has poles z_1, $1/z_1$, where

$$z_1 = e^{-\alpha}\,, \quad \alpha = 2\,\text{asinh}\left(\frac{1}{2\sqrt{\lambda}}\right)$$

For the case $s = 2$, show that the filter has poles $\{z_1, z_1^*, 1/z_1, 1/z_1^*\}$, where

$$z_1 = \left(\sqrt{1-ja^2} + jae^{j\pi/4}\right)^2 = \frac{1}{2}D^2\left(1-\frac{a}{D}\right)^2\left(1+j\frac{a}{D}\right)^2\,, \quad D = \sqrt{1+\sqrt{1+a^4}}\,, \quad a = \frac{1}{2\lambda^{1/4}}$$

Show that in both cases $|z_1| < 1$.

26.2 Determine explicit expressions in terms of λ for the quantities σ^2 and z_1 that appear in the factorization of the denominator of the Hodrick-Prescott filter:

$$1+\lambda(1-z^{-1})^2(1-z)^2 = \sigma^2(1-z_1z^{-1})(1-z_1^*z^{-1})(1-z_1z)(1-z_1^*z)$$

What are the numerical values of σ^2, z_1 for $\lambda = 1600$? What are the values of the coefficients of the second-order filter $\left(1-2\,\text{Re}(z_1)z^{-1} + |z_1|^2z^{-2}\right)$?

26.3 Consider the performance index (26.6.1) for a regularized deconvolution problem. Making enough assumptions, show that the performance index can be written in terms of frequency responses as follows,

$$\begin{aligned}\mathcal{J} &= \sum_n |y_n - f_n * x_n|^2 + \lambda\sum_n |d_n * x_n|^2 \\ &= \int_{-\pi}^{\pi}\left[\,\big|Y(\omega)-F(\omega)X(\omega)\big|^2 + \lambda\,\big|D(\omega)X(\omega)\big|^2\right]\frac{d\omega}{2\pi} = \min\end{aligned}$$

Determine the optimum $X(\omega)$ that minimizes this index. Then, show that the corresponding optimum deconvolution filter $H(z) = X(z)/Y(z)$ is given by:

$$H(z) = \frac{F(z^{-1})}{F(z)F(z^{-1}) + \lambda D(z)D(z^{-1})}$$

What would be the stochastic state-space model for x_n, y_n that has this $H(z)$ as its optimum double-sided (unrealizable) Wiener filter for estimating x_n from y_n?

27

Periodic Signal Extraction

27.1 Introduction

Many physical, financial, and social time series have a natural periodicity in them, such as daily, monthly, quarterly, yearly. The observed signal can be regarded as having three components: a periodic (or nearly periodic) seasonal part s_n, a smooth trend t_n, and a residual irregular part v_n that typically represents noise,

$$y_n = s_n + t_n + v_n$$

The model can also be assumed to be multiplicative, $y_n = s_n t_n v_n$. The signal processing task is to extract both the trend and the seasonal components, t_n and s_n, from the observed signal y_n.

For example, many climatic signals, such as CO_2 emissions, are characterized by an annual periodicity. Government agencies routinely estimate and remove the seasonal component from business and financial data and only the "seasonally-adjusted" signal $a_n = t_n + v_n$ is available, such as the US GDP that we considered in Example 26.4.1. Further processing of the deseasonalized signal a_n, using for example a trend extraction filter such as the Hodrick-Prescott filter, can reveal additional information, such as business cycles.

Periodic signals appear also in many engineering applications. Some examples are: (a) Electrocardiogram recordings are subject to power frequency interference (e.g., 60 Hz and its higher harmonics) which must be removed by appropriate filters. (b) All biomedical signals require some sort of signal processing for their enhancement. Often weak biomedical signals, such as brain signals from visual responses or muscle signals, can be evoked periodically with the responses accumulated (averaged) to enhance their SNR; (c) TV video signals have two types of periodicities in them, one due to line-scanning and one due to the frame rate. In the pre-HDTV days, the chrominance (color) TV signals were put on a subcarrier signal and added to the luminance (black & white) signal, and the composite signal was then placed on another carrier for transmission. The subcarrier's frequency was chosen carefully so as to shift the line- and frame-harmonics of the chrominance signal relative to those of the luminance so that at the receiving end the two could be separated by appropriately designed comb filters. (d) GPS signals contain a repetitive code word that repeats with a period of one millisecond. The use of comb filters can enhance their reception. (e) Radars send out repetitive pulses so that the returns from slowly moving targets have a quasi-periodic character. By accumulating these returns, the SNR can be enhanced. As we see below, signal averaging is a form of comb filtering.

In this chapter,[†] we discuss the design of comb and notch filters for extracting periodic signals or canceling periodic interference. We discuss also the specialized comb filters, referred to as "seasonal filters," that are used by standard seasonal decomposition methods, such as the census X-11 method, and others.

27.2 Notch and Comb Filters for Periodic Signals

We review briefly the notch and comb filters for extracting periodic signals, discussed in Sec. 15.11. Consider the signal plus interference model,

$$y_n = s_n + v_n$$

in which either the signal or the noise is periodic, but not both. If the noise v_n is periodic, its spectrum will be concentrated at the harmonics of some fundamental frequency, say ω_1. The noise reduction filter must be an ideal *notch filter* with notches at the harmonics $k\omega_1$, $k = 0, 1, \dots$, as shown in Fig. 27.2.1. If the filter notches are narrow, then the distortion of the desired signal s_n will be minimized.

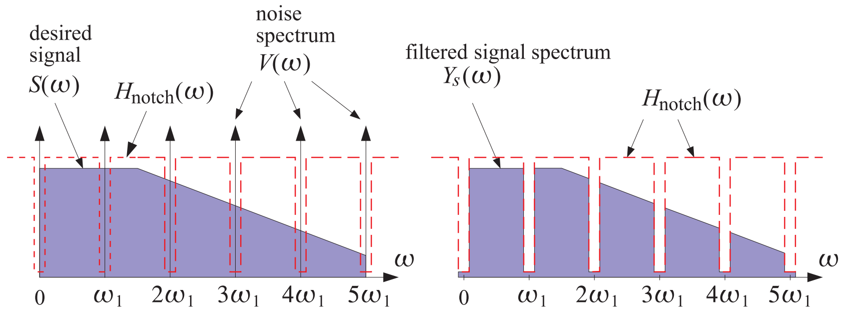


Fig. 27.2.1 Notch filter for reducing periodic interference.

On the other hand, if the desired signal s_n is periodic and the noise is a wideband signal, the signal enhancement filter for extracting s_n must be an ideal *comb filter* with peaks at the harmonics of the desired signal, as shown in Fig. 27.2.2. If the comb peaks are narrow, then only a minimal amount of noise will pass through the filter (that is, the portion of the noise whose power lies within the narrow peaks.)

A discrete-time periodic signal s_n with a period of D samples admits the following finite D-point DFT and inverse DFT representation [2] in terms of the D harmonics that lie within the Nyquist interval, $\omega_k = 2\pi k/D = k\omega_1$, for $k = 0, 1, \dots, D-1$,

[†] adapted from the author's book on *Applied Optimum Signal Processing* [45]

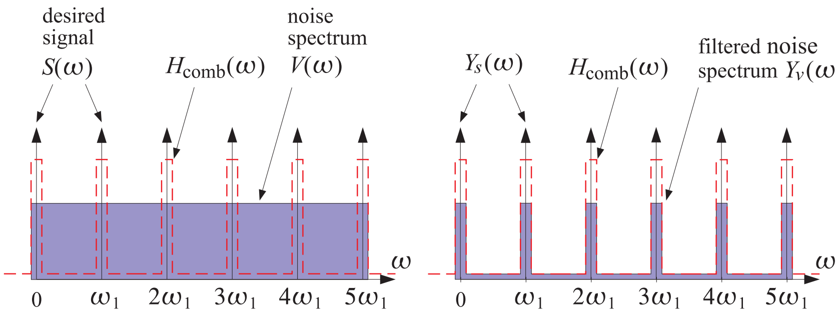


Fig. 27.2.2 Comb filter for enhancing a periodic signal.

$$
\begin{aligned}
&\text{(DFT)} \qquad S_k = \sum_{n=0}^{D-1} s_n e^{-j\omega_k n}\,, \quad k = 0,1,\dots,D-1\\
&\text{(IDFT)} \qquad s_n = \frac{1}{D}\sum_{k=0}^{D-1} S_k e^{j\omega_k n}\,, \quad n = 0,1,\dots,D-1
\end{aligned}
\tag{27.2.1}
$$

where S_k is the D-point DFT of one period $[s_0, s_1, \dots, s_{D-1}]$ of the time signal. Because of the periodicity, the IDFT formula is actually valid for all n in the interval $-\infty < n < \infty$.

We note that a periodic continuous-time signal $s(t)$ does not necessarily result into a periodic discrete-time signal when sampled at some arbitrary rate. For the sampled signal $s_n = s(nT)$ to be periodic in n with a period of D samples, where T is the sampling interval, the sampling rate $f_s = 1/T$ must be D times the fundamental harmonic f_1, that is, $f_s = Df_1$, or equivalently, one period $T_{\text{per}} = 1/f_1$ must contain D samples, $T_{\text{per}} = DT$. This implies periodicity in n,

$$s_{n+D} = s\big((n+D)T\big) = s(nT + DT) = s(nT + T_{\text{per}}) = s(nT) = s_n$$

The assumed periodicity of s_n implies that the sum of any D successive samples, $(s_n + s_{n-1} + \cdots + s_{n-D+1})$, is a constant independent of n. In fact, it is equal to the DFT component S_0 at DC $(\omega_k = 0)$,

$$s_n + s_{n-1} + \cdots + s_{n-D+1} = S_0\,, \quad -\infty < n < \infty \tag{27.2.2}$$

In a seasonal + trend model such as, $y_n = s_n + t_n + v_n$, we may be inclined to associate any DC term with the trend t_n rather with the periodic signal s_n. Therefore, it is common to assume that the DC component of s_n is absent, that is, the sum (27.2.2) is zero, $S_0 = 0$. In such cases, the comb filter for extracting s_n must be designed to have peaks only at the non-zero harmonics, $\omega_k = k\omega_1$, $k = 1,2,\dots,D-1$. Similarly, the notch filter for removing periodic noise must not have a notch at DC.

The typical technique for designing notch and comb filters for periodic signals is by frequency scaling, that is, the mapping of frequencies $\omega \to \omega D$, or equivalently, the mapping of the z-domain variable

$$z \to z^D \tag{27.2.3}$$

The effect of the transformation is to shrink the spectrum by a factor of D and then replicate it D times to fill the new Nyquist interval. An example is shown in Fig. 27.2.3 for $D = 4$. Starting with a lowpass filter $H_{\text{LP}}(\omega)$, the frequency-scaled filter will be a comb filter, $H_{\text{comb}}(\omega) = H_{\text{LP}}(\omega D)$. Similarly, a highpass filter is transformed into a notch filter $H_{\text{notch}}(\omega) = H_{\text{HP}}(\omega D)$.

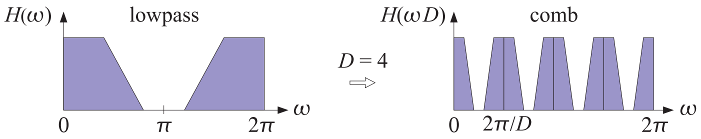


Fig. 27.2.3 Mapping of a lowpass filter to a comb filter by frequency scaling.

In the z-domain, we have the following simple prescriptions for turning lowpass and highpass filters into comb and notch filters:

$$
\boxed{
\begin{aligned}
H_{\text{comb}}(z) &= H_{\text{LP}}(z^D)\\
H_{\text{notch}}(z) &= H_{\text{HP}}(z^D)
\end{aligned}
}
\tag{27.2.4}
$$

For example, the simplest comb and notch filters are generated by,

$$
\begin{aligned}
H_{\text{LP}}(z) &= \frac{1}{2}(1 + z^{-1}) & & H_{\text{comb}}(z) = \frac{1}{2}(1 + z^{-D})\\
& & \Rightarrow & \\
H_{\text{HP}}(z) &= \frac{1}{2}(1 - z^{-1}) & & H_{\text{notch}}(z) = \frac{1}{2}(1 - z^{-D})
\end{aligned}
\tag{27.2.5}
$$

Their magnitude responses are shown in Fig. 27.2.4 for $D = 10$. The harmonics $\omega_k = 2\pi k/D = 2\pi k/10$, $k = 0,1,\dots,9$ are the peaks/notches of the comb/notch filters. The original lowpass and highpass filter responses are shown as the dashed lines. The factors $1/2$ in Eq. (27.2.5) normalize the peak gains to unity. The magnitude responses of the two filters are:

$$|H_{\text{comb}}(\omega)|^2 = \cos^2(\omega D/2)\,, \quad |H_{\text{notch}}(\omega)|^2 = \sin^2(\omega D/2) \tag{27.2.6}$$

The filters are complementary, as well as power-complementary, in the sense,

$$H_{\text{comb}}(z) + H_{\text{notch}}(z) = 1\,, \quad |H_{\text{comb}}(\omega)|^2 + |H_{\text{notch}}(\omega)|^2 = 1 \tag{27.2.7}$$

The 3-dB widths $\Delta\omega$ of the comb peaks or the notch dips are fixed by the period D. Indeed, they are defined by the condition $\sin^2(D\Delta\omega/4) = 1/2$, which gives $\Delta\omega = \pi/D$. They are indicated on Fig. 27.2.4 as the short horizontal lines at the half-power level.

In order to control the width, we must consider IIR or higher order FIR filters. For example, we may start with the lowpass filter given in Eq. (15.2.2), and its highpass version,

$$H_{\text{LP}}(z) = b\,\frac{1 + z^{-1}}{1 - az^{-1}}\,, \quad b = \frac{1-a}{2}\,, \qquad H_{\text{HP}}(z) = b\,\frac{1 - z^{-1}}{1 - az^{-1}}\,, \quad b = \frac{1+a}{2} \tag{27.2.8}$$

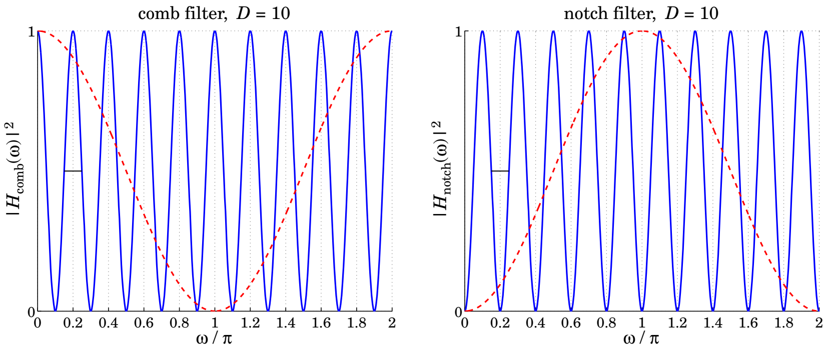


Fig. 27.2.4 Simple comb and notch filters with $D = 10$.

where $0 < a < 1$. The transformation $z \to z^D$ gives the comb and notch filters:

$$\begin{aligned} H_{\mathrm{comb}}(z) &= b\,\frac{1+z^{-D}}{1-az^{-D}}\,, \qquad b = \frac{1-a}{2} \\ H_{\mathrm{notch}}(z) &= b\,\frac{1-z^{-D}}{1-az^{-D}}\,, \qquad b = \frac{1+a}{2} \end{aligned} \tag{27.2.9}$$

The filters remain complementary, and power-complementary, with magnitude responses:

$$\begin{aligned} |H_{\mathrm{comb}}(\omega)|^2 &= \frac{\beta^2}{\beta^2 + \tan^2(\omega D/2)}\,, \qquad \beta \equiv \frac{1-a}{1+a} \\ |H_{\mathrm{notch}}(\omega)|^2 &= \frac{\tan^2(\omega D/2)}{\beta^2 + \tan^2(\omega D/2)} \end{aligned} \tag{27.2.10}$$

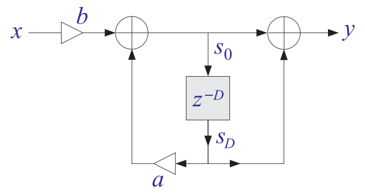
Their 3-dB width $\Delta\omega$ is controlled by the pole parameter a through the relation:

$$\tan\left(\frac{D\Delta\omega}{4}\right) = \frac{1-a}{1+a} = \beta \tag{27.2.11}$$

The noise reduction ratio of the comb filter is the same as that of the lowpass filter $H_{\mathrm{LP}}(z)$, which was calculated in Chap. 15,

$$\mathcal{R} = \frac{1-a}{2} = \frac{\beta}{1+\beta} \tag{27.2.12}$$

and can be made as small as desired by increasing a towards unity, but at the expense of also increasing the time constant of the filter. The canonical (direct-form II) realization of the comb filter and its sample processing algorithm using a circular buffer implementation of the multiple delay z^{-D} is as follows, where p is the circular pointer,



```
for each x do:
    $s_D = *(p + D)$
    $s_0 = bx + as_D$
    $y = s_0 + s_D$
    $*p = s_0$
    $p--$
```

Example 27.2.1: Fig. 27.2.5 shows two examples designed with $D = 10$ and 3-dB widths $\Delta\omega = 0.05\pi$ and $\Delta\omega = 0.01\pi$. By comparison, the simple designs had $\Delta\omega = \pi/D = 0.1\pi$. For $\Delta\omega = \pi/D$, we have $\tan(D\Delta\omega/4) = \tan(\pi/4) = 1$, which implies that $a = 0$ and $b = 1/2$, reducing to the simple designs of Eq. (27.2.5). □

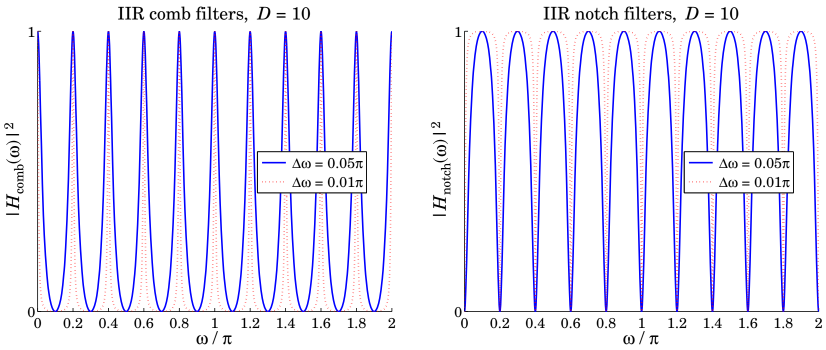


Fig. 27.2.5 Recursive comb and notch filters with $D = 10$.

Example 27.2.2: Fig. 27.2.6 shows on the left a simulated electrocardiogram (ECG) signal corrupted by 60 Hz power frequency interference and its harmonics. On the right, it shows the result of filtering by an IIR notch filter. The underlying ECG is recovered well after the initial transients die out.

The sampling rate was $f_s = 600$ Hz and the fundamental frequency of the noise, $f_1 = 60$ Hz. This gives for the period $D = f_s/f_1 = 10$. The ECG beat was taken to be 1 sec and therefore there were 600 samples in each beat for a total of 1200 samples in the two beats shown in the figure.

The IIR notch filter was designed to achieve a 3-dB width of $\Delta f = f_1/50$, that is, a Q-factor of $Q = f_1/\Delta f = 50$. Therefore, in units of rads/sample, the notch width is $\Delta\omega = 2\pi\Delta f/f_s = 2\pi/(DQ) = 0.004\pi$, which results in the filter parameters $a = 0.9391$ and $b = (1+a)/2 = 0.9695$. Thus, the designed notch filter was:

$$H_{\mathrm{notch}}(z) = 0.9695\,\frac{1-z^{-10}}{1-0.9391z^{-10}}$$

The noise was simulated by adding the following harmonic components,

$$v_n = \sum_{k=1}^{D/2-1} A_k \sin(\omega_k n)\,, \quad \text{with} \quad \omega_k = \frac{2\pi k}{D}\,, \quad A_k = \frac{1}{2k^2}$$

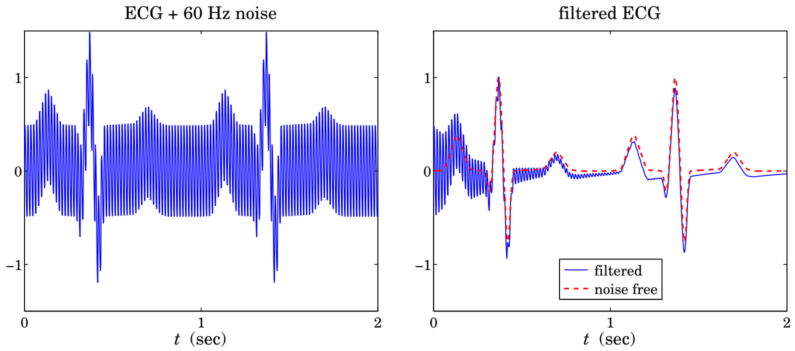


Fig. 27.2.6 Eliminating 60 Hz harmonics from ECG signal.

where the amplitudes A_k were arbitrarily chosen. Note that only the non-zero harmonics that lie in the interval $0 < \omega < \pi$ were used.

The 40-dB time-constant of the notch filter was $n_{\text{eff}} = D\,\ln(0.01)\,/\,\ln(a) = 732$ samples or equivalently, $\tau = n_{\text{eff}}/f_s = 732/600 = 1.22$ sec. It is evident from the figure that beyond this time, the transients essentially die out. The MATLAB code for generating these graphs was as follows:

```
nbeats = 2; L = 600; M = 15;                        % 600 samples per beat
s = ecgsim(nbeats,L,M);                             % simulated ECG
n = (0:length(s)-1)'; t = n/L;                      % time in seconds

D = 10;  v = 0;
for k=1:D/2-1,
   v = v + (0.5/k^2) * sin(2*pi*k*n/D);             % generate noise
end

y = s + v;                                          % noisy ECG

Q = 50; beta = tan(pi/2/Q);
a = (1-beta)/(1+beta); b = (1+a)/2;                 % filter parameters
aD = up([1,-a],D);                                  % upsampled denominator coefficients
bD = up([b,-b],D);                                  % upsampled numerator coefficients

x = filter(bD,aD,y);                                % filtered ECG

figure; plot(t,y);                                  % left graph
figure; plot(t,x, t,s,':');                         % right graph
```

The MATLAB function `ecgsim`, which is part of the AOSP toolbox [45], was used to generate the simulated ECG. It is based on the ISP function `ecg`. The function `up` is used to upsample the highpass filter's coefficient vectors by a factor of D, generating the coefficient vectors of the notch filter, so that the built-in filtering function `filter` can be used. □

The upsampling operation used in the previous example is the time-domain equivalent of the transformation $z \to z^D$ and it amounts to inserting $D-1$ zeros between any two original filter coefficients. For example, applied to the vector $[h_0, h_1, h_2, h_3]$ with $D = 4$, it generates the upsampled vector:

$$[h_0, h_1, h_2, h_3] \to [h_0, 0, 0, 0, h_1, 0, 0, 0, h_2, 0, 0, 0, h_3]$$

The function `up` implements this operation,

```
g = up(h,D);          % upsampled vector
```

It is similar to MATLAB's built-in function `upsample`, except it does not append $D-1$ zeros at the end. The difference is illustrated by the following example,

```
      up([1,2,3,4],4) = [1  0  0  0  2  0  0  0  3  0  0  0  4]
upsample([1,2,3,4],4) = [1  0  0  0  2  0  0  0  3  0  0  0  4  0  0  0]
```

In addition to enhancing periodic signals or removing periodic interference, comb and notch filters have many other applications. The transformation $z \to z^D$ is widely used in audio signal processing for the design of reverberation algorithms emulating the delays arising from reflected signals within rooms or concert halls or in other types of audio effects. The mapping $z \to z^D$ is also used in multirate signal processing applications, such as decimation or interpolation. The connection to multirate applications can be seen by writing the frequency mapping $\omega' = \omega D$ in terms of the physical frequency f in Hz and sampling rate f_s,

$$\frac{2\pi f'}{f_s'} = \frac{2\pi f}{f_s} D$$

In the signal enhancement context, the sampling rates are the same $f_s' = f_s$, but we have frequency scaling $f' = fD$. On the other hand, in multirate applications, the frequencies remain the same $f' = f$ and the above condition implies the sampling rate change $f_s' = f_s/D$, which can be thought of as decimation by a factor of D from the high rate f_s to the low rate f_s', or interpolation from the low to the high rate.

27.3 Notch and Comb Filters with Fractional Delay

The implementation of comb and notch filters requires that the sampling rate be related to the fundamental harmonic by $f_s = Df_1$ with D an integer, so that z^{-D} represents a D-fold multiple delay. In some applications, one may not have the freedom of choosing the sampling rate and the equation $D = f_s/f_1$ may result into a non-integer number.

One possible approach, discussed at the end of this section, is simply to design individual comb/notch filters for each desired harmonic $f_k = kf_1 = kf_s/D$, $k = 1, 2, \dots,$ that lies within the Nyquist interval, and then either cascade the filters together in the notch case, or add them in parallel in the comb case.

Another approach is to approximate the desired non-integer delay z^{-D} by an FIR filter and then use the IIR comb/notch structures of Eq. (27.2.9). Separating D into its integer and fractional parts, we may write:

$$D = D_{\text{int}} + d \tag{27.3.1}$$

where $D_{\text{int}} = \text{floor}(D)$ and $0 < d < 1$. The required multiple delay can be written then as $z^{-D} = z^{-D_{\text{int}}} z^{-d}$. The fractional part z^{-d} can be implemented by replacing it with an

FIR filter that approximates it, that is, $H(z)\approx z^{-d}$, so that $z^{-D}\approx z^{-D_{\text{int}}}H(z)$. Then, the corresponding IIR comb/notch filters (27.2.9) will be approximated by

$$\begin{aligned}
H_{\text{comb}}(z) &= b\,\frac{1+z^{-D_{\text{int}}}H(z)}{1-a\,z^{-D_{\text{int}}}H(z)}\,, \quad b=\frac{1-a}{2}\\
H_{\text{notch}}(z) &= b\,\frac{1-z^{-D_{\text{int}}}H(z)}{1-a\,z^{-D_{\text{int}}}H(z)}\,, \quad b=\frac{1+a}{2}
\end{aligned} \tag{27.3.2}$$

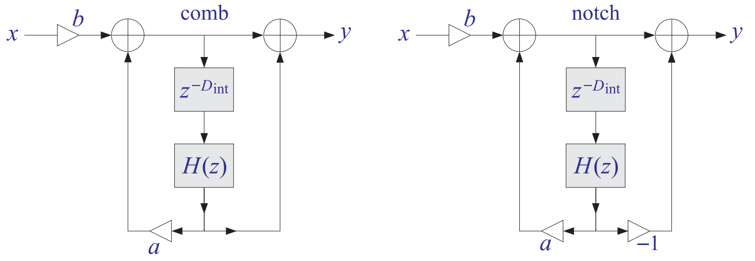


Fig. 27.3.1 Comb and notch filters with fractional delay.

Fig. 27.3.1 shows a possible realization. There exist many design methods for such approximate fractional delay filters [746]. We encountered some in Sec. 23.6. For example, the transfer functions of the causal Lagrange interpolation filters of orders 1 and 2 approximating the required non-integer delay d can be obtained from Eq. (23.7.17),

$$\begin{aligned}
H(z) &= d+(1-d)z^{-1}\\
H(z) &= \frac{1}{2}(d-1)(d-2)-d(d-2)z^{-1}+\frac{1}{2}d(d-1)z^{-2}
\end{aligned} \tag{27.3.3}$$

Such interpolation filters accurately cover only a fraction, typically 10–20%, of the Nyquist interval, and therefore, would be appropriate only if the first few harmonics are significant. A more effective approach suggested by [752] is to impose linear constraints on the design of $H(z)$ that preserve the required filter response at all the harmonics.

For integer delay D, the comb filter peaks or the notch filter nulls occur at the D-th roots of unity $z_k=e^{j\omega_k}$, $\omega_k=2\pi k/D$, which satisfy $z_k^D=1$.

For non-integer D, we require the same constraints for the delay filter $z^{-D_{\text{int}}}H(z)$, that is, $z_k^{-D_{\text{int}}}H(z_k)=1$, or in terms of the frequency response, $e^{-j\omega_k D_{\text{int}}}H(\omega_k)=1$, where again $z_k=e^{j\omega_k}$, $\omega_k=2\pi k/D$. Since $e^{-j\omega_k D}=1$, we have,

$$e^{-j\omega_k D_{\text{int}}}H(\omega_k)=1=e^{-j\omega_k D}=e^{-j\omega_k(D_{\text{int}}+d)} \quad\Rightarrow\quad H(\omega_k)=e^{-j\omega_k d}$$

These are the constraints to be imposed on the design of $H(z)$. In order to obtain a real-valued impulse response for this filter, we must work with the harmonics that lie in the symmetric Nyquist interval, that is, $-\pi\le\omega_k\le\pi$, or,

$$-\pi\le\frac{2\pi k}{D}\le\pi \quad\Rightarrow\quad -\frac{D}{2}\le k\le\frac{D}{2}$$

Writing $D_{\text{int}}=2p+q$ and $D=D_{\text{int}}+d=2p+q+d$, with integer p and $q=0,1$, the above condition reads:

$$-p-\frac{1}{2}(q+d)\le k\le p+\frac{1}{2}(q+d)$$

Since $0<d<1$ and k must be an integer, we obtain,

$$-p\le k\le p \tag{27.3.4}$$

Thus, the design problem is to determine an FIR filter $H(z)$ such that $H(\omega)\approx e^{-j\omega d}$, and subject to the constraints:

$$H(\omega_k)=e^{-j\omega_k d}\,,\quad -p\le k\le p \tag{27.3.5}$$

When $q=d=0$, we must choose $-p\le k\le p-1$, because $k=\pm p$ both are mapped onto the Nyquist frequency $\omega=\pm\pi$ and need be counted only once. In this case, of course, we expect the design method to produce the identity filter $H(z)=1$.

Following [752], we use a constrained least-squares design criterion with the following performance index into which the constraints have been incorporated by means of complex-valued Lagrange multipliers λ_k:

$$\mathcal{J}=\int_{-\alpha\pi}^{\alpha\pi}\big|H(\omega)-e^{-j\omega d}\big|^2\frac{d\omega}{2\pi}+\sum_{k=-p}^{p}\big[e^{-j\omega_k d}-H(\omega_k)\big]\lambda_k^*+\text{c.c.}=\min \tag{27.3.6}$$

where "c.c." denotes the complex conjugate of the second term. The approximation $H(\omega)\approx e^{-j\omega d}$ is enforced in the least-squares sense over a portion of the Nyquist interval, $[-\alpha\pi,\alpha\pi]$, where typically, $0.9\le\alpha\le1$, with $\alpha=1$ covering the full interval. Assuming an Mth order filter $\mathbf{h}=[h_0,h_1,\dots,h_M]^T$, we can write the frequency response in terms of the $(M+1)$-dimensional vectors,

$$H(\omega)=\sum_{n=0}^{M}h_n e^{-jn\omega}=\mathbf{s}_\omega^\dagger\mathbf{h}\,,\quad \mathbf{s}_\omega=\begin{bmatrix}1\\ e^{j\omega}\\ \vdots\\ e^{jM\omega}\end{bmatrix},\quad \mathbf{h}=\begin{bmatrix}h_0\\ h_1\\ \vdots\\ h_M\end{bmatrix} \tag{27.3.7}$$

Similarly, we can express the gain constraints in the vector form,

$$S^\dagger\mathbf{h}=\mathbf{g} \tag{27.3.8}$$

where S is an $(M+1)\times(2p+1)$ matrix and $\mathbf{g}$ a $(2p+1)$-dimensional column vector defined component-wise by

$$\begin{aligned}
S_{nk} &= e^{jn\omega_k}\,,\quad 0\le n\le M\,,\quad -p\le k\le p\\
g_k &= e^{-j\omega_k d}\,,\quad -p\le k\le p
\end{aligned} \tag{27.3.9}$$

that is,

$$S=[\dots,\mathbf{s}_{\omega_k},\dots]\,,\quad \mathbf{g}=\begin{bmatrix}\vdots\\ e^{-j\omega_k d}\\ \vdots\end{bmatrix} \tag{27.3.10}$$

It follows that the performance index can be written compactly as,

$$\mathcal{J} = \int_{-\alpha\pi}^{\alpha\pi} \big|\mathbf{s}_\omega^\dagger \mathbf{h} - e^{-j\omega d}\big|^2 \frac{d\omega}{2\pi} + \boldsymbol{\lambda}^\dagger(\mathbf{g} - S^\dagger\mathbf{h}) + (\mathbf{g} - S^\dagger\mathbf{h})^\dagger\boldsymbol{\lambda} = \min \tag{27.3.11}$$

where $\boldsymbol{\lambda} = [\dots, \lambda_k, \dots]^T$ is the vector of Lagrange multipliers. Expanding the first term of $\mathcal{J}$, we obtain,

$$\mathcal{J} = \mathbf{h}^\dagger R\mathbf{h} - \mathbf{h}^\dagger\mathbf{r} - \mathbf{r}^\dagger\mathbf{h} + \alpha + \boldsymbol{\lambda}^\dagger(\mathbf{g} - S^\dagger\mathbf{h}) + (\mathbf{g}^\dagger - \mathbf{h}^\dagger S)\boldsymbol{\lambda} = \min \tag{27.3.12}$$

where the matrix R and vector $\mathbf{r}$ are defined by,

$$R = \frac{1}{2\pi}\int_{-\alpha\pi}^{\alpha\pi} \mathbf{s}_\omega \mathbf{s}_\omega^\dagger \, d\omega\,, \quad \mathbf{r} = \frac{1}{2\pi}\int_{-\alpha\pi}^{\alpha\pi} \mathbf{s}_\omega e^{-j\omega d}\, d\omega \tag{27.3.13}$$

and component-wise,

$$\begin{aligned} R_{nm} &= \int_{-\alpha\pi}^{\alpha\pi} e^{j\omega(n-m)}\frac{d\omega}{2\pi} = \frac{\sin\big(\alpha\pi(n-m)\big)}{\pi(n-m)}\,, \quad n,m = 0,1,\dots,M \\ r_n &= \int_{-\alpha\pi}^{\alpha\pi} e^{j\omega(n-d)}\frac{d\omega}{2\pi} = \frac{\sin\big(\alpha\pi(n-d)\big)}{\pi(n-d)}\,, \quad n = 0,1,\dots,M \end{aligned} \tag{27.3.14}$$

We note that for $\alpha = \pi$, R reduces to the identity matrix. The optimal solution for $\mathbf{h}$ is obtained by setting the gradient of $\mathcal{J}$ to zero:

$$\frac{\partial\mathcal{J}}{\partial\mathbf{h}^*} = R\mathbf{h} - \mathbf{r} - S\boldsymbol{\lambda} = 0 \quad\Rightarrow\quad \mathbf{h} = R^{-1}\mathbf{r} + R^{-1}S\boldsymbol{\lambda} = \mathbf{h}_{\mathrm{u}} + R^{-1}S\boldsymbol{\lambda}$$

where $\mathbf{h}_{\mathrm{u}} = R^{-1}\mathbf{r}$ is the unconstrained solution of the least-squares problem. The Lagrange multiplier $\boldsymbol{\lambda}$ can be determined by multiplying both sides by $S^\dagger$ and using the constraint (27.3.8):

$$\mathbf{g} = S^\dagger\mathbf{h} = S^\dagger\mathbf{h}_{\mathrm{u}} + S^\dagger R^{-1}S\boldsymbol{\lambda} \quad\Rightarrow\quad \boldsymbol{\lambda} = (S^\dagger R^{-1}S)^{-1}(\mathbf{g} - S^\dagger\mathbf{h}_{\mathrm{u}})$$

Finally, substituting $\boldsymbol{\lambda}$ into the solution for $\mathbf{h}$, we obtain,

$$\boxed{\mathbf{h} = \mathbf{h}_{\mathrm{u}} + R^{-1}S(S^\dagger R^{-1}S)^{-1}(\mathbf{g} - S^\dagger\mathbf{h}_{\mathrm{u}})} \tag{27.3.15}$$

This type of constrained least-squares problem appears in many applications. We will encounter it again in the context of designing linearly constrained minimum variance beamformers for interference suppression, and in the problem of optimum stock portfolio design.

The MATLAB function combfd implements the above design method. Its inputs are the fractional period D, the order M of the filter $H(z)$, the comb/notch pole parameter a, and the Nyquist factor α,

```
[bD,aD,h,zmax] = combfd(D,M,a,alpha);      % comb/notch filter design with fractional delay
```

Entering the parameter a as negative indicates the design of a notch filter. The outputs bD,aD are the coefficients of the numerator and denominator polynomials of the comb/notch filters (27.3.2):

$$\begin{aligned} B_D(z) &= b\big[1 \pm z^{-D_{\mathrm{int}}}H(z)\big] \\ A_D(z) &= 1 - a\,z^{-D_{\mathrm{int}}}H(z) = 1 - az^{-D_{\mathrm{int}}}\big(h_0 + h_1z^{-1} + \cdots + h_Mz^{-M}\big) \end{aligned} \tag{27.3.16}$$

The output h is the impulse response vector $\mathbf{h}$, and zmax is the maximum pole radius of the denominator filter $A_D(z)$, which can be used to monitor the stability of the designed comb/notch filter. The pole parameter a can be fixed using the bandwidth equation (27.2.11), which is still approximately valid.

Fig. 27.3.2 shows a design example with fractional period $D = 9.1$, so that $D_{\mathrm{int}} = 9$ and $d = 0.1$. The other parameters were $M = 8$, $a = R^D$ with $R = 0.95$, and $\alpha = 1$.

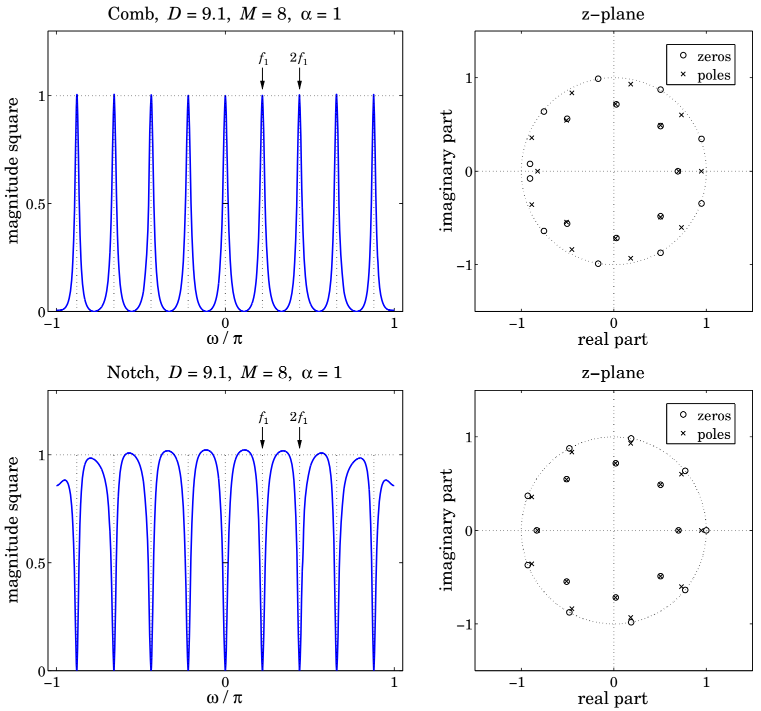


Fig. 27.3.2 Comb and notch filters with $D = 9.1$, and their pole/zero patterns.

The 3-dB widths obtained from Eq. (27.2.11) are indicated on the graphs by the two short horizontal lines at the half-power levels. The frequency plots are over the symmetric interval $-\pi \le \omega \le \pi$. The comb peaks have unity gain at the harmonics. For the notch case, the response between the notch dips is not very flat, but can be made flatter by decreasing the bandwidth, i.e., increasing the parameter a towards unity.

The right graphs depict the pole/zero patterns of the polynomials $B_D(z)$ and $A_D(z)$. These polynomials have orders $D_{\text{int}} + M = 9 + 8 = 17$. For the comb filter, we observe how the $D_{\text{int}} = 9$ poles arrange themselves around the unit circle at the harmonic frequencies, while the remaining 8 poles lie inside the unit circle.

The zeros of $B_D(z)$ also arrange themselves in two groups, 8 of them lying on the unit circle halfway between the comb peak poles, and the remaining 9 lying inside the unit circle, with a group of 7 poles and 7 zeros almost falling on top of each other, almost canceling each other.

A similar pattern occurs for the notch filter, except now the notch zeros at the harmonics have poles lying almost behind them in order to sharpen the notch widths, while the remaining pole/zero pairs arrange themselves inside the unit-circle as in the comb case.

Generally, this design method tends to work well whenever D is near an odd integer, such as in the above example and in the top graphs of Fig. 27.3.4, which have $D = 9.1$ and $D = 8.9$. The method has some difficulty when D is near an even integer, such as $D = 9.9$ or $D = 8.1$, as shown in Figs. 27.3.3 and the bottom of 27.3.4.

In such cases, the method tends to place a pole or pole/zero pair on the real axis near $z = -1$ resulting in an unwanted peak or dip at the Nyquist frequency $\omega = \pi$. Such poles are evident in the pole/zero plots of Fig. 27.3.3. If D were exactly an even integer, then such pole/zero pair at Nyquist would be present, but for non-integer D, the Nyquist frequency is not one of the harmonics. Removing that pole/zero pair from the design, does not improve the problem.

The MATLAB code for generating the magnitude responses and pole/zero plots is the same for all three figures. In particular, Fig. 27.3.2 was generated by,

```
f = linspace(-1,1,4001); w = pi*f;                 % frequency range

D=9.1; R=0.95; a=R^D; M=8; alpha=1;                % design parameters
beta = (1-a)/(1+a); Dw = 4/D * atan(beta);         % bandwidth calculation

[bD,aD,h,zmax] = combfd(D,M,a,alpha);              % comb, param a entered as positive
Hcomb = abs(freqz(bD,aD,w)).^2;                    % comb's magnitude rresponse

figure; plot(w/pi,Hcomb); figure; zplane(bD,aD);   % upper two graphs

[bD,aD,h,zmax] = combfd(D,M,-a,alpha);             % notch, param a entered as negative
Hnotch = abs(freqz(bD,aD,w)).^2;

figure; plot(w/pi,Hnotch); figure; zplane(bD,aD);  % lower two graphs
```

27.4 Parallel and Cascade Realizations

As we mentioned in the beginning of the previous section, an alternative approach is to design individual peak or notch filters at the harmonics and then combine the filters in parallel for the comb case, and in cascade for the notch case. Fig. 27.4.1 illustrates this type of design for the two "difficult" cases of $D = 9.9$ and $D = 8.1$ using second-order peak/notch filters designed to have the same bandwidth as in Fig. 27.3.3.

Let $H_k(z)$ be the peak/notch filter for the kth harmonic $\omega_k = k\omega_1 = 2\pi k/D$, $k = 0, 1, \dots, p$ and its negative $-\omega_k$. Then, the transfer functions of the comb and notch filters will be:

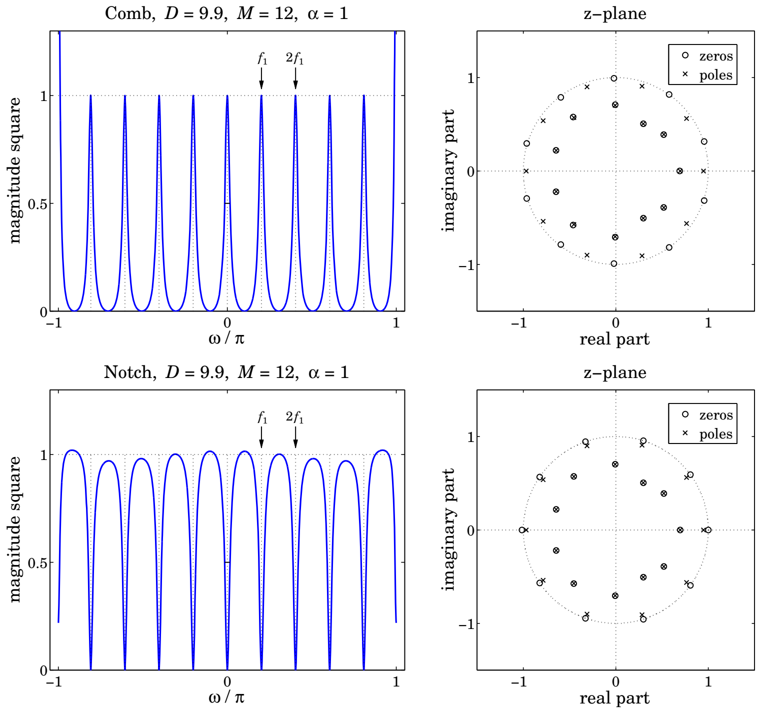


Fig. 27.3.3 Comb and notch filters with $D = 9.9$, and their pole/zero patterns.

$$H_{\text{comb}}(z) = \sum_{k=0}^{p} H_k(z)\,, \qquad H_{\text{notch}}(z) = \prod_{k=0}^{p} H_k(z) \tag{27.4.1}$$

In their simplest form, the individual filters $H_k(z)$ are second-order and can be obtained from the lowpass and highpass filters (27.2.8) by the lowpass-to-bandpass z-domain transformation:

$$z \;\to\; z' = \frac{z(\cos\omega_k - z)}{1 - z\cos\omega_k} \tag{27.4.2}$$

The resulting second-order peaking and notch filters are, for $k = 1, 2, \dots, p$:

$$\begin{aligned} &\text{peak:} \quad H_k(z) = b\,\frac{1 - z^{-2}}{1 - (1+a)\cos\omega_k\, z^{-1} + az^{-2}}\,, \quad b = \frac{1-a}{2} \\ &\text{notch:} \quad H_k(z) = b\,\frac{1 - 2\cos\omega_k\, z^{-1} + z^{-2}}{1 - (1+a)\cos\omega_k\, z^{-1} + az^{-2}}\,, \quad b = \frac{1+a}{2} \end{aligned} \tag{27.4.3}$$

The filter parameter a is fixed in terms of the 3-dB width of the peak or the notch by,

$$\tan\left(\frac{\Delta\omega}{2}\right) = \frac{1-a}{1+a} = \beta \tag{27.4.4}$$

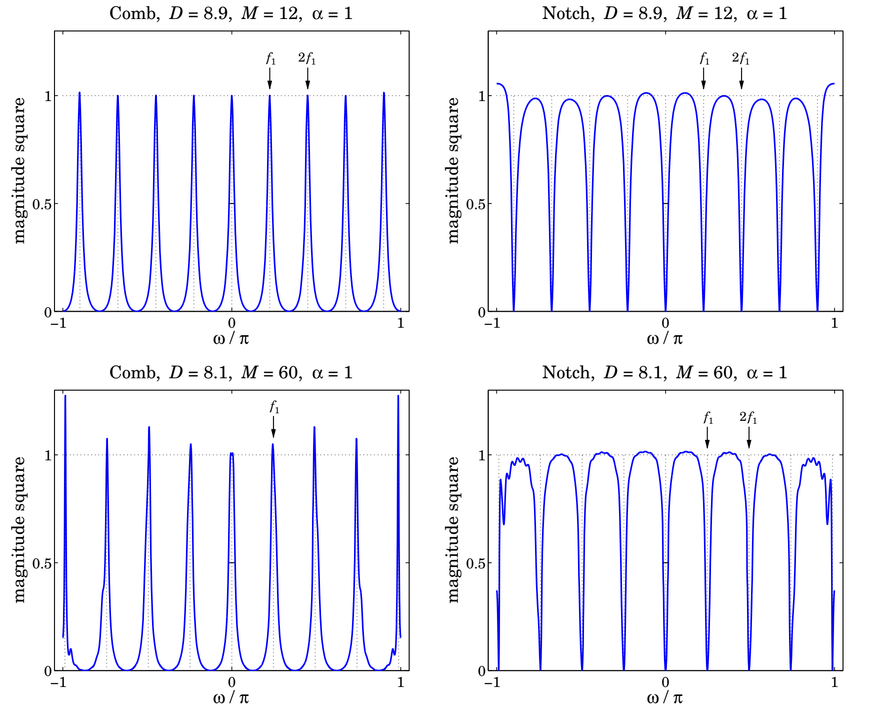


Fig. 27.3.4 Comb and notch filters with $D = 8.9$ and $D = 8.1$.

For $k = 0$, we may use the first-order lowpass/highpass filters of Eq. (27.2.8) without any z-domain transformation. But in order for their 3-dB frequency to match the specified 3-dB width $\Delta\omega$, their parameter a must be redefined as follows:

$$\tan\left(\frac{\Delta\omega}{4}\right) = \frac{1-a}{1+a} = \beta \tag{27.4.5}$$

To clarify the construction, we give below the MATLAB code for generating the left graphs of Fig. 27.4.1,

```
f = linspace(-1,1,4001); w = pi*f;                 % frequency range −π ≤ ω ≤ π

D = 9.9; p = floor(D/2); w1 = 2*pi/D;              % design parameters
R = 0.95; a = R^2;

beta = (1-a)/(1+a); dw = 2*atan(beta);             % 3-dB width calculation
beta0 = tan(dw/4);  a0 = (1-beta0)/(1+beta0);      % bandwidth parameter for k = 0 section

A = [1, -a0, 0];                                   % denominator coefficients for k = 0
Bcomb =  [1, 1, 0] * (1-a0)/2;                     % numerator coefficients for k = 0
Bnotch = [1,-1, 0] * (1+a0)/2;
```

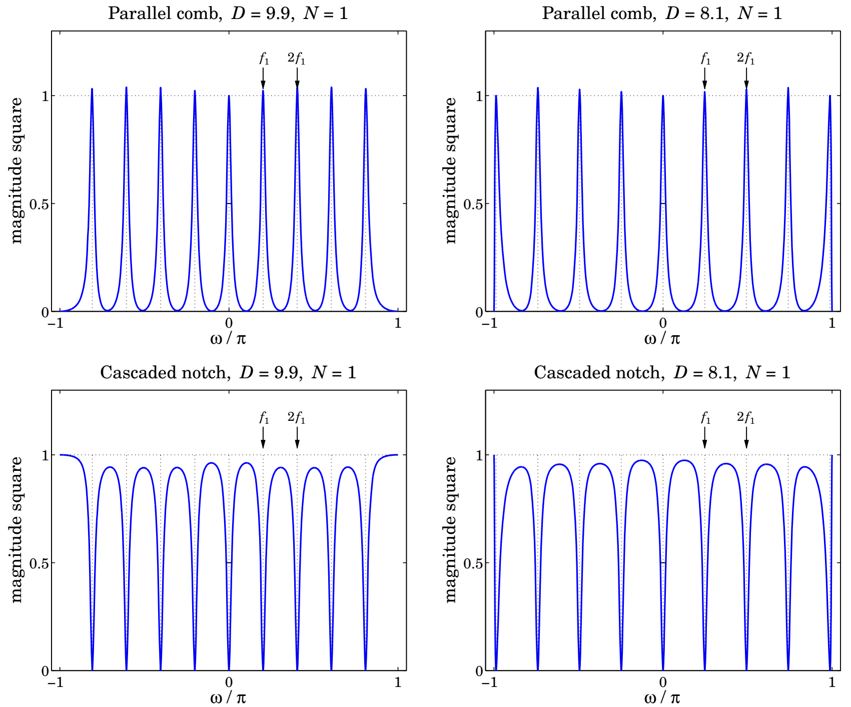


Fig. 27.4.1 Second-order parallel comb and cascaded notch filters.

```
Hcomb  = freqz(Bcomb,A,w);                         % k = 0 section, H0(ω)
Hnotch = freqz(Bnotch,A,w);

for k=1:p,                                         % non-zero harmonics
   A = [1, -(1+a)*cos(k*w1), a];                   % denominator of Hk(z)
   Bcomb =  [1, 0, -1] * (1-a)/2;                  % numerator of peak Hk(z)
   Bnotch = [1, -2*cos(k*w1), 1] * (1+a)/2;        % numerator of notch Hk(z)
   Hcomb =  Hcomb + freqz(Bcomb,A,w);              % add in parallel for comb
   Hnotch = Hnotch .* freqz(Bnotch,A,w);           % cascade for notch
end

figure; plot(w/pi, abs(Hcomb).^2, '-');            left graphs
figure; plot(w/pi, abs(Hnotch).^2,'-');
```

It is evident from Fig. 27.4.1 that this design method is flexible enough to correctly handle any values of the fractional period D. However, because of the mutual interaction between the individual filters, the peaks of the comb do not quite have unity gains, and the segments between the nulls of the notch filter are not quite flat.

This behavior can be fixed by decreasing the width $\Delta\omega$. However, for a fixed value of $\Delta\omega$, the only way to improve the response is by using higher-order filters. For example, Fig. 27.4.2 illustrates the cases of designing the individual filters using Butterworth filter prototypes of orders $N = 2$ and $N = 3$, whereas Fig. 27.4.1 corresponds to $N = 1$.

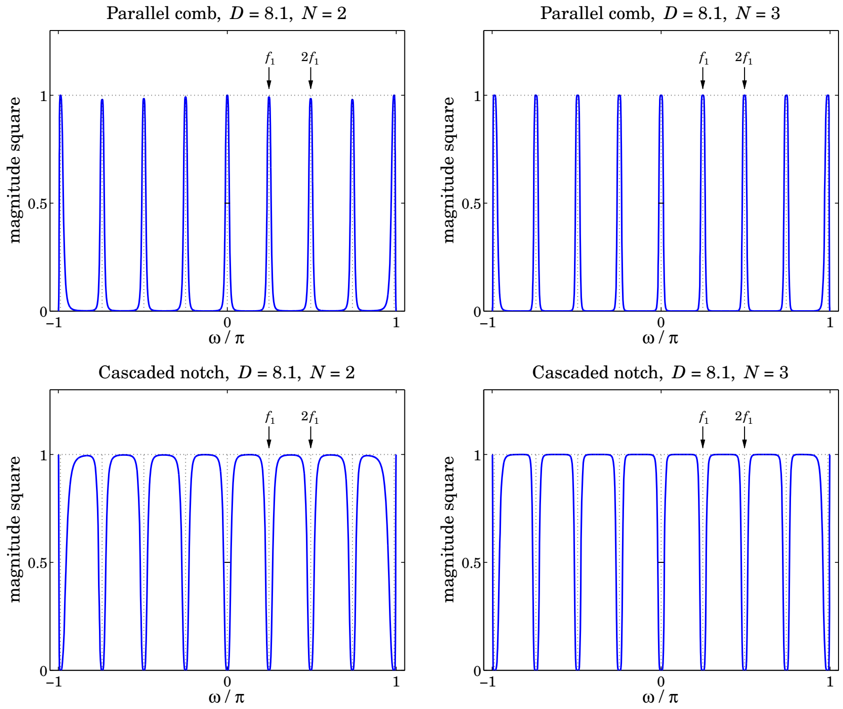


Fig. 27.4.2 High-order Butterworth parallel comb and cascaded notch filters.

The following MATLAB code illustrates the generation of the left graphs in Fig. 27.4.2, and uses the functions `hpeq` and `frespc` from the high-order equalizer design toolbox in [329], which is also included in the AOSP toolbox:

```
f = linspace(-1,1,4001); w = pi*f;

D = 8.1; p = floor(D/2); w1 = 2*pi/D;
R = 0.95; a = R^2;

beta = (1-a)/(1+a); dw = 2*atan(beta);          % 3-dB width

N = 2; GB = -20*log10(2);                        % Butterworth order and bandwidth gain

[B0,A0] = hpeq(N, -inf, 0, GB, 0, dw/2);         % k = 0 for comb, cutoff = half-bandwidth
Hcomb = frespc(B0,A0,w);

for k=1:p
   [B,A] = hpeq(N, -inf, 0, GB, k*w1, dw);       % non-zero harmonics
    Hcomb = Hcomb + frespc(B,A,w);               % add in parallel
end
```

```
figure; plot(w/pi,abs(Hcomb).^2,'-');            upper-left graph

[B0,A0] = hpeq(N, 0, -inf, GB, 0, dw/2);         % k = 0 for notch
Hnotch = frespc(B0,A0,w);

for k=1:p
   [B,A] = hpeq(N, 0, -inf, GB, k*w1, dw);
    Hnotch = Hnotch .* frespc(B,A,w);            % cascade in series
end

figure; plot(w/pi,abs(Hnotch).^2,'-');           % lower-left graph
```

The higher-order designs can also be based on Chebyshev or elliptic filters. In all cases, the starting point is a lowpass (or highpass) analog prototype filter $H_a(s)$, which is transformed into a peaking (or notch) filter centered at ω_k using the s-to-z domain bandpass transformation:

$$H(z) = H_a(s)\,, \qquad s = \frac{z'-1}{z'+1} = \frac{z^2 - 2\cos\omega_k\, z + 1}{z^2 - 1} \tag{27.4.6}$$

where z' is given by Eq. (27.4.2). For example, the analog Butterworth prototype filters of orders $N = 1, 2, 3$ are:

$$H_a(s) = \frac{\beta}{\beta + s}\,, \quad H_a(s) = \frac{\beta^2}{\beta^2 + \sqrt{2}\beta s + s^2}\,, \quad H_a(s) = \frac{\beta}{\beta + s}\cdot\frac{\beta^2}{\beta^2 + \beta s + s^2}$$

Similarly, for the notch filters, the analog prototypes are the highpass filters:

$$H_a(s) = \frac{s}{\beta + s}\,, \quad H_a(s) = \frac{s^2}{\beta^2 + \sqrt{2}\beta s + s^2}\,, \quad H_a(s) = \frac{s}{\beta + s}\cdot\frac{s^2}{\beta^2 + \beta s + s^2}$$

For arbitrary N, the Butterworth lowpass and highpass filters are:

$$H_a(s) = \left[\frac{\sigma}{\beta + s}\right]^r \prod_{i=1}^{L}\left[\frac{\sigma^2}{\beta^2 + 2\beta s\sin\phi_i + s^2}\right], \quad \phi_i = \frac{\pi(2i-1)}{2N} \tag{27.4.7}$$

where $N = 2L + r$, with integer L and $r = 0, 1$, and with $\sigma = \beta$ in the lowpass case, and $\sigma = s$ in the highpass one. The parameter β is related to the 3-dB width through $\beta = \tan(\Delta\omega/2)$. The filters of Eq. (27.4.3) are obtained by applying the transformation (27.4.6) to the $N = 1$ case.

Each peaking or notching filter is the cascade of L second-order sections in s or fourth-order sections in z (and possibly a second-order section in z if $r = 1$). The function `frespc` is used to calculate the corresponding frequency responses in such cascaded form. Further details on high-order designs and a description of the function `hpeq` can be found in [329].

27.5 Signal Averaging

Signal averaging is a technique for estimating a repetitive signal in noise. Evoked biological signals, GPS, and radar were some applications mentioned at the beginning of this chapter. A variant of the method can also be used to deseasonalize business, social, and climate data—the difference being here that the non-periodic part of the measured signal is not only noise but it can also contain a trend component. The typical assumed noise model in signal averaging has the form:

$$y_n = s_n + v_n \tag{27.5.1}$$

where s_n is periodic with some period D, assumed to be an integer, and v_n is zero-mean white noise. The periodic signal s_n can be extracted by filtering y_n through any comb filter, such as the IIR filter of Eq. (27.2.9).

Signal averaging is equivalent to comb filtering derived by applying the D-fold replicating transformation $z \to z^D$ to an ordinary, length-N, lowpass FIR averaging filter:

$$H_{\text{LP}}(z) = \frac{1}{N}\big[1 + z^{-1} + z^{-2} + \cdots + z^{-(N-1)}\big] = \frac{1}{N}\frac{1 - z^{-N}}{1 - z^{-1}} \tag{27.5.2}$$

The definition $H(z) = H_{\text{LP}}(z^D)$, then gives the comb filter:

$$H(z) = \frac{1}{N}\big[1 + z^{-D} + z^{-2D} + \cdots + z^{-(N-1)D}\big] = \frac{1}{N}\frac{1 - z^{-ND}}{1 - z^{-D}} \tag{27.5.3}$$

The latter equation shows that $H(z)$ has zeros at all the (ND)-th roots of unity that are not D-th roots of unity. At the latter, the filter has unity-gain peaks.

An example is shown in Fig. 27.5.1, with period $D = 10$ and $N = 5$ and $N = 10$. The comb peaks are at the D-th roots of unity $\omega_k = 2\pi k/D = 2\pi k/10$, $k = 0, 1, \dots, 9$. The 3-dB width of the peaks is indicated on the graphs by the short horizontal lines at the half-power level centered around the first harmonic.

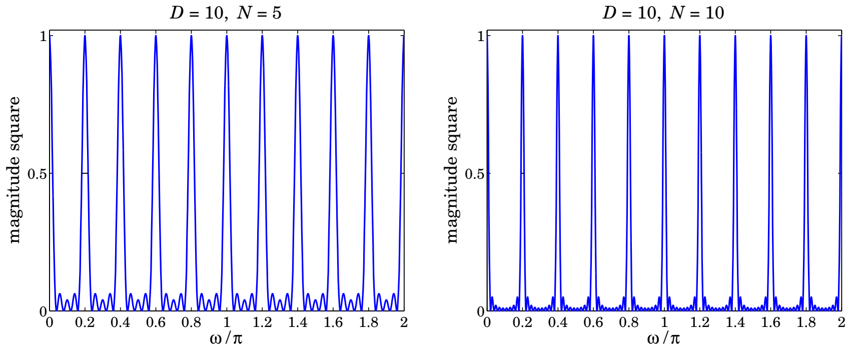


Fig. 27.5.1 Signal averaging filters with $D = 10$ and $N = 5, 10$.

The 3-dB width is given by

$$\Delta\omega = 0.886\,\frac{2\pi}{ND} \tag{27.5.4}$$

which follows from the frequency response of $H(z)$:

$$H(\omega) = \frac{1}{N}\frac{1 - e^{-j\omega ND}}{1 - e^{-j\omega D}} = \frac{1}{N}\frac{\sin(ND\omega/2)}{\sin(D\omega/2)}\,e^{-j(\omega(N-1)D/2} \tag{27.5.5}$$

Thus, the peaks get narrower with increasing number N of averaging periods. This has the effect of decreasing the noise, while letting through the periodic signal s_n.

The signal averaging interpretation can be seen from the time-domain operation of the filter. The corresponding output is the estimated periodic signal,

$$\hat{s}_n = \frac{1}{N}\big[y_n + y_{n-D} + y_{n-2D} + \cdots + y_{n-(N-1)D}\big] \tag{27.5.6}$$

Inserting $y_n = s_n + v_n$ and using the periodicity property $s_{n-D} = s_n$, we obtain,

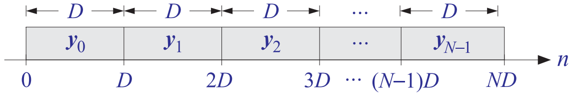
$$\hat{s}_n = s_n + \frac{1}{N}\big[v_n + v_{n-D} + \cdots + v_{n-(N-1)D}\big] \equiv s_n + \hat{v}_n \tag{27.5.7}$$

Because v_n was assumed to be stationary uncorrelated white noise, the variance of the filtered noise $\hat{v}_n$ will be reduced by a factor of N,

$$\sigma_{\hat{v}}^2 = \frac{1}{N^2}\big[\text{var}(v_n) + \text{var}(v_{n-D}) + \cdots + \text{var}(v_{n-(N-1)D})\big] = \frac{1}{N^2}(N\sigma_v^2) = \frac{1}{N}\sigma_v^2 \tag{27.5.8}$$

which implies that the NRR of the comb filter is $\mathcal{R} = 1/N$. Thus, by choosing N sufficiently large, the noise can be reduced, enabling the estimation of s_n.

Let the signal y_n be collected over N periods, that is, $0 \le n \le ND - 1$, and divide the signal into N length-D period segments as shown below,



The filtering operation (27.5.6) can be thought of as the averaging the N subblocks together. Indeed, let $y_i(n) = y_{iD+n}$, for $n = 0, 1, \dots, D - 1$, be the samples within the i-th subblock, $i = 0, 1, \dots, N - 1$. Then, we have

$$\frac{1}{N}\sum_{i=0}^{N-1} y_i(n) = \frac{1}{N}\sum_{i=0}^{N-1} y_{iD+n} = \frac{1}{N}\sum_{k=0}^{N-1} y_{(N-1)D+n-kD} = \hat{s}_{(N-1)D+n} \tag{27.5.9}$$

or, in words, the last D filter output samples, that is, over the period $[(N-1)D,\, ND-1]$, are the average of the samples over the last N periods. This can also be seen more simply by writing (27.5.6) in recursive form, which follows from Eq. (27.5.3),

$$\hat{s}_n = \hat{s}_{n-D} + \frac{1}{N}(y_n - y_{n-ND}) = \hat{s}_{n-D} + \frac{1}{N}y_n\,, \qquad 0 \le n \le ND - 1 \tag{27.5.10}$$

where the term y_{n-ND} was dropped because of the causal nature of y_n and the assumed range of n, that is, $0 \le n \le ND-1$. Thus, Eq. (27.5.10) shows that $\hat{s}_n$ is the accumulation and averaging of the N period segments of y_n.

The MATLAB implementation of signal averaging is straightforward, for example, assuming that the array y has length at least ND,

```
s = 0;
for i=0:N-1,
   yi = y(i*D+1 : i*D+D);        % extract i-th period
   s = s + yi;                   % accumulate i-th period
end
s = s/N;                         % average of N periods
```

So far we have not imposed the constraint $S_0 = s_n + s_{n-1} + \cdots + s_{n-D+1} = 0$. If in addition to the noise component v_n, there is a slowly-varying background or trend present, say, t_n, so that the observation signal is $y_n = s_n + t_n + v_n$, then we may associate the constant S_0 with the trend and assume that $S_0 = 0$. To guarantee this constraint, we may subtract from each block $y_i(n)$ its local average, and compute the estimated periodic component by:

$$\hat{s}_n = \frac{1}{N}\sum_{i=0}^{N-1}\big[y_i(n) - \mu_i\big]\,, \quad \mu_i = \frac{1}{D}\sum_{n=0}^{D-1} y_i(n) \tag{27.5.11}$$

which does satisfy $S_0 = 0$. By replicating the μ_i by D times within the i-th time period $[iD, iD + D - 1]$, and stringing the replicated values together over all the periods, we obtain a step-wise estimate of the trend component t_n. The following MATLAB code illustrates how to do that:

```
y = y(:);
L = length(y);
N = floor(L/D);                       % number of periods in y
r = mod(L,D);                         % L = ND + r

s = 0;
for i=0:N-1,
   yi = y(i*D+1 : i*D+D);             % i-th period
   m(i+1) = mean(yi);                 % mean to be removed
   s = s + yi - m(i+1);               % accumulate i-th period
end
s = s / N;                            % estimated period

ys = repmat(s,N,1);                   % replicate N periods
ys(end+1:end+r) = s(1:r);             % extend to length L by appending a portion of s

yt = repmat(m,D,1); yt = yt(:);       % repeat each mean D times within its period
yt(end+1:end+r) = yt(end);            % extend to length L by replicating last mean r times
```

where ys represents the estimated periodic signal, replicated over N periods, and yt is the estimated step-function trend. These above steps have been incorporated into the MATLAB function sigav:

```
[ys,s,yt] = sigav(y,D);       % signal averaging
```

Example 27.5.1: Fig. 27.5.2 shows a simulated signal averaging example. The period is $D = 10$ and the total number of periods $N = 100$. The graphs display only the first 10 periods to improve visibility. The periodic signal was superimposed on a slowly-varying trend and noise was added:

$$y_n = s_n + t_n + v_n\,, \quad s_n = 0.5\sin\left(\frac{4\pi n}{D}\right) + 0.5\sin\left(\frac{6\pi n}{D}\right)\,, \quad t_n = \sin\left(\frac{2\pi n}{10D}\right)$$

where $n = 0, 1, \dots, ND - 1$, and v_n is zero-mean, unit-variance, white noise. The upper row shows the noise-free case (with $v_n = 0$). The upper-right graph shows the periodic signal s_n. The estimated one resulting from the output of sigav is essentially identical to s_n and thus not displayed. The step-function estimated trend is shown on the upper-left.

The lower-left graph shows the noisy case, including the estimated step-trend signal. The lower-right graph shows the estimated periodic signal from the output of sigav. The following MATLAB code illustrates the generation of the bottom graphs:

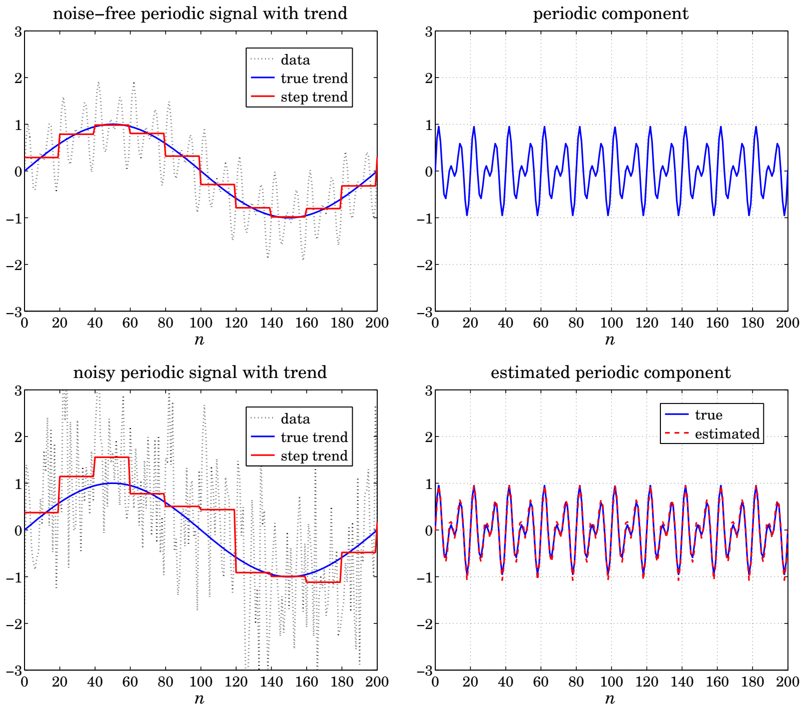


Fig. 27.5.2 Signal averaging of noisy periodic signal with slowly-varying trend.

```
D = 20; N = 100; n = 0:N*D;

s = (sin(4*pi*n/D) + sin(6*pi*n/D))/2;          % periodic component
t = sin(2*pi*n/D/10);                           % trend component

seed = 2008; randn('state',seed);
v = randn(size(n));

y = s + t + v;                                  % noisy observations

[ys,p,yt] = sigav(y,D);                         % signal averaging, p = one period

figure; plot(n,y,'--', n,t,'-.', n,yt,'-');     % yt is the estimated trend
xlim([0,200]);                                  % show only the first 10 periods
figure; plot(n,ys, '-');                        % estimated periodic component
xlim([0,200]);
```

Example 27.5.2: *Housing Starts.* Fig. 27.5.3 shows the application of signal averaging to the monthly, not seasonally adjusted, new privately-owned housing starts, for the 25 year period from January 1984 to December 2008. The data are from the US Census Bureau from the web link: http://www.census.gov/ftp/pub/const/starts_cust.xls.

The upper graphs show the estimated step-wise trend and the seasonal, periodic, compo-

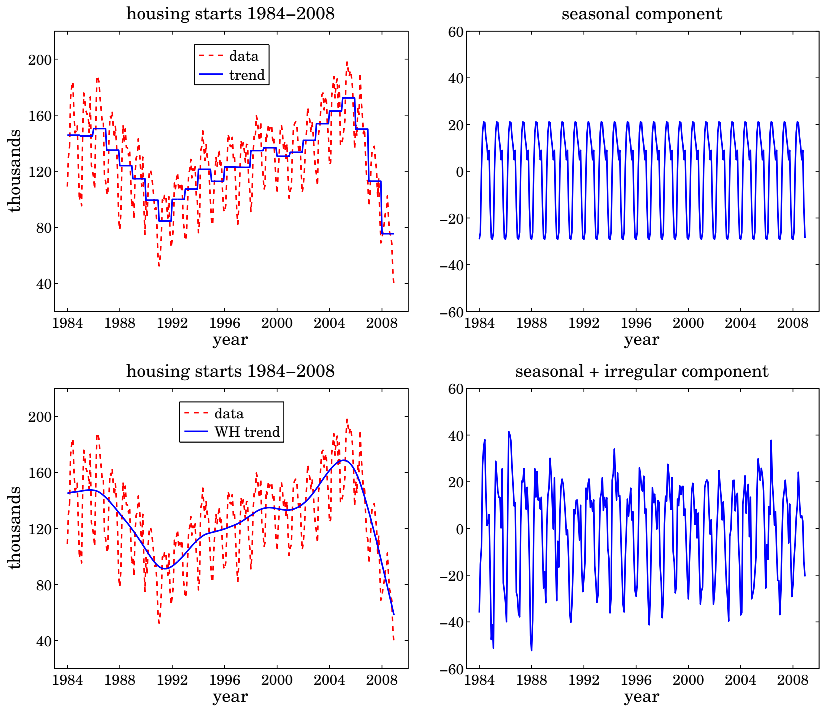


Fig. 27.5.3 Signal averaging and smoothing of monthly housing data.

nent. Although there is clear annual periodicity in the data, the signal averaging method is not the best approach to this application because it does not result into a smooth trend. We consider better methods to deseasonalize such data in the next sections.

As an alternative method, the bottom graphs show the application of the Whittaker Henderson smoothing method to estimate the smooth trend. The optimal smoothing parameter was determined by the GCV criterion to be $\lambda = 6850$ and the smoothing order was $s = 2$.

The difference between the raw data and the estimated trend represents the seasonal plus irregular component and is plotted in the bottom-right graph. Further application of signal averaging to this component will generate an estimate of the seasonal component. It is not plotted because it is essentially identical to that shown in the upper-right graph.

The following MATLAB code illustrates the generation of the four graphs, including, but commented out, the computation of the seasonal part for the bottom graphs:

```
Y = loadfile('newhouse.dat');                 % data file available in the AOSP toolbox

i = find(Y(:,1)==109.1);                      % finds the beginning of the year 1984

y = Y(i:end-4,1);                             % keep data from Jan.1984 to Dec.2008
t = taxis(y,12,1984);                         % define time axis
```

```
[ys,s,yt] = sigav(y,12);                      % signal averaging with period 12

figure; plot(t,y,'--', t,yt,'-');             % upper-left graph
figure; plot(t,ys,'-');                       % upper-right graph

s = 2; la = 6800:2:6900;                      % smoothing order and search-range of λ's
[gcv,lopt] = whgcv(y,la,s);                   % optimum smoothing parameter, λ_opt = 6850

yt = whsm(y,lopt,s);                          % Whittaker-Henderson smoothing
ysi = y-yt;                                   % seasonal + irregular component
% ys = sigav(ysi,12);                         % seasonal component, not shown

figure; plot(t,y,'--', t,yt,'-');             % bottom-left graph
figure; plot(t,ysi,'-');                      % bottom-right graph
% figure; plot(t,ys,'-');                     % essentially the same as upper-right graph
```

27.6 Ideal Seasonal Decomposition Filters

A possible approach for separating the three components of the signal $y_n = s_n + t_n + v_n$ is to first estimate the trend t_n using a lowpass filter, and then extract the seasonal component s_n by applying a comb filter to the residual $r_n = y_n - t_n = s_n + v_n$, which consists of the seasonal and irregular parts.

The technique assumes of course that the trend is a slowly-varying, low-frequency, signal. Fig. 27.6.1 illustrates some typical frequency spectra for the three components and the ideal filters that might be used to extract them.

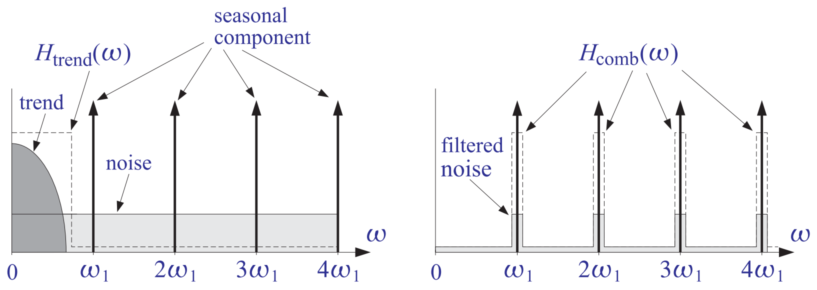


Fig. 27.6.1 Ideal filters for decomposition into trend and seasonal components.

Let $H_{\rm trend}(z)$ be the trend-extraction filter and $H_{\rm comb}(z)$ the comb filter with peaks at the seasonal harmonics (excluding the one at DC). Then, the filtering equations for extracting the three components from y_n can be expressed in the z-domain as follows:

$$
\begin{aligned}
T(z) &= H_{\rm trend}(z)Y(z) \\
R(z) &= S(z)+V(z) = Y(z)-T(z) = \big[1 - H_{\rm trend}(z)\big]Y(z) \\
S(z) &= H_{\rm comb}(z)R(z) = H_{\rm comb}(z)\big[1 - H_{\rm trend}(z)\big]Y(z) \equiv H_S(z)Y(z) \\
V(z) &= R(z)-S(z) = \big[1 - H_{\rm comb}(z)\big]\big[1 - H_{\rm trend}(z)\big]Y(z) \equiv H_I(z)Y(z)
\end{aligned}
$$

where $Y(z), S(z), T(z), V(z), R(z)$ are the z-transforms of y_n, s_n, t_n, v_n, r_n. Thus, the filters for extracting the three components are:

$$\begin{aligned} H_T(z) &= H_{\text{trend}}(z) && \text{(trend)} \\ H_S(z) &= H_{\text{comb}}(z)\big[1-H_{\text{trend}}(z)\big] && \text{(seasonal)} \\ H_I(z) &= \big[1-H_{\text{comb}}(z)\big]\big[1-H_{\text{trend}}(z)\big] && \text{(irregular)} \end{aligned} \tag{27.6.1}$$

The three filters satisfy the complementarity property:

$$H_T(z)+H_S(z)+H_I(z)=1 \tag{27.6.2}$$

In Example 27.5.2, we followed exactly this approach where the trend filter was implemented as a Whittaker-Henderson smoother and the comb filter as a signal averager. Other possibilities exist for these filters and a lot of research has gone into making choices that try to balance a good filter response versus the ability to work well with short data records, including the handling of the end-point problem.

Example 27.6.1: *Housing Starts.* The housing starts signal considered in Example 27.5.2 displays the typical frequency spectra shown in Fig. 27.6.1.

The left graph in Fig. 27.6.2 shows the corresponding magnitude spectrum of the original data signal y_n, normalized to unity maximum and plotted over the symmetric Nyquist interval $[-\pi, \pi]$ in units of the fundamental harmonic $\omega_1 = 2\pi/12$. The spectrum is dominated by the low-frequency trend signal. The right graph shows the spectrum of the seasonal plus irregular component $r_n = y_n - t_n = s_n + v_n$, which displays the harmonics more clearly.

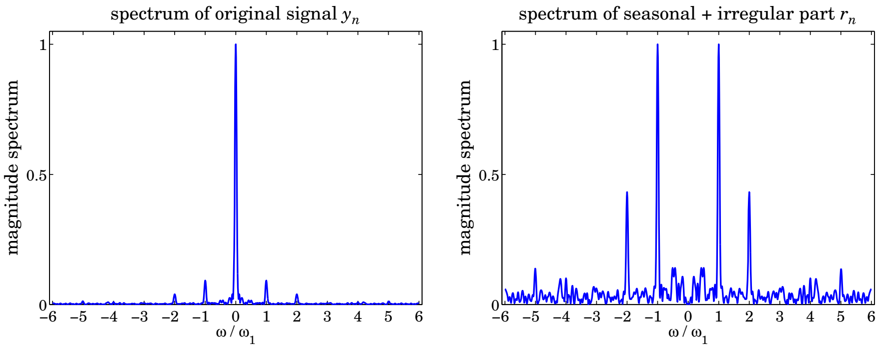


Fig. 27.6.2 Spectra of monthly housing data with and without trend.

The following MATLAB code illustrates the computation of the spectra:

```
Y = loadfile('newhouse.dat');            % data file available in the AOSP toolbox
i = find(Y(:,1)==109.1);                 % finds the beginning of the year 1984
y = Y(i:end-4,1);                        % keep data from Jan.1984 to Dec.2008

s = 2; lopt = 6850                       % use optimum λ from Example 27.5.2
```

```
yt = whsm(y,lopt,s);                     % Whittaker-Henderson smoothing
ysi = y - yt;                            % seasonal + irregular component

k = linspace(-6,6,1201); w = 2*pi*k/12;                   % frequency ω = kω1
L = length(y);
wind = 0.54 - 0.46*cos(2*pi*(0:L-1)/(L-1))';              % Hamming window

Y = abs(freqz(y.*wind,1,w));   Y = Y/max(Y);              % normalized spectrum
Ysi = abs(freqz(ysi.*wind,1,w)); Ysi = Ysi/max(Ysi);

figure; plot(k,Y); figure; plot(k,Ysi);                   % left and right graphs
```

The signals were windowed by a Hamming window prior to computing their DTFTs. □

Ideally, it does not matter if $H_{\text{comb}}(z)$ excludes or not the peak at DC because it would be canceled from $H_S(z)$ by the presence of the factor $\big[1-H_{\text{trend}}(z)\big]$. However, in practice because the filters are non-ideal, an extra step is usually taken to ensure that this peak is absent or minimized from s_n. For example, an additional de-trending step may be applied to $S(z)$, that is,

$$\begin{aligned} &S_{\text{prelim}}(z) = H_{\text{comb}}(z)R(z) \\ &S(z) = S_{\text{prelim}}(z) - H_{\text{trend}}(z)S_{\text{prelim}}(z) = H_{\text{comb}}(z)\big[1-H_{\text{trend}}(z)\big]^2 Y(z) \end{aligned} \tag{27.6.3}$$

This results in the modified extraction filters, which still satisfy (27.6.2):

$$\begin{aligned} H_T(z) &= H_{\text{trend}}(z) && \text{(trend)} \\ H_S(z) &= H_{\text{comb}}(z)\big[1-H_{\text{trend}}(z)\big]^2 && \text{(seasonal)} \\ H_I(z) &= \big[1-H_{\text{trend}}(z)\big]\big\{1-H_{\text{comb}}(z)\big[1-H_{\text{trend}}(z)\big]\big\} && \text{(irregular)} \end{aligned} \tag{27.6.4}$$

Further refinements will be discussed later on.

27.7 Classical Seasonal Decomposition

The classical seasonal decomposition method is the simplest realization of the procedure outlined in the previous section. Consider the following two possible lowpass trend-extraction filters:

$$\begin{aligned} H_{\text{trend}}(z) &= \frac{1}{D}\big[1+z^{-1}+z^{-2}+\cdots+z^{-(D-1)}\big] \\ H_{\text{trend}}(z) &= \frac{1}{D}\big[1+z^{-1}+z^{-2}+\cdots+z^{-(D-1)}\big]\cdot\frac{1}{2}(1+z^{-1}) \end{aligned} \tag{27.7.1}$$

where D is the period of the seasonal component. The first is typically used when D is odd, and the second, when D is even. They are referred to as the $1\times D$ and $2\times D$ trend filters, the notation $N_1\times N_2$ denoting the convolution of a length-N_1 with a length-N_2 averaging filter:

$$\frac{1}{N_1}\big[1+z^{-1}+\cdots+z^{-(N_1-1)}\big]\cdot\frac{1}{N_2}\big[1+z^{-1}+\cdots+z^{-(N_2-1)}\big] \tag{27.7.2}$$

The filters (27.7.1) are not perfect but are widely used. They have the desirable property of having nulls at the non-zero harmonics $\omega_k = k\omega_1 = 2\pi k/D$, $k = 1, 2, \dots, D-1$. Their 3-dB cutoff frequency is about one-half the fundamental harmonic ω_1, that is,

$$\omega_c = 0.886\,\frac{\pi}{D} \tag{27.7.3}$$

Eq. (27.7.3) can easily be derived for the $1{\times}D$ case and is a good approximation for the $2{\times}D$ case. Fig. 27.7.1 shows the magnitude response $|H_{\text{trend}}(\omega)|$ versus ω over the symmetric Nyquist interval, $-\pi \le \omega \le \pi$. The 3-dB frequency is indicated on the graph at the $1/\sqrt{2}$ level.

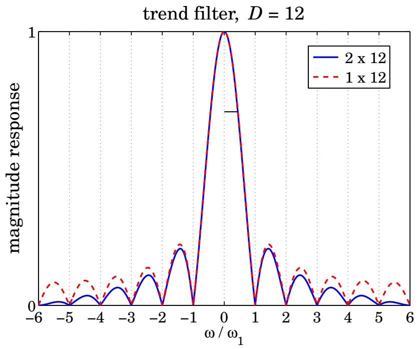


Fig. 27.7.1 Trend-extraction filters with $D = 12$.

In order to avoid delays introduced by the filters, the filters can be made symmetric with respect to the time origin. Let $D = 2p+1$ or $D = 2p$ in the even or odd case. Then, the symmetrized versions of the filters (27.7.1) are obtained by advancing them by p time units, that is, multiplying them by a factor of z^p:

$$\begin{aligned} D &= 2p+1\,, \quad H_{\text{trend}}(z) = z^p\,\frac{1}{D}\big[1 + z^{-1} + \cdots + z^{-(D-1)}\big] \\ D &= 2p\,, \qquad\;\; H_{\text{trend}}(z) = z^p\,\frac{1}{D}\big[1 + z^{-1} + \cdots + z^{-(D-1)}\big]\cdot\frac{1}{2}(1+z^{-1}) \end{aligned} \tag{27.7.4}$$

The corresponding frequency responses are obtained by setting $z = e^{j\omega}$:

$$\begin{aligned} D &= 2p+1\,, \quad H_{\text{trend}}(\omega) = \frac{\sin(D\omega/2)}{D\sin(\omega/2)} \\ D &= 2p\,, \qquad\;\; H_{\text{trend}}(\omega) = \frac{\sin(D\omega/2)}{D\sin(\omega/2)}\cdot\cos(\omega/2) \end{aligned} \tag{27.7.5}$$

where we used the identity $1+z^{-1}+\cdots+z^{-(D-1)} = (1-z^{-D})/(1-z^{-1})$. The symmetric impulse responses are:

$$\begin{aligned} D &= 2p+1\,, \quad \mathbf{h}_{\text{trend}} = \frac{1}{D}\big[1, \underbrace{1, \dots, 1}_{2p-1 \text{ ones}}, 1\big] \\ D &= 2p\,, \qquad\;\; \mathbf{h}_{\text{trend}} = \frac{1}{D}\big[0.5, \underbrace{1, \dots, 1}_{2p-1 \text{ ones}}, 0.5\big] \end{aligned} \tag{27.7.6}$$

In both cases, the filter length is $2p+1$, and the time-domain operation for calculating the estimated trend is by the symmetric convolutional equation:

$$\hat{t}_n = \sum_{i=-p}^{p} h_{\text{trend}}(i)\,y_{n-i} \tag{27.7.7}$$

The issues of filtering with double-sided filters were discussed in Sec. 23.10. We recall that for a length-L input signal y_n, the steady-state filtered output is over the time range $p \le n \le L-1-p$. The first p and last p output transients can be computed using appropriate asymmetric filters, and there exist many possibilities for these. Musgrave's minimum-revision method, discussed in Sec. 27.10, constructs such asymmetric filters from a given symmetric filter such as $\mathbf{h}_{\text{trend}}$.

The calculation of the trend estimate, incorporating also the end-point asymmetric filters, can be carried out with the MATLAB functions `trendma`, `minrev`, and `lpfilt`,

```
htrend = trendma(D);        % trend filters of Eq. (27.7.6)
B = minrev(htrend,R);       % corresponding smoothing matrix
t_hat = lpfilt(B,y);        % filtering operation
```

where `y` denotes the input data vector, and R is the Musgrave parameter to be explained in Sec. 27.10. The use of asymmetric filters affects only the first p and last p outputs.

In the so-called *classical decomposition method*, we apply the above filtering procedure to calculate the trend, and then apply ordinary signal averaging on the residual $r_n = y_n - t_n$ to calculate the seasonal component. The following computational steps describe the method:

```
B = minrev(trendma(D),R);    % trend moving-average, with minimum-revision end-filters
yt = lpfilt(B,y);            % trend component
yr = y - yt;                 % seasonal + irregular components
ys = sigav(yr,D);            % seasonal component
yi = yr - ys;                % irregular component
```

For a multiplicative decomposition, $y_n = s_n t_n v_n$, the last three steps are replaced by,

```
yr = y ./ yt;                % seasonal + irregular components
ys = sigav(yr,D);            % seasonal component
yi = yr ./ ys;               % irregular component
```

The function `cldec` implements the above steps,

```
[yt,ys,yi] = cldec(y,D,R,type);     % classical decomposition method
```

where the string `type` takes on the values `'a'` or `'m'` for additive (the default) or multiplicative decomposition. The default value of R is zero, which simply omits the computation of the first and last p transients and replaces them with the corresponding samples of the input signal y_n.

Example 27.7.1: *Housing Starts.* Fig. 27.7.2 the trend and seasonal components of the housing starts data extracted by the classical decomposition method versus the methods discussed in Example 27.5.2.

The Musgrave parameter was chosen to be $R = 10$. Since $D = 12$, the value of R affects only the first and last 6 outputs. The MATLAB code for generating these graphs was,

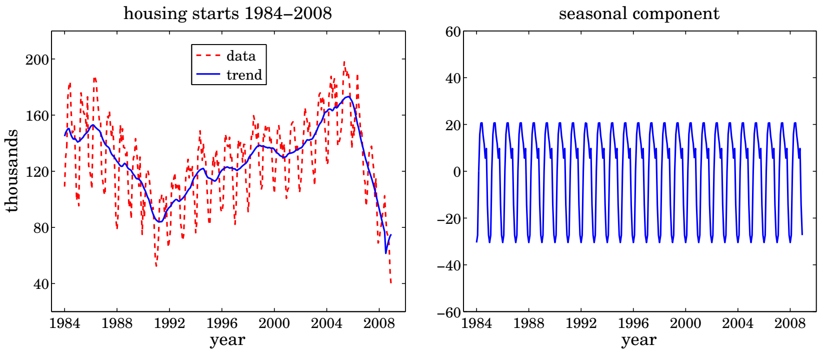


Fig. 27.7.2 Classical decomposition of monthly housing data.

```
Y = loadfile('newhouse.dat');
i = find(Y(:,1)==109.1);
y = Y(i:end-4,1); t = taxis(y,12,1984);

D=12; R=10;
[yt,ys,yi] = cldec(y,D,R);                 % classical decomposition method

figure; plot(t,y,'--', t,yt,'-');          % left graph
figure; plot(t,ys,'-');                    % right graph
```

The estimated trend is not as smooth as that of the Whittaker-Henderson method, but the estimated seasonal component is essentially the same as that of Example 27.5.2. □

Example 27.7.2: *Global Carbon Dioxide Data.* Figure 27.7.3 shows on the upper-left the monthly global CO$_2$ data for the period of January 1980 to March 2009, obtained from the NOAA web site: `http://www.esrl.noaa.gov/gmd/ccgg/trends/`.

The vertical axis is in parts per million (ppm), which represents the dry air mole fraction, that is, the number of CO$_2$ molecules divided by the number of all air molecules, after water vapor has been removed.

The upper graphs show the application of the classical seasonal decomposition method. The upper-left graph shows the trend t_n extracted by a 2×12 moving-average filter, while the right graph shows the seasonal component s_n.

The middle graphs show the spectra of the original data on the left, and of the residual part $r_n = y_n - t_n = s_n + v_n$ on the right. The frequency axis is the symmetric Nyquist interval $[-\pi, \pi]$ in units of the fundamental harmonic $\omega_1 = 2\pi/12$. The trend dominates the spectrum of y_n and swamps the smaller harmonic peaks of the seasonal part. Indeed, the level of the seasonal component relative to the trend can be estimated in dB to be:

$$20\log_{10}\left(\frac{\text{std}(s_n)}{\text{mean}(y_n)}\right) = 20\log_{10}\left(\frac{1.46}{360}\right) = -47.8 \text{ dB}$$

Therefore, the spectrum of the seasonal component is too small to be visible if plotted in absolute units. In order to make it visible, a Kaiser window with an 80-dB sidelobe level was applied to y_n prior to computing its spectrum and then plotted in dB. On the other

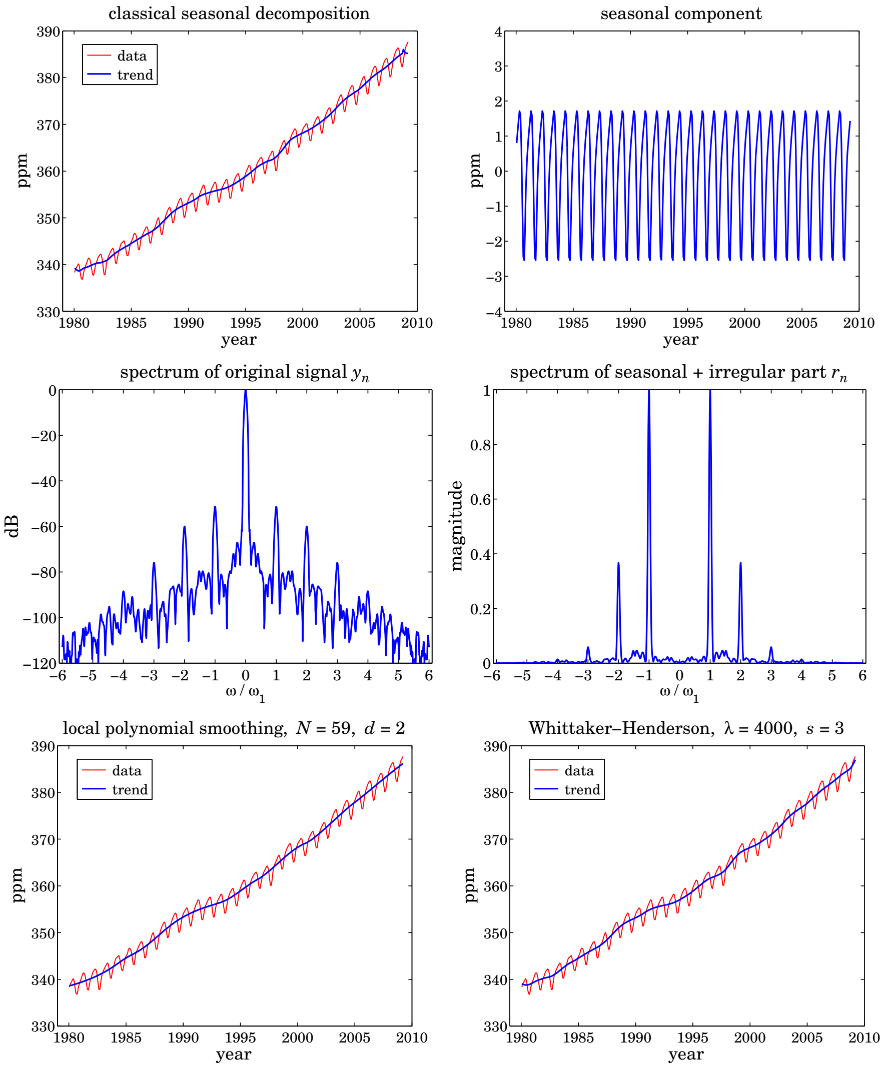


Fig. 27.7.3 Monthly global CO$_2$ data and spectra.

hand, after the trend is removed, the harmonics in the residual component r_n are quite visible if plotted in absolute units as in the middle-right graph.

The bottom two graphs show the trend component t_n extracted by a local polynomial smoothing filter on the left (with length $N = 59$ and length $d = 2$), and by a Whittaker-Henderson smoother on the right (with $\lambda = 4000$ and $s = 3$). The corresponding seasonal components obtained by signal averaging of the residual $r_n = y_n - t_n$ are not shown because they are essentially the same as that of the upper-right graph. The MATLAB code used to generate these six graphs was as follows:

```
Y = loadfile('co2_mm_gl.dat');                        % data file in the AOSP toolbox
t = Y(:,3); y = Y(:,4); yt0 = Y(:,5);                 % extract times and signals

R = 15; [yt,ys,yi] = cldec(y,12,R);                   % classical decomposition

figure; plot(t,y, t,yt);                              % upper-left graph
figure; plot(t,ys);                                   % upper-right graph

k = linspace(-6,6,1201); w = 2*pi*k/12;               % frequency in units of ω1

L = length(y); Rdb = 80;                              % Kaiser window parameters
wind = kwindow(L,Rdb)';                               % Kaiser window in the AOSP toolbox

ysi = y - yt;                                         % seasonal + irregular component

Y =   abs(freqz(y.*wind,1,w));    Y = Y/max(Y);               % DTFT computation
Ysi = abs(freqz(ysi.*wind,1,w)); Ysi = Ysi/max(Ysi);

figure; plot(k, 20*log10(Y));                         % middle-left graph
figure; plot(k, Ysi);                                 % middle-right graph

N=59; d=2; yt = lpfilt(lpsm(N,d),y);                  % LPSM smoother

figure; plot(t,y, t,yt);                              % bottom-left graph
% ys = sigav(y-yt,12);                                % seasonal part, not shown
% figure; plot(t,ys);

la=4000; s=3; yt = whsm(y,la,s);                      % Whittaker-Henderson smoother

figure; plot(t,y, t,yt);                              % bottom-right graph
```

The signal yt0 extracted from the 5th column of the data file (as in the second line of code above) represents the already de-seasonalized data, and therefore, we can compare it to the trend extracted by the above three methods. It is not plotted because it is virtually identical to the above extracted trends.

The percentage error defined as 100*norm(yt-yt0)/norm(yt0) is found to be 0.05%, 0.07%, and 0.05% for the classical, LPSM, and WH methods, respectively. □

To gain some further insight into the nature of the filtering operations for the classical decomposition method, we show in Fig. 27.7.4 the magnitude responses of the filters $H_S(\omega)$, $H_T(\omega)$, and $H_I(\omega)$ for extracting the seasonal, trend, and irregular components, as defined by Eq. (27.6.1). The trend filter $H_{\text{trend}}(\omega)$ is given by Eq. (27.7.5), and the comb filter $H_{\text{comb}}(\omega)$ by Eq. (27.5.5) with the phase factor removed to make it symmetric.

The upper graphs in Fig. 27.7.4 show the case of $D = 12$ and $N = 15$. We observe the absence of the harmonic at DC in $H_S(\omega)$. The irregular filter does not quite extract the noise component v_n, but rather a filtered version thereof. Ideally, the irregular filter $H_I(\omega)$ should have zeros at the harmonics, be very small in the passband of $H_T(\omega)$, and be flat between the harmonics. The actual filter $H_I(\omega)$ does approximate these features.

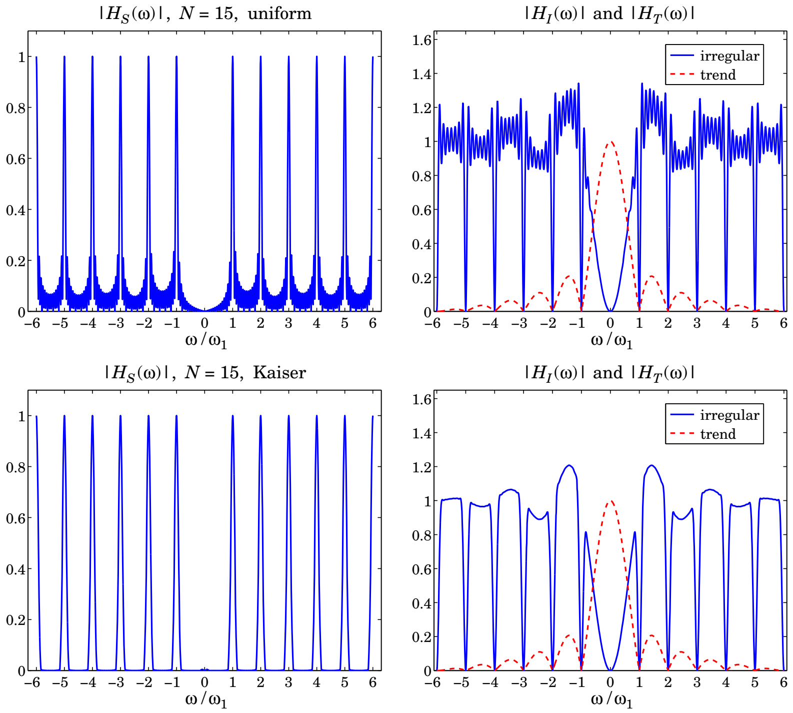


Fig. 27.7.4 Component extraction filters $H_S(\omega)$, $H_T(\omega)$, and $H_I(\omega)$.

The sidelobe behavior about the harmonics in $H_S(\omega)$, or about the nulls in $H_I(\omega)$, is due to the sidelobes introduced by the signal averaging filter $H_{\text{comb}}(\omega)$ of Eq. (27.5.5), which was obtained by applying the seasonalizing transformation $z \to z^D$ to a length-N FIR filter with uniform weights—the sidelobes being effectively the D-fold replicated versions of the sidelobes of a length-N rectangular window.

Such sidelobes are suppressed only by about 13 dB relative to the main peaks and are quire visible (at the level of $10^{-13/20} = 0.22$). The sidelobes can be suppressed further by replacing the rectangular FIR filter by a length-N windowed version thereof, using for example a Hamming or a Kaiser window. To be precise, the comb filter obtained from a window $w(n)$, $-M \le n \le M$, where $N = 2M + 1$, is defined by

$$W(z) = \sum_{n=-M}^{M} w(n) z^{-n} \quad \Rightarrow \quad H_{\text{comb}}(z) = W(z^D) = \sum_{n=-M}^{M} w(n) z^{-nD} \tag{27.7.8}$$

where $w(n)$ must be normalized to add up to unity. The two lower graphs of Fig. 27.7.4 show the filters obtained from a Kaiser window of length $N = 15$ and sidelobe level $R_{\text{dB}} = 50$ dB. The sidelobes are suppressed to the level of $10^{-50/20} = 0.003$ and are not visible if plotted in absolute scales. The price one pays for suppressing the sidelobes is, of course, the widening of the harmonic peaks. To clarify these ideas, we give below the MATLAB code for generating the graphs in Fig. 27.7.4:

```
D = 12; N = 15;
k = linspace(-6,6,1201); w = 2*pi*k/D;                            % frequency axis

ht = trendma(D);                   Ht = abs(freqz(ht,1,w));       % trend filter H_trend(ω)
hc = up(ones(1,N)/N, D);           Hc = abs(freqz(hc,1,w));       % comb filter H_comb(ω)
hs = conv(hc, compl(ht));          Hs = abs(freqz(hs,1,w));       % seasonal filter H_S(ω)
hi = conv(compl(hs), compl(ht));   Hi = abs(freqz(hi,1,w));       % irregular filter H_I(ω)
ha = compl(hs);                    Ha = abs(freqz(ha,1,w));       % seasonal adjustment filter

figure; plot(k, Hs); figure; plot(k, Hi, k,Ht,'r--');             % upper graphs
figure; plot(k, Ha);                                              % left graph in Fig. 27.7.5

Rdb=50; hk = kwindow(N,Rdb); hk = hk/sum(hk);                     % Kaiser window

hc = up(hk, D);                    Hc = abs(freqz(hc,1,w));       % new H_comb(ω)
hs = conv(hc, compl(ht));          Hs = abs(freqz(hs,1,w));       % new H_S(ω)
hi = conv(compl(hs), compl(ht));   Hi = abs(freqz(hi,1,w));       % new H_I(ω)
ha = compl(hs);                    Ha = abs(freqz(ha,1,w));       % seasonal adjustment filter

figure; plot(k, Hs); figure; plot(k, Hi, k,Ht,'r--');             % lower graphs
figure; plot(k, Ha);                                              % right graph in Fig. 27.7.5
```

The impulse response definitions in this code implement Eq. (27.6.1) in the time domain. The upsampling function up was described in Sec. 27.2. The function compl computes the impulse response of the complement of a double-sided symmetric filter, that is, $H(z)\to 1-H(z)$, or $h_n \to \delta_n - h_n$. The function kwindow computes the Kaiser window (for spectral analysis) [224] for a given length N and sidelobe level R_{db} in dB, and it is part of the AOSP toolbox.

Fig. 27.7.5 illustrates the complementarity property more clearly by showing the seasonal adjustment filter $H_A(\omega)= 1-H_S(\omega)= H_T(\omega)+H_I(\omega)$, that is, the filter that removes the seasonal component from the data. As expected, the filter has nulls at the harmonics and is essentially flat in-between.

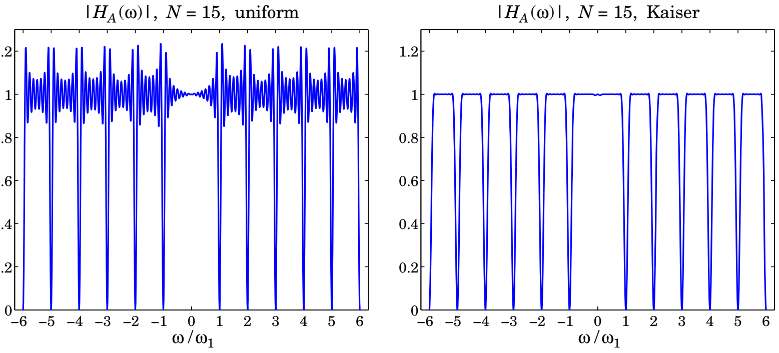


Fig. 27.7.5 Seasonal adjustment filter $H_A(\omega)=1-H_S(\omega)=H_T(\omega)+H_I(\omega)$.

27.8 Seasonal Moving-Average Filters

Signal averaging can be thought of as ordinary filtering by the seasonalized FIR averager filter of Eq. (27.5.3). However, as we saw in Eq. (27.5.9), the averaged period builds up gradually at the filter output and becomes available only as the last D output points. This is so because the filter length ND is essentially the same as the signal length so that the filter operates mostly in its transient state. Indeed, if L is the length of the signal y_n, the number of periods is $N=\text{floor}(L/D)$ so that $L\approx ND$.

In the classical decomposition method, the final accumulated period is replicated N times to make up the seasonal component s_n. This procedure is appropriate only if s_n is truly periodic. However, in many practical applications s_n is only quasi-periodic with slowly changing periods. In order to be able to estimate s_n more accurately we must use a shorter seasonal moving-average filter that tracks the local (i.e., within the filter's moving window) periodic component.

Example 27.8.1: Fig. 27.8.1 illustrates the filtering point of view for extracting the seasonal part s_n. The same CO$_2$ data are used as in Example 27.7.2. The classical decomposition method is applied first to determine the trend t_n, and then the residual signal is formed $r_n = y_n - t_n$. In this example, the number of periods contained in the y_n signal is $N=29$.

The upper-left graph shows the result of ordinary causal filtering of the residual signal r_n by the signal averaging comb filter (27.5.3) using MATLAB's built-in function filter. We observe that the transients eventually build up to the same final period as that obtained by signal averaging (shown as the dotted line.)

In the upper-right graph, the residual r_n was filtered by the double-sided filtering function filtdbl discussed in Sec. 23.10, which is ordinary causal convolution followed by advancing the result by $(N-1)D/2$ samples. Again, we observe the input-on and input-off transients and the build-up of the correct period at the middle.

The transient portions of the double-sided filter output can be adjusted by using Musgrave's minimum-revision asymmetric filters for the left and right end points. The resulting filter output is shown in the lower-left graph, in which the Musgrave parameter was chosen to be $R=\infty$ (see Sec. 27.10 for more on that.)

The lower-right graph shows the result of filtering r_n through a so-called 3×3 double-sided seasonal moving-average filter, which is discussed below. The MATLAB code for generating these graphs is as follows:

```
Y = loadfile('co2_mm_gl.dat');
t = Y(:,3); y = Y(:,4);                      % CO2 data

D=12; N=floor(length(y)/D); M=33; R=inf;     % filter parameters

[yt,ys,yi] = cldec(y,D,R); yr = y - yt;      % yr = residual component r_n

h = ones(1,N)/N;                             % length-N moving-average
hc = up(h, D);                               % seasonalized comb filter obtained from h
Bc = upmat(minrev(h,R), D);                  % seasonalized minimum-revision filter matrix

ys1 = filter(hc,1,yr);                       % ordinary causal filtering by the comb filter hc
ys2 = filtdbl(hc,yr);                        % double-sided filtering
ys3 = lpfilt(Bc, yr);                        % double-sided filtering and end-point filters
[yt4,ys4] = smadec(y, D, M, R);              % ys4 is the 3×3 moving-average output
```

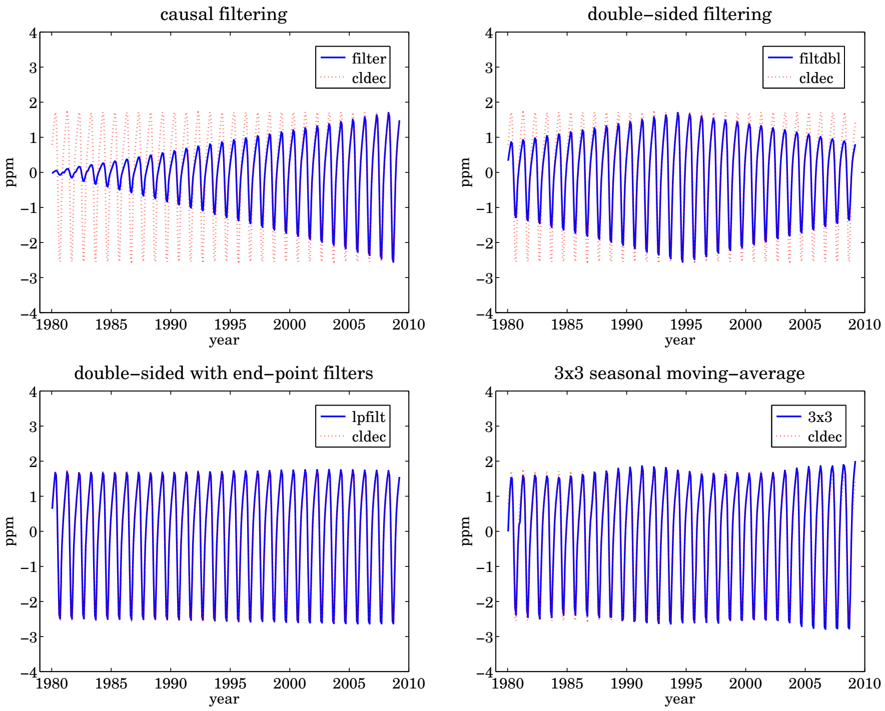


Fig. 27.8.1 Filtering versions of seasonal filter.

```
figure; plot(t,ys1, t,ys,':'); figure; plot(t,ys2, t,ys,':');     % upper graphs
figure; plot(t,ys3, t,ys,':'); figure; plot(t,ys4, t,ys,':');     % lower graphs
```

The `smadec` function is a simple alternative to `cldec` and is discussed below. The function `upmat` upsamples a filter matrix by a factor of D for its use in comb filtering. It upsamples each row and then each column by D and then, it replaces each group of D columns by the corresponding convolution matrix arising from the first column in each group. It can be passed directly into the filtering function `lpfilt`,

```
Bup = upmat(B,D);     % upsampling a filtering matrix
```

For example, the asymmetric filters associated with the 3×3 seasonal moving-average filter [1210] are as follows for $D = 3$, where the middle column is the 3×3 filter and the other columns, the asymmetric filters to be used at the ends of the data record, and the

function `smat` is described below:

$$
B = \text{smat}(1,33) = \frac{1}{27}\begin{bmatrix} 11 & 7 & 3 & 0 & 0 \\ 11 & 10 & 6 & 3 & 0 \\ 5 & 7 & 9 & 7 & 5 \\ 0 & 3 & 6 & 10 & 11 \\ 0 & 0 & 3 & 7 & 11 \end{bmatrix}
$$

$$
B_{\text{up}} = \text{upmat}(B,3) = \frac{1}{27}\begin{bmatrix} 11 & 0 & 0 & 7 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 11 & 0 & 0 & 7 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 11 & 0 & 0 & 7 & 0 & 0 & 3 & 0 & 0 & 0 & 0 \\ 11 & 0 & 0 & 10 & 0 & 0 & 6 & 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 11 & 0 & 0 & 10 & 0 & 0 & 7 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 11 & 0 & 0 & 10 & 0 & 0 & 7 & 0 & 0 & 5 & 0 \\ 5 & 0 & 0 & 7 & 0 & 0 & 9 & 0 & 0 & 7 & 0 & 0 & 5 \\ 0 & 5 & 0 & 0 & 7 & 0 & 0 & 10 & 0 & 0 & 11 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 & 7 & 0 & 0 & 10 & 0 & 0 & 11 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 & 6 & 0 & 0 & 10 & 0 & 0 & 11 \\ 0 & 0 & 0 & 0 & 3 & 0 & 0 & 7 & 0 & 0 & 11 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 7 & 0 & 0 & 11 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 7 & 0 & 0 & 11 \end{bmatrix} \tag{27.8.1}
$$

The simple 3×3, 3×5, and 3×9 seasonal moving-average filters are widely used in deseasonalizing business, government, and census data. They are obtained by symmetrizing the $N_1 \times N_2$ filters of Eq. (27.7.2) and then applying the transformation $z \to z^D$. For example, the resulting 3×3 and 3×5 comb filters are:

$$
\begin{aligned}
H_{33}(z) &= \frac{1}{3}(z^D + 1 + z^{-D})\cdot\frac{1}{3}(z^D + 1 + z^{-D}) \\
H_{35}(z) &= \frac{1}{3}(z^D + 1 + z^{-D})\cdot\frac{1}{5}(z^{2D} + z^D + 1 + z^{-D} + z^{-2D})
\end{aligned} \tag{27.8.2}
$$

with symmetric impulse responses,

$$
\begin{aligned}
\mathbf{h}_{33} &= \frac{1}{9}[1, \underbrace{0, \dots, 0}_{D-1\text{ zeros}}, 2, 0, \dots, 0, 3, 0, \dots, 0, 2, 0, \dots, 0, 1] \\
\mathbf{h}_{35} &= \frac{1}{15}[1, 2, 0, \dots, 0, 3, 0, \dots, 0, 3, 0, \dots, 0, 3, 0, \dots, 0, 2, 0, \dots, 0, 1]
\end{aligned} \tag{27.8.3}
$$

The MATLAB function `smav` calculates such impulse responses,

```
h = smav(N1,N2,D);     % seasonal moving-average filters
```

It is simply:

```
h  = up(conv(ones(1,N1), ones(1,N2))/(N1*N2), D);
```

These filters are to be applied to the residual signal $r_n = y_n - t_n$. Their end-point effects can be handled by using Musgrave's minimum-revision filters or by any other appropriate asymmetric filters. In fact, the census X-11/X-12 methods use asymmetric filters that are specially constructed for the 3×3, 3×5, and 3×9 filters, and may be found in Ref. [1210]. They have been incorporated into the `smadec` and `x11dec`. For example, Eq. (27.8.1) shows the 3×3 filter matrix before and after it is upsampled.

To summarize, the filtering approach for de-seasonalizing a signal $y_n = s_n + t_n + v_n$ with period D consists of the following two basic steps:

1. Apply a lowpass filter to extract the trend component t_n, incorporating also asymmetric end-point filters. The trend-extraction filter can be a simple $1\times D$ or $2\times D$ moving average, or, any other lowpass filter such as a local-polynomial or Whittaker-Henderson smoother.
2. Apply a comb filter to the de-trended residual signal $r_n = y_n - t_n$ to extract the seasonal part s_n, incorporating asymmetric filters for the end-points. The comb filter can be a simple seasonalized 3×3, 3×5, or 3×9 lowpass filter, or a more general seasonalized filter such as one obtained from a non-rectangular window. The de-seasonalized, or seasonally adjusted, signal is then $a_n = y_n - s_n$.

The MATLAB function `smadec` carries out this program using the simple $1\times D$ or $2\times D$ moving-average filter for de-trending and the 3×3, 3×5, or 3×9 comb filters for the seasonal part. It has usage:

```
[yt,ys,yi] = smadec(y,D,M,R,iter,type);      % seasonal moving-average decomposition
```

where `yt, ys, yi` are the estimated components t_n, s_n, v_n, and `y` is the input data vector. The integer values $M = 33, 35, 39$ select the 3×3, 3×5, or 3×9 seasonal comb filters, other values of M can also be used. The Musgrave parameter defaults to $R = \infty$, the parameter `iter` specifies the number of iterations of the filtering process, which correspond to applying the trend filter `iter` times. The string `type` takes on the values `'a'`, `'m'` for additive or multiplicative decomposition. To clarify the operations, we give below the essential part of the code in `smadec` for the additive case:

```
F = minrev(trendma(D),R);     % trend moving-average, with minimum-revision end-filters
B = smat(D,M,R);              % seasonal moving averages, with end-filters

yt = y;                       % initialize iteration
for i=1:iter,
  yt = lpfilt(F,yt);          % T component
  yr = y - yt;                % S+I component
  ys = lpfilt(B,yr);          % S component
  yi = yr - ys;               % I component
end
```

The function `smat` generates the filtering matrix of the seasonalized comb filters, including the specific asymmetric filters for the 3×3, 3×5, or 3×9 cases, as well for other cases.

Example 27.8.2: *Unemployment Data 1965–1979.* The data set representing the monthly number of unemployed 16–19 year old men for the period Jan. 1965 to Dec. 1979 has served as a benchmark for comparing seasonal adjustment methods [1225,1229]. The data set is available from the US Bureau of Labor Statistics web site: `http://www.bls.gov/data/` (series ID: `LNU03000013`, under category: Unemployment > Labor Force Statistics > on-screen data search).

The upper graphs of Fig. 27.8.2 illustrate the application of the `smadec` function using $D = 12$, $M = 35$ (which selects the 3×5 comb), one iteration, Musgrave parameter $R = \infty$ for the $2\times D$ trend filter, and additive decomposition type. The left graph shows the trend t_n and the right, the estimated seasonal component s_n, which is not exactly periodic but exhibits quasi-periodicity. The results are comparable to those of Refs. [1225,1229].

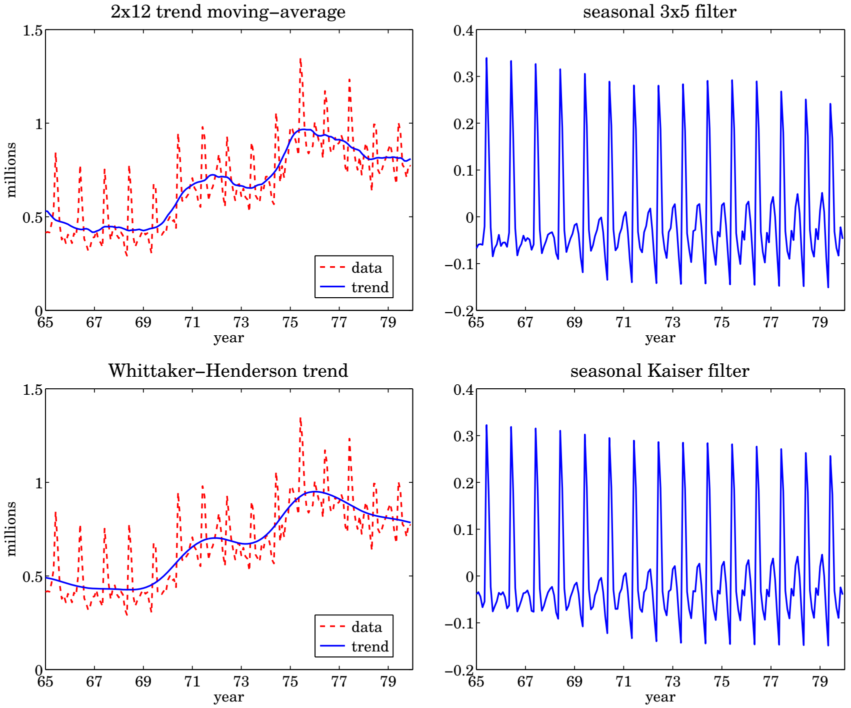


Fig. 27.8.2 Trend/seasonal decomposition of monthly unemployment data for 1965–1979.

The lower graphs show the decomposition obtained by de-trending using a Whittaker-Henderson smoother of order $s = 2$, followed by a Kaiser comb filter. The function `whgcv` was used to determine the optimum smoothing parameter, $\lambda_{\text{opt}} = 2039$. The Kaiser window had length $N = 15$ and relative sidelobe level of $R_{\text{db}} = 50$ dB. The MATLAB code for generating these graphs was as follows:

```
Y = loadfile('unemp-1619-nsa.dat');             % data file in AOSP toolbox
i=find(Y==1965); Y = Y(i:i+14,2:13)';           % extract 1965-1979 data
y = Y(:)/1000; t = taxis(y,12,65);              % y units in millions

D=12; M=35; R=inf; iter=1; type='a';            % smadec input parameters

[yt,ys,yi] = smadec(y,D,M,R,iter,type);         % yt,ys represent t_n, s_n
```

```
figure; plot(t,y, t,yt); figure; plot(t,ys);          % upper graphs

s = 2; la = 2000:2050;                                 % search range of λ's
[gcv,lopt] = whgcv(y,la,s);                            % optimum λopt = 2039

yt = whsm(y,lopt,s);                                   % extract tn component
yr = y-yt;                                             % residual S+I component

Rdb=50; N=15; h = kwindow(N,Rdb); hk = h/sum(h);      % Kaiser window
B = upmat(minrev(hk,R), D);                            % Kaiser comb with end-filters

ys = lpfilt(B, yr);                                    % extract sn component

figure; plot(t,y, t,yt); figure; plot(t,ys);          % bottom graphs
```

The Whittaker-Henderson method results in a smoother trend. However, the trend from `smadec` can be made equally smooth by increasing the number of iterations, for example, setting `iter=3`. The frequency responses of the various filters are shown in Eq. (27.8.3).

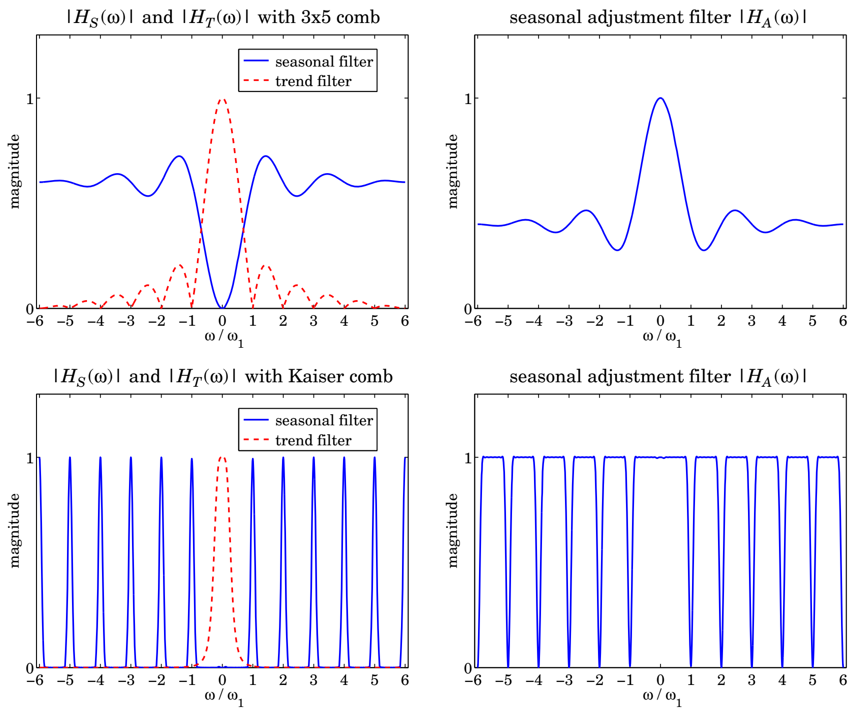


Fig. 27.8.3 Frequency responses of trend, seasonal, and seasonal-adjustment filters.

The filters $H_T(\omega), H_S(\omega)$ for extracting the trend and seasonal components, and the seasonal-adjustment filter $H_A(\omega) = 1 - H_S(\omega)$ are constructed from Eq. (27.6.1). The MATLAB code for generating these graphs was:

```
k = linspace(-6,6,1201); w = 2*pi*k/12;               % frequency range [−π, π]

ht = trendma(12);          Ht = abs(freqz(ht,1,w));   % 2×12 trend filter
hc = smav(3,5,12);                                     % upsampled 3×5 comb filter
hs = conv(hc,compl(ht));   Hs = abs(freqz(hs,1,w));   % seasonal filter, HS(ω)
ha = compl(hs);            Ha = abs(freqz(ha,1,w));   % adjustment filter, 1 − HS(ω)

figure; plot(k,Hs, k,Ht,'--'); figure; plot(k,Ha);    % upper graphs

Ht = 1 ./ (1 + lopt * (2*sin(w/2)).^(2*s));           % Whittaker-Henderson trend filter
hc = up(hk,12);                                        % Kaiser comb impulse response
Hc = freqz(hc,1,w) .* exp(j*(N-1)*D*w/2);             % Kaiser comb frequency response
Hs = Hc .* (1-Ht); Ha = 1 - Hs;                       % HS(ω) and HA(ω) = 1 − HS(ω)
Hs = abs(Hs); Ha = abs(Ha);

figure; plot(k,Hs, k,Ht,'--'); figure; plot(k,Ha);     % bottom graphs
```

The Whittaker-Henderson trend filter was computed using Eq. (26.3.7). The frequency response of the Kaiser comb filter was multiplied by $e^{j\omega(N-1)D/2}$ to make the filter symmetric. It is evident that the WH/Kaiser filters perform better. □

The steps implementing the Whittaker-Henderson/Kaiser decomposition have been incorporated into the MATLAB function `whkdec`,

```
[yt,ys,yi] = whkdec(y,D,s,la,N,Rdb,R,type);     % WH/Kaiser decomposition
```

The WH parameters `s,la` must be selected in advance, for example, λ can be tentatively estimated using the GCV function `whgcv`, but it should be noted that the GCV does not always give a "good" value for λ. The Kaiser window length `N` must be odd and the sidelobe level must be restricted to the range $[13, 120]$ dB. The Musgrave parameter R affects only the Kaiser comb filter because the WH trend already takes into account the end points. The parameter `type` is as in the function `smadec`.

27.9 Census X-11 Decomposition Filters

The Census X-11/X-12 seasonal adjustment procedures have become a standard for de-seasonalizing economic data [1197–1213]. They are based on a series of filtering operations that represent a refined version of the procedures outlined in the previous section.

Here, we only discuss the relevant filtering operations, leaving out details such as adjustments for outliers or calendar effects. The most recent version, X-12-ARIMA, is available from the web site [1199]. The web pages [1200,1201] contain a number of papers on the development of the X-11/X-12 methods.

As outlined in [1202,1205], the X-11 method involves the repeated application of the 2×12 trend filter of Eq. (27.7.1), the 3×3 and 3×5 comb filters of Eq. (27.8.2), and the Henderson filters of lengths 9, 13, or 25, with polynomial and smoothing orders $d = s = 3$ given by Eq. (11.1.1) of Sec. 23.12. The basic X-11 filtering steps are as follows, assuming an additive model $y_n = s_n + t_n + v_n$,

1. Apply a 2×12 trend filter to y_n to get a preliminary estimate of the trend t_n.

2. Subtract t_n from y_n to get a preliminary estimate of the residual $r_n = y_n - t_n$.
3. Apply the 3×3 comb filter to r_n to get a preliminary estimate of s_n.
4. Get an improved s_n by removing its filtered version by the 2×12 trend filter.
5. Subtract s_n from y_n to get a preliminary adjusted signal $a_n = y_n - s_n = t_n + v_n$.
6. Filter a_n by a Henderson filter to get an improved estimate of the trend t_n.
7. Subtract t_n from y_n to get an improved residual $r_n = y_n - t_n$.
8. Apply the 3×5 comb filter to r_n to get an improved estimate of s_n.
9. Get the final s_n by removing its filtered version by the 2×12 trend filter.
10. Subtract s_n from y_n to get the final adjusted signal $a_n = y_n - s_n = t_n + v_n$.
11. Filter a_n by a Henderson filter to get the final estimate of the trend t_n.
12. Subtract t_n from a_n to get the final estimate of the irregular component v_n.

These steps are for monthly data. For quarterly data, replace the 2×12 trend filter by a 2×4 filter. The steps can be expressed concisely in the z-domain as follows:

$$
\begin{array}{rl}
1. & Y_T^{\rm pre} = FY \\
2. & Y_R^{\rm pre} = Y - Y_T^{\rm pre} = (1-F)Y \\
3. & Y_S^{\rm pre} = H_{33}Y_R^{\rm pre} = H_{33}(1-F)Y \\
4. & Y_S^{\rm imp} = Y_S^{\rm pre} - FY_S^{\rm pre} = H_{33}(1-F)^2Y \\
5. & Y_A^{\rm pre} = Y - Y_S^{\rm imp} = \big[1 - H_{33}(1-F)^2\big]Y \\
6. & Y_T^{\rm imp} = HY_A^{\rm pre} = H\big[1 - H_{33}(1-F)^2\big]Y \\
7. & Y_R^{\rm imp} = Y - Y_T^{\rm imp} = \Big[1 - H\big[1 - H_{33}(1-F)^2\big]\Big]Y \\
8. & Y_S^{\rm imp} = H_{35}Y_R^{\rm imp} = H_{35}\Big[1 - H\big[1 - H_{33}(1-F)^2\big]\Big]Y \\
9. & Y_S = Y_S^{\rm imp} - FY_S^{\rm imp} = (1-F)H_{35}\Big[1 - H\big[1 - H_{33}(1-F)^2\big]\Big]Y \equiv H_SY \\
10. & Y_A = Y - Y_S = (1-H_S)Y \equiv H_AY \\
11. & Y_T = HY_A = H(1-H_S)Y \equiv H_TY \\
12. & Y_I = Y_A - Y_T = (1-H)(1-H_S)Y \equiv H_IY
\end{array}
\qquad (27.9.1)
$$

where $Y_T^{\rm pre} = FY$ stands for $Y_Y^{\rm pre}(z) = F(z)Y(z)$, etc., and $F(z), H_{33}(z), H_{35}(z), H(z)$ denote the 2×12 trend filter, the 3×3 and 3×5 comb filters, and the Henderson filter, and the z-transforms of the data, trend, seasonal, adjusted, and irregular components are denoted by $Y(z)$, $Y_T(z)$, $Y_S(z)$, $Y_A(z)$, and $Y_I(z)$.

It follows that the effective filters for extracting the seasonal, seasonally-adjusted, trend, and irregular components are:

$$
\begin{array}{ll}
H_S = (1-F)H_{35}\Big[1 - H\big[1 - H_{33}(1-F)^2\big]\Big] & \text{(seasonal)} \\
H_A = 1 - H_S & \text{(seasonally-adjusted)} \\
H_T = H(1-H_S) & \text{(trend)} \\
H_I = (1-H)(1-H_S) & \text{(irregular)}
\end{array}
\qquad (27.9.2)
$$

They satisfy the complementarity property $H_T(z) + H_S(z) + H_I(z) = 1$.

Example 27.9.1: *X-11 Filters.* The construction of the time-domain impulse responses of the X-11 decomposition filters (27.9.2) is straightforward. For example, the following MATLAB code evaluates the impulse responses (using a 13-term Henderson filter), as well as the corresponding frequency responses shown in Fig. 27.9.1,

```
k = linspace(-6,6,1201); w = 2*pi*k/12;     % frequency axis −π ≤ ω ≤ π

hf  = trendma(12);                               % 2×12 trend filter, F
hfc = compl(hf);                                 % complement of trend filter, 1 − F
h33 = smav(3,3,12);                              % 3×3 comb filter, H33
h35 = smav(3,5,12);                              % 3×5 comb filter, H35
N=13; he = lprs2(N,3,3);                         % 13-term Henderson filter, H
g = conv(hfc,hfc);                               % G = (1 − F)^2, G is temporary variable
g = compl(conv(h33, g));                         % G = 1 − H33 (1 − F)^2
g = compl(conv(he,g));                           % G = 1 − H[1 − H33 (1 − F)^2]
g = conv(h35,g);                                 % G = H35 {1 − H[1 − H33 (1 − F)^2]}
                                                 % HS = (1 − F)H35 {1 − H[1 − H33 (1 − F)^2]}
hs = conv(hfc,g);          Hs = abs(freqz(hs,1,w));     % seasonal
ha = compl(hs);            Ha = abs(freqz(ha,1,w));     % adjustment
ht = conv(he,ha);          Ht = abs(freqz(ht,1,w));     % trend
hi = conv(compl(he),ha);   Hi = abs(freqz(hi,1,w));     % irregular

figure; plot(k, Hs); figure; plot(k, Ht);               % upper graphs
figure; plot(k, Ha); figure; plot(k, Hi);               % lower graphs
```

We note that the filters have the expected shapes. All cases described in [1205] can be generated by variations of this code. □

The MATLAB function `x11dec` implements the above steps, amended by the use of asymmetric filters to handle the end-points of the time series,

```
[yt,ys,yi] = x11dec(y,D,M1,M2,N1,N2,R,type);     % X-11 decomposition method
```

where `D` is the seasonal period, `M1,M2` the sizes of the first and second comb filters (entered as 33, 35, or 39), `N1,N2` are the lengths of the first and second Henderson filters, `R` is the Musgrave minimum-revision parameter affecting both the Henderson filters and the trend filter, and `type` designates an additive or multiplicative decomposition. The Musgrave parameter R is usually assigned the following values, depending on the length N of the Henderson filter [1210]:

N	R
5	0.001
7	4.5
9	1.0
13	3.5
23	4.5

(27.9.3)

Example 27.9.2: *Unemployment Data 1980–2008.* Fig. 27.9.2 shows the X-11 decomposition of the monthly unemployment data for 20 year and older men for the period Jan. 1980 to Dec. 2008. The data are from the US BLS web site: `http://www.bls.gov/data/`, series `LNU03000025`, under category: Unemployment > Labor Force Statistics > on-screen data search. The already seasonally adjusted data are also available as series `LNS13000025`.

The upper-left graph shows the original data and the extracted trend t_n assuming an additive model $y_n = t_n + s_n + v_n$. The lower-left graph is the extracted seasonal component s_n.

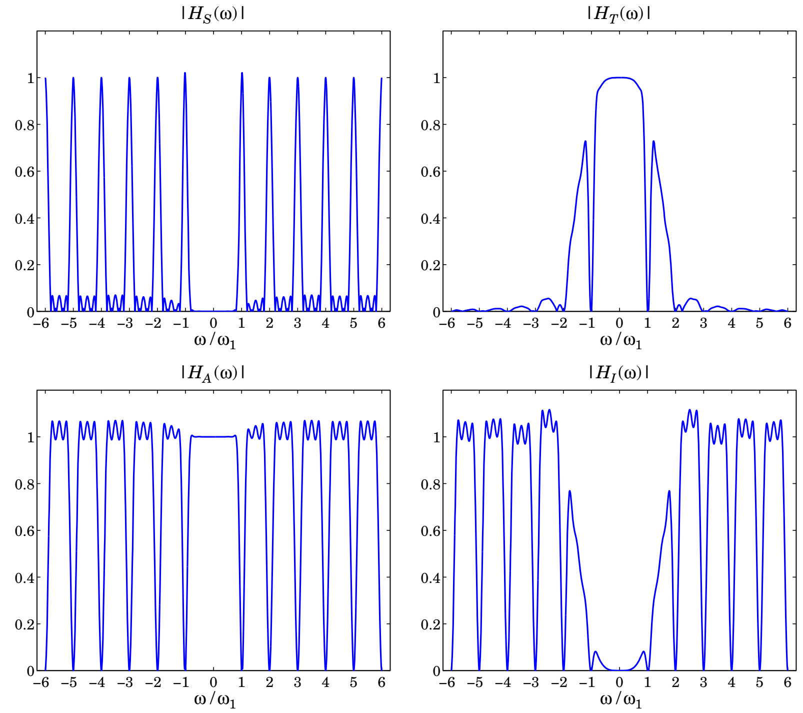


Fig. 27.9.1 X-11 decomposition filters (with 13-term Henderson).

The upper-right graph shows the seasonally-adjusted signal $a_n = y_n - s_n = t_n + v_n$ to be compared with that of the lower-right graph, which shows the already available adjusted signal— the two agreeing fairly well. The graphs were generated by the following code:

```
Y = loadfile('unemp-20-nsa.dat');               % not-seasonally adjusted data
i = find(Y==1980); Y = Y(i:end,2:13)';          % select years 1980-2008
y = Y(:)/1000; t = taxis(y,12,1980);            % data vector y, and time axis
                                                % data sets available in the AOSP toolbox
Y = loadfile('unemp-20-sa.dat');                % seasonally adjusted data
i = find(Y==1980); Y = Y(i:end,2:13)';
yadj = Y(:)/1000;                               % yadj = already available adjusted data

D=12; M1=33; M2=35; N1=13; N2=13; R=3.5; type='a';      % X-11 parameters
[yt,ys,yi] = x11dec(y,D,M1,M2,N1,N2,R,type);            % X-11 method
ya = y-ys;                                              % seasonally adjusted

%s = 2; la = 1000; N=15; Rdb=50;                      % WH/K parameters
%[yt,ys,yi] = whkdec(y,D,s,la,N,Rdb,R,type);          % Whittaker-Henderson/Kaiser
%ya = y-ys;                                           % seasonally adjusted

figure; plot(t,y,':', t,yt,'-'); figure; plot(t,ya);    % upper graphs
figure; plot(t,ys); figure; plot(t,yadj);               % lower graphs
```

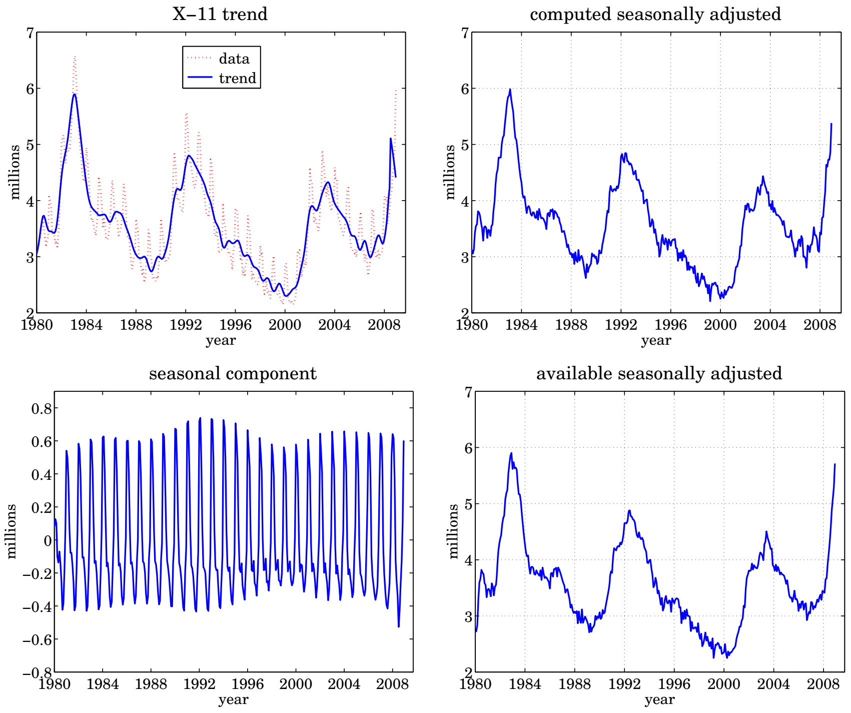


Fig. 27.9.2 X-11 decomposition of unemployment data 1980-2008.

The value of R was 3.5 because a 13-term Henderson filter was used. The purpose of this example was to compare the performance of our simplified X-11 implementation with the results that are already available from the Bureau of Labor Statistics. We note that the use of the Whittaker-Henderson/Kaiser decomposition method also works comparably well, for example with parameters $s = 2$, $\lambda = 1000$, Kaiser length $N = 15$, and $R_{\text{db}} = 50$ dB. The code for that is included above but it is commented out. □

27.10 Musgrave Asymmetric Filters

The handling of the end-point problem by the use of asymmetric filters was discussed in Sec. 23.10. We saw that the output y_n of filtering a length-L signal x_n, $0 \le n \le L-1$, by a double-sided filter h_m, $-M \le m \le M$, using for example the function `filtdbl`, consists of M initial and M final transient output samples, and $L-2M$ steady-state samples, the latter being computed by the steady-state version of the convolutional equation:

$$y_n = \sum_{m=-M}^{M} h_m x_{n-m}\,, \quad M \le n \le L-1-M \tag{27.10.1}$$

The overall operation can be cast in convolution matrix form. For example, for $L = 8$ and $M = 2$, we have:

$$\begin{bmatrix} y_0 \\ y_1 \\ \cdots \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ \cdots \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} h_0 & h_{-1} & h_{-2} & 0 & 0 & 0 & 0 & 0 \\ h_1 & h_0 & h_{-1} & h_{-2} & 0 & 0 & 0 & 0 \\ \hdotsfor{8} \\ h_2 & h_1 & h_0 & h_{-1} & h_{-2} & 0 & 0 & 0 \\ 0 & h_2 & h_1 & h_0 & h_{-1} & h_{-2} & 0 & 0 \\ 0 & 0 & h_2 & h_1 & h_0 & h_{-1} & h_{-2} & 0 \\ 0 & 0 & 0 & h_2 & h_1 & h_0 & h_{-1} & h_{-2} \\ \hdotsfor{8} \\ 0 & 0 & 0 & 0 & h_2 & h_1 & h_0 & h_{-1} \\ 0 & 0 & 0 & 0 & 0 & h_2 & h_1 & h_0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ \cdots \\ x_6 \\ x_7 \end{bmatrix} \tag{27.10.2}$$

The middle $L - 2M = 4$ output samples are steady, while the first and last $M = 2$ are transient and are computed by using fewer filter weights that the steady ones. The transient and steady filters can be arranged into a matrix B, which is for the above example,

$$B = \begin{bmatrix} h_0 & h_1 & h_2 & 0 & 0 \\ h_{-1} & h_0 & h_1 & h_2 & 0 \\ h_{-2} & h_{-1} & h_0 & h_1 & h_2 \\ 0 & h_{-2} & h_{-1} & h_0 & h_1 \\ 0 & 0 & h_{-2} & h_{-1} & h_0 \end{bmatrix} \tag{27.10.3}$$

The convolution matrix H can be built from the knowledge of B as described in Sec. 23.10. The matrix B conveniently summarizes the relevant filters and can be used as an input to the filtering function `lpfilt`.

As discussed in Sec. 23.10, local polynomial smoothing filters, including Henderson filters, generate their own matrix B to handle the series end-points, with the non-central columns of B consisting of the corresponding prediction filters.

However, when one does not have available such prediction filters, but only the central filter h_m, $-M \le m \le M$, one must use appropriately designed end-point filters. For example, Eq. (27.10.2) would change to:

$$\begin{bmatrix} y_0 \\ y_1 \\ \cdots \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ \cdots \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} f_0^0 & f_{-1}^0 & f_{-2}^0 & 0 & 0 & 0 & 0 & 0 \\ f_1^1 & f_0^1 & f_{-1}^1 & f_{-2}^1 & 0 & 0 & 0 & 0 \\ \hdotsfor{8} \\ h_2 & h_1 & h_0 & h_{-1} & h_{-2} & 0 & 0 & 0 \\ 0 & h_2 & h_1 & h_0 & h_{-1} & h_{-2} & 0 & 0 \\ 0 & 0 & h_2 & h_1 & h_0 & h_{-1} & h_{-2} & 0 \\ 0 & 0 & 0 & h_2 & h_1 & h_0 & h_{-1} & h_{-2} \\ \hdotsfor{8} \\ 0 & 0 & 0 & 0 & g_2^1 & g_1^1 & g_0^1 & g_{-1}^1 \\ 0 & 0 & 0 & 0 & 0 & g_2^0 & g_1^0 & g_0^0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ \cdots \\ x_6 \\ x_7 \end{bmatrix} \tag{27.10.4}$$

where the filters f_m^0 and f_m^1 are used for computing the first two transient outputs y_0, y_1, and the filters g_m^0 and g_m^1 are for the last two outputs y_7, y_6. The corresponding B matrix

would be in this case:

$$B = \begin{bmatrix} f_0^0 & f_1^1 & h_2 & 0 & 0 \\ f_{-1}^0 & f_0^1 & h_1 & g_2^1 & 0 \\ f_{-2}^0 & f_{-1}^1 & h_0 & g_1^1 & g_2^0 \\ 0 & f_{-2}^1 & h_{-1} & g_0^1 & g_1^0 \\ 0 & 0 & h_{-2} & g_{-1}^1 & g_0^0 \end{bmatrix} \tag{27.10.5}$$

More generally, the filters f_m^i , g_m^i , $i = 0, 1, \dots, M - 1$, compute the first M and last M output samples y_i , y_{L-1-i} , respectively, through the convolutional equations:

$$\begin{aligned} y_i &= \sum_{m=-M}^{i} f_m^i x_{i-m} = \sum_{m=-i}^{M} f_{-m}^i x_{i+m} = f_i^i x_0 + \cdots + f_0^i x_i + \cdots + f_{-M}^i x_{i+M} \\ y_{L-1-i} &= \sum_{m=-i}^{M} g_m^i x_{L-1-i-m} = g_M^i x_{L-1-i-M} + \cdots + g_0^i x_{L-1-i} + \cdots + g_{-i}^i x_{L-1} \end{aligned} \tag{27.10.6}$$

for $i = 0, 1, \dots, M - 1$, where the limits of summations follow by the requirement that only available x_n samples appear in the sums.

Musgrave's method [1207,1208] constructs such asymmetric filters from the knowledge only of the central filter h_m. The construction applies to filters h_m that are symmetric, $h_m = h_{-m}$, and are normalized to unity gain at DC, such as lowpass trend filters,

$$\sum_{m=-M}^{M} h_m = 1 \tag{27.10.7}$$

The asymmetric filters f_m^i, g_m^i are required to satisfy similar moment constraints:

$$\sum_{m=-M}^{i} f_m^i = 1\,, \quad \sum_{m=-i}^{M} g_m^i = 1 \tag{27.10.8}$$

The design is based on a minimum-revision criterion. When the data record has length L, the ith output from the end, y_{L-1-i}, is computed with the filter g_m^i. If or when the series is extended to length $L - 1 + M$, then the same output can actually be computed with the symmetric filter h_m resulting in a revised output y_{L-1-i}^{rev}, that is,

$$y_{L-1-i} = \sum_{m=-i}^{M} g_m^i x_{L-1-i-m}\,, \quad y_{L-1-i}^{\text{rev}} = \sum_{m=-M}^{M} h_m x_{L-1-i-m}$$

Musgrave's criterion selects g_m^i to minimize the mean-square revision error $E[e_{L-1-i}^2]$, where $e_{L-1-i} = y_{L-1-i} - y_{L-1-i}^{\text{rev}}$, under the assumption that locally the input series is linear, that is, $x_{L-1-i-m} = a + bm + v_m$, with a, b constant parameters, and v_m zero-mean white noise with variance σ^2. The mean-square error becomes then,

$$\begin{aligned} E[e_{L-1-i}^2] &= E\Big[\big[\sum_{m=-i}^{M} g_m^i(a+bm+\nu_m) - \sum_{m=-M}^{M} h_m(a+bm+\nu_m)\big]^2\Big] \\ &= E\Big[\big[b\sum_{m=-i}^{M} m g_m^i + \sum_{m=-i}^{M}(g_m^i - h_m)\nu_m - \sum_{m=-M}^{-i-1} h_m\nu_m\big]^2\Big] \\ &= \sigma^2\sum_{m=-i}^{M}(g_m^i - h_m)^2 + b^2\Big(\sum_{m=-i}^{M} m g_m^i\Big)^2 + \text{const.} \end{aligned} \tag{27.10.9}$$

where "const." is a positive term independent of g_m^i. In deriving this, we used the moment constraints (27.10.7) and (27.10.8), and the property $\sum_{m=-M}^{M} m h_m = 0$, which follows from the assumed symmetry of h_m. Defining the constant $\beta^2 = b^2/\sigma^2$, it follows that the optimum filter g_m^i will be the solution of the following optimization criterion, which incorporates the constraint (27.10.8) by means of a Lagrange multiplier λ:

$$\mathcal{J} = \sum_{m=-i}^{M}(g_m - h_m)^2 + \beta^2\Big(\sum_{m=-i}^{M} m g_m\Big)^2 + \lambda\Big(1 - \sum_{m=-i}^{M} g_m\Big) = \min \tag{27.10.10}$$

In a similar fashion, we can show that the filters f_m^i are the solutions of

$$\mathcal{J} = \sum_{m=-M}^{i}(f_m - h_m)^2 + \beta^2\Big(\sum_{m=-M}^{i} m f_m\Big)^2 + \lambda\Big(1 - \sum_{m=-M}^{i} f_m\Big) = \min \tag{27.10.11}$$

Because h_m is even in m it follows (by changing variables $m \to -m$ in the sums) that $f_m^i = g_{-m}^i$, that is, the beginning filters are the reverse of the end filters. Thus, only g_m^i need be determined and is found to be [1208]:

$$g_m^i = h_m + \frac{A_i}{M+i+1} + \frac{\beta^2 B_i}{D_i}(m - \mu_i)\,, \quad -i \le m \le M \tag{27.10.12}$$

for $i = 0, 1, \dots, M-1$, with the constants A_i, B_i, D_i, μ_i defined by,

$$\begin{aligned} &A_i = \sum_{m=-M}^{-i-1} h_m\,, \quad B_i = \sum_{m=-M}^{-i-1}(m - \mu_i)h_m\,, \quad i = 0, 1, \dots, M-1 \\ &\mu_i = \frac{M-i}{2}\,, \quad D_i = 1 + \frac{\beta^2}{12}(M+i)(M+i+1)(M+i+2) \end{aligned} \tag{27.10.13}$$

To show Eq. (27.10.12), we set the gradient of $\mathcal{J}$ in (27.10.10) to zero, $\partial\mathcal{J}/\partial g_m = 0$, to get,

$$g_m = h_m + \lambda - \beta^2 G m\,, \quad G = \sum_{m=-i}^{M} m g_m \tag{27.10.14}$$

Summing up over m and using the constraint (27.10.8), and then, multiplying by m and summing up over m, results in two equations for the two unknowns λ, G:

$$\begin{aligned} 1 &= \sum_{m=-i}^{M} h_m + \Big(\sum_{m=-i}^{M} 1\Big)\lambda - \Big(\sum_{m=-i}^{M} m\Big)\beta^2 G \\ G &= \sum_{m=-i}^{M} m h_m + \Big(\sum_{m=-i}^{M} m\Big)\lambda - \Big(\sum_{m=-i}^{M} m^2\Big)\beta^2 G \end{aligned} \tag{27.10.15}$$

Using the properties,

$$1 - \sum_{m=-i}^{M} h_m = \sum_{m=-M}^{-i-1} h_m\,, \quad \sum_{m=-i}^{M} m h_m = -\sum_{m=-M}^{-i-1} m h_m$$

and the identities,

$$\begin{aligned} &\sum_{m=-i}^{M} 1 = M + i + 1 \\ &\sum_{m=-i}^{M} m = \frac{1}{2}(M+i+1)(M-i) = (M+i+1)\mu_i \\ &\sum_{m=-i}^{M} m^2 = (M+i+1)\mu_i^2 + \frac{1}{12}(M+i)(M+i+1)(M+i+2) \end{aligned} \tag{27.10.16}$$

and solving Eqs. (27.10.15) for the constants λ, G and substituting them in (27.10.14), gives the solution (27.10.12). The parameter β is usually computed in terms of the Musgrave parameter R, the two being related by

$$R^2 = \frac{4}{\pi\beta^2} \quad \Rightarrow \quad \beta^2 = \frac{4}{\pi R^2} \tag{27.10.17}$$

The MATLAB function `minrev` implements Eq. (27.10.12) and arranges the asymmetric filters into a filtering matrix B, which can be passed into the filtering function `lpfilt`,

```
B = minrev(h,R);          % minimum-revision asymmetric filters
```

The input is any odd-length symmetric filter h_m and the parameter R. Typical values of R are given in Eq. (27.9.3). The value $R = \infty$ corresponds to slope $\beta = 0$. For $R = 0$ or $\beta = \infty$, the limit of the solution (27.10.12) is ignored and, instead, the function `minrev` generates the usual convolutional transients for the filter h_m, resulting in a matrix B such that in Eqs. (27.10.3).

We have made extensive use of this function since Chap. 23. As a further example, we compare the filtering matrix B for a 7-term Henderson filter resulting from `minrev` with the standard value $R = 4.5$ to that resulting from the function `lprs` using the corresponding prediction filters for the same Henderson filter:

$$h = \mathrm{lprs2}(7,3,3) = [-0.0587,\ 0.0587,\ 0.2937,\ 0.4126,\ 0.2937,\ 0.0587,\ -0.0587]$$

$$B = \mathrm{minrev}(h,4.5) = \begin{bmatrix} 0.5345 & 0.2892 & 0.0336 & -0.0587 & 0 & 0 & 0 \\ 0.3833 & 0.4103 & 0.2747 & 0.0587 & -0.0531 & 0 & 0 \\ 0.1160 & 0.2937 & 0.3997 & 0.2937 & 0.0582 & -0.0542 & 0 \\ -0.0338 & 0.0610 & 0.2870 & 0.4126 & 0.2870 & 0.0610 & -0.0338 \\ 0 & -0.0542 & 0.0582 & 0.2937 & 0.3997 & 0.2937 & 0.1160 \\ 0 & 0 & -0.0531 & 0.0587 & 0.2747 & 0.4103 & 0.3833 \\ 0 & 0 & 0 & -0.0587 & 0.0336 & 0.2892 & 0.5345 \end{bmatrix}$$

$$B = \mathrm{lprs}(7,3,3) = \begin{bmatrix} 0.8182 & 0.1836 & -0.0587 & -0.0587 & 0.0336 & 0.0682 & -0.1049 \\ 0.4895 & 0.4510 & 0.2741 & 0.0587 & -0.0951 & -0.0874 & 0.1818 \\ -0.2448 & 0.4283 & 0.5245 & 0.2937 & -0.0140 & -0.1486 & 0.1399 \\ -0.2797 & 0.1049 & 0.3357 & 0.4126 & 0.3357 & 0.1049 & -0.2797 \\ 0.1399 & -0.1486 & -0.0140 & 0.2937 & 0.5245 & 0.4283 & -0.2448 \\ 0.1818 & -0.0874 & -0.0951 & 0.0587 & 0.2741 & 0.4510 & 0.4895 \\ -0.1049 & 0.0682 & 0.0336 & -0.0587 & -0.0587 & 0.1836 & 0.8182 \end{bmatrix}$$

The mean-square revision error (27.10.9) was calculated assuming a local linear function of time for the input. The criterion can be generalized to higher-order polynomials. For example, for a second-order polynomial,

$$x_{L-1-i-m} = a + bm + cm^2 + v_m$$

the mean-square revision error will be:

$$\begin{aligned} E[e_{L-1-i}^2] &= E\Big[\big[\sum_{m=-i}^{M} g_m(a+bm+cm^2+v_m) - \sum_{m=-M}^{M} h_m(a+bm+cm^2+v_m)\big]^2\Big] \\ &= \sigma^2 \sum_{m=-i}^{M}(g_m-h_m)^2 + c^2\Big(\sum_{m=-i}^{M} m^2 g_m\Big)^2 + \text{const.} \end{aligned} \tag{27.10.18}$$

where we assumed that h_m is symmetric, has unity gain at DC, and zero second moment (i.e., it reproduces second-order polynomials). Similarly, we assumed that g_m has unity gain at DC and zero first moment (so that it reproduces first-order polynomials). Thus, the expression (27.10.18) was obtained under the constraints:

$$h_m = h_{-m}\,, \quad \sum_{m=-M}^{M}\begin{bmatrix}1\\ m\\ m^2\end{bmatrix} h_m = \begin{bmatrix}1\\0\\0\end{bmatrix}, \quad \sum_{m=-i}^{M}\begin{bmatrix}1\\ m\end{bmatrix} g_m = \begin{bmatrix}1\\0\end{bmatrix} \tag{27.10.19}$$

Defining $\gamma^2 = c^2/\sigma^2$, we obtain the following optimization criterion, which incorporates the above constraints on g_m with two Lagrange multipliers λ_1, λ_2:

$$\mathcal{J} = \sum_{m=-i}^{M}(g_m-h_m)^2 + \gamma^2\Big(\sum_{m=-i}^{M} m^2 g_m\Big)^2 + \lambda_1\Big(1-\sum_{m=-i}^{M} g_m\Big) - \lambda_2\Big(\sum_{m=-i}^{M} m g_m\Big) \tag{27.10.20}$$

The vanishing of the gradient gives:

$$g_m = h_m + \lambda_1 + \lambda_2 m - \gamma^2 G m^2\,, \quad G = \sum_{m=-i}^{M} m^2 g_m \tag{27.10.21}$$

By multiplying by m^0, m^1, m^2 and summing up over m, we obtain three equations for the three unknowns λ_1, λ_2, G,

$$\begin{aligned} 1 &= \sum_{m=-i}^{M} h_m + \Big(\sum_{m=-i}^{M} 1\Big)\lambda_1 + \Big(\sum_{m=-i}^{M} m\Big)\lambda_2 - \Big(\sum_{m=-i}^{M} m^2\Big)\gamma^2 G \\ 0 &= \sum_{m=-i}^{M} m h_m + \Big(\sum_{m=-i}^{M} m\Big)\lambda_1 + \Big(\sum_{m=-i}^{M} m^2\Big)\lambda_2 - \Big(\sum_{m=-i}^{M} m^3\Big)\gamma^2 G \\ G &= \sum_{m=-i}^{M} m^2 h_m + \Big(\sum_{m=-i}^{M} m^2\Big)\lambda_1 + \Big(\sum_{m=-i}^{M} m^3\Big)\lambda_2 - \Big(\sum_{m=-i}^{M} m^4\Big)\gamma^2 G \end{aligned} \tag{27.10.22}$$

Substituting the solutions for λ_1, λ_2, G into (27.10.21) gives the solution:

$$g_m^i = h_m + \frac{A_i}{M+i+1} + \frac{B_i}{\Sigma_i}(m-\mu_i) + \frac{\gamma^2 C_i}{\Delta_i}\big[(m-\mu_i)^2 - v_i^2\big]\,, \quad -i \le m \le M \tag{27.10.23}$$

for $i = 0, 1, \dots, M-1$, with the same constants A_i, B_i, μ_i as in Eq. (27.10.13), and with $\Sigma_i, \Delta_i, v_i, C_i$ are defined by

$$\begin{aligned} \Sigma_i &= \frac{1}{12}(M+i)(M+i+1)(M+i+2)\,, \quad v_i^2 = \frac{1}{12}(M+i)(M+i+2) \\ \Delta_i &= 1 + \frac{\gamma^2}{180}(M+i-1)(M+i)(M+i+1)(M+i+2) \\ C_i &= \sum_{m=-M}^{-i-1}\big[(m-\mu_i)^2 - v_i^2\big]h_m \end{aligned} \tag{27.10.24}$$

27.11 Seasonal Whittaker-Henderson Decomposition

There are several other seasonal decomposition methods. The Holt-Winters exponential smoothing method [823–825], which was briefly discussed in Eq. (24.13.6), is a simple, effective, method of simultaneously tracking trend and seasonal components.

Another method is based on a seasonal generalization of the Whittaker-Henderson method [1214–1217] and we discuss it a more detail in this section.

Model-based methods of seasonal adjustment [1218–1234] are widely used and are often preferred over the X-11/X-12 methods. They are based on making ARIMA-type models for the trend and seasonal components and then estimating the components using optimum Wiener filters, or their more practical implementation as Kalman filters [1235–1256]. We encountered some examples in making signal models of exponential-smoothing, spline, and Whittaker-Henderson filters. The state-space approach is discussed in greater detail in [45].

The seasonal generalization of the Whittaker-Henderson method, which was originally introduced by Leser, Akaike, and Schlicht [1214–1216], differs from the Whittaker-Henderson/Kaiser method that we discussed earlier in that the latter determines the trend t_n using ordinary Whittaker-Henderson smoothing, and then applies a Kaiser-window comb filter to the residual $r_n = y_n - t_n$ to extract the seasonal part s_n. By contrast, in the seasonalized version, t_n and s_n are determined simultaneously from a single optimization criterion. We recall that the ordinary Whittaker-Henderson performance index for estimating the trend t_n is,

$$\mathcal{J}=\sum_{n=0}^{N-1}(y_n-t_n)^2+\lambda\sum_{n=s}^{N-1}\big(\nabla^s t_n\big)^2 = \min \tag{27.11.1}$$

where s is the smoothing order and N, the length of y_n. The seasonalized version with period D replaces this by,

$$\mathcal{J}=\sum_{n=0}^{N-1}(y_n-t_n-s_n)^2+\lambda\sum_{n=s}^{N-1}(\nabla^s t_n)^2+\alpha\sum_{n=D-1}^{N-1}(s_n+s_{n-1}+\cdots+s_{n-D+1})^2=\min \tag{27.11.2}$$

A fourth term, $\beta\sum_{n=D}^{N-1}(s_n-s_{n-D})^2$, may be added [1215,1216], but it is generally not necessary for the following reason. The minimization of $\mathcal{J}$ forces the sum

$$S_n = s_n + s_{n-1} + \cdots + s_{n-D+1} \tag{27.11.3}$$

to become small, ideally zero, and as a consequence the quantity $s_n - s_{n-D} = S_n - S_{n-1}$ will also be made small. Nevertheless, such a term has been implemented as an option in the function `swhdec` below. Eq. (27.11.2) can be written in a compact vectorial form as,

$$\mathcal{J}=(\mathbf{y}-\mathbf{t}-\mathbf{s})^T(\mathbf{y}-\mathbf{t}-\mathbf{s})+\lambda\,\mathbf{t}^T(D_s^T D_s)\,\mathbf{t}+\alpha\,\mathbf{s}^T(A^TA)\mathbf{s}=\min \tag{27.11.4}$$

where, as discussed in general terms in Sec. 26.2, the matrices D_s, A have dimensions $(N-s)\times N$ and $(N-D+1)\times N$, respectively, and are the steady-state versions of the convolution matrices of the corresponding filters, that is,

$$\begin{aligned}
&D_s(z)=(1-z^{-1})^s\,, && \mathbf{d}_s=\mathrm{binom}(s)\,, && D_s=\mathrm{convmat}\big(\mathrm{flip}(\mathbf{d}_s),\,N-s\big)^T\\
&A(z)=\sum_{k=0}^{D-1}z^{-k}\,, && \mathbf{a}=[\underbrace{1,1,\dots,1}_{D\text{ ones}}]\,, && A=\mathrm{convmat}\big(\mathrm{flip}(\mathbf{a}),\,N-D+1\big)^T
\end{aligned} \tag{27.11.5}$$

For example, we have for $N=7$, $s=2$, and $D=4$:

$$D_s=\begin{bmatrix}1&-2&1&0&0&0&0\\0&1&-2&1&0&0&0\\0&0&1&-2&1&0&0\\0&0&0&1&-2&1&0\\0&0&0&0&1&-2&1\end{bmatrix},\quad A=\begin{bmatrix}1&1&1&1&0&0&0\\0&1&1&1&1&0&0\\0&0&1&1&1&1&0\\0&0&0&1&1&1&1\end{bmatrix} \tag{27.11.6}$$

The solution of the minimization problem (27.11.4) is obtained from the vanishing of the gradient of $\mathcal{J}$ with respect to $\mathbf{t}$ and $\mathbf{s}$, which results in the system and its solution:

$$\begin{aligned}(I+P)\mathbf{t}+\mathbf{s}&=\mathbf{y}\\ \mathbf{t}+(I+Q)\mathbf{s}&=\mathbf{y}\end{aligned}\quad\Rightarrow\quad\boxed{\begin{aligned}\mathbf{t}&=(Q+P+QP)^{-1}Q\mathbf{y}\\ \mathbf{s}&=\mathbf{y}-(I+P)\mathbf{t}\end{aligned}} \tag{27.11.7}$$

where we defined $P=\lambda\,(D_s^TD_s)$ and $Q=\alpha\,(A^TA)$. The matrices P, Q and $(Q+P+QP)$ are *banded sparse* matrices and therefore the indicated inverse[†] in (27.11.7) can be computed very efficiently with $O(N)$ operations (provided it is implemented by the backslash operator in MATLAB.)

From the above system we also have, $\mathbf{t}=(I+P)^{-1}(\mathbf{y}-\mathbf{s})$, which has the appealing interpretation that the trend is obtained by an ordinary Whittaker-Henderson smoother, i.e., the operator $(I+P)^{-1}$, applied to the seasonally-adjusted signal $(\mathbf{y}-\mathbf{s})$, which is similar to how the X-11 method obtains the final trend by applying a Henderson filter.

The function `swhdec` implements this method. It has an optional argument for the fourth β-term mentioned above:

```
[yt,ys,yi] = swhdec(y,D,s,lambda,alpha,beta);      % seasonal Whittaker-Henderson
```

The larger the parameters α,β, the closer to zero the quantity (27.11.3), and the “more periodic” the seasonal component. Thus, if one wants to extract a slowly evolving periodic component, one should choose smaller values for these parameters, relative to λ. The latter, can be estimated using the GCV criterion. The simultaneous estimation of λ,α,β can be accomplished by maximizing an appropriate likelihood function in a Bayesian formulation of this method [1215,1228,1230].

27.12 Sparse Seasonal Whittaker-Henderson Decomposition

The ℓ_1-regularized version can be obtained by replacing the ℓ_2 norms of the regularizing parts by their ℓ_1 norms, that is,

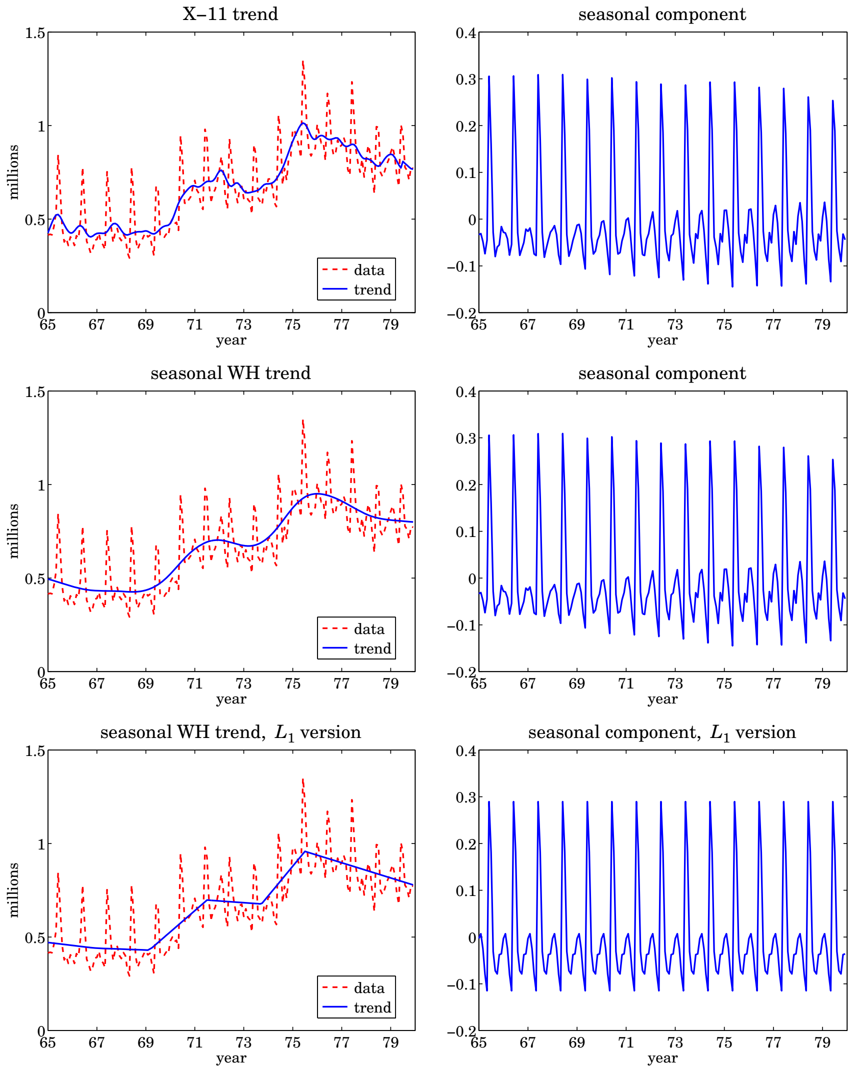
$$\mathcal{J}=\sum_{n=0}^{N-1}(y_n-t_n-s_n)^2+\lambda\sum_{n=s}^{N-1}\big|\nabla^s t_n\big|+\alpha\sum_{n=D-1}^{N-1}\big|s_n+s_{n-1}+\cdots+s_{n-D+1}\big|=\min$$

$$\boxed{\mathcal{J}=\|\mathbf{y}-\mathbf{t}-\mathbf{s})\|_2^2+\lambda\,\|D_s\mathbf{t}\|_1+\alpha\,\|A\,\mathbf{s}\|_1=\min} \tag{27.12.1}$$

and can be solved easily with the CVX package.[‡]

Example 27.12.1: We revisit the unemployment data for 16–19 year old men for the 1965–79 period, which we encountered in Example 27.8.2. Fig. 27.12.1 compares the trend/seasonal decomposition obtained by the X-11 method (top graphs) and by the seasonal Whittaker-Henderson (middle graphs), as well as the corresponding L_1 version (bottom graphs). The input parameters were as follows, where λ was determined in Example 27.8.2 by the GCV criterion,

$$D=12,\quad s=2,\quad \lambda=2039,\quad \alpha=10,\quad \beta=0$$



X–11 trend

seasonal component

seasonal WH trend

seasonal component

seasonal WH trend, L_1 version

seasonal component, L_1 version

Fig. 27.12.1 X-11 and seasonal Whittaker-Henderson decomposition methods.

The MATLAB code used to generate the six graphs was as follows:

```
Y = loadfile('unemp-1619-nsa.dat'); i=find(Y==1965);         % read data
Y = Y(i:i+14,2:13)'; y = Y(:)/1000; t = taxis(y,12,65);     % extract 1965-79 range
```

[†] It can be shown [1216] that the inverse exists for all positive values of λ, α.

[‡] http://cvxr.com/cvx

```
D=12; M1=33; M2=35; N1=13; N2=13; R=3.5; type='a';         % X-11 input parameters
[yt,ys,yi] = x11dec(y,D,M1,M2,N1,N2,R,type);                 % X-11 decomposition

figure; plot(t,y, t,yt); figure; plot(t,ys);                 % upper graphs

D=12; s=2; la=2039; alpha=10;                      % input parameters
[yt,ys,yi] = swhdec(y,D,s,la,alpha);               % seasonal WH decomposition

figure; plot(t,y, t,yt); figure; plot(t,ys);       % middle graphs


la=5; alpha=10;                                    % L1 version
N = length(y); s=2; Ds = diff(eye(N),s);           % construct matrices Ds and A
A = convmat(ones(1,D), N-D+1)';

cvx_quiet(true);                                   % CVX package
cvx_begin
    variable X(2*N)                                % pack trend and seasonal into X
    T = X(1:N); S = X(N+1:2*N);
    minimize( sum_square(y-T-S) + la * norm(Ds*T,1) + alpha * norm(A*S,1) );
cvx_end

yt = X(1:N); ys = X(N+1:2*N);                      % extract trend and seasonal parts

figure; plot(t,y, t,yt); figure; plot(t,ys);       % lower graphs
```

The seasonal components extracted by the methods are comparable, as are the outputs of this method and the Whittaker-Henderson/Kaiser method plotted in Fig. 27.8.2. □

In Sec. 26.3 we obtained the equivalent Whittaker-Henderson trend-extraction filter and showed that it could be thought of as the optimum unrealizable Wiener filter of a particular state-space model. The optimum filter had frequency response:

$$H(\omega)=\frac{1}{1+\lambda\,|D_s(\omega)|^2}\,,\quad \text{where}\quad D_s(\omega)=\left(1-e^{-j\omega}\right)^s \tag{27.12.2}$$

and the state-space model was defined by

$$y_n=t_n+v_n\,,\quad \nabla^s t_n=w_n \tag{27.12.3}$$

where v_n, w_n were zero-mean, mutually-uncorrelated, white-noise signals of variances σ_v^2, σ_w^2, and the smoothing parameter was identified as $\lambda=\sigma_v^2/\sigma_w^2$.

All of these results carry over to the seasonal case. First, we obtain the effective trend and seasonal filters $H_T(\omega), H_S(\omega)$ for extracting t_n, s_n. Then, we show that they are optimal in the Wiener sense. As we did in Sec. 26.3, we consider a double-sided infinitely-long signal y_n and using Parseval's identity, we may write the performance index (27.11.2) in the frequency domain, as follows:

$$\mathcal{J}=\int_{-\pi}^{\pi}\Big[\,|Y(\omega)-T(\omega)-S(\omega)|^2+\lambda\,|D_s(\omega)T(\omega)|^2+\alpha\,|A(\omega)S(\omega)|^2\Big]\frac{d\omega}{2\pi} \tag{27.12.4}$$

where $D_s(\omega)$ and $A(\omega)$ are the frequency responses of the filters in Eq. (27.11.5). From the vanishing of the gradients $\partial\mathcal{J}/\partial T^*$ and $\partial\mathcal{J}/\partial S^*$, we obtain the equations:

$$\begin{aligned} T(\omega)+\lambda\,\big|D_s(\omega)\big|^2 T(\omega)+S(\omega) &= Y(\omega)\\ S(\omega)+\alpha\,\big|A(\omega)\big|^2 S(\omega)+T(\omega) &= Y(\omega)\end{aligned}\tag{27.12.5}$$

which may be solved for the transfer functions $H_T(\omega)= T(\omega)/Y(\omega)$ and $H_S(\omega)= S(\omega)/Y(\omega)$, resulting in,

$$\begin{aligned} H_T(\omega) &= \frac{\alpha\,\big|A(\omega)\big|^2}{\lambda\,\big|D_s(\omega)\big|^2+\alpha\,\big|A(\omega)\big|^2+\lambda\alpha\,\big|D_s(\omega)\big|^2\big|A(\omega)\big|^2}\\ H_S(\omega) &= \frac{\lambda\,\big|D_s(\omega)\big|^2}{\lambda\,\big|D_s(\omega)\big|^2+\alpha\,\big|A(\omega)\big|^2+\lambda\alpha\,\big|D_s(\omega)\big|^2\big|A(\omega)\big|^2}\end{aligned}\tag{27.12.6}$$

with $\big|D_s(\omega)\big|^2$ and $\big|A(\omega)\big|^2$ given by,

$$\begin{aligned} \big|D_s(\omega)\big|^2 &= \big|1-e^{-j\omega}\big|^{2s} = \big|2\sin(\omega/2)\big|^{2s}\\ \big|A(\omega)\big|^2 &= \big|1+e^{-j\omega}+\cdots+e^{-j(D-1)\omega}\big|^2 = \left|\frac{\sin(\omega D/2)}{\sin(\omega/2)}\right|^2\end{aligned}\tag{27.12.7}$$

The filters (27.12.6) generalize the Whittaker-Henderson, or Hodrick-Prescott filter (27.12.2) to the seasonal case. The filters may be identified as the optimum Wiener filters for the following signal model:

$$y_n = t_n + s_n + v_n\,,\quad \nabla^s t_n = w_n\,,\quad s_n + s_{n-1}+\cdots+s_{n-D+1} = u_n\tag{27.12.8}$$

where v_n, w_n, u_n are mutually-uncorrelated, zero-mean, white noises. The model can be written symbolically in operator form:

$$y_n = t_n + s_n + v_n\,,\quad D_s(z)t_n = w_n\,,\quad A(z)s_n = u_n\tag{27.12.9}$$

The signals t_n, s_n are not stationary, but nevertheless the optimum Wiener filters can be derived as though the signals were stationary [1235–1241]. Alternatively, multiplication by $D_s(z)A(z)$ acts as a stationarity-inducing transformation, resulting in the stationary signal model,

$$\begin{aligned} \bar{y}_n &= D_s(z)A(z)y_n = \bar{t}_n+\bar{s}_n+\bar{v}_n = A(z)w_n + D_s(z)u_n + D_s(z)A(z)v_n\\ \bar{t}_n &= D_s(z)A(z)t_n = A(z)w_n\\ \bar{s}_n &= D_s(z)A(z)s_n = D_s(z)u_n\\ \bar{v}_n &= D_s(z)A(z)v_n\end{aligned}\tag{27.12.10}$$

with spectral densities:

$$\begin{aligned} S_{\bar{t}\bar{y}}(\omega) &= S_{\bar{t}\bar{t}}(\omega)= \sigma_w^2\big|A(\omega)\big|^2\\ S_{\bar{s}\bar{y}}(\omega) &= S_{\bar{s}\bar{s}}(\omega)= \sigma_u^2\big|D_s(\omega)\big|^2\\ S_{\bar{y}\bar{y}}(\omega) &= \sigma_u^2\big|D_s(\omega)\big|^2+\sigma_w^2\big|A(\omega)\big|^2+\sigma_v^2\big|D_s(\omega)A(\omega)\big|^2\end{aligned}$$

It follows from [1235–1241] that the optimum Wiener filters for estimating t_n, s_n will be:

$$\begin{aligned} H_T(\omega) &= \frac{S_{\bar{t}\bar{y}}(\omega)}{S_{\bar{y}\bar{y}}(\omega)} = \frac{\sigma_w^2\big|A(\omega)\big|^2}{\sigma_u^2\big|D_s(\omega)\big|^2+\sigma_w^2\big|A(\omega)\big|^2+\sigma_v^2\big|D_s(\omega)A(\omega)\big|^2}\\ H_S(\omega) &= \frac{S_{\bar{s}\bar{y}}(\omega)}{S_{\bar{y}\bar{y}}(\omega)} = \frac{\sigma_u^2\big|D_s(\omega)\big|^2}{\sigma_u^2\big|D_s(\omega)\big|^2+\sigma_w^2\big|A(\omega)\big|^2+\sigma_v^2\big|D_s(\omega)A(\omega)\big|^2}\end{aligned}\tag{27.12.11}$$

It is evident that these are identical to (27.12.6) with the identifications $\lambda = \sigma_v^2/\sigma_w^2$ and $\alpha = \sigma_v^2/\sigma_u^2$. For a finite, length-$N$, signal y_n, the model (27.12.8) has been used to derive Kalman smoothing algorithms for estimating t_n, s_n with $O(N)$ operations, and for efficiently evaluating the model's likelihood function [1228,1230]. We note, however, that the matrix solutions (27.11.7) are equally efficient.

Example 27.12.2: Fig. 27.12.2 plots the frequency responses $H_T(\omega)$ and $H_S(\omega)$ of Eq. (27.12.6). For the upper graphs, the parameter values were the same as those of Example 27.12.1, that is, $D = 12$, $s = 2$, $\lambda = 2039$, $\alpha = 10$. We note that the responses have the expected shapes.

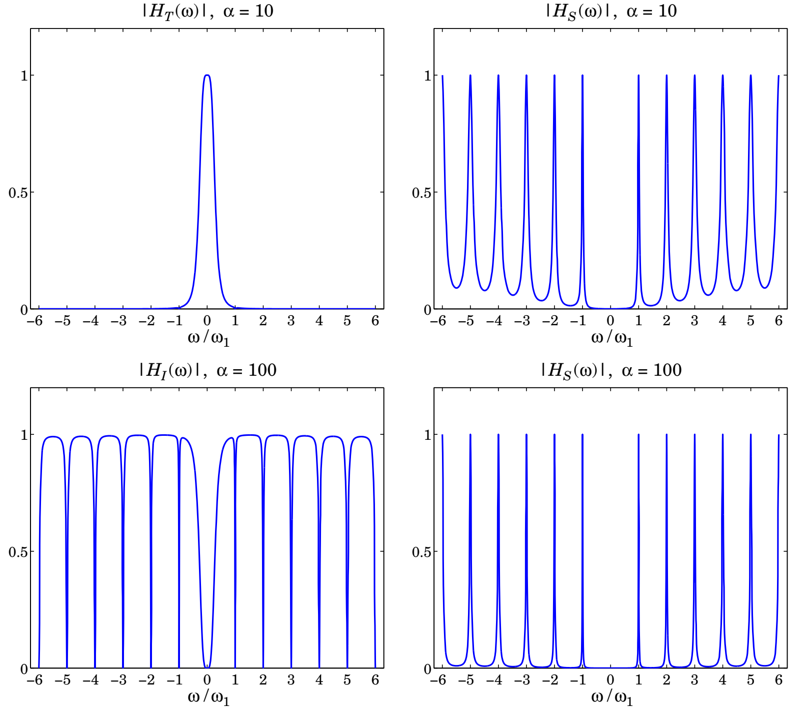


Fig. 27.12.2 Frequency responses of seasonal Whittaker-Henderson filters.

In the lower graphs, we increased the parameter α to 100 in order to sharpen the comb peaks. The lower-left graph depicts the filter $H_I(\omega)= 1 - H_T(\omega) - H_S(\omega)$ for extracting the irregular component, and the right graph depicts $H_S(\omega)$. The trend filter is not shown since it is virtually identical to that of the upper-left graph. The MATLAB code used to generate the upper graphs was as follows:

```
k = linspace(-6,6,2401); w = 2*pi*k/12;          % frequencies −π ≤ ω ≤ π

D = 12; s = 2; la = 2039; alpha = 10;

a = ones(D,1); A = freqz(a,1,w);                 % calculate A(ω)

P = la * abs(1 - exp(-j*w)).^(2*s);              % evaluate P(ω) = λ |D_S(ω)|^2
Q = alpha * abs(A).^2;                           % evaluate Q(ω) = α |A(ω)|^2
R = Q + P + Q.*P;

HT = Q./R; HS = P./R; HI = 1-HS-HT;

figure; plot(k,HT);  figure; plot(k,HS);         % upper graphs
```

27.13 Problems

27.1 First prove Eq. (27.2.2) for all n. Then, using the DFT/IDFT pair in Eq. (27.2.1), show that a more general form of (27.2.2) is,

$$\sum_{m=0}^{D-1} s_{n-m} e^{j\omega_k m} = e^{j\omega_k n} S_k\,, \quad k = 0, 1, \dots, D-1, \quad -\infty < n < \infty$$

27.2 Consider the analog signal $s(t) = \cos(2\pi f_1 t)$ and its sampled version $s_n = \cos(2\pi f_1 nT)$, where T is the sampling interval related to the sampling rate by $f_s = 1/T$. It is required that s_n be periodic in n with period of D samples, that is, $\cos\big(2\pi f_1(n+D)T\big) = \cos(2\pi f_1 nT)$, for all n. How does this requirement constrain f_s and f_1?

27.3 Show that the IIR comb and notch filters defined in Eq. (27.2.9) are complementary and power complementary in the sense that they satisfy Eqs. (27.2.7).

Working with the magnitude response $\big|H_{\text{comb}}(\omega)\big|^2$ show that the 3-dB width of the comb peaks is given by Eq. (27.2.11).

27.4 Show that the solution of the system (27.11.7) can be written in the more symmetric, but computationally less efficient, form:

$$\mathbf{t} = (Q + P + QP)^{-1} Q\mathbf{y}$$

$$\mathbf{s} = (P + Q + PQ)^{-1} P\mathbf{y}$$

28 Neural Networks

28.1 Introduction

Neural networks are the underlying engine for most Machine Learning and Artificial Intelligence (AI) applications. There are several online resources, textbooks, bibliographies, and software implementations [1415–1437].

The subject has expanded considerably in the past decade with several variations of neural network architectures, and efficient computational procedures. Some typical variations are:

- Multilayer feedforward neural networks
- Recurrent neural networks, LSTM, GRU
- Deep learning, convolutional neural networks
- Deep residual networks, ResNet, ResNet50
- Hopfield networks
- Boltzman machines, including restricted
- Auto encoders, including sparse, denoising, variational
- Deep belief networks
- Generative adversarial networks
- Kohonen networks
- Extreme learning machine
- Reinforcement learning

In this chapter, we present only a brief introduction to multilayer feedforward neural networks, discuss the backpropagation algorithm, and carry out some computer experiments illustrating the convergence and learning properties of neural networks, including neural networks for time-series prediction [1430–1437].

28.2 Multilayer Feedforward Neural Networks

A typical multilayer feedforward neural network is depicted in Fig. 28.2.1. For clarity, only two hidden layers are shown, but any additional layers can be added as necessary, as in deep learning networks.

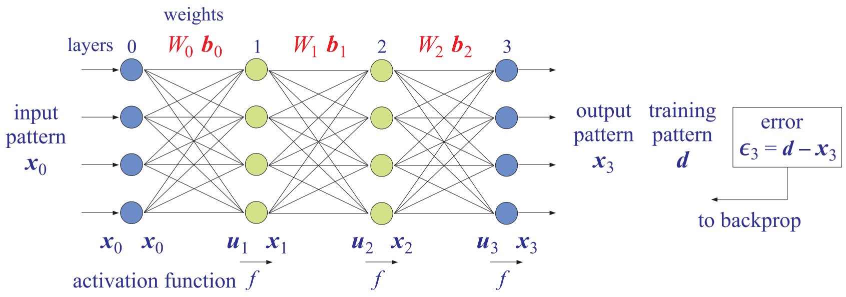


Fig. 28.2.1 Neural network with two hidden layers.

Starting with an applied input pattern $\mathbf{x}_0$ (column vector of dimension M_0), the output pattern $\mathbf{x}_3$ (column vector of dimension M_3) is computed by the following sequence of operations defined by the *connection* matrices W_r and *bias* weights $\mathbf{b}_r$, $r = 0, 1, 2$, and activation function $f(u)$,

$$\boxed{\begin{array}{l} \text{forward pass} \\ \hline \mathbf{u}_1 = W_0\mathbf{x}_0 + \mathbf{b}_0 \\ \mathbf{x}_1 = f(\mathbf{u}_1) \\ \hline \mathbf{u}_2 = W_1\mathbf{x}_1 + \mathbf{b}_1 \\ \mathbf{x}_2 = f(\mathbf{u}_2) \\ \hline \mathbf{u}_3 = W_2\mathbf{x}_2 + \mathbf{b}_2 \\ \mathbf{x}_3 = f(\mathbf{u}_3) \end{array}} \tag{28.2.1}$$

The vector dimensions of the layers are,

$$\begin{array}{lll} M_0 \times 1, & \text{input layer}, & \mathbf{x}_0 \\ M_1 \times 1, & \text{hidden layer 1}, & \mathbf{u}_1, \mathbf{x}_1 = f(\mathbf{u}_1) \\ M_2 \times 1, & \text{hidden layer 2}, & \mathbf{u}_2, \mathbf{x}_2 = f(\mathbf{u}_2) \\ M_3 \times 1, & \text{output layer}, & \mathbf{u}_3, \mathbf{x}_3 = f(\mathbf{u}_3) \end{array}$$

so that the dimensions of the connection matrices and bias weights will be,

$$\begin{array}{lllll} W_0, & M_1 \times M_0 & \text{and} & \mathbf{b}_0\,, & M_1 \times 1 \\ W_1, & M_2 \times M_1 & \text{and} & \mathbf{b}_1\,, & M_2 \times 1 \\ W_2, & M_3 \times M_2 & \text{and} & \mathbf{b}_2\,, & M_3 \times 1 \end{array}$$

and the activation function operations are element-wise, that is, if u_i is the i-th component of a vector $\mathbf{u}$, then, the i-th component of the vector $f(\mathbf{u})$ is $f(u_i)$. The activation functions could also be chosen to be different for each layer, and also, often the activation function for the last layer is omitted, that is, the output is, $\mathbf{x}_3 = \mathbf{u}_3$.

Several types of activation functions $f(u)$ have been used, some of which are listed below together with their derivatives $f'(u)$, with the most common being the sigmoid logistic and tanh functions and ReLU (rectified linear unit),

logistic:	$f(u) = \dfrac{1}{1+e^{-u}}\,,$	$f'(u) = f(u)\big[1 - f(u)\big]$
tanh:	$f(u) = \tanh(u)\,,$	$f'(u) = 1 - f^2(u)$
softplus:	$f(u) = \ln(1 + e^u)\,,$	$f'(u) = \dfrac{1}{1+e^{-u}}$
linear:	$f(u) = u\,,$	$f'(u) = 1$
ReLU:	$f(u) = \max(u, 0)\,,$	$f'(u) = (u \ge 0)$
leaky ReLU:	$f(u) = \max(u, \alpha u)\,,$	$f'(u) = (u \ge 0) + \alpha\,(u < 0)$
sinusoid:	$f(u) = \sin u\,,$	$f'(u) = \cos u$

where for the ReLU cases, the notations, $(u \ge 0)$ and $(u < 0)$, denote MATLAB-like logical operations[†], and the parameter α must be in the range, $0 < \alpha < 1$.

During the *training phase* of the neural network, a set of training input/output pattern pairs, $\{\mathbf{x}_0, \mathbf{d}\}$, are applied to the network, and the connection matrices and bias weights are adapted to minimize the square deviations of the actual outputs $\mathbf{x}_3$ from the desired outputs $\mathbf{d}$, that is, minimizing the following typical performance index,[‡] where, $\boldsymbol{\epsilon}_3 = \mathbf{d} - \mathbf{x}_3$, is the output error,

$$\boxed{J = \sum_{\text{patterns}} \boldsymbol{\epsilon}_3^T \boldsymbol{\epsilon}_3 = \sum_{\text{patterns}} (\mathbf{d} - \mathbf{x}_3)^T (\mathbf{d} - \mathbf{x}_3) = \min} \tag{28.2.2}$$

The adaptation of the weights is usually implemented with the gradient-descent algorithm, that is, the weight updates are computed by the following rule with a small positive adaptation constant μ, where W_{rij} denotes the ij matrix element of the r-th connection matrix W_r, and b_{ri} denotes the i-th component of the bias weight $\mathbf{b}_r$,

$$\boxed{\Delta W_{rij} = -\mu\,\frac{\partial J}{\partial W_{rij}}\,, \quad \Delta b_{ri} = -\mu\,\frac{\partial J}{\partial b_{ri}}}\,, \quad r = 0, 1, 2 \tag{28.2.3}$$

The updated weights are then,

$$\begin{aligned} W_r &= W_r + \Delta W_r\,, \quad r = 0, 1, 2 \\ \mathbf{b}_r &= \mathbf{b}_r + \Delta\mathbf{b}_r\,, \quad r = 0, 1, 2 \end{aligned} \tag{28.2.4}$$

The updates (28.2.4) can be applied in two ways:

[†] being equal to 1 if their argument is true, and 0, otherwise.

[‡] more generally, the negative-log-likelihood function based on maximum likelihood is used

(i) On a *pattern-basis*, that is, J arises only from one pattern pair in the sum (28.2.2), that is, $J_{\text{patt}} = (\mathbf{d}-\mathbf{x}_3)^T(\mathbf{d}-\mathbf{x}_3)$, and the updates are applied immediately for that pattern, then, the next pattern pair is used to perform the next update, and so on. After all the pattern pairs in the training set have been used, the whole process is repeated multiple times till convergence.

(ii) On an *epoch-basis*, that is, a whole series of input/output patterns called an epoch (which could be the whole training set) is applied and the corrections, $\Delta W_{rij}, \Delta b_{ri}$, are accumulated over all the patterns in the epoch before the updated weights are computed, and then, the whole epoch is repeated till convergence.

To see how the gradient-descent works, we may expand the performance index, considered as a function of the weights, $J(W,\mathbf{b})$, to first-order in Taylor series (justified if μ is sufficiently small), and apply Eq. (28.2.3), so that component-wise we have,

$$J(W+\Delta W,\mathbf{b}+\Delta\mathbf{b}) = J(W,\mathbf{b}) + \sum_{r,i,j}\frac{\partial J}{\partial W_{rij}}\Delta W_{rij} + \sum_{r,i}\frac{\partial J}{\partial b_{ri}}\Delta b_{ri}$$

$$= J(W,\mathbf{b}) - \mu\sum_{r,i,j}\left|\frac{\partial J}{\partial W_{rij}}\right|^2 - \mu\sum_{r,i}\left|\frac{\partial J}{\partial b_{ri}}\right|^2 \le J(W,\mathbf{b})\,, \quad \text{since } \mu > 0$$

Thus, the value of J at the updated weights is smaller than its value at the original weights, $J(W+\Delta W,\mathbf{b}+\Delta\mathbf{b}) \le J(W,\mathbf{b})$, so that the iteration of the updating process will drive J to smaller and smaller values, such as a local minimum. If a local minimum is reached, then both summation terms are zero, and the performance index will no longer change. Because of the highly nonlinear dependence of J on the weights, there may be several local minima, but often choosing any of these tends to be good enough for many applications — another way to state this is that J is fairly flat in the neighborhoods of the local or global minima.

28.3 Backpropagation Algorithm

The *backpropagation algorithm* [1427] is a convenient way of calculating the updates (28.2.3) based on each pattern $\{\mathbf{x}_0, \mathbf{d}\}$. It makes use of the following gradients of the performance index,

$$\mathbf{e}_r = -\frac{\partial J}{\partial \mathbf{u}_r}\,, \quad r = 1,2,3\,, \quad \text{or, component-wise}\,, \quad e_{ri} = -\frac{\partial J}{\partial u_{ri}} \tag{28.3.1}$$

To see how this works, start with the updates of the last connection weights, $W_2, \mathbf{b}_2$. Given an input pattern, $\mathbf{x}_0$, then after the forward pass of Eq. (28.2.1), we have computed all layer vectors, $\mathbf{u}_r$, $\mathbf{x}_r$, $r = 1,2,3$. Because J depends on $W_2, \mathbf{b}_2$ only through the variable $\mathbf{u}_3$, we have for the partial derivatives of the updates,

$$\begin{aligned} \Delta W_{2ij} &= -\mu\frac{\partial J}{\partial W_{2ij}} = -\mu\frac{\partial J}{\partial u_{3i}}\frac{\partial u_{3i}}{\partial W_{2ij}} = \mu e_{3i}x_{2j} \\ \Delta b_{2i} &= -\mu\frac{\partial J}{\partial b_{2i}} = -\mu\frac{\partial J}{\partial u_{3i}}\frac{\partial u_{3i}}{\partial b_{2i}} = \mu e_{3i} \end{aligned} \tag{28.3.2}$$

where we used the definitions (28.3.1) and the partial derivatives of the connection formula,

$$\mathbf{u}_3 = W_2\mathbf{x}_2 + \mathbf{b}_2 \quad\Rightarrow\quad u_{3i} = \sum_j W_{2ij}x_{2j} + b_{2i}$$

$$\frac{\partial u_{3i}}{\partial W_{2ij}} = x_{2j}\,, \quad \frac{\partial u_{3i}}{\partial b_{2i}} = 1$$

Eqs. (28.3.2) can be written in the compact matrix forms,[†]

$$\Delta W_2 = \mu\,\mathbf{e}_3\mathbf{x}_2^T\,, \quad \Delta\mathbf{b}_2 = \mu\,\mathbf{e}_3 \tag{28.3.3}$$

The other updates can similarly be expressed in terms of the gradients (28.3.1). In summary, we have,

$$\begin{aligned} \Delta W_2 &= \mu\,\mathbf{e}_3\mathbf{x}_2^T\,, \quad \Delta\mathbf{b}_2 = \mu\,\mathbf{e}_3 \\ \Delta W_1 &= \mu\,\mathbf{e}_2\mathbf{x}_1^T\,, \quad \Delta\mathbf{b}_1 = \mu\,\mathbf{e}_2 \\ \Delta W_0 &= \mu\,\mathbf{e}_1\mathbf{x}_0^T\,, \quad \Delta\mathbf{b}_0 = \mu\,\mathbf{e}_1 \end{aligned} \tag{28.3.4}$$

The backpropagation algorithm efficiently calculates the quantities, $\mathbf{e}_3, \mathbf{e}_2, \mathbf{e}_1$, starting from the output layer and proceeding backwards to the input. The operations involve the diagonal matrix of the derivatives of the activation function, that is,

$$D(\mathbf{u}) = \text{diag}\big[f'(\mathbf{u})\big]\,, \quad \text{or, element-wise}\,, \quad D_{ij}(\mathbf{u}) = \delta_{ij}f'(u_i) \tag{28.3.5}$$

Given the output $\mathbf{x}_3$ of the network corresponding to the input pattern $\mathbf{x}_0$, the error relative to the training pattern, $\mathbf{d}$, will be, $\boldsymbol{\epsilon}_3 = \mathbf{d} - \mathbf{x}_3$, and its contribution to the performance index J of Eq. (28.2.2), will be, $J_{\text{patt}} = (\mathbf{d}-\mathbf{x}_3)^T(\mathbf{d}-\mathbf{x}_3) = \boldsymbol{\epsilon}_3^T\boldsymbol{\epsilon}_3$. Starting with $\mathbf{e}_3$, we have component-wise,

$$e_{3i} = -\frac{\partial J_{\text{patt}}}{\partial u_{3i}} = -\frac{\partial x_{3i}}{\partial u_{3i}}\frac{\partial J_{\text{patt}}}{\partial x_{3i}} = f'(u_{3i})(d_i - x_{3i})\,, \quad \text{or,}$$

$$\mathbf{e}_3 = D(\mathbf{u}_3)(\mathbf{d}-\mathbf{x}_3) = D(\mathbf{u}_3)\boldsymbol{\epsilon}_3$$

Next, for $\mathbf{e}_2$, because J_{patt} depends on $\mathbf{x}_2$ through $\mathbf{u}_3$, and for $\mathbf{e}_1$, because J_{patt} depends on $\mathbf{x}_1$ through $\mathbf{u}_2$, we have,

$$\begin{aligned} e_{2i} &= -\frac{\partial J_{\text{patt}}}{\partial u_{2i}} = -\frac{\partial x_{2i}}{\partial u_{2i}}\frac{\partial J_{\text{patt}}}{\partial x_{2i}} = -D(u_{2i})\frac{\partial J_{\text{patt}}}{\partial x_{2i}} \\ &= -D(u_{2i})\sum_j\frac{\partial J_{\text{patt}}}{\partial u_{3j}}\frac{\partial u_{3j}}{\partial x_{2i}} = D(u_{2i})\sum_j W_{2ji}e_{3j} \\ e_{1i} &= -\frac{\partial J_{\text{patt}}}{\partial u_{1i}} = -\frac{\partial x_{1i}}{\partial u_{1i}}\frac{\partial J_{\text{patt}}}{\partial x_{1i}} = -D(u_{1i})\frac{\partial J_{\text{patt}}}{\partial x_{1i}} \\ &= -D(u_{1i})\sum_j\frac{\partial J_{\text{patt}}}{\partial u_{2j}}\frac{\partial u_{2j}}{\partial x_{1i}} = D(u_{1i})\sum_j W_{1ji}e_{2j} \end{aligned}$$

[†] supescript T denotes transposition

These can be written in matrix forms using the *transposed* connection matrices,

$$\begin{aligned}\mathbf{e}_2 &= D(\mathbf{u}_2)\,W_2^T\mathbf{e}_3\\ \mathbf{e}_1 &= D(\mathbf{u}_1)\,W_1^T\mathbf{e}_2\end{aligned}\tag{28.3.6}$$

Finally, putting all the computational steps together, we have for the two-hidden-layer network, for each input/output pattern, $\{\mathbf{x}_0, \mathbf{d}\}$,

$$\begin{array}{|l|}\hline \text{forward pass}\\ \hline \mathbf{u}_1 = W_0\mathbf{x}_0+\mathbf{b}_0\\ \mathbf{x}_1 = f(\mathbf{u}_1)\\ \hline \mathbf{u}_2 = W_1\mathbf{x}_1+\mathbf{b}_1\\ \mathbf{x}_2 = f(\mathbf{u}_2)\\ \hline \mathbf{u}_3 = W_2\mathbf{x}_2+\mathbf{b}_2\\ \mathbf{x}_3 = f(\mathbf{u}_3)\\ \hline\end{array}\;\Rightarrow\;\begin{array}{|l|}\hline \text{backpropagation}\\ \hline \boldsymbol{\epsilon}_3 = \mathbf{d}-\mathbf{x}_3\\ \mathbf{e}_3 = D(\mathbf{u}_3)\,\boldsymbol{\epsilon}_3\\ \hline \boldsymbol{\epsilon}_2 = W_2^T\mathbf{e}_3\\ \mathbf{e}_2 = D(\mathbf{u}_2)\,\boldsymbol{\epsilon}_2\\ \hline \boldsymbol{\epsilon}_1 = W_1^T\mathbf{e}_2\\ \mathbf{e}_1 = D(\mathbf{u}_1)\,\boldsymbol{\epsilon}_1\\ \hline\end{array}\;\Rightarrow\;\begin{array}{|l|}\hline \text{weight updates}\\ \hline \Delta W_2 = \mu\,\mathbf{e}_3\mathbf{x}_2^T\\ \Delta\mathbf{b}_2 = \mu\,\mathbf{e}_3\\ \hline \Delta W_1 = \mu\,\mathbf{e}_2\mathbf{x}_1^T\\ \Delta\mathbf{b}_1 = \mu\,\mathbf{e}_2\\ \hline \Delta W_0 = \mu\,\mathbf{e}_1\mathbf{x}_0^T\\ \Delta\mathbf{b}_0 = \mu\,\mathbf{e}_1\\ \hline\end{array}\tag{28.3.7}$$

If the last activation function is omitted, then we simply set, $\mathbf{x}_3 = \mathbf{u}_3$ and $\mathbf{e}_3 = \boldsymbol{\epsilon}_3$. For a single hidden layer network depicted in Fig. 28.3.1, we have the simpler version,

$$\begin{array}{|l|}\hline \text{forward pass}\\ \hline \mathbf{u}_1 = W_0\mathbf{x}_0+\mathbf{b}_0\\ \mathbf{x}_1 = f(\mathbf{u}_1)\\ \hline \mathbf{u}_2 = W_1\mathbf{x}_1+\mathbf{b}_1\\ \mathbf{x}_2 = f(\mathbf{u}_2)\\ \hline\end{array}\;\Rightarrow\;\begin{array}{|l|}\hline \text{backpropagation}\\ \hline \boldsymbol{\epsilon}_2 = \mathbf{d}-\mathbf{x}_2\\ \mathbf{e}_2 = D(\mathbf{u}_2)\,\boldsymbol{\epsilon}_2\\ \hline \boldsymbol{\epsilon}_1 = W_1^T\mathbf{e}_2\\ \mathbf{e}_1 = D(\mathbf{u}_1)\,\boldsymbol{\epsilon}_1\\ \hline\end{array}\;\Rightarrow\;\begin{array}{|l|}\hline \text{weight updates}\\ \hline \Delta W_1 = \mu\,\mathbf{e}_2\mathbf{x}_1^T\\ \Delta\mathbf{b}_1 = \mu\,\mathbf{e}_2\\ \hline \Delta W_0 = \mu\,\mathbf{e}_1\mathbf{x}_0^T\\ \Delta\mathbf{b}_0 = \mu\,\mathbf{e}_1\\ \hline\end{array}\tag{28.3.8}$$

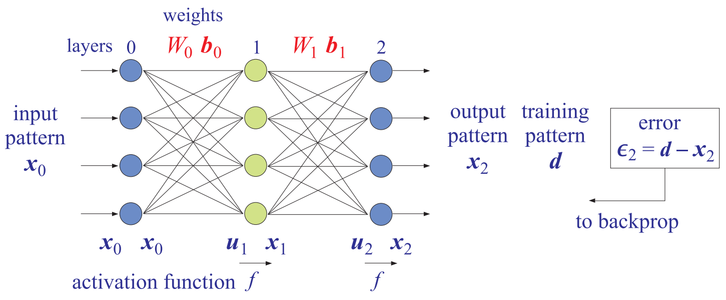


Fig. 28.3.1 Neural network with one hidden layer.

In practice, the weight updates, ΔW_r, $\Delta\mathbf{b}_r$, are smoothed using a simple EMA with a forgetting factor λ (referred to as *momentum* updating) before the new weights are computed, that is, instead of using the corrections of Eq. (28.3.4),

$$\begin{aligned}\Delta W_r &= \mu\,\mathbf{e}_{r+1}\mathbf{x}_r^T\,, \quad r = 0,1,2\\ \Delta\mathbf{b}_r &= \mu\,\mathbf{e}_{r+1}\end{aligned}$$

we use their EMA-smoothed versions, with some forgetting factor, $0 < \lambda < 1$,[†]

$$\begin{aligned}\Delta W_r &= \lambda\,\Delta W_r + \mu\,\mathbf{e}_{r+1}\mathbf{x}_r^T\,, \quad r = 0,1,2\\ \Delta\mathbf{b}_r &= \lambda\,\Delta\mathbf{b}_r + \mu\,\mathbf{e}_{r+1}\end{aligned}$$

Depending on whether one uses pattern-updating or epoch-updating, the iterative computational algorithm for minimizing the performance index J may be summarized as follows, where we use the value $\lambda = 1$ for epoch updating, and $0 \le \lambda < 1$ for pattern updating,[‡] and the weights are usually initialized to random values prior to starting the iteration,

```
for each epoch, do,
    for each pattern, do,
        forward pass, calculate, $\mathbf{x}_r$, $r = 1,2,3$
        backpropagation pass, calculate, $\mathbf{e}_{r+1}$, $r = 2,1,0$
        $\Delta W_r = \lambda\Delta W_r + \mu\mathbf{e}_{r+1}\mathbf{x}_r^T\,, \quad r = 0,1,2$
        $\Delta\mathbf{b}_r = \lambda\Delta\mathbf{b}_r + \mu\mathbf{e}_{r+1}$
        if pattern-updating, update now
            $W_r = W_r + \Delta W_r\,, \quad r = 0,1,2$
            $\mathbf{b}_r = \mathbf{b}_r + \Delta\mathbf{b}_r$
        end
    end pattern loop
    if epoch-updating, update after all epoch patterns
        $W_r = W_r + \Delta W_r\,, \quad r = 0,1,2$
        $\mathbf{b}_r = \mathbf{b}_r + \Delta\mathbf{b}_r$
    end
end epoch loop                                          (28.3.9)
```

In the pattern-updating case, the smoothed weight corrections are applied to update the weights after each pattern presentation, whereas in the epoch-updating case, the weight corrections are accumulated over all the patterns and applied to update the weights after all patterns have been presented. The epoch loop is then repeated several times (typically, thousands of times) until J has been minimized.

28.4 Computer Experiments

28.4.1 3:3:2 network for 3-bit parity problem

The 2-bit XOR[†] problem has served as a prototypical example for neural networks. Here, we consider its 3-bit generalization, so that the input patterns are length-3 vectors of

[†] we can also define, $\Delta W_r = \lambda\,\Delta W_r + (1-\lambda)\,\mu\,\mathbf{e}_{r+1}\mathbf{x}_r^T$, but this amounts to a redefinition of μ.
[‡] $\lambda = 0$ corresponds to no EMA smoothing
[†] exclusive OR

bits (0 or 1), and there are two outputs, one corresponding to the XOR operation of the 3 bits, i.e., having odd parity, while the other output is the complement of the first one, i.e., having even parity. There are, $2^3 = 8$, input/output pattern pairs, contained column-wise in the matrices X_0 and D,

$$X_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

As a first example, we use a neural network with one hidden layer of dimension 3, i.e., a 3:3:2 network. One epoch consists of all 8 patterns, and the algorithm of Eqs. (28.3.8) and (28.3.9) is applied repeatedly, with $\mu = 0.7$, to train the network.

Fig. 28.4.1 shows the convergence of the performance index J as a function of epoch iterations for the four cases of using epoch updating ($\lambda = 1$) versus pattern updating ($\lambda < 1$), and using standard sigmoid activation function versus tanh. Because tanh lies in the symmetric interval $[-1, 1]$, the input/output patterns are also transformed in that case to the same interval before applying the algorithm. For plotting purposes, in Fig. 28.4.1 the performance index J was normalized to unity value at its beginning.

In all cases, the initial connection matrices and bias weights are initialized to random values. We note that the algorithm is quite sensitive to the choice of initial values.

After convergence, the output of the network (i.e., layer 2) is calculated by performing a forward pass using the converged weights, and the results are collected in the 2×8 matrix X_2. As an example, for the case $\lambda = 1$ with sigmoid activation function, the calculated or learned output X_2 is listed below (row-wise) together with the input X_0 and theoretical output D, where we observe that the rounding of X_2 reproduces the correct D output,

```
     activation = sigmoid
-----------------------------
   X0     |   D  |      X2
-----------------------------
 0  0  0 | 0  1 | 0.01  0.99
 0  0  1 | 1  0 | 0.99  0.01
 0  1  0 | 1  0 | 0.99  0.01
 0  1  1 | 0  1 | 0.01  0.99
 1  0  0 | 1  0 | 0.99  0.01
 1  0  1 | 0  1 | 0.01  0.99
 1  1  0 | 0  1 | 0.01  0.99
 1  1  1 | 1  0 | 0.99  0.01
```

The complete MATLAB code for this example can be found in the script file, **nn332.m**, located in the NN folder of the ISP2e toolbox.

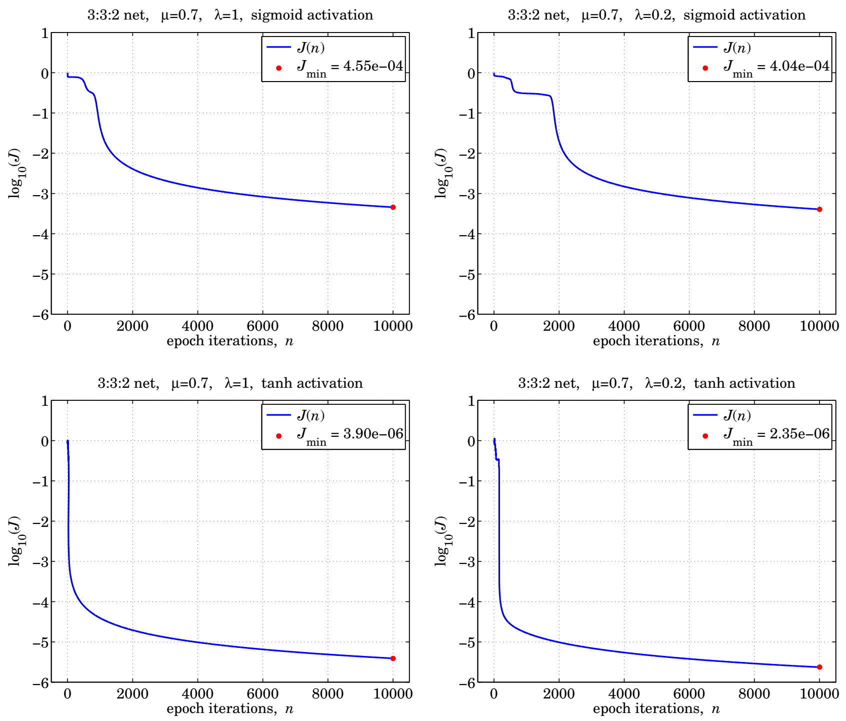


Fig. 28.4.1 3-bit parity problem – performance index for 3:3:2 network for epoch ($\lambda = 1$) vs. pattern ($\lambda < 1$) updating with sigmoid vs. tanh activation functions, but same initializations.

28.4.2 3:3:2:2 network for 3-bit parity problem

Here, we consider the same 3-bit parity problem, but using a neural network with two hidden layers of dimensions 3 and 2, i.e., a 3:3:2:2 network, adapted with Eqs. (28.3.7) and (28.3.9). Fig. 28.4.2 shows the convergence of the performance index J as a function of epoch iterations for the four cases of using epoch updating ($\lambda = 1$) vs. pattern updating ($\lambda < 1$), and using standard sigmoid vs. tanh activation functions. Again, the initial connection matrices and bias weights were initialized to random values, and as before, the algorithm was sensitive to the choice of initial values.

The complete MATLAB code for this example can be found in the file, **nn3322.m**, located in the NN folder of the ISP2e toolbox.

28.4.3 4:4:1 network for prediction of sunspot time series

In addition to their Machine Learning and AI applications, neural networks can also be used for time series forecasting or prediction. Some of the earlier and later references discussing a variety of approaches and different architectures are [1427,1430–1436].

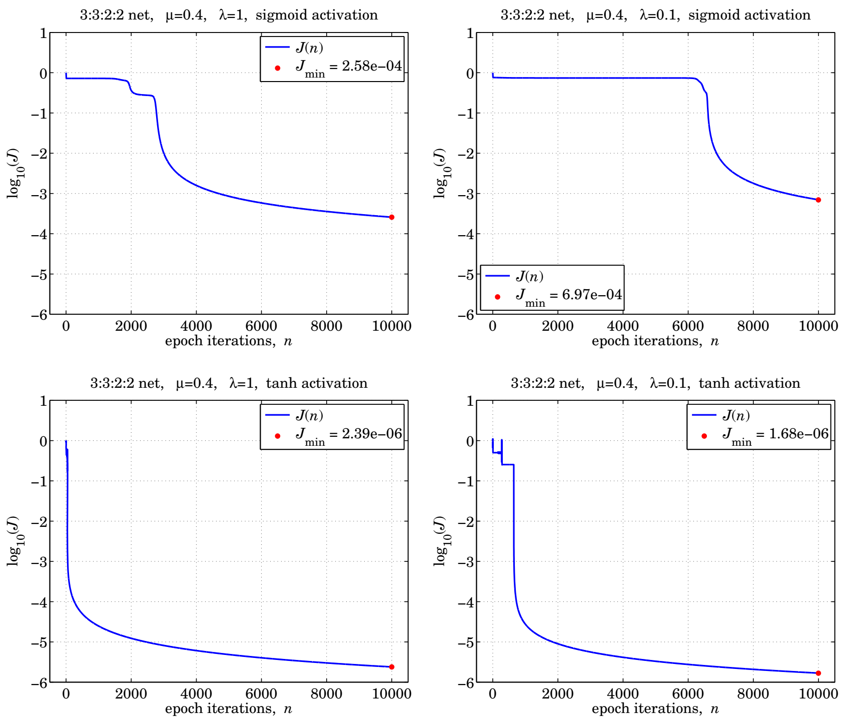


Fig. 28.4.2 3-bit parity problem – performance index for 3:3:2:2 network for epoch ($\lambda = 1$) vs. pattern ($\lambda < 1$) updating with sigmoid vs. tanh activation functions, but same initializations.

One possible approach is to use a nonlinear autoregressive (NAR) neural network model, as shown in Fig. 28.4.3, in which a time series y_n of length N, say, $\{y_0, y_1, \dots, y_{N-1}\}$, is being estimated, or predicted, in terms of its past-p time samples, $y_{n-1}, y_{n-2}, \dots, y_{n-p}$, which are fed as the inputs to a multilayer neural network that computes an estimate of y_n, denoted here by, $\hat{y}_{n/n-1}$, as a nonlinear function of these p delayed inputs.

This generalizes the standard linear autoregressive model [45], which determines the coefficients of the model by minimizing the mean-square prediction error,

$$\begin{aligned} J &= E[e_n^2] = \min \\ e_n &= y_n - \hat{y}_{n/n-1} = \text{prediction error} \end{aligned} \tag{28.4.1}$$

In the neural network case, the criterion (28.4.1) is replaced by a time average over a training set, consisting of a subset of length N_t the sequence y_n, that is, defined over the N_t time periods, $p \le n \le N_t + p - 1$,

$$J = \frac{1}{N_t}\sum_{n=p}^{N_t+p-1} e_n^2 = \frac{1}{N_t}\sum_{n=p}^{N_t+p-1}\big(y_n - \hat{y}_{n/n-1}\big)^2 = \min \tag{28.4.2}$$

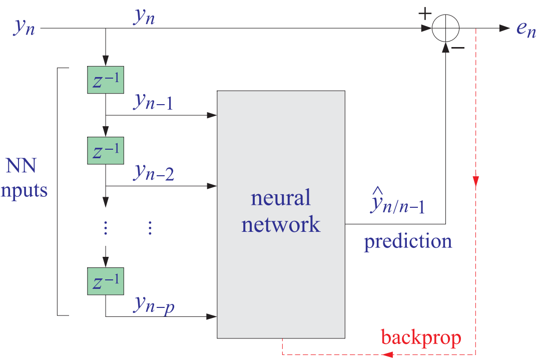


Fig. 28.4.3 Times series prediction with a neural network.

where the minimization is with respect to the network connection matrices and bias weights. If the size N_t of the training set is such that, $N_t + p < N$, then, once the neural network has been adapted, the rest of the sequence samples over the time period $N_t + p \le n \le N - 1$, can be used as a prediction set, testing the ability of the trained network to predict the future values of the time series.

If the adapted network makes a good prediction, $\hat{y}_{n/n-1}$, of y_n, then the corresponding prediction error, $e_n = y_n - \hat{y}_{n/n-1}$, should be unpredictable and resemble white noise. Thus, as an additional test of the performance of the network, one may calculate the autocorrelation function of the prediction error, and verify that it resembles that of white noise, that is, a delta-function autocorrelation.

The input and output patterns of the training set will be the following $p\times 1$ vectors and scalars, over the training time range, $p \le n \le N_t + p - 1$,

$$\text{input vectors} = \begin{bmatrix} y_{n-1} \\ y_{n-2} \\ \vdots \\ y_{n-p} \end{bmatrix}, \quad \text{scalar outputs} = y_n$$

For example, if $p = 4$ and $N_t = 8$, then, the set of input and output training pairs can be collected together in the matrices, X_0, D,

$$X_0 = \begin{bmatrix} y_3 & y_4 & y_5 & y_6 & y_7 & y_8 & y_9 & y_{10} \\ y_2 & y_3 & y_4 & y_5 & y_6 & y_7 & y_8 & y_9 \\ y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 & y_8 \\ y_0 & y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 \end{bmatrix}$$

$$D = \begin{bmatrix} y_4 & y_5 & y_6 & y_7 & y_8 & y_9 & y_{10} & y_{11} \end{bmatrix}$$

These matrices can be easily constructed with the help of the MATLAB function, **datamat.m**, found in the NN folder of the ISP2e toolbox.

Such neural networks, driven by time-delayed inputs, can also be applied to the more general problem of estimating one signal, say x_n, from another, say y_n, using a nonlinear generalization of the usual linear Wiener filtering problem [45], as depicted in Fig. 28.4.4.

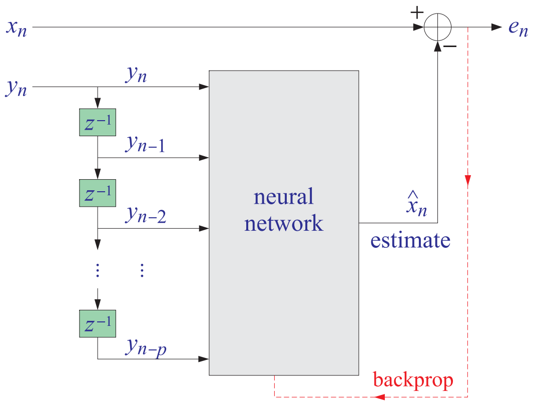


Fig. 28.4.4 Times series estimation with a neural network.

In the present example, we discuss the prediction of the sunspot time series, which has served as a benchmark for time series prediction problems. The time series y_n consists of $N = 300$ time samples, and we will use a 4:4:1 network, i.e., $p = 4$, so that there will be, $N - p = 296$, input/output patterns, over the time range, $4 \le n \le 299$. The first $N_t = 250$ of these, that is, for $4 \le n \le 253$, will be taken as the training set, and the remaining 46 ones, $254 \le n \le 299$, will be used for testing the predictive ability of the network.

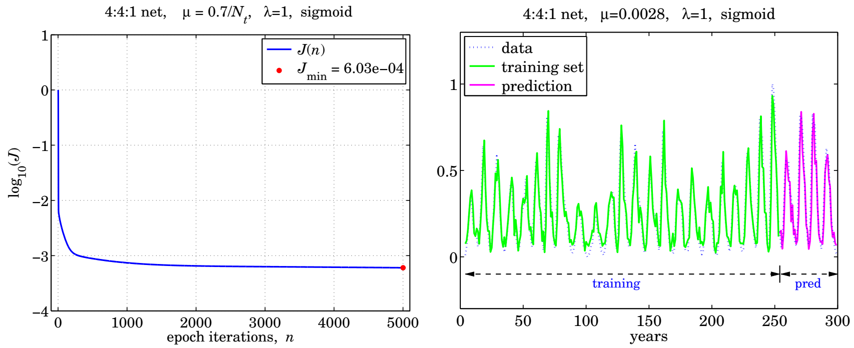
Two further simplifications in the present case are (i) to normalize the time series to unity maximum prior to applying the training algorithm, and (ii) to not use an activation function for the output layer. For the middle layer, we used a plain logistic sigmoid, and applied the training algorithm of Eqs. (28.3.8) and (28.3.9) on an epoch basis ($\lambda = 1$). Because of the normalization factor $1/N_t$ in the definition of the performance index (28.4.2), we have used the following effective adaptation constant μ in applying Eqs. (28.3.8),

$$\mu = \frac{0.7}{N_t} = 0.0028$$

The left graph of Fig. 28.4.5 shows the performance index vs. epoch iterations, and the right graph, the output of trained network over the training set, as well as the predicted output of the prediction set. We observe the initial rapid decrease of the performance index.

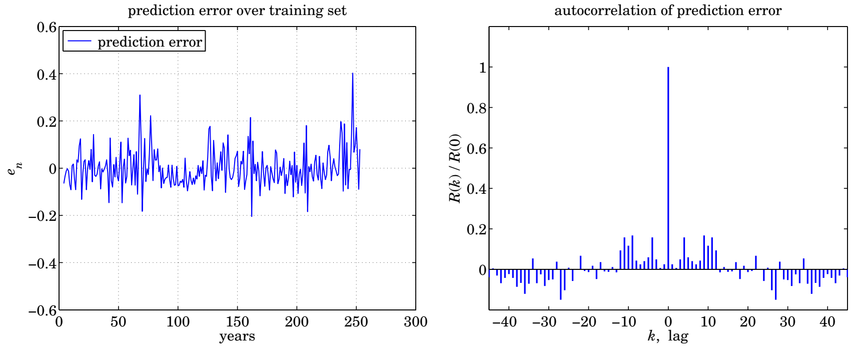
Fig. 28.4.6 shows the prediction error, $e_n = y_n - \hat{y}_{n/n-1}$, calculated only over the training set, as well as 45 lags of its autocorrelation function, resembling a delta function.

The complete MATLAB code for this example is in the file, **nn441sun.m**, located in the NN folder of the ISP2e toolbox.


4:4:1 net, μ = 0.7/N_t, λ=1, sigmoid

4:4:1 net, μ=0.0028, λ=1, sigmoid

Fig. 28.4.5 Performance index vs. epoch iterations, and output of trained network over the training set as well as the predicted output (color online).


prediction error over training set

autocorrelation of prediction error

Fig. 28.4.6 Prediction error and its autocorrelation function.

29

Appendices

A Random Number Generators

A.1 Uniform and Gaussian Generators

Random number generators are useful in DSP for performing *simulations* of various algorithms, for example, in simulating noisy data. They are also useful in real-time applications, such as adding dither noise to eliminate quantization distortions as we saw in Chapter 2, or in computer music synthesis and in the implementation of digital audio effects, such as chorusing.

Most computer systems and languages have built-in routines for the generation of random numbers. Typically, these routines generate random numbers that are distributed *uniformly* over the standardized interval $[0,1)$, although Gaussian-distributed random numbers can be generated just as easily. Figure A.1 shows the probability density functions in the uniform and Gaussian cases.

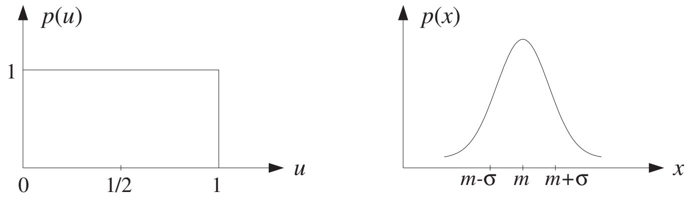


Fig. A.1 Uniform and Gaussian probability distributions.

There is a large literature on random number generators; see [1257–1275] and references therein. As reviewed by Park and Miller [1260], it is hard to find good random number generators, that is, generators that pass all or most criteria of randomness.

By far the most common generators are the so-called *linear congruential generators* (LCG). They can generate fairly long sequences of independent random numbers, typically, of the order of two billion numbers before repeating. For longer sequences, one may use *shift-register* and *lagged-Fibonacci* generators [1264–1268], which can generate astronomically long sequences of order of 2^{250} or 2^{931}.

In C, a typical call to a random number generator routine takes the form:

```
u = ran(&iseed);
```

where the output is a real number in the interval $0 \le u < 1$.

The input is an integer seed, `iseed`, which is passed by *address* because it is modified by the routine internally and its new value serves as the *next* seed.[†] Thus, the routine has one input, namely `iseed`, and two outputs, `u` and the new value of `iseed`. Figure A.2 shows the effect of a single call to such a routine, as well as successive calls which generate a sequence of independent random numbers, starting from an arbitrary initial seed.

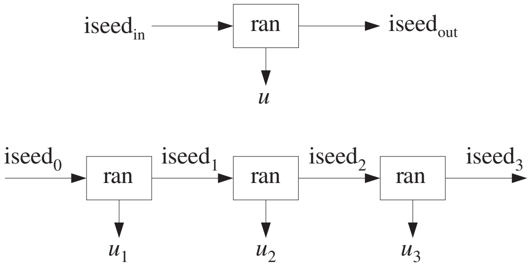


Fig. A.2 Single and successive calls to routine `ran`.

The LCG algorithm for generating u and updating the seed is defined by three integer parameters, $\{a, c, m\}$, called the multiplier, the displacement, and the modulus. Given an initial integer seed I_0 in the interval[‡] $0 \le I_0 \le m-1$, the LCG algorithm is the recursion:

$$\boxed{\begin{aligned} I_n &= (aI_{n-1} + c)\,\mathrm{mod}\,(m) \\ u_n &= \frac{I_n}{m} \end{aligned}} \qquad \text{(LCG algorithm)} \qquad (A.1)$$

Because of the modulo-m operation, all the seeds I_n are restricted to the interval:

$$0 \le I_n \le m-1$$

This implies that u_n will be in the interval $0 \le u_n < 1$, and that the length of such a sequence can be *at most* $m-1$. The parameters $\{a, c, m\}$ must be chosen carefully, such that every initial seed must result in a maximal-length sequence [1257]. We will use the following generator which has maximal length $m-1$. It was originally proposed in [1269] and has withstood the test of time [1260]. It has $c = 0$ and:

$$\boxed{a = 7^5 = 16807, \quad m = 2^{31} - 1 = 2147483647} \qquad (A.2)$$

[†] In some implementations, the seed is hidden from the user.
[‡] If $c = 0$, one must pick $I_0 \neq 0$.

With these parameters, Eq. (A.1) cannot be implemented in a straightforward fashion because the product aI can take on extremely large values that exceed the integer range of many computers. For example, if $I = m-1 = 2147483646$, then $aI \simeq 3.6\times 10^{13} \simeq 2^{45}$, which exceeds the typical signed "long" (4-byte) integer range of most micros:

$$-2^{31} \le I \le 2^{31}-1 \qquad \text{(A.3)}$$

A *portable* implementation suggested by Schrage [1258,1270] rearranges the computation of $(aI)\bmod(m)$ in such a way that *all* intermediate results remain bounded by the range (A.3). The technique is based on using the quotient q and remainder r of the division of m by a, that is,

$$\boxed{m = aq + r} \qquad \text{(A.4)}$$

where r is in the range $0 \le r \le a-1$. The key requirement for the method to work is that r satisfy the additional constraint:

$$r < q \qquad \text{(A.5)}$$

For the choice (A.2), we have the values for q and r satisfying (A.4) and (A.5):

$$\boxed{q = 127773, \quad r = 2836} \qquad \text{(A.6)}$$

Given an integer seed I in the range $0 \le I \le m-1$, the quantity $J = (aI)\bmod(m)$ is the remainder of the division of aI by m, that is,

$$aI = mK + J, \qquad \text{where} \quad 0 \le J \le m-1 \qquad \text{(A.7)}$$

Schrage's method calculates J without directly performing the multiplication aI. As a preliminary step, the seed I is divided by q, giving the quotient and remainder:

$$I = qk + j, \qquad \text{where} \quad 0 \le j \le q-1 \qquad \text{(A.8)}$$

Then, the quantity aI can be expressed as follows:

$$aI = a(qk+j) = aqk + aj = (m-r)k + aj = mk + (aj - rk) \qquad \text{(A.9)}$$

where we used $aq = m - r$ from Eq. (A.4).

Comparing Eqs. (A.9) and (A.7), it appears that $K = k$ and $J = aj - rk$. This would be true by the uniqueness of Eq. (A.7) if $aj - rk$ were in the range $0 \le aj - rk \le m-1$. Note that both quantities aj and rk lie in this range:

$$0 \le aj \le m-1, \qquad 0 \le rk \le m-1 \qquad \text{(A.10)}$$

Indeed, the first follows from the fact that $j < q$ from Eq. (A.8), so that

$$0 \le aj < aq = m - r < m$$

The second one follows from Eqs. (A.5) and (A.8):

$$0 \le rk < qk = I - j \le I \le m-1$$

Combining the inequalities (A.10) we find the range of the quantity $aj - rk$:

$$-(m-1) \le aj - rk \le m-1$$

If $aj - rk$ lies in the positive half, $0 \le aj - rk \le m-1$, then we must necessarily have $J = aj - rk$ and $K = k$. But, if it lies in the negative half, $-m+1 \le aj - rk \le -1$, we must *shift* it by m so that $1 \le aj - rk + m \le m-1$. In this case, we have $aI = mk + aj - rk = m(k-1) + (m + aj - rk)$; therefore, $J = m + aj - rk$ and $K = k-1$.

Denoting $k = \lfloor I/q \rfloor$ and $j = I\%q$, we can summarize the computation of the new seed $J = (aI)\bmod(m)$ as follows:

```
given a seed $I$ in the range $0 \le I \le m-1$ do:
    compute $J = a(I\%q) - r\lfloor I/q \rfloor$
    if $J < 0$, then shift
        $J = J + m$
```

The following routine `ran.c` is a C implementation based on Schrage's Fortran version [1258]. Note that `iseed` is declared `long` and passed by reference:

```
/* ran.c - uniform random number generator in [0, 1) */

#define  a    16807                                   that is, a = 7^5
#define  m    2147483647                              that is, m = 2^31 - 1
#define  q    127773                                  note, q = m/a = quotient
#define  r    2836                                    note, r = m%a = remainder

double ran(iseed)                                     usage: u = ran(&iseed);
long *iseed;                                          iseed passed by address
{
     *iseed = a * (*iseed % q) - r * (*iseed / q);           update seed

     if (*iseed < 0)                                  wrap to positive values
            *iseed += m;

     return (double) *iseed / (double) m;
}
```

The following program segment illustrates the usage of the routine. It generates an array of N uniform random numbers. The initial value of the seed is arbitrary:

```
long iseed;                                   seed must be long int

iseed = 123456;                               initial seed is arbitrary

for (n=0; n<N; n++)
       u[n] = ran(&iseed);
```

There exist methods that improve the quality of random number generators by making them "more random" than they already are [1257–1275].

The generated random numbers u are uniformly distributed over the interval $0 \le u < 1$, as shown in Fig. A.1. Over this interval, the probability density is flat, $p(u) = 1$. Therefore, the mean and variance of u will be:

$$
\begin{aligned}
E[u] &= \int_0^1 up(u)\,du = \int_0^1 u\,du = \frac{1}{2} \\
\sigma_u^2 &= E[u^2]-E[u]^2 = \int_0^1 u^2\,du - \frac{1}{4} = \frac{1}{3} - \frac{1}{4} = \frac{1}{12}
\end{aligned}
\tag{A.11}
$$

To generate a random number which is uniformly distributed over a different interval, say $a \le v < b$, we generate a uniform u over $[0,1)$ and then shift and scale it to obtain:

$$\boxed{v = a + (b-a)u} \tag{A.12}$$

The mean and variance of v will be:

$$E[v] = a + (b-a)\frac{1}{2} = \frac{a+b}{2}, \qquad \sigma_v^2 = \frac{(b-a)^2}{12}$$

In particular, the transformation $v = u - 0.5$ will generate *zero-mean* random numbers over the unit interval $[-0.5, 0.5)$, and the transformation $v = 2u - 1$ will generate *zero-mean* random numbers over the length-2 interval $[-1, 1)$.

More complicated transformations and combinations of uniform random numbers can be used to generate random numbers that are distributed according to other probability distributions, such as Gaussian, exponential, Poisson, binomial, etc. [1257–1275].

A method of generating *Gaussian-distributed* random numbers is based on the *central limit theorem*, which states that the sum of a large number of independent random variables is Gaussian. In particular, summing only 12 independent *uniform* random numbers gives a very good approximation to a Gaussian:

$$v = u_1 + u_2 + \cdots + u_{12} \tag{A.13}$$

The mean of v is the sum of the individual means, and because u_i are independent, the variance of v will be the sum of the variances:

$$
\begin{aligned}
E[v] &= E[u_1] + \cdots + E[u_{12}] = \frac{1}{2} + \cdots + \frac{1}{2} = 12 \times \frac{1}{2} = 6 \\
\sigma_v^2 &= \sigma_{u_1}^2 + \cdots + \sigma_{u_{12}}^2 = \frac{1}{12} + \cdots + \frac{1}{12} = 12 \times \frac{1}{12} = 1
\end{aligned}
$$

Because each u_i has finite range $0 \le u_i < 1$, the range of v will also be finite: $0 \le v < 12$, with mean at 6. Even though v has finite range, it represents an adequate approximation to a Gaussian because there are $\pm 6\sigma_v$ on either side of the mean, and we know that for a Gaussian distribution more than 99.99% of the values fall within $\pm 4\sigma_v$.

To generate a Gaussian random number x with a given mean $E[x] = m$ and variance $\sigma_x^2 = s^2$, we may shift and scale v:

$$x = m + s(v - 6)$$

The following routine `gran.c` implements this method using the uniform routine `ran`. Its inputs are $\{m, s\}$ and a seed. Its outputs are the random number x and the updated seed:

```
/* gran.c - gaussian random number generator */

double ran();                                   uniform generator

double gran(m, s, iseed)                        usage: x = gran(m, s, &iseed);
double m, s;                                    m = mean, s^2 = variance
long *iseed;                                    iseed passed by address
{
       double v = 0;
       int i;

       for (i = 0; i < 12; i++)                 sum 12 uniform random numbers
              v += ran(iseed);

       return s * (v - 6) + m;                  adjust mean and variance
}
```

Its usage is demonstrated by the following program segment. As in the case of `ran`, the seed must be declared to be `long`:

```
iseed = 123456;                                 initial seed is arbitrary

for (n=0; n<N; n++)
       x[n] = gran(m, s, &iseed);
```

A.2 Low-Frequency Noise Generators

A sequence of zero-mean uniform random numbers generated by successive calls to `ran`, such as,

$$u_n = \text{ran}(\&\text{iseed}) - 0.5, \qquad n = 0, 1, 2, \dots$$

corresponds to a *white noise* signal because the generated numbers are mutually independent. The autocorrelation function and power spectral density of such signal are,

$$R_{uu}(k) = \sigma_u^2 \delta(k), \qquad S_{uu}(f) = \sigma_u^2 \tag{A.14}$$

with variance $\sigma_u^2 = 1/12$.

Such a sequence is *purely random* in the sense that each sample has no memory or dependence on the previous samples. Because $S_{uu}(f)$ is flat, the sequence will contain all frequencies in equal proportions and will exhibit equally slow and rapid variations in time.

The rate at which this sequence is produced is equal to the sampling rate f_s, that is, one random number per sampling instant. In some applications, such as computer music [110–114,116], or for generating $1/f$ noise, it is desired to generate random numbers at a *slower* rate, for example, one random number every D sampling instants. This corresponds to a generation frequency of f_s/D random numbers per second.

If a new random number is generated every D sampling instants, that is, at times $n = 0, D, 2D, 3D, \dots$, then the signal values filling the gaps between these random numbers must be calculated by *interpolation*. Two simple ways of interpolating are to use hold or linear interpolators [110]. They are shown in Fig. A.3 for $D = 5$.

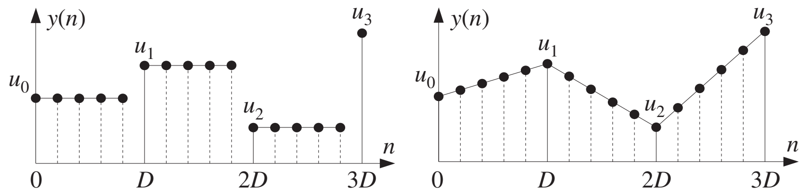


Fig. A.3 Low-frequency noise generation using hold and linear interpolators.

In the hold interpolator, each random number is held constant for D sampling instants. In the linear interpolator, two successive random numbers, separated by D time units, are connected by a straight line and the intermediate samples lie on that line.

The following routine `ranh.c` implements the hold generator. Its inputs are the desired period D and a seed. Its outputs are a *zero-mean* random number and an updated seed.

```
/* ranh.c - hold random number generator of period D */

double ran();                                   uniform generator
void cdelay2();                                 circular delay

double ranh(D, u, q, iseed)                     usage: y = ranh(D, u, &q, &iseed);
int D, *q;                                      q is cycled modulo-D
double *u;                                      u = 1-dimensional array
long *iseed;                                    q, iseed are passed by address
{
       double y;

       y = u[0];                                hold sample for D calls

       cdelay2(D-1, q);                         decrement q and wrap mod-D

       if (*q == 0)                             every D calls,
              u[0] = ran(iseed) - 0.5;          get new u[0] (zero mean)

       return y;
}
```

The temporary variable u is a 1-dimensional array that holds the current value of the random number for D calls. The index q is cycled modulo-D with the help of the circular delay routine `cdelay2`, which decrements it circularly during each call. Every D calls, the index q cycles through zero, and a new zero-mean random number is obtained by a call to `ran`, which overwrites the value of $u[0]$ and also updates the seed. Before the first call, the array $u[0]$ must be filled with an initial (zero-mean) random number. The initialization and usage of the routine are illustrated by the following program segment:

```
double *u;                                      u is a 1-dimensional array
int D, q;
long iseed = 654321;                            initial seed is arbitrary

u[0] = ran(&iseed) - 0.5;                       initialize u (zero mean)
q = 0;                                          initialize q
```

```
for (n=0; n<N; n++)
       y[n] = ranh(D, u, &q, &iseed);           q, iseed are passed by address
```

For the linear interpolation case, we need to keep track of *two* successive random values u, say $u[0]$ and $u[1]$, and connect them linearly. Because the slope of the straight line between $u[0]$ and $u[1]$ is $(u[1]-u[0])/D$, the linearly interpolated samples will be

$$y = u[0]+\big(u[1]-u[0]\big)\frac{i}{D}, \qquad i = 0,1,\dots,D-1$$

Because we use the routine `cdelay2`, the circular index q takes periodically the successive values: $q = 0, D-1, D-2, \dots, 1$. These may be mapped to the interpolation index i by:

$$i = (D-q)\%D = 0,1,\dots,D-1$$

where the modulo-D operation is effective only when $q = 0$ giving in that case $i = D\%D = 0$. The following routine `ranl.c` implements the linearly interpolated periodic generator, where now the temporary variable u is a *two-dimensional* array:

```
/* ranl.c - linearly interpolated random generator of period D */

double ran();                                   uniform generator
void cdelay2();                                 circular delay

double ranl(D, u, q, iseed)                     usage: y = ranl(D, u, &q, &iseed);
int D, *q;                                      q is cycled modulo-D
double *u;                                      u = 2-dimensional array
long *iseed;                                    q, iseed are passed by address
{
       double y;
       int i;

       i = (D - *q) % D;                        interpolation index

       y = u[0] + (u[1] - u[0]) * i / D;        linear interpolation

       cdelay2(D-1, q);                         decrement q and wrap mod-D

       if (*q == 0) {                           every D calls,
              u[0] = u[1];                      set new u[0] and
              u[1] = ran(iseed) - 0.5;          get new u[1] (zero mean)
              }

       return y;
}
```

Every D calls, as q cycles through zero, the value of $u[1]$ is shifted into $u[0]$ and $u[1]$ is replaced by a new zero-mean random number by a call to `ran`. The initialization and usage of the routine are illustrated by the program segment:

```
double u[2];                                    u is a 2-dimensional array
int D, q;
```

```
long iseed = 654321;                             initial seed is arbitrary

u[0] = ran(&iseed) - 0.5;                        initialize u[0] and u[1]
u[1] = ran(&iseed) - 0.5;                        zero-mean initial values
q = 0;                                           initialize q

for (n=0; n<N; n++)
      y[n] = ranl(D, u, &q, &iseed);             q, iseed are passed by address
```

Figure A.4 shows typical sequences $y[n]$ both for the hold and linear interpolator generators, for the cases $D = 5$ and $D = 10$. The sequence length was $N = 100$.

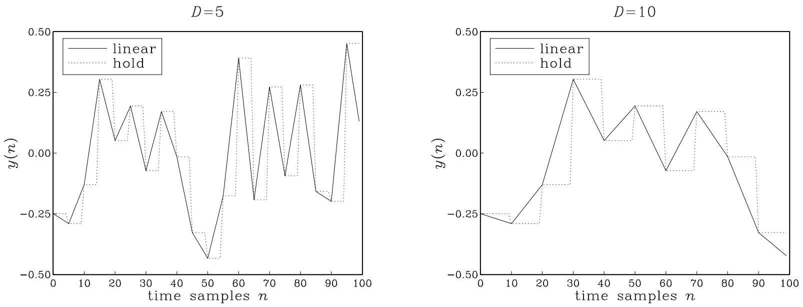


Fig. A.4 Hold and linearly interpolated low-frequency random sequences.

The routines `ranh` and `ranl` generate random numbers y in the range $-0.5 \le y < 0.5$, with mean $E[y] = 0$. In the hold case, the variance is $\sigma_y^2 = \sigma_u^2 = 1/12$, and in the linear case $\sigma_y^2 = (2D^2+1)\sigma_u^2/3D^2$.

The hold and linear interpolator generators can be given a convenient *filtering* interpretation, as shown in Fig. A.5. The interpolated random sequence $y(n)$ can be thought of as the *output* of an *interpolation filter* whose input is the *low-rate* sequence of random numbers u_m occurring at rate f_s/D and being separated from each other by $D-1$ zeros. Each input random number causes the filter to produce its impulse response filling the gap till the next random number D samples later. The impulse responses for the hold and linear cases are shown in Fig. 14.3.1.

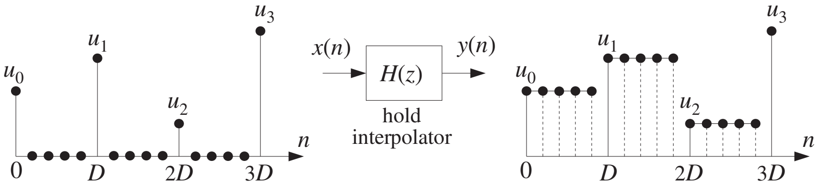


Fig. A.5 Filtering interpretation of hold interpolator.

To turn the input and output sequences $x(n)$ and $y(n)$ into stationary random sequences, we must make a slight modification to the generation model [1276]. Instead of

assuming that the random numbers u_m, $m = 0, 1, \dots$ are generated exactly at multiples of D, that is, at times $n = mD$, we introduce a random delay shift in the entire sequence and assume that the u_m's are generated at times:

$$n = mD + d$$

where the delay d is a *discrete-valued random variable* taking on the possible values $\{0, 1, \dots, D-1\}$ with *uniform probability*, that is, $p(d) = 1/D$. In this case, the sequences $x(n)$ and $y(n)$ defining the generation model are obtained by:

$$\begin{aligned} x(n) &= \sum_{m=-\infty}^{\infty} \delta(n - mD - d)\,u_m \\ y(n) &= \sum_{m=-\infty}^{\infty} h(n - mD - d)\,u_m \end{aligned} \tag{A.15}$$

Each realization of $x(n)$ and $y(n)$ is defined by a random value of d from the set $\{0, 1, \dots, D-1\}$ and the random numbers u_m. The particular case $D = 5$ and $d = 3$ is shown in Fig. A.6 for the hold interpolator.

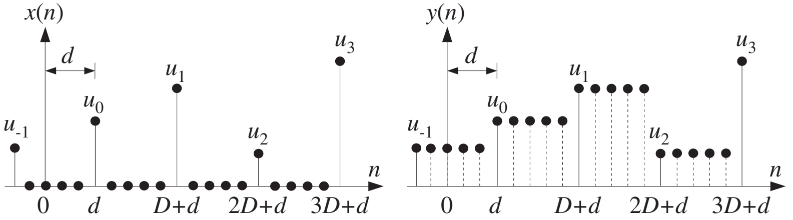


Fig. A.6 Randomly delayed input and output sequences.

If we assume that the random numbers u_m are zero-mean, mutually independent, and uniformly distributed over the interval $[-0.5, 0.5)$, then the sequence $x(n)$ becomes a zero-mean *white noise* sequence with variance $\sigma_x^2 = \sigma_u^2/D$, and therefore, its filtered version $y(n)$ will be stationary (in the steady state) and have power spectrum as given by Eq. (9.7.2):

$$\boxed{S_{yy}(f) = |H(f)|^2 \frac{\sigma_u^2}{D}} \tag{A.16}$$

where for the hold and linear interpolators, we have from Section 14.3:

$$H(f) = \frac{\sin(\pi f D/f_s)}{\sin(\pi f/f_s)} e^{-j\pi f(D-1)/f_s}, \qquad H(f) = \frac{1}{D}\left[\frac{\sin(\pi f D/f_s)}{\sin(\pi f/f_s)}\right]^2 \tag{A.17}$$

The above technique of introducing a random delay to guarantee the stationarity of the output process is standard in digital communication applications [1277]. For the purpose of generating low-frequency random sequences, such a delay is a welcome feature that improves the flexibility of the model.

The implementation of an overall random delay shift in the generated sequence can be accomplished by initializing the routines `ranh` or `ranl` not at $q = 0$, but at $q = d$, where the random integer $d = 0, 1, \dots, D-1$ can be generated by an initial call to `ran`:

$$d = \lfloor D \cdot \text{ran}(\&\text{iseed}) \rfloor$$

The interpolation between the low-rate random numbers u_m can also be accomplished using more sophisticated interpolation filters, whose frequency response $H(f)$ closely approximates an ideal lowpass filter with cutoff $f_s/2D$. The subject of interpolation filter design was discussed in Chapter 14. Any of those designs and their efficient, so-called polyphase, realizations can be used in place of the hold and linear interpolators. For example, using the polyphase sample processing algorithm of Eq. (14.2.20), we may write the low-frequency random number generation algorithm:

```
repeat forever:
    if ($q = 0$)
        $w[0] = \text{ran}(\&\text{iseed}) - 0.5$
    $i = (D - q)\%D$
    $y = \text{dot}(P, \mathbf{h}_i, \mathbf{w}) = \text{output}$
    $\text{cdelay2}(D-1, \&q)$
    if ($q = 0$)
        $\text{delay}(P, \mathbf{w})$
```

(A.18)

where $\mathbf{h}_i$ is the ith polyphase subfilter of order P, $\mathbf{w}$ is the low-rate delay line holding the random numbers u_m. The length of the interpolation filter is $N = 2DM + 1$, and $P = 2M - 1$. As in the routines `ranh` and `ranl`, q is passed by address and gets decremented circularly with the help of `cdelay2`. As q cycles through zero every D calls, the low-rate delay line is shifted and a new (zero-mean) random number is entered into $\mathbf{w}$. See Problem A.5 for a simulation.

A.3 1/f Noise Generators

$1/f$*-noise* is also known as *flicker* or *pink* noise, depending on the context. It is characterized by a power spectrum that falls in frequency like $1/f$:

$$S(f) = \frac{A}{f} \tag{A.19}$$

To avoid the infinity at $f = 0$, this behavior is assumed valid for $f \geq f_{\min}$, where $f_{\min}$ is a desired minimum frequency. The spectrum (A.19) is characterized by a 3-dB *per octave* drop, that is, whenever f doubles, $S(f)$ drops by a factor of $1/2$. Indeed, we have:

$$S(2f) = \frac{A}{2f} = \frac{1}{2}S(f) \quad \Rightarrow \quad 10\log_{10}\left[\frac{S(f)}{S(2f)}\right] = 10\log_{10}(2) = 3 \text{ dB}$$

The amount of power contained within a frequency interval $[f_1, f_2]$ is

$$\int_{f_1}^{f_2} S(f)\,df = A\ln\left(\frac{f_2}{f_1}\right)$$

This implies that the amount of power contained in *any* octave interval is the same. That is, if $f_2 = 2f_1$, then $A\ln(f_2/f_1) = A\ln(2f_1/f_1) = A\ln(2)$, which is *independent* of the octave interval $[f_1, 2f_1]$.

$1/f$ noise is ubiquitous in nature. It is observed in solid-state circuits, astrophysics, oceanography, geophysics, fractals, and music; see [1278–1287] and references therein. In audio engineering, it is known as *pink noise* and is used to test the frequency response of audio equipment such as loudspeakers. It represents the *psychoacoustic equivalent of white noise* because our auditory system is better matched to the logarithmic octave frequency scale than the linear scale.

In this section, we present a $1/f$ noise generator that uses the low-frequency generator `ranh`. It is based on an algorithm by Voss, mentioned in [1279]. The algorithm is a variant of the so-called "spreading of time constants" models that have been proposed to explain the physics of $1/f$ noise [1278,1283,1287].

Such models assume that the noise consists of the sum of several white noise processes that are filtered through first-order lowpass filters having time constants that are successively larger and larger, forming a geometric progression. In Voss's algorithm, the role of the lowpass filters is played by the hold interpolation filters.

In our notation, Voss's $1/f$-noise generator is defined by taking the *average* of several periodically held random numbers with periods that form a geometric progression, $D_b = 2^b$, with $b = 0, 1, 2, \dots, B-1$. That is, we define the random number sequence:

$$\boxed{y(n) = \frac{1}{B}\sum_{b=0}^{B-1} y_b(n) = \frac{1}{B}\big[y_0(n) + y_1(n) + \cdots + y_{B-1}(n)\big]} \tag{A.20}$$

where the term $y_b(n)$ is a periodically held random number with period $D_b = 2^b$, produced by calling the routine `ranh`:

$$y_b(n) = \text{ranh}(D_b, \&u[b], \&q[b], \&\text{iseed}) \tag{A.21}$$

Each term $y_b(n)$ must have its own 1-dimensional array $u[b]$ and circular index $q[b]$. The same seed variable `iseed` is used by all terms, but because it is updated at different periods, the terms $y_b(n)$ will be mutually independent.

The following routine `ran1f.c` implements this algorithm. Its inputs are the number of "bits" B, the B-dimensional arrays u and q, and a seed. Its outputs are a $1/f$-noise random number and an updated seed:

```
/* ran1f.c - 1/f random number generator */

double ranh();                                  random hold periodic generator

double ran1f(B, u, q, iseed)                    usage: y = ran1f(B, u, q, &iseed);
int B, *q;                                      q, u are B-dimensional
double *u;
long *iseed;                                    passed by address
```

```
{
    double y;
    int b;

    for(y=0, b=0; b<B; b++)
          y += ranh(1<<b, u+b, q+b, iseed);          period = (1<<b) = 2^b

    return y / B;
}
```

Because the component signals $y_b(n)$ are mutually independent with mean $E[y_b]= 0$ and variance $\sigma_{y_b}^2 = \sigma_u^2 = 1/12$, it follows that the mean and variance of the $1/f$-noise sequence $y(n)$ will be:

$$E[y]= 0, \qquad \sigma_y^2 = \frac{\sigma_u^2}{B} = \frac{1}{12B} \tag{A.22}$$

The initialization and usage of the routine are illustrated by the following program segment, which generates N random numbers $y(n)$:

```
double *u;
int *q;
long iseed=123456;                                   initial seed is arbitrary

u = (double *) calloc(B, sizeof(double));            B-dimensional
q = (int *) calloc(B, sizeof(int));                  B-dimensional

for (b=0; b<B; b++) {
       u[b] = ran(&iseed) - 0.5;                     initialize u's
       q[b] = (1<<b) * ran(&iseed);                  random initial q's
       }

for (n=0; n<N; n++)                                  N is arbitrary
       y[n] = ran1f(B, u, q, &iseed);
```

As discussed in the previous section, to guarantee stationarity the initial values of q must be selected randomly from the set $\{0, 1, \dots, D-1\}$. To see how the various terms $y_b(n)$ combine to generate $y(n)$, consider the case $B = 4$

$$y(n)= \frac{1}{4}\big[y_0(n)+y_1(n)+y_2(n)+y_3(n)\big]$$

Figures A.7 and A.8 show the generated signal $y(n)$ and its four component signals $y_b(n)$, for $n = 0, 1, \dots, 199$. The periods of the four component signals are 1, 2, 2^2, 2^3. For convenience, all initial random delays were set to zero, that is, $q[b]= 0$, $b = 0, 1, 2, 3$.

The power spectrum of the model (A.20) does not have an exact $1/f$ shape, but is close to it. Therefore, it can be used in practice to simulate $1/f$ noise. The $1/f$ shape is approximated for frequencies $f \ge f_{\min}$, where:

$$\boxed{f_{\min} = \frac{f_s}{2^B}} \tag{A.23}$$

This expression can be used to pick the proper value of B for a particular simulation. That is, given a desired minimum frequency we calculate $B = \log_2(f_s/f_{\min})$.

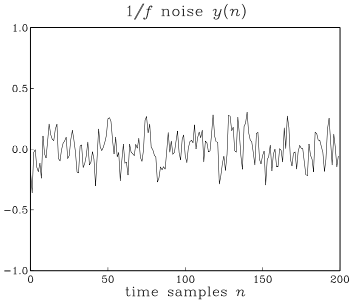


Fig. A.7 $1/f$-noise with $B = 4$.

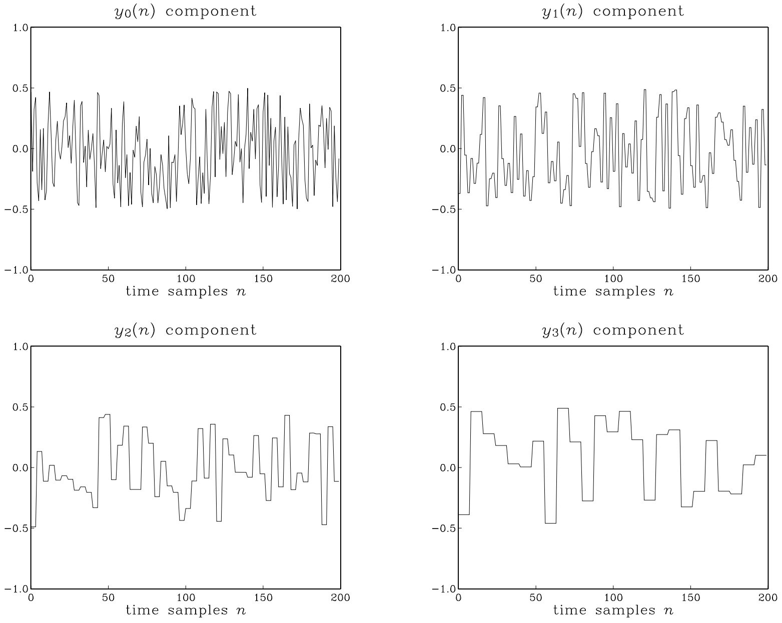


Fig. A.8 Components of $1/f$ noise.

Because the $y_b(n)$ components are mutually independent, the power spectrum of $y(n)$ will be equal to the *sum* of the power spectra of the individual parts. Using Eqs. (A.16) and (A.17), we have:

$$\boxed{S_{yy}(f)=\frac{1}{B^2}\sum_{b=0}^{B-1}S_{y_by_b}(f)=\frac{1}{B^2}\sum_{b=0}^{B-1}\frac{1}{2^b}\frac{\sin^2(\pi f 2^b/f_s)}{\sin^2(\pi f/f_s)}\,\sigma_u^2} \tag{A.24}$$

The DC value of the spectrum at $f = 0$ is not infinite as suggested by Eq. (A.19); it is finite, but large. Taking the limit $f \to 0$, we find: $S_{yy}(0)= (2^B - 1)\sigma_u^2/B^2$. Figure A.9 shows the theoretical spectrum computed via Eq. (A.24) for $B = 8$, together with the exact $1/f$ curve, and the estimated spectra obtained by the periodogram averaging method, for the two cases of averaging $K = 2$ and $K = 200$ zero-mean blocks of length $N = 256$, generated by calls to `ran1f`.

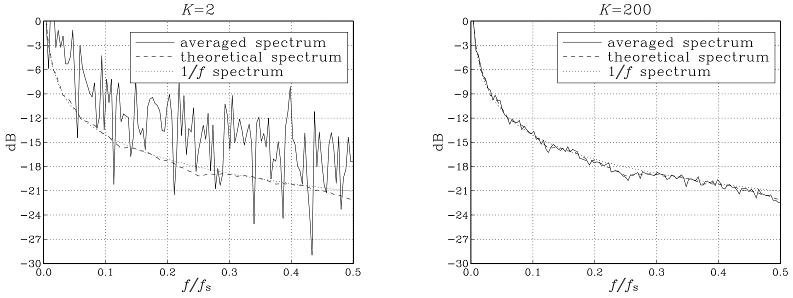


Fig. A.9 Theoretical and estimated $1/f$ power spectra, for $B = 8$.

All spectra have been normalized to unity at $f = f_{\min} = f_s 2^{-B} = f_s/256$, and are plotted in dB, that is, $10\log_{10}\big(S(f)/S(f_{\min})\big)$. Basically, they attenuate by 3 dB per octave. The calculation of the averaged periodograms was done by the following program segment, which implements the periodogram averaging method,

```
for (k=0; k<K; k++) {                                    average K periodograms
     for (b=0; b<B; b++) {                               initialize kth block
          u[b] = ran(&iseed) - 0.5;
          q[b] = (1<<b) * ran(&iseed);                   randomized initial q's
          }

     for (n=0; n<N; n++)                                 generate kth block
          Y[n] = cmplx(ran1f(B,u,q,&iseed), 0.0);        complexify for FFT

     fft(N, Y);                                          FFT of kth block

     for(i=0; i<N; i++)
          S[i] += cabs(Y[i]) * cabs(Y[i]);              accumulate kth periodogram
     }
```

A.4 Problems

A.1 Consider the random signal model of Eq. (A.15). Show that $x(n)$ is a zero-mean white noise sequence with variance $\sigma_x^2 = \sigma_u^2/D$. Show that the output signal $y(n)$ has zero mean and variance is in the two cases:

$$\sigma_y^2 = \sigma_u^2 \quad \text{(hold)}, \qquad \sigma_y^2 = \frac{2D^2+1}{3D^2}\sigma_u^2 \quad \text{(linear)}$$

A.2 For the hold interpolator case of Eq. (A.15), show that the autocorrelation function of the output signal is:

$$R_{yy}(k)= E[y(n+k)y(n)]= \left(1-\frac{|k|}{D}\right)\sigma_u^2, \quad -D \le k \le D$$

A.3 *Computer Experiment: Autocorrelation Function of Held Noise.* Generate a length-100 block of randomly held zero-mean random numbers with period $D = 5$. Using the routine `corr`, compute the sample autocorrelation $\hat{R}_{yy}(k)$ of the block for $k = 0,1,\dots,99$, and plot it together with the theoretical autocorrelation of Problem A.2.

A.4 *Computer Experiment: Power Spectrum of Held Noise.* For the cases $D = 2$, 5, and 10, plot the theoretical power spectrum of the hold interpolation noise given by Eq. (A.16), over the interval $0 \le f \le f_s$. Then, for the case $D = 5$, generate $K = 200$ blocks of held numbers $y(n)$ of length $N = 256$, compute the periodogram of each block using a 256-point FFT, and average the K periodograms to get an estimate of the power spectrum. Plot that estimate together with the theoretical power spectrum. Use absolute scales (not dB) and normalize all spectra to unity at DC. (The steps for such a computation were illustrated at the end of Section A.3.)

A.5 *Computer Experiment: Interpolated Random Number Generators.* The algorithm of Eq. (A.18) generates low-frequency random numbers using a general interpolation filter. The input random numbers are generated at a rate f_s/L and are interpolated by an L-fold interpolator resulting in a random sequence at rate f_s. Write a general routine, say `rani.c`, that implements this algorithm. Its inputs should be the $L\times(P+1)$ polyphase filter matrix $h[i][n]$ (designed independently), the low-rate delay line vector $\mathbf{w}$, an input/output seed variable declared as in `ranh` or `ranl`, and the circular index q that cycles modulo-L.

Using this routine, write a test program that generates $N_{\text{tot}} = 150$ random numbers of frequency $f_s/10$, i.e., $L = 10$. The delay line $\mathbf{w}$ must be initialized as in Eq. (14.2.19). Use three interpolator designs—all implemented by your routine `rani`: a hold, a linear, and a Kaiser interpolator with given stopband attenuation A and transition width Δf (you may choose values such that $P = 5$). Plot and compare the three length-150 random number sequences.

A.6 Using the result of Problem A.2, show that the autocorrelation of the $1/f$ noise model of Eq. (A.20) is:

$$R_{yy}(k)=\left[1-\frac{b(k)}{B}-\frac{2|k|}{B}\left(2^{-b(k)}-2^{-B}\right)\right]\frac{\sigma_u^2}{B} \tag{A.25}$$

where $b(k)$ is the ceiling quantity $b(k)=\lceil\log_2(|k|+1)\rceil$. Draw a sketch of $R_{yy}(k)$ for $B = 4$. Show that the maximum correlation length is $k_{\max} = 2^{B-1} - 1$.

A.7 *Computer Experiment: Autocorrelation Function of* $1/f$ *Noise.* Generate a block of $1/f$ noise samples $y(n)$, $n = 0,1,\dots,N-1$, where $N = 2000$ assuming $B = 8$. Using the correlation routine `corr.c`, compute and plot the sample autocorrelation of the sequence for lags $0 \le k \le 150$. Compare it Eq. (A.25).

A.8 *Computer Experiment: Alternative* $1/f$ *Noise Generator.* An alternative $1/f$ noise generator is based on the "spreading of time constants" model [1278,1283,1287] in which white noise signals are filtered through first-order lowpass filters with time constants in geometric progression, $\tau_b = \tau_0 c^b$, $b = 0, 1, \dots$, where $c > 1$. Discrete-time versions of such filters are of the form $H(z) = G/(1 - az^{-1})$, where a is related to the sampling interval and time constant by $a = e^{-T/\tau}$ and the gain is chosen to be $G = \sqrt{1 - a^2}$ in order for the NRR of the filter to be unity. For small T/τ, we may use the first-order approximation $a = 1 - T/\tau$. The generation model is based on summing the outputs of B such filters:

$$\begin{aligned} y(n) &= \frac{1}{\sqrt{B}} \sum_{b=0}^{B-1} y_b(n) \\ y_b(n) &= a_b y_b(n-1) + G_b x_b(n)\,, \quad b = 0, 1, \dots, B-1 \end{aligned} \tag{A.26}$$

where $G_b = \sqrt{1 - a_b^2}$ and $x_b(n)$ are mutually independent, zero-mean, unit-variance, white Gaussian signals that can be generated by calls to gran. The $1/\sqrt{B}$ factor normalizes $y(n)$ to unit variance. The power spectrum of the signal $y(n)$ will be:

$$S(\omega) = \frac{1}{B}\sum_{b=0}^{B-1} |H_b(\omega)|^2 = \frac{1}{B}\sum_{b=0}^{B-1} \frac{1 - a_b^2}{1 - 2a_b\cos\omega + a_b^2} \tag{A.27}$$

The filter parameters can be expressed as $a_b = e^{-T/\tau_b} \simeq 1 - T/\tau_b = 1 - c^{-b}T/\tau_0$. As a practical matter, we would like the model to approximate $1/f$ noise over a given interval $\omega_{\min} \le \omega \le \omega_{\max}$. These limits are inversely proportional to the longest and shortest time constants τ_b [1287]. Thus, we may set $c^{B-1} = \omega_{\max}/\omega_{\min}$. Using the approximation $a = 1 - \omega_c$ of Example 15.2, we obtain the "design" equations for the filter parameters:

$$c = \left(\frac{\omega_{\max}}{\omega_{\min}}\right)^{1/(B-1)}, \qquad a_b = 1 - \omega_{\min} c^{B-1-b} = 1 - \omega_{\max} c^{-b}, \tag{A.28}$$

for $b = 0, 1, \dots, B-1$. The model works well over a wide range of frequencies, especially when $\omega_{\min}$ is very small [1283,1288]. The positivity of a_b requires $\omega_{\max} < 1$ in rads/sample. However, the model also works if we allow negative a_b's as long as they have $|a_b| < 1$. This requires that $\omega_{\max} < 2$ or in terms of the sampling frequency $f_{\max} < f_s/\pi$. To get a feeling for the range of applicability of this model consider the values:

$$\begin{aligned} \omega_{\min} &= 0.01\pi,\ 0.001\pi \\ \omega_{\max} &= 0.1\pi,\ 0.2\pi,\ 0.3\pi,\ 0.4\pi, 0.5\pi,\ 0.6\pi \end{aligned}$$

For each pair $\{\omega_{\min}, \omega_{\max}\}$, compute the model spectrum (A.27) over the interval $\omega_{\min} \le \omega \le \omega_{\max}$ and plot it together with the desired $1/\omega$ spectrum. Use dB scales and normalize each spectrum to 0 dB at $\omega_{\min}$. In each case, you need to experiment to find the best value for B, but typically, $B = 2$–6.

Next, for each frequency pair, generate K sequences $y(n)$ each of length L by the Eqs. (A.26). For each sequence compute its L-point FFT periodogram and average the K periodograms; (such a computation was outlined at the end of Section A.3). Use $K = 200$ and $L = 256$. Plot the averaged periodogram together with the model spectrum. Use dB scales and normalize all spectra to the same DFT frequency (for this problem, it is better to normalize them at the second DFT frequency; that is, $\omega = 2\pi/L$.)

A.9 *Computer Experiment: Yet Another $1/f$ Noise Generator.* A simple and effective $1/f$ noise generator that covers almost the entire Nyquist interval is obtained by sending a zero-mean white noise sequence $x(n)$ of variance σ_x^2 through the following third-order filter [1289]:

$$H(z) = G\frac{(1 - 0.98444z^{-1})(1 - 0.83392z^{-1})(1 - 0.07568z^{-1})}{(1 - 0.99574z^{-1})(1 - 0.94791z^{-1})(1 - 0.53568z^{-1})} \tag{A.29}$$

The resulting output sequence $y(n)$ has power spectrum $S_{yy}(\omega) = |H(\omega)|^2\sigma_x^2$, according to Eq. (9.7.1). The filter is lowpass with finite gain at DC; its 3-dB frequency is approximately $\omega_c = 0.0015\pi$ (e.g., 30 Hz at the audio rate $f_s = 40$ kHz). Beyond ω_c the filter's magnitude response squared behaves approximately like $1/\omega$ over the rest of the Nyquist interval, that is,

$$|H(\omega)|^2 \simeq \frac{\text{const.}}{\omega}, \qquad \text{for} \quad \omega_c \lesssim \omega \le \pi$$

Thus, the output sequence $y(n)$ will imitate $1/f$ noise. To generate $y(n)$, one needs to filter a white noise input $x(n)$ through $H(z)$. To avoid the transients introduced by the filter, the first n_{eff} outputs must be discarded, where $n_{\text{eff}} = \log\epsilon/\log a$ is the ϵ-level time constant of the filter (for example, with $\epsilon = 0.05$, $a = \max_i |p_i| = 0.99574$, we have $n_{\text{eff}} = 702$).

This model is similar to the spreading of time constants model discussed in Problem A.8, except it is the cascade instead of the sum of factors with time constants increasing in geometric progression. Indeed, a more general such model would be of the form:

$$H(z) = G\frac{(1 - bz^{-1})(1 - b^c z^{-1})(1 - b^{c^2} z^{-1})}{(1 - az^{-1})(1 - a^c z^{-1})(1 - a^{c^2} z^{-1})} \tag{A.30}$$

The time constants n_{eff} of the three poles are in geometric proportions $1 : c : c^2$.

a. Determine the parameters $\{a, b, c\}$ of the model (A.30) by matching them to those of the model (A.29), that is, set $a = 0.99574$, $a^c = 0.94791$, and $b = 0.98444$. Then solve for c and determine the remaining pole and zeros: a^{c^2}, b^c, b^{c^2}.

b. Evaluate $|H(\omega)|^2$ of the filters (A.29) and (A.30) at 500 equally spaced frequencies over the interval $0.002\pi \le \omega \le \pi$ and plot them on the same graph together with the curve $1/\omega$ evaluated over the same frequencies. For plotting convenience, use dB scales for all responses and normalize them to 0 dB at $\omega = 0.01\pi$. Note the characteristic 3 dB/octave drop.

c. Define the overall gain factor G such that the *NRR* of the filter (A.29) is unity and therefore the generated $1/f$ noise sequence $y(n)$ has variance $\sigma_y^2 = \sigma_x^2$. Verify that $G = 0.57534$. Then, write the sample processing algorithm for generating the output samples $y(n)$ using the cascade realization of the three sections.

B Prolate Spheroidal Wave Functions

Prolate spheroidal wave functions (PSWF) of order zero provide an ideal basis for representing *bandlimited* signals that are also *maximally concentrated* in a finite time interval. They were extensively studied by Slepian, Pollak, and Landau in a series of papers [1290–1294] and have since been applied to a wide variety of applications, such as signal extrapolation, deconvolution, communication systems, waveform design, antennas, diffraction-limited optical systems, laser resonators, and acoustics.

In this Appendix,[†] we summarize their properties and provide a MATLAB function for their computation. We have used them in [46] in our discussion of superresolution

[†] adapted from the author's book on *Electromagnetic Waves and Antennas* [46]

and its dual, supergain, and their relationship to superoscillations. Further details on superoscillations may be found in references [1327–1340]. More information on superresolution, signal restoration, degrees of freedom, and focusing of evanescent plane waves may be found in [1341–1393]. References on superdirectivity are [1394–1414].

B.1 Definition

The PSWF functions are defined with respect to two intervals: a frequency interval, $[-\omega_0, \omega_0]$ (rad/sec), over which they are bandlimited, and a time interval, $[-t_0, t_0]$ (sec), over which they are concentrated (but not limited to).

For notational convenience let us define the following three function spaces: (a) the space $\mathcal{L}^2_\infty$ of functions $f(t)$ that are square-integrable over the real line, $-\infty < t < \infty$, (b) the space $\mathcal{L}^2_{t_0}$ of functions $f(t)$ that are square-integrable over the finite interval $[-t_0, t_0]$, and (c) the subspace $\mathcal{B}_{\omega_0}$ of $\mathcal{L}^2_\infty$ of bandlimited functions $f(t)$ whose Fourier transform $\hat{f}(\omega)$ vanishes outside the interval $[-\omega_0, \omega_0]$, that is, $\hat{f}(\omega) = 0$ for $|\omega| > \omega_0$, so that they are representable in the form,

$$f(t) = \int_{-\omega_0}^{\omega_0} \hat{f}(\omega)\, e^{j\omega t}\, \frac{d\omega}{2\pi}\,, \quad \hat{f}(\omega) = \int_{-\infty}^{\infty} f(t)\, e^{-j\omega t}\, dt \tag{B.1}$$

The PSWF functions, denoted here by $\psi_n(t)$ with $n = 0, 1, 2, \dots$, belong to the subspace $\mathcal{B}_{\omega_0}$ and are defined by the following equivalent expansions in terms of Legendre polynomials or spherical Bessel functions:

$$\boxed{\psi_n(t) = \sqrt{\frac{\lambda_n}{t_0}} \sum_k \beta_{nk} \sqrt{k + \tfrac{1}{2}}\, P_k\Big(\frac{t}{t_0}\Big) = \sqrt{\frac{c}{2\pi t_0}} \sum_k \beta_{nk} \sqrt{k + \tfrac{1}{2}}\; 2\, i^{k-n} j_k(\omega_0 t)} \tag{B.2}$$

for $n = 0, 1, 2, \dots$, where β_{nk} are the expansion coefficients, c is the *time-bandwidth* product, $c = t_0\omega_0$, and λ_n are the positive eigenvalues (listed in decreasing order) and $\psi_n(t)$ the corresponding eigenfunctions of the following linear integral operator with the sinc-kernel:

$$\boxed{\int_{-t_0}^{t_0} \frac{\sin\big(\omega_0(t - t')\big)}{\pi(t - t')}\, \psi_n(t')\, dt' = \lambda_n\, \psi_n(t)} \quad n = 0, 1, 2, \dots, \quad \text{for all } t \tag{B.3}$$

The Legendre polynomial expansion in (B.2) is numerically accurate for $|t| \le t_0$, whereas the spherical Bessel function expansion is valid for all t, and we use that in our MATLAB implementation. The unnormalized Legendre polynomials $P_k(x)$ are defined as follows [40],

$$P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n} \big[(x^2 - 1)^n\big]\,, \quad n = 0, 1, 2, \dots \tag{B.4}$$

The first few of them are listed below,

$$\begin{aligned}
P_0(x) &= 1 \\
P_1(x) &= x \\
P_2(x) &= \frac{3}{2}\left[x^2 - \frac{1}{3}\right] \\
P_3(x) &= \frac{5}{2}\left[x^3 - \frac{3}{5}x\right] \\
P_4(x) &= \frac{35}{8}\left[x^4 - \frac{6}{7}x^2 + \frac{3}{35}\right]
\end{aligned} \tag{B.5}$$

They are normalized such that $P_n(1) = 1$ and satisfy the orthogonality property:

$$\int_{-1}^{1} P_n(x) P_m(x)\, dx = \frac{2}{2n+1}\, \delta_{nm} \tag{B.6}$$

and the three-term recurrence relation:

$$x P_n(x) = \left(\frac{n}{2n+1}\right) P_{n-1}(x) + \left(\frac{n+1}{2n+1}\right) P_{n+1}(x) \tag{B.7}$$

The normalized Legendre polynomials are $\sqrt{k + \frac{1}{2}}\, P_k(x)$. The spherical Bessel functions $j_k(x)$ are defined in terms of the ordinary Bessel functions of half-integer order:

$$j_k(x) = \sqrt{\frac{\pi}{2x}}\, J_{k+\frac{1}{2}}(x) \tag{B.8}$$

The eigenvalues λ_n are distinct and lie in the interval, $0 < \lambda_n < 1$. Typically, they have values near unity, $\lambda_n \lessapprox 1$, for $n = 0, 1, 2, \dots$, up to about the so-called Shannon number, $N_c = 2c/\pi$, and after that they drop rapidly towards zero. The Shannon number represents roughly the number of *degrees of freedom* for characterizing a signal of total frequency bandwidth $\Omega = 2\omega_0$ and total time duration $T = 2t_0$. If F is the total bandwidth in Hz, that is, $F = \Omega/2\pi$, then, $N_c = FT$,

$$N_c = \frac{2c}{\pi} = \frac{2\omega_0 t_0}{\pi} = \frac{2\omega_0\, 2t_0}{2\pi} = \frac{\Omega T}{2\pi} = FT \tag{B.9}$$

Since, $j_k(\omega_0 t) = j_k(ct/t_0)$, we note that up to a scale factor, $\psi_n(t)$ is a function of c and the scaled variable $\eta = t/t_0$. Indeed, we have, $\psi_n(t) = t_0^{-1/2}\, \phi_n(c, t/t_0)$, where,

$$\boxed{\phi_n(c, \eta) = \sqrt{\lambda_n} \sum_k \beta_{nk} \sqrt{k + \tfrac{1}{2}}\, P_k(\eta) = \sqrt{\frac{c}{2\pi}} \sum_k \beta_{nk} \sqrt{k + \tfrac{1}{2}}\; 2 i^{k-n} j_k(c\eta)} \tag{B.10}$$

We will also use the following notation to indicate explicitly the dependence on the parameters t_0, ω_0 and $c = t_0\omega_0$,

$$\psi_n(t_0, \omega_0, t) = \frac{1}{\sqrt{t_0}}\, \phi_n\left(c, \frac{t}{t_0}\right) \tag{B.11}$$

The k-summation in (B.2) goes over $0 \le k < \infty$. However, k takes only even values, $k = 0, 2, 4, \dots$, when n is even or zero, and only odd values, $k = 1, 3, 5, \dots$, when n is odd. This also implies that $\psi_n(t)$ is an even function of t, if n is even, and odd in t, if n is odd, so that, $\psi_n(-t) = (-1)^n \psi_n(t)$. The expansion coefficients β_{nk} are real-valued and because n, k have the same parity, i.e., $n - k$ is even, it follows that the number i^{k-n} will be real, $i^{k-n} = (-1)^{(k-n)/2}$. Therefore, all $\psi_n(t)$ are real-valued.

The k-summation can be extended to all $k \ge 0$ by redefining the expansion coefficients β_{nk} for all k by appropriately interlacing zeros as follows:

$$\beta_{nk} = \begin{array}{ll} [\,\beta_{n0},\; 0,\; \beta_{n2},\; 0,\; \beta_{n4},\; 0,\; \cdots\,] & (n \text{ even}) \\ [\;0,\; \beta_{n1},\; 0,\; \beta_{n3},\; 0,\; \beta_{n5},\; \cdots\,] & (n \text{ odd}) \end{array} \tag{B.12}$$

The expansion coefficients are chosen to satisfy the orthogonality property,

$$\boxed{\sum_{k=0}^{\infty} \beta_{nk}\beta_{mk} = \delta_{nm}} \qquad n, m = 0, 1, 2, \dots \tag{B.13}$$

This particular normalization is computationally convenient and enables the orthogonality properties of the $\psi_n(t)$ functions, that is, Eqs. (B.28) and (B.30). The coefficients β_{nk} may be constructed as the orthonormal eigenvectors of a real symmetric tridiagonal matrix as we discuss below. We note also that for large t, the PSWF functions behave like sinc-functions [1290,1305],

$$\psi_n(t) \approx \sqrt{\frac{2c}{\pi\lambda_n}}\, \psi_n(t_0)\, \frac{\sin(\omega_0 t - \frac{1}{2}\pi n)}{\omega_0 t}\,, \qquad \text{for large } |t| \tag{B.14}$$

This follows from the asymptotic expansion of the spherical Bessel functions,

$$j_k(x) \approx \frac{\sin(x - \frac{1}{2}\pi k)}{x}\,, \qquad \text{for large } |x|$$

B.2 Fourier Transform

The bandlimited Fourier transform of $\psi_n(t)$ can be constructed as follows. First, we note that $P_k(x)$ and $j_k(x)$ satisfy the following Fourier transform relationships [40]:

$$\begin{aligned} &\int_{-1}^{1} e^{j\omega t}\, \pi\, i^{-k} P_k(\omega)\, \frac{d\omega}{2\pi} = j_k(t)\,, \quad \text{for all real } t \\ &\int_{-\infty}^{\infty} e^{-j\omega t} j_k(t)\, dt = \pi\, i^{-k} P_k(\omega) \cdot \chi_1(\omega) = \begin{cases} \pi\, i^{-k} P_k(\omega)\,, & |\omega| < 1 \\ 0\,, & |\omega| > 1 \end{cases} \end{aligned} \tag{B.15}$$

where $\chi_1(\omega)$ is the *indicator function* for the interval $[-1, 1]$, defined in terms of the unit-step function $u(x)$ as follows for a more general interval $[-\omega_0, \omega_0]$,[†]

$$\chi_{\omega_0}(\omega) = u(\omega_0 - |\omega|) = \begin{cases} 1\,, & |\omega| < \omega_0 \\ 0\,, & |\omega| > \omega_0 \end{cases} \tag{B.16}$$

[†] $\chi_{\omega_0}(\omega)$ may be defined to have the value $\frac{1}{2}$ at $\omega = \pm\omega_0$ corresponding to the unit-step value $u(0) = \frac{1}{2}$.

It follows that the Fourier transform $\hat{j}_k(\omega)$ of $j_k(\omega_0 t)$ is bandlimited over $[-\omega_0, \omega_0]$,

$$\begin{aligned} \hat{j}_k(\omega) &= \int_{-\infty}^{\infty} e^{-j\omega t} j_k(\omega_0 t)\, dt = \frac{\pi}{\omega_0\, i^k} P_k\left(\frac{\omega}{\omega_0}\right) \cdot \chi_{\omega_0}(\omega) \\ j_k(\omega_0 t) &= \int_{-\omega_0}^{\omega_0} e^{j\omega t}\, \hat{j}_k(\omega)\, \frac{d\omega}{2\pi} = \int_{-\omega_0}^{\omega_0} e^{j\omega t}\, \frac{\pi}{\omega_0\, i^k} P_k\left(\frac{\omega}{\omega_0}\right) \frac{d\omega}{2\pi} \end{aligned} \tag{B.17}$$

In fact, the functions $j_k(\omega_0 t)$, like the $\psi_n(t)$, form a *complete and orthogonal basis* of the subspace $\mathcal{B}_{\omega_0}$, see [1325,1326]. Their mutual orthogonality follows from Parseval's identity and the orthogonality property (B.6) of the Legendre polynomials:

$$\int_{-\infty}^{\infty} j_k(\omega_0 t)\, j_n(\omega_0 t)\, dt = \int_{-\omega_0}^{\omega_0} \hat{j}_k^*(\omega) \hat{j}_n(\omega)\, \frac{d\omega}{2\pi} = \frac{\pi}{\omega_0}\, \frac{\delta_{kn}}{2k+1} \tag{B.18}$$

The bandlimited Fourier transform $\hat{\psi}_n(\omega)$ of $\psi_n(t)$ can now be obtained by Fourier-transforming the spherical Bessel function expansion in (B.2), then using (B.17), and comparing the result with the Legendre expansion of (B.2), that is,

$$\begin{aligned} \hat{\psi}_n(\omega) &= \sqrt{\frac{c}{2\pi t_0}} \sum_k \beta_{nk} \sqrt{k + \tfrac{1}{2}}\; 2\, i^{k-n}\, \hat{j}_k(\omega) \\ &= \sqrt{\frac{c}{2\pi t_0}} \sum_k \beta_{nk} \sqrt{k + \tfrac{1}{2}}\; 2\, i^{k-n}\, \frac{\pi}{\omega_0\, i^k} P_k\left(\frac{\omega}{\omega_0}\right) \cdot \chi_{\omega_0}(\omega) \\ &= \frac{2\pi}{\omega_0}\, \frac{1}{\mu_n}\, \underbrace{\sqrt{\frac{\lambda_n}{t_0}} \sum_k \beta_{nk} \sqrt{k + \tfrac{1}{2}}\, P_k\left(\frac{\omega}{\omega_0}\right)}_{\psi_n(\omega t_0/\omega_0)} \cdot \chi_{\omega_0}(\omega) = \frac{2\pi}{\omega_0}\, \frac{1}{\mu_n}\, \psi_n\left(\frac{\omega t_0}{\omega_0}\right) \cdot \chi_{\omega_0}(\omega) \end{aligned}$$

where μ_n was defined in terms of λ_n as follows:

$$\mu_n = i^n |\mu_n|\,, \quad |\mu_n| = \sqrt{\frac{2\pi\lambda_n}{c}} \quad \Rightarrow \quad \boxed{\lambda_n = \frac{c}{2\pi} |\mu_n|^2} \tag{B.19}$$

Thus, we find that $\hat{\psi}_n(\omega)$ is a scaled version of $\psi_n(t)$ itself,

$$\boxed{\hat{\psi}_n(\omega) = \int_{-\infty}^{\infty} e^{-j\omega t}\, \psi_n(t)\, dt = \frac{2\pi}{\omega_0}\, \frac{1}{\mu_n}\, \psi_n\left(\frac{\omega t_0}{\omega_0}\right) \cdot \chi_{\omega_0}(\omega)} \tag{B.20}$$

The inverse Fourier transform brings out more clearly the meaning of μ_n:

$$\psi_n(t) = \int_{-\omega_0}^{\omega_0} e^{j\omega t}\, \hat{\psi}_n(\omega)\, \frac{d\omega}{2\pi} \quad \Rightarrow \quad \boxed{\int_{-\omega_0}^{\omega_0} e^{j\omega t}\, \psi_n\left(\frac{\omega t_0}{\omega_0}\right) \frac{d\omega}{\omega_0} = \mu_n\, \psi_n(t)} \tag{B.21}$$

for all t. By changing variables to $\omega \to t\omega_0/t_0$ and $t \to \omega t_0/\omega_0$, we also have,

$$\boxed{\int_{-t_0}^{t_0} e^{j\omega t}\, \psi_n(t)\, \frac{dt}{t_0} = \mu_n\, \psi_n\left(\frac{\omega t_0}{\omega_0}\right)} \quad \text{for all } \omega \tag{B.22}$$

If (B.21) is written in terms of the scaled function $\phi_n(\eta)$ of (B.10), then, μ_n is the eigenvalue and $\phi_n(\eta)$ the eigenfunction of the following integral operator with exponential kernel:

$$\boxed{\int_{-1}^{1} e^{jc\eta\xi}\,\phi_n(\xi)\,d\xi = \mu_n\,\phi_n(\eta)} \qquad n = 0,1,2,\dots, \quad \text{and all } \eta \tag{B.23}$$

Similarly, (B.3) reads as follows with respect to $\phi_n(\eta)$,

$$\boxed{\int_{-1}^{1} \frac{\sin\big(c(\eta-\xi)\big)}{\pi(\eta-\xi)}\,\phi_n(\xi)\,d\xi = \lambda_n\,\phi_n(\eta)} \quad \text{for all } \eta \tag{B.24}$$

Eqs. (B.3) and (B.19) can be derived from Eqs. (B.21) and (B.22). Indeed, multiplying both sides of (B.21) by μ_n^* and taking the complex conjugate of (B.22), we have,

$$|\mu_n|^2\,\psi_n(t) = \int_{-\omega_0}^{\omega_0} \mu_n^*\,\psi_n\Big(\frac{\omega t_0}{\omega_0}\Big)\,e^{j\omega t}\,\frac{d\omega}{\omega_0} = \int_{-\omega_0}^{\omega_0}\Bigg[\int_{-t_0}^{t_0} e^{-j\omega t'}\,\psi_n(t')\,\frac{dt'}{t_0}\Bigg]e^{j\omega t}\,\frac{d\omega}{\omega_0}$$

$$= \frac{2\pi}{t_0\omega_0}\int_{-t_0}^{t_0}\Bigg[\int_{-\omega_0}^{\omega_0} e^{j\omega(t-t')}\,\frac{d\omega}{2\pi}\Bigg]\psi_n(t')\,dt' = \frac{2\pi}{c}\int_{-t_0}^{t_0}\frac{\sin\big(\omega_0(t-t')\big)}{\pi(t-t')}\,\psi_n(t')\,dt'$$

where we used the sinc-function transform,

$$\frac{\sin(\omega_0 t)}{\pi t} = \int_{-\infty}^{\infty}\chi_{\omega_0}(\omega)\,e^{j\omega t}\,\frac{d\omega}{2\pi} = \int_{-\omega_0}^{\omega_0} e^{j\omega t}\,\frac{d\omega}{2\pi} \tag{B.25}$$

Eq. (B.3) follows now by multiplying both sides by $c/2\pi$ and using the definition (B.19). Any function $f(t)$ in $\mathcal{B}_{\omega_0}$ with a bandlimited Fourier transform $\hat{f}(\omega)$ over $[-\omega_0,\omega_0]$ satisfies a similar sinc-kernel integral equation, but over the infinite time interval, $-\infty < t < \infty$. Indeed, using the convolution theorem of Fourier transforms and (B.25), we have,

$$f(t) = \int_{-\omega_0}^{\omega_0}\hat{f}(\omega)e^{j\omega t}\,\frac{d\omega}{2\pi} = \int_{-\infty}^{\infty}\chi_{\omega_0}(\omega)\hat{f}(\omega)e^{j\omega t}\,\frac{d\omega}{2\pi} = \int_{-\infty}^{\infty}\frac{\sin\big(\omega_0(t-t')\big)}{\pi(t-t')}\,f(t')dt'$$

that is, for $f(t)\in\mathcal{B}_{\omega_0}$ and for all t,

$$f(t) = \int_{-\infty}^{\infty}\frac{\sin\big(\omega_0(t-t')\big)}{\pi(t-t')}\,f(t')dt' \tag{B.26}$$

Thus, because they lie in $\mathcal{B}_{\omega_0}$, all $\psi_n(t)$ satisfy a similar condition,

$$\int_{-\infty}^{\infty}\frac{\sin\big(\omega_0(t-t')\big)}{\pi(t-t')}\,\psi_n(t')dt' = \psi_n(t)\,, \quad \text{for all } t \tag{B.27}$$

B.3 Orthogonality and Completeness Properties

The PSWF functions $\psi_n(t)$ satisfy *dual* orthogonality and completeness properties, that is, the $\psi_n(t)$ functions form an orthogonal and complete basis for both $\mathcal{L}^2_{t_0}$ *and* $\mathcal{B}_{\omega_0}$. With respect to the space $\mathcal{L}^2_{t_0}$, we have,

$$\boxed{\int_{-t_0}^{t_0}\psi_n(t)\psi_m(t)\,dt = \lambda_n\,\delta_{nm}} \quad \text{(orthogonality)} \tag{B.28}$$

$$\boxed{\sum_{n=0}^{\infty}\frac{1}{\lambda_n}\,\psi_n(t)\psi_n(t') = \delta(t-t')} \quad \text{(completeness)} \tag{B.29}$$

for $t,t'\in[-t_0,t_0]$. And, with respect to the subspace $\mathcal{B}_{\omega_0}$, we have for the infinite time interval, $-\infty < t < \infty$,

$$\boxed{\int_{-\infty}^{\infty}\psi_n(t)\psi_m(t)\,dt = \delta_{nm}} \tag{B.30}$$

$$\boxed{\sum_{n=0}^{\infty}\psi_n(t)\psi_n(t') = \frac{\sin\big(\omega_0(t-t')\big)}{\pi(t-t')}} \quad \text{for all } t,t' \tag{B.31}$$

Eq. (B.28) can be derived from the Legendre expansion in (B.2) as a consequence of the normalization condition (B.13) and the Legendre polynomial orthogonality (B.6). Similarly, (B.30) can be derived from the spherical Bessel function expansion and (B.18).

The sinc-kernel in (B.31) plays the role of the identity operator for functions in $\mathcal{B}_{\omega_0}$, as implied by (B.26). Taking the limit $t'\to t$ on both sides of (B.31), we obtain the following relationship, valid for all t,

$$\sum_{n=0}^{\infty}\psi_n^2(t) = \frac{\omega_0}{\pi} \tag{B.32}$$

Integrating this over $[-t_0,t_0]$ and using (B.28) for $n=m$, we obtain the sums,

$$\sum_{n=0}^{\infty}\lambda_n = \frac{2c}{\pi} = N_c \quad\Rightarrow\quad \sum_{n=0}^{\infty}|\mu_n|^2 = 4 \tag{B.33}$$

Another identity can be derived from (B.31) by taking Fourier transforms of both sides with respect to the variable t' and using the delay theorem of Fourier transforms on the right-hand-side, resulting in,

$$e^{-j\omega t}\cdot\chi_{\omega_0}(\omega) = \sum_{n=0}^{\infty}\psi_n(t)\hat{\psi}_n(\omega) = \frac{2\pi}{\omega_0}\chi_{\omega_0}(\omega)\sum_{n=0}^{\infty}\frac{1}{\mu_n}\,\psi_n(t)\psi_n\Big(\frac{\omega t_0}{\omega_0}\Big)\,, \quad \text{or,}$$

$$\frac{2\pi}{\omega_0}\sum_{n=0}^{\infty}\frac{1}{\mu_n}\,\psi_n(t)\psi_n\Big(\frac{\omega t_0}{\omega_0}\Big) = e^{-j\omega t}\,, \quad \text{for all } t \text{ and } |\omega|<\omega_0 \tag{B.34}$$

and in particular, setting $\omega = 0$, we have for all t,

$$\frac{2\pi}{\omega_0}\sum_{n=0}^{\infty}\frac{1}{\mu_n}\,\psi_n(0)\psi_n(t) = 1 \tag{B.35}$$

Eqs. (B.31)–(B.35) are demonstrated in Examples B.1–B.2. The completeness property (B.29) can also be derived by complex-conjugating both sides of (B.34) and taking Fourier transforms with respect to the variable t, denoting the corresponding frequency by ω', with $|\omega'|<\omega_0$,

$$\frac{2\pi}{\omega_0}\sum_{n=0}^{\infty}\frac{1}{\mu_n^*}\,\hat{\psi}_n(\omega')\psi_n\Big(\frac{\omega t_0}{\omega_0}\Big) = 2\pi\delta(\omega-\omega')\,, \quad \text{or,}$$

$$\frac{(2\pi)^2}{\omega_0^2}\sum_{n=0}^{\infty}\frac{1}{|\mu_n|^2}\,\psi_n\left(\frac{\omega t_0}{\omega_0}\right)\psi_n\left(\frac{\omega' t_0}{\omega_0}\right) = 2\pi\delta(\omega-\omega')$$

or, using the relationship, $|\mu_n|^2 = 2\pi\lambda_n/c$,

$$\sum_{n=0}^{\infty}\frac{1}{\lambda_n}\,\psi_n\left(\frac{\omega t_0}{\omega_0}\right)\psi_n\left(\frac{\omega' t_0}{\omega_0}\right) = \delta\left(\frac{\omega t_0}{\omega_0}-\frac{\omega' t_0}{\omega_0}\right) \tag{B.36}$$

for $\omega,\omega'\in[-\omega_0,\omega_0]$. This becomes equivalent to Eq. (B.29) after replacing, $\omega t_0/\omega_0 \to t$ and $\omega' t_0/\omega_0 \to t'$, with $t,t'\in[-t_0,t_0]$. In a similar fashion, Eq. (B.28) can be derived by applying Parseval's identity to (B.30).

B.4 Signal Restoration

Another application is in *signal restoration*, such as image restoration through a finite-aperture diffraction-limited optical system, involving the inversion of the sinc-kernel over the finite interval $[-t_0,t_0]$, that is, finding a kernel function, say, $K(t,t')$, that performs the inverse operation,

$$g(t)=\int_{-t_0}^{t_0}\frac{\sin\big(\omega_0(t-t')\big)}{\pi(t-t')}\,f(t')\,dt' \quad\Rightarrow\quad f(t)=\int_{-t_0}^{t_0}K(t,t')\,g(t')\,dt' \tag{B.37}$$

for $|t|\le t_0$. It is easily verified that $K(t,t')$ is given formally by [1365],[†]

$$K(t,t')=\sum_n\frac{1}{\lambda_n^2}\,\psi_n(t)\psi_n(t')\,,\qquad \text{for } t,t'\in[-t_0,t_0] \tag{B.38}$$

The operation of $K(t,t')$ on the sinc-kernel generates the identity kernel, that is,

$$\int_{-t_0}^{t_0}K(t,t'')\,\frac{\sin\big(\omega_0(t''-t')\big)}{\pi(t''-t')}\,dt'' = \delta(t-t')\,,\quad \text{for } t,t'\in[-t_0,t_0] \tag{B.39}$$

Indeed, using (B.28), (B.29), (B.31), and (B.38), we have,

$$\begin{aligned}
&\int_{-t_0}^{t_0}K(t,t'')\,\frac{\sin\big(\omega_0(t''-t')\big)}{\pi(t''-t')}\,dt'' = \\
&\quad= \int_{-t_0}^{t_0}\left[\sum_n\frac{1}{\lambda_n^2}\,\psi_n(t)\psi_n(t'')\right]\left[\sum_m\psi_m(t'')\psi_m(t')\right]dt'' \\
&\quad= \sum_n\sum_m\frac{1}{\lambda_n^2}\,\psi_n(t)\underbrace{\left[\int_{-t_0}^{t_0}\psi_n(t'')\psi_m(t'')\,dt''\right]}_{\lambda_n\delta_{nm}}\psi_m(t') \\
&\quad= \sum_n\frac{1}{\lambda_n}\,\psi_n(t)\psi_n(t') = \delta(t-t')\,,\quad \text{for } t,t'\in[-t_0,t_0]
\end{aligned}$$

Such inversion method will most surely fail in practice if there is even a tiny amount of noise in the observed data. Suppose, for example, that we add a small noise component $v(t)$ to Eq. (B.37),

[†] A more precise meaning may be given to (B.38) by the regularized versions discussed below.

$$g(t)=\int_{-t_0}^{t_0}\frac{\sin\big(\omega_0(t-t')\big)}{\pi(t-t')}\,f(t')\,dt' + v(t) \tag{B.40}$$

then, the restored signal will be,

$$f_{\text{rest}}(t)=\int_{-t_0}^{t_0}K(t,t')\,g(t')\,dt' = f(t)+\int_{-t_0}^{t_0}K(t,t')\,v(t')\,dt' \equiv f(t)+u(t)$$

where the noise $v(t)$ and its inverse-filtered version $u(t)$ can be expanded in the following forms over $[-t_0,t_0]$,

$$v(t)=\sum_n v_n\,\psi_n(t) \quad\Rightarrow\quad u(t)=\int_{-t_0}^{t_0}K(t,t')\,v(t')\,dt' = \sum_n\frac{v_n}{\lambda_n}\,\psi_n(t)$$

Thus, even if all the v_n were tiny, the ratios v_n/λ_n can become very large, because the λ_n tend to zero for large n, and the filtered noise $u(t)$ will be amplified and may completely mask the desired signal component $f(t)$.

A way out of this, which provides only an approximation to the inverse kernel, is to limit the summation over n to those eigenvalues λ_n that are large and near unity, that is, for n less than about the Shannon number N_c. For example, using $M+1$ terms, with M near N_c, we have the following approximation to $K(t,t')$ and to the delta function in (B.39), for t,t' in $[-t_0,t_0]$,

$$\hat K(t,t')=\sum_{n=0}^{M}\frac{1}{\lambda_n^2}\,\psi_n(t)\psi_n(t') \tag{B.41}$$

$$\hat\delta(t,t')=\int_{-t_0}^{t_0}\hat K(t,t'')\,\frac{\sin\big(\omega_0(t''-t')\big)}{\pi(t''-t')}\,dt'' = \sum_{n=0}^{M}\frac{1}{\lambda_n}\,\psi_n(t)\psi_n(t') \tag{B.42}$$

where $\hat\delta(t,t')$ approximates $\delta(t-t')$, for $t,t'\in[-t_0,t_0]$. With $f(t)$ expanded as,

$$f(t)=\sum_{n=0}^{\infty}f_n\psi_n(t)\,,\quad |t|\le t_0 \tag{B.43}$$

it follows that the restoration approximations correspond to the finite sums,

$$f_{\text{rest}}(t)=\hat f(t)+\hat u(t)=\sum_{n=0}^{M}f_n\psi_n(t)+\sum_{n=0}^{M}\frac{v_n}{\lambda_n}\,\psi_n(t) \tag{B.44}$$

Keeping only a finite number of terms is a form of *regularization* of the inverse filtering operation. Other regularization schemes are possible, for example, the Tikhonov-type regularization approximating $\delta(t-t')$ over $[-t_0,t_0]$,

$$\hat\delta(t,t')=\sum_{n=0}^{\infty}\frac{\lambda_n}{\lambda_n^2+\varepsilon^2}\,\psi_n(t)\psi_n(t') \tag{B.45}$$

where ε small regularization parameter, $\varepsilon^2 \ll 1$. The summation in (B.45) effectively cuts off as soon as $\lambda_n\approx\varepsilon$, while it behaves like (B.42) for $1\ge\lambda_n\gg\varepsilon$.

Setting $t' = 0$ in Eqs. (B.29), (B.42), and (B.45), we obtain the following bandlimited approximations to a delta function $\delta(t)$, over the finite interval $[-t_0, t_0]$,

$$\delta(t) = \sum_{n=0}^{\infty} \frac{1}{\lambda_n}\,\psi_n(0)\,\psi_n(t)\,, \qquad -t_0 \le t \le t_0 \tag{B.46}$$

$$\hat{\delta}(t) = \sum_{n=0}^{M} \frac{1}{\lambda_n}\,\psi_n(0)\,\psi_n(t)\,, \qquad -t_0 \le t \le t_0 \tag{B.47}$$

$$\hat{\delta}(t) = \sum_{n=0}^{\infty} \frac{\lambda_n}{\lambda_n^2 + \varepsilon^2}\,\psi_n(0)\,\psi_n(t)\,, \qquad -t_0 \le t \le t_0 \tag{B.48}$$

Given that $\delta(t)$ has a flat spectrum extending over $-\infty < \omega < \infty$, referring to a “bandlimited” delta function is an oxymoron. However, the above expressions approximate $\delta(t)$ only over the finite interval, $-t_0 \le t \le t_0$, while they have bandlimited spectrum over $-\omega_0 < \omega < \omega_0$. Outside the $[-t_0, t_0]$ interval, they result in extremely large values. In fact, these expressions provide the ultimate example of *superoscillations* [1327–1340], which are bandlimited signals that, over a finite time interval, appear to oscillate faster than their highest frequency, but typically, exhibit much higher values outside that interval. Indeed, the above $\delta(t)$ is the fastest varying signal in $[-t_0, t_0]$ while at the same time it remains bandlimited.

Example B.3 explores the approximation of Eq. (B.47). We will used it in [46] in the design of superresolving pupil masks for achieving highly focused fields, and in the design of supergain aperture antennas for achieving very high directivity.

B.5 Representation and Extrapolation of Bandlimited Functions

If a function $f(t)$ is *bandlimited* over $[-\omega_0, \omega_0]$, then the completeness of the $\psi_n(t)$ basis implies that $f(t)$ can be expanded in the following form, for all t,

$$f(t) = \sum_n c_n \psi_n(t)\,, \quad \text{with} \quad c_n = \int_{-\infty}^{\infty} \psi_n(t) f(t)\,dt \tag{B.49}$$

Using the expansion (B.49) and the orthogonality property (B.28), we may obtain an alternative way of calculating the coefficients c_n that involves knowledge of $f(t)$ only over the finite interval $[-t_0, t_0]$,

$$c_n = \frac{1}{\lambda_n} \int_{-t_0}^{t_0} \psi_n(t) f(t)\,dt \tag{B.50}$$

Thus, once c_n are determined, $f(t)$ can be extrapolated outside $[-t_0, t_0]$ using (B.49). The validity of this procedure rests on the assumption that $f(t)$ is a bandlimited function in $\mathcal{B}_{\omega_0}$ and a segment of it is known over the interval $[-t_0, t_0]$.

If $f(t)$ is an arbitrary function in $\mathcal{L}^2_{t_0}$, but is not necessarily a segment of a bandlimited function in $\mathcal{B}_{\omega_0}$, then the orthogonality and completeness properties (B.28) and (B.29) still allow an expansion in the following form, but valid only for $|t| \le t_0$,

$$f(t) = \sum_n c_n \psi_n(t)\,, \quad \text{with} \quad c_n = \frac{1}{\lambda_n} \int_{-t_0}^{t_0} \psi_n(t) f(t)\,dt \tag{B.51}$$

The attempt to extrapolate $f(t)$ beyond $[-t_0, t_0]$ using (B.51) could diverge and lead to extremely large values for $f(t)$ outside the $[-t_0, t_0]$ interval. This, again, is an example of superoscillations.

The representation (B.49) of a bandlimited function $f(t)$ in the $\psi_n(t)$ basis is convenient, but is not the only one. One could also expand $f(t)$ in the spherical Bessel function basis $j_n(\omega_0 t)$, or in the familiar sinc-function basis that appears in the sampling theorem. We summarize these expansions and their Fourier transforms below.

The expansion coefficients c_n can be expressed either in the time domain or in the frequency domain. Let $f(t)$ be in $\mathcal{B}_{\omega_0}$ with a bandlimited Fourier transform $\hat{f}(\omega)$, then, in the $\psi_n(t)$ basis,

$$\boxed{\begin{aligned} f(t) &= \sum_n c_n \psi_n(t) \\ \hat{f}(\omega) &= \sum_n c_n \frac{2\pi}{\omega_0}\frac{1}{\mu_n}\,\psi_n\Big(\frac{\omega t_0}{\omega_0}\Big)\cdot \chi_{\omega_0}(\omega) \end{aligned}} \qquad \text{(PSWF basis)} \tag{B.52}$$

$$c_n = \int_{-\infty}^{\infty} \psi_n(t) f(t)\,dt = \frac{1}{\lambda_n}\int_{-t_0}^{t_0} \psi_n(t) f(t)\,dt = \frac{1}{\mu_n^*}\int_{-\omega_0}^{\omega_0} \psi_n\Big(\frac{\omega t_0}{\omega_0}\Big)\hat{f}(\omega)\,\frac{d\omega}{\omega_0} \tag{B.53}$$

Similarly, in the $j_n(\omega_0 t)$ basis, with Legendre polynomial Fourier transform,

$$\boxed{\begin{aligned} f(t) &= \sum_n c_n j_n(\omega_0 t) \\ \hat{f}(\omega) &= \sum_n c_n \frac{\pi}{\omega_0\, i^n}\,P_n\Big(\frac{\omega}{\omega_0}\Big)\cdot \chi_{\omega_0}(\omega) \end{aligned}} \qquad \text{(spherical Bessel basis)} \tag{B.54}$$

$$c_n = \frac{(2n+1)\,\omega_0}{\pi}\int_{-\infty}^{\infty} j_n(\omega_0 t)\,f(t)\,dt = \frac{i^n(2n+1)}{2\pi}\int_{-\omega_0}^{\omega_0} \hat{f}(\omega) P_n\Big(\frac{\omega}{\omega_0}\Big)\,d\omega \tag{B.55}$$

and in the sinc-function basis, with $T_s = \pi/\omega_0$ denoting the sampling time interval,

$$\boxed{\begin{aligned} f(t) &= \sum_n c_n \frac{\sin\big(\omega_0(t-nT_s)\big)}{\pi(t-nT_s)} = T_s \sum_n f(nT_s)\,\frac{\sin\big(\omega_0(t-nT_s)\big)}{\pi(t-nT_s)} \\ \hat{f}(\omega) &= \sum_n c_n\, e^{-j\omega nT_s}\cdot\chi_{\omega_0}(\omega) = T_s \sum_n f(nT_s)\, e^{-j\omega nT_s}\cdot\chi_{\omega_0}(\omega) \end{aligned}} \qquad \text{(sinc)} \tag{B.56}$$

$$c_n = T_s f(nT_s) = T_s \int_{-\omega_0}^{\omega_0} \hat{f}(\omega)\, e^{j\omega nT_s}\,\frac{d\omega}{2\pi} \tag{B.57}$$

The expansion of $f(t)$ in Eq. (B.56) is the same as Eq. (1.6.4) of the Shannon sampling theorem discussed in Chap. 1, whereas the expansion of $\hat{f}(\omega)$ is equivalent to Eq. (1.5.19). Note that the sinc-basis satisfies the orthogonality condition:

$$\int_{-\infty}^{\infty} \frac{\sin\big(\omega_0(t-nT_s)\big)}{\pi(t-nT_s)}\,\frac{\sin\big(\omega_0(t-mT_s)\big)}{\pi(t-mT_s)}\,dt = \frac{1}{T_s}\,\delta_{nm} \tag{B.58}$$

The result, $c_n = T_s f(nT_s)$, follows from (B.58) by applying (B.26) at $t = nT_s$, indeed,

$$\begin{aligned} f(nT_s) &= \int_{-\infty}^{\infty} \frac{\sin\big(\omega_0(t-nT_s)\big)}{\pi(t-nT_s)}\, f(t)\,dt \\ &= \int_{-\infty}^{\infty} \frac{\sin\big(\omega_0(t-nT_s)\big)}{\pi(t-nT_s)} \left[\sum_m c_m \frac{\sin\big(\omega_0(t-mT_s)\big)}{\pi(t-mT_s)}\right] dt \\ &= \sum_m c_m \int_{-\infty}^{\infty} \frac{\sin\big(\omega_0(t-nT_s)\big)}{\pi(t-nT_s)}\, \frac{\sin\big(\omega_0(t-mT_s)\big)}{\pi(t-mT_s)}\, dt \\ &= \sum_m c_m \frac{1}{T_s}\,\delta_{nm} = \frac{c_n}{T_s} \quad\Rightarrow\quad c_n = T_s f(nT_s) \end{aligned}$$

Up to the factor T_s, the c_n are the time samples of $f(t)$, and $\hat{f}(\omega)$ in (B.56) is recognized as the central Nyquist replica of the DTFT of the discrete-time signal $f(nT_s)$. Moreover, Eq. (B.57) is the inverse DTFT of $\hat{f}(\omega)$ integrated over the Nyquist interval.

The Shannon number can be understood with the help of (B.56). In this context, we are assuming that the bandlimited function $f(t)$ is sampled at its Nyquist rate, which is $2\omega_0$ in units of radians/sec, or, $f_s = 2\omega_0/2\pi = \omega_0/\pi$ in samples/sec, so that the sampling time interval is $T_s = 1/f_s = \pi/\omega_0$ in seconds. Thus, the sampling rate f_s plays the role of F in (B.9).

Because $f(t)$ is frequency-limited, it cannot be time-limited. However, if we assume that the *most significant* time samples of $f(t)$ are contained within a total time interval T, say, $0 \le nT_s \le T$, then the maximum time index, and hence the number of significant time samples will be, $n_{\max}T_s = T$, or, $n_{\max} = T/T_s = f_sT$, that is, the Shannon number.

B.6 Energy Concentration Properties

Consider a bandlimited signal $f(t)$ in $\mathcal{B}_{\omega_0}$ with an expansion of the form (B.52),

$$f(t) = \sum_n c_n\psi_n(t)\,, \quad \text{with}\quad c_n = \int_{-\infty}^{\infty}\psi_n(t)f(t)\,dt = \frac{1}{\lambda_n}\int_{-t_0}^{t_0}\psi_n(t)f(t)\,dt \tag{B.59}$$

Then, its energy contained in $[-t_0, t_0]$ and in the infinite interval are given by the norms,

$$\begin{aligned} \int_{-t_0}^{t_0} f^2(t)\,dt &= \sum_{n=0}^{\infty}\lambda_n c_n^2\,, \\ \int_{-\infty}^{\infty} f^2(t)\,dt &= \sum_{n=0}^{\infty} c_n^2 \end{aligned} \tag{B.60}$$

The proportion of the total energy contained in the interval $[-t_0, t_0]$ is the ratio,

$$\mathcal{R}(f) = \frac{\displaystyle\int_{-t_0}^{t_0} f^2(t)\,dt}{\displaystyle\int_{-\infty}^{\infty} f^2(t)\,dt} = \frac{\displaystyle\sum_{n=0}^{\infty}\lambda_n c_n^2}{\displaystyle\sum_{n=0}^{\infty} c_n^2} \tag{B.61}$$

In the context of designing apodization functions [46], it is known as the “encircled energy” ratio, while in the context of superdirective antennas, its inverse, $1/\mathcal{R}(f)$, is known as Taylor’s supergain or superdirectivity ratio [233].

The following extremal properties of the PSWF functions can be derived from (B.61). Because the eigenvalues λ_n are in decreasing order, $1 > \lambda_0 > \lambda_1 > \lambda_2 > \cdots > 0$, it is easily seen that $\mathcal{R}(f)$ is maximized when $c_0 \neq 0$, and $c_n = 0$, for $n \ge 1$, that is, when $f(t) = \psi_0(t)$, and the maximized value is $\mathcal{R}(\psi_0) = \lambda_0$. This follows from the inequality,

$$\lambda_0 - \mathcal{R}(f) = \frac{\displaystyle\sum_{n=1}^{\infty}(\lambda_0 - \lambda_n)c_n^2}{\displaystyle\sum_{n=0}^{\infty} c_n^2} \ge 0$$

with equality realized when $c_n = 0$ for all $n \ge 1$. Similarly, among all the functions in $\mathcal{B}_{\omega_0}$ that are orthogonal to $\psi_0(t)$, the energy ratio $\mathcal{R}(f)$ is maximized when $f(t) = \psi_1(t)$, with maximum value $\mathcal{R}(\psi_1) = \lambda_1$. In this case the c_0 term is absent because of the orthogonality to ψ_0, and $\mathcal{R}(f)$ becomes,

$$\mathcal{R}(f) = \frac{\displaystyle\sum_{n=1}^{\infty}\lambda_n c_n^2}{\displaystyle\sum_{n=1}^{\infty} c_n^2} \quad\Rightarrow\quad \lambda_1 - \mathcal{R}(f) = \frac{\displaystyle\sum_{n=2}^{\infty}(\lambda_1 - \lambda_n)c_n^2}{\displaystyle\sum_{n=1}^{\infty} c_n^2} \ge 0$$

More generally, among all the functions in $\mathcal{B}_{\omega_0}$ that are simultaneously orthogonal to $\psi_0(t), \psi_1(t), \dots, \psi_{n-1}(t)$, the function $f(t) = \psi_n(t)$ maximizes $\mathcal{R}(f)$ with maximum value $\mathcal{R}(\psi_n) = \lambda_n$.

If $f(t)$ is not necessarily bandlimited in $\mathcal{B}_{\omega_0}$ then, as we mentioned earlier, its values outside the interval $[-t_0, t_0]$ could become very large causing the energy concentration ratio $\mathcal{R}(f)$ to become very small.

Thus, a related optimization problem is to find that function $g(t)$ in $\mathcal{B}_{\omega_0}$ that has a prescribed value of the energy ratio, say, $\mathcal{R}(g) = \mathcal{R}_0 < 1$, and provides the best approximation to a given $f(t)$ within the $[-t_0, t_0]$ interval. Using a mean-square criterion, the problem can be stated as finding $g(t) \in \mathcal{B}_{\omega_0}$ that minimizes the following performance index subject to the energy ratio constraint,

$$\mathcal{J} = \int_{-t_0}^{t_0}\big(f(t) - g(t)\big)^2 = \min\,, \quad \text{subject to}\quad \mathcal{R}_0 = \frac{\displaystyle\int_{-t_0}^{t_0} g^2(t)\,dt}{\displaystyle\int_{-\infty}^{\infty} g^2(t)\,dt} \tag{B.62}$$

or equivalently, we may introduce a Lagrange multiplier μ for the constraint,

$$\mathcal{J} = \int_{-t_0}^{t_0}\big(f(t) - g(t)\big)^2 + \mu\left(\mathcal{R}_0\int_{-\infty}^{\infty} g^2(t) - \int_{-t_0}^{t_0} g^2(t)\,dt\right) = \min \tag{B.63}$$

Expand $f(t)$ and $g(t)$ in the $\psi_n(t)$ basis,

$$\begin{aligned} f(t) &= \sum_n f_n\,\psi_n(t)\,, \quad \text{for } |t|\le t_0 \\ g(t) &= \sum_n g_n\,\psi_n(t)\,, \quad \text{for all } t \end{aligned} \tag{B.64}$$

where f_n may be assumed to be known since $f(t)$ is given. Then, (B.63) becomes,

$$\mathcal{J} = \sum_n \lambda_n(g_n - f_n)^2 + \mu\left(\mathcal{R}_0\sum_n g_n^2 - \sum_n \lambda_n g_n^2\right) = \min \tag{B.65}$$

The minimization condition with respect to g_n gives,

$$\frac{\partial\mathcal{J}}{\partial g_n} = 0 \quad\Rightarrow\quad \lambda_n(g_n - f_n) + \mu(\mathcal{R}_0 - \lambda_n)g_n = 0\,, \quad \text{or,}$$

$$g_n = \frac{\lambda_n f_n}{\lambda_n + \mu(\mathcal{R}_0 - \lambda_n)} \tag{B.66}$$

with the Lagrange multiplier μ determined by solving the constraint equation,

$$\mathcal{R}_0\sum_n g_n^2 - \sum_n \lambda_n g_n^2 = \sum_n(\mathcal{R}_0 - \lambda_n)g_n^2 = 0\,, \quad \text{or,}$$

$$\sum_n(\mathcal{R}_0 - \lambda_n)\left[\frac{\lambda_n f_n}{\lambda_n + \mu(\mathcal{R}_0 - \lambda_n)}\right]^2 = 0 \tag{B.67}$$

B.7 Computation

The numerical computation of $\psi_n(t)$ is based on keeping only a finite number of terms in the summation of Eq. (B.2), e.g., $0 \le k \le K-1$,

$$\boxed{\psi_n(t) = \sqrt{\frac{\lambda_n}{t_0}}\sum_{k=0}^{K-1}\beta_{nk}\sqrt{k+\tfrac{1}{2}}\,P_k\!\left(\frac{t}{t_0}\right) = \sqrt{\frac{c}{2\pi t_0}}\sum_{k=0}^{K-1}\beta_{nk}\sqrt{k+\tfrac{1}{2}}\;2i^{k-n}j_k(\omega_0 t)} \tag{B.68}$$

where K is chosen to be sufficiently large. In practice, the choice, $K = 2N+30$, is adequate [1317], where $N = M+1$ is the number of PSWF functions to be calculated, that is, $\psi_n(t)$, $n = 0, 1, 2, \dots, M$.

The computation of β_{nk} is based on the observation [1290] that the PSWF functions $\psi_n(t)$ are also the eigenfunctions of the following Sturm-Liouville differential operator eigenproblem, that arises in solving the Helmholtz equation in spheroidal coordinates,

$$\left[(t^2 - t_0^2)\frac{d^2}{dt^2} + 2t\frac{d}{dt} + \omega_0^2 t^2\right]\psi_n(t) = \chi_n\,\psi_n(t)\,, \quad n = 0, 1, 2, \dots \tag{B.69}$$

The eigenvalues χ_n are real and positive and when listed in increasing order, they match the eigenvalues λ_n of Eq. (B.3) listed in decreasing order, that is, we have the correspondence,

$$\begin{aligned} &0 < \chi_0 < \chi_1 < \chi_2 < \dots\,, \\ &1 > \lambda_0 > \lambda_1 > \lambda_2 > \dots\,, \end{aligned}$$

There is extensive literature on the computation of the PSWF solutions of (B.69), including its generalization to higher order PSWFs, with implementations in FORTRAN, MATLAB, and Mathematica [1301–1324].

Our MATLAB implementation is based on Rhodes [1305] who gives normalization-independent expressions, Hodge [1307] who was the first to suggest using a tridiagonal eigenvalue problem for calculating the coefficients β_{nk}, Kozin et al. [1309] who applied Hodge's method specifically to order-zero PSWFs, and Xiao et al [1311] and Boyd [1317] for some more recent implementations. Accurate computation of the eigenvalues λ_n and eigenfunctions $\psi_n(t)$ is very demanding for large values of c and large values of n because the λ_n quickly get smaller than the machine epsilon, even using double precision as in MATLAB.

If one inserts the Legendre polynomial expansion of Eq. (B.2) into (B.69), one finds that the coefficients β_{nk} satisfy the following recursion formula, for $k \ge 0$,

$$A_{k,k-2}\,\beta_{n,k-2} + A_{k,k}\,\beta_{nk} + A_{k,k+2}\,\beta_{n,k+2} = \chi_n\,\beta_{nk} \tag{B.70}$$

where the first term is present only for $k \ge 2$, and,

$$\begin{aligned} A_{k,k} &= k(k+1) + c^2\cdot\frac{2k(k+1)-1}{(2k+3)(2k-1)}\,, && k \ge 0 \\ A_{k,k+2} &= c^2\cdot\frac{(k+2)(k+1)}{(2k+3)\sqrt{(2k+1)(2k+5)}}\,, && k \ge 0 \\ A_{k,k-2} = A_{k-2,k} &= c^2\cdot\frac{k(k-1)}{(2k-1)\sqrt{(2k-3)(2k+1)}}\,, && k \ge 2 \end{aligned} \tag{B.71}$$

If a finite number of coefficients is kept, such as, $K = 2N+30$, then, we may define a $K\times K$ symmetric tridiagonal matrix A whose main diagonal is $A_{k,k}$, for $k = 0, 1, \dots, K-1$, and whose upper and lower second diagonals are given by $A_{k,k+2}$, for $k = 0, 1, \dots, K-3$. For example, if $K = 8$, the matrix will have the following structure,

$$A = \begin{bmatrix} A_{00} & 0 & A_{02} & 0 & 0 & 0 & 0 & 0 \\ 0 & A_{11} & 0 & A_{13} & 0 & 0 & 0 & 0 \\ A_{02} & 0 & A_{22} & 0 & A_{24} & 0 & 0 & 0 \\ 0 & A_{13} & 0 & A_{33} & 0 & A_{35} & 0 & 0 \\ 0 & 0 & A_{24} & 0 & A_{44} & 0 & A_{46} & 0 \\ 0 & 0 & 0 & A_{35} & 0 & A_{55} & 0 & A_{57} \\ 0 & 0 & 0 & 0 & A_{46} & 0 & A_{66} & 0 \\ 0 & 0 & 0 & 0 & 0 & A_{57} & 0 & A_{77} \end{bmatrix}$$

The difference equation (B.69) is equivalent then to the following eigenvalue equation, where the coefficients $\beta_{n,k}$ are arranged into the corresponding eigenvectors,

$$\boxed{A\boldsymbol{\beta}_n = \chi_n\boldsymbol{\beta}_n}\,, \qquad \boldsymbol{\beta}_n = \begin{bmatrix} \beta_{n,0} \\ \beta_{n,1} \\ \vdots \\ \beta_{n,K-1} \end{bmatrix} \tag{B.72}$$

Because A is real and symmetric, its eigenvectors may be chosen to be real-valued and orthonormal, that is, satisfying Eq. (B.13). And because of the particular structure of A, having only a nonzero second upper/lower subdiagonal, every other element of the eigenvectors may be set to zero, that is, $\beta_{n,k} = 0$, if $n-k$ is odd. For example, if $K = 8$, the eigenvectors, will have the following structure,

$$\underset{n=0,2,4,6}{\boldsymbol{\beta}_{n,\text{even}}} = \begin{bmatrix} \beta_{n0} \\ 0 \\ \beta_{n2} \\ 0 \\ \beta_{n4} \\ 0 \\ \beta_{n6} \\ 0 \end{bmatrix}, \quad \underset{n=1,3,5,7}{\boldsymbol{\beta}_{n,\text{odd}}} = \begin{bmatrix} 0 \\ \beta_{n1} \\ 0 \\ \beta_{n3} \\ 0 \\ \beta_{n5} \\ 0 \\ \beta_{n7} \end{bmatrix}$$

MATLAB can very efficiently and accurately solve the above tridiagonal eigenvalue problem using the built-in function **eig**, and obtain all K eigenvectors $\boldsymbol{\beta}_n$ and eigenvalues χ_n (even for very large values of c, K, e.g., $c = 100$, $K = 1000$).

Once the $K{\times}K$ eigenvalue problem (B.72) is solved, we may retain the first $M + 1$ eigenvectors $\boldsymbol{\beta}_n$, for $n = 0, 1, \dots, M$, that are needed in the computation of $\psi_n(t)$. One still has the task of determining the corresponding eigenvalues λ_n of the eigenproblem (B.3). Since λ_n appears in (B.2), we may solve for it by evaluating both sides of (B.68) at a particular value of t. It's is simpler to solve for μ_n and then calculate λ_n from (B.19). Multiplying both sides of (B.68) by $i^n\sqrt{2\pi t_0/c}$, we obtain the relationship:

$$i^n\sqrt{\frac{2\pi t_0}{c}}\,\psi_n(t) = \mu_n\sum_{k=0}^{K-1}\beta_{nk}\sqrt{k+\tfrac{1}{2}}\,P_k\Big(\frac{t}{t_0}\Big) = \sum_{k=0}^{K-1}\beta_{nk}\sqrt{k+\tfrac{1}{2}}\;2i^k\,j_k(\omega_0 t) \tag{B.73}$$

One obvious choice is to set $t = t_0$, and since $P_n(1) = 1$, we may solve for μ_n,

$$\boxed{\mu_n = \frac{\displaystyle\sum_{k=0}^{K-1}\beta_{nk}\sqrt{k+\tfrac{1}{2}}\;2i^k\,j_k(c)}{\displaystyle\sum_{k=0}^{K-1}\beta_{nk}\sqrt{k+\tfrac{1}{2}}}}\qquad n = 0, 1, \dots, M \tag{B.74}$$

Another choice is to set $t = 0$, resulting in

$$\mu_n\sum_{k=0}^{K-1}\beta_{nk}\sqrt{k+\tfrac{1}{2}}\,P_k(0) = \sum_{k=0}^{K-1}\beta_{nk}\sqrt{k+\tfrac{1}{2}}\;2i^k\,j_k(0) \tag{B.75}$$

However, this works only for n even, because if n and k are odd, both $P_k(0)$ and $j_k(0)$ are zero. In this case, one may work with the time derivative of both sides of (B.68),

$$\mu_n\sum_{k=0}^{K-1}\beta_{nk}\sqrt{k+\tfrac{1}{2}}\,\frac{1}{t_0}\,P'_k\Big(\frac{t}{t_0}\Big) = \sum_{k=0}^{K-1}\beta_{nk}\sqrt{k+\tfrac{1}{2}}\;2i^k\,\omega_0\,j'_k(\omega_0 t)\,,\quad\text{or,}$$

$$\mu_n\sum_{k=0}^{K-1}\beta_{nk}\sqrt{k+\tfrac{1}{2}}\,P'_k\Big(\frac{t}{t_0}\Big) = \sum_{k=0}^{K-1}\beta_{nk}\sqrt{k+\tfrac{1}{2}}\;2i^k\,c\,j'_k(\omega_0 t)$$

Evaluating this at $t = 0$ will work for n odd, but not for n even,

$$\mu_n\sum_{k=0}^{K-1}\beta_{nk}\sqrt{k+\tfrac{1}{2}}\,P'_k(0) = \sum_{k=0}^{K-1}\beta_{nk}\sqrt{k+\tfrac{1}{2}}\;2i^k\,c\,j'_k(0) \tag{B.76}$$

We may add (B.75) and (B.76) together to get a combined formula that works for all n, even or odd, and solve for μ_n,

$$\boxed{\mu_n = \frac{\displaystyle\sum_{k=0}^{K-1}\beta_{nk}\sqrt{k+\tfrac{1}{2}}\;2i^k\,\big[j_k(0) + c\,j'_k(0)\big]}{\displaystyle\sum_{k=0}^{K-1}\beta_{nk}\sqrt{k+\tfrac{1}{2}}\,\big[P_k(0) + P'_k(0)\big]}}\qquad n = 0, 1, \dots, M \tag{B.77}$$

The spherical Bessel functions $j_k(x)$ for $k \ge 0$ have the following limiting behavior,

$$j_k(x) \underset{x\to 0}{\longrightarrow} \frac{x^k}{1\cdot 3\cdot 5\cdots(2k+1)} \quad\Rightarrow\quad j_k(0) = \delta_k = \begin{cases} 1, & k = 0 \\ 0, & k \neq 0 \end{cases} \tag{B.78}$$

The derivative, $j'_k(x)$, can be computed from the recursion:

$$j'_k(x) = \frac{k\,j_{k-1}(x) - (k+1)\,j_{k+1}(x)}{2k+1} \tag{B.79}$$

and using the result (B.78),

$$j'_k(0) = \frac{k\,\delta_{k-1} - (k+1)\,\delta_{k+1}}{2k+1} = \frac{1}{3}\,\delta_{k-1} \tag{B.80}$$

Using (B.78) and (B.80), the numerator in (B.77) simplifies to the $k = 0$ and $k = 1$ terms,

$$\boxed{\mu_n = \frac{\sqrt{2}\,\beta_{n0} + \sqrt{\dfrac{2}{3}}\,ic\,\beta_{n1}}{\displaystyle\sum_{k=0}^{K-1}\beta_{nk}\sqrt{k+\tfrac{1}{2}}\,\big[P_k(0) + P'_k(0)\big]}}\qquad n = 0, 1, \dots, M \tag{B.81}$$

The quantities, $P_k(0), P'_k(0)$, can be calculated in terms of gamma-functions,

$$P_k(0) = \frac{\sqrt{\pi}}{\Gamma\Big(\dfrac{k}{2}+1\Big)\,\Gamma\Big(\dfrac{1}{2}-\dfrac{k}{2}\Big)}\,,\quad P'_k(0) = kP_{k-1}(0) = \frac{-2\sqrt{\pi}}{\Gamma\Big(\dfrac{k}{2}+\dfrac{1}{2}\Big)\,\Gamma\Big(-\dfrac{k}{2}\Big)} \tag{B.82}$$

These are valid for all $k \ge 0$ and give $P_k(0) = 0$, for odd k, and $P'_k(0) = 0$, for even k. Moreover, they can be implemented as vectorized functions in MATLAB. For a given n, either the left or the right terms will be non-zero in the numerator and denominator of (B.81). We have found that the choice (B.81) produces more accurate results than (B.74) for moderate values of c, M, up to about, $c = 50$, $M = 50$.

The following MATLAB function, `pswf.m`, implements the above computational steps. It also computes the derivatives $\psi'_n(t)$. It has usage:

```
[Psi,La,dPsi,Chi,B] = pswf(t0,w0,M,t);      % prolate spheroidal wave functions
[Psi,La,dPsi,Chi,B] = pswf(t0,w0,M,t,K);
```

```
t0 = time limit, [-t0,t0], (sec)
w0 = freq limit, [-w0,w0], (rad/sec)
M = max order computed, evaluating psi_n(t), n = 0,1,...,M
t = length-L vector of time instants (sec), e.g., t = [t1,t2,...,tL]

Psi  = (M+1)xL matrix of prolate function values
La   = (M+1)x1 vector of eigenvalues
dPsi = (M+1)xL matrix of derivatives of prolate functions
Chi  = (M+1)x1 vector of eigenvalues of spheroidal differential operator
B    = (M+1)xK matrix of expansion coefficients
K    = number of expansion terms (default, K=2*N+30, N=M+1)
```

For a given maximum order M and time vector $t = [t_1, t_2, \dots, t_L]$, the output matrix `Psi` contains the computed values $\psi_n(t_i)$, arranged into an $(M+1)\times L$ matrix,

$$\Psi = \begin{bmatrix} \psi_0(t_1) & \psi_0(t_2) & \psi_0(t_3) & \cdots & \psi_0(t_L) \\ \psi_1(t_1) & \psi_1(t_2) & \psi_1(t_3) & \cdots & \psi_1(t_L) \\ \psi_2(t_1) & \psi_2(t_2) & \psi_2(t_3) & \cdots & \psi_2(t_L) \\ \vdots & \vdots & \vdots & \dots & \vdots \\ \psi_M(t_1) & \psi_M(t_2) & \psi_M(t_3) & \cdots & \psi_M(t_L) \end{bmatrix}$$

The output `dPsi` contains the derivatives $\psi_n'(t_i)$ arranged in a similar fashion. The outputs `La,Chi` are column vectors containing the eigenvalues λ_n, χ_n, $n = 0, 1, \dots, M$. The $(M+1)\times K$ matrix `B` contains the expansion coefficients, $B_{nk} = \beta_{nk}$, in its rows, with the appropriate interlacing of zeros for even or odd n, that is,

$$B = \begin{bmatrix} \beta_{0,0} & 0 & \beta_{0,2} & 0 & \beta_{0,4} & 0 & \cdots & \beta_{0,K-1} \\ 0 & \beta_{1,1} & 0 & \beta_{1,3} & 0 & \beta_{1,5} & \cdots & \beta_{1,K-1} \\ \beta_{2,0} & 0 & \beta_{2,2} & 0 & \beta_{2,4} & 0 & \cdots & \beta_{2,K-1} \\ 0 & \beta_{3,1} & 0 & \beta_{3,3} & 0 & \beta_{3,5} & \cdots & \beta_{3,K-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \dots & \vdots \end{bmatrix}$$

The eigenvector matrix is actually the transposed, $B^T = [\boldsymbol{\beta}_0, \boldsymbol{\beta}_1, \dots, \boldsymbol{\beta}_M]$, which satisfies the eigenvalue equation, $AB^T = B^T\mathcal{X}$, where $\mathcal{X} = \mathrm{diag}\{\chi_0, \chi_1, \dots, \chi_M\}$, and the orthonormality property,

$$BB^T = I_{M+1} \quad \Rightarrow \quad \sum_{k=0}^{K-1} \beta_{nk}\beta_{mk} = \delta_{nm}\,, \quad n, m = 0, 1, \dots, M$$

The required spherical Bessel function computations are implemented by the function, `spherj.m`, which also computes the derivatives using the recursion (B.79). It arranges its output similarly to `pswf`, and has usage:

```
[J,dJ] = spherj(k,x);      % spherical Bessel functions and derivatives
```

The derivatives, $\psi_n'(t)$, are computed using the expansion,

$$\psi_n'(t) = \sqrt{\frac{c}{2\pi t_0}} \sum_{k=0}^{K-1} \beta_{nk} \sqrt{k+\tfrac{1}{2}}\; 2i^{k-n}\, \omega_0 j_k'(\omega_0 t) \tag{B.83}$$

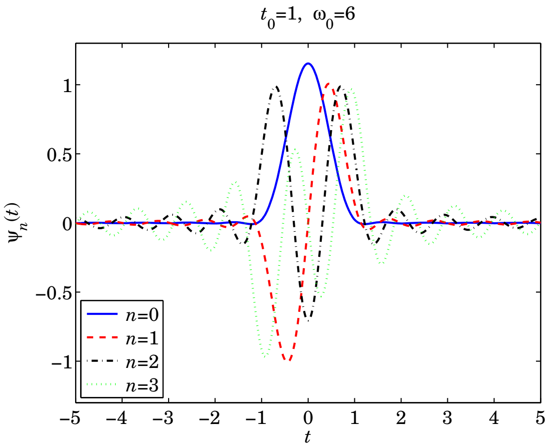
The functions **pswf** and **spherj** are included in the EWA-toolbox of this book, and work well for parameter values up to about $c = 50$ and $M = 50$. A related function that evaluates Legendre polynomials and can be used in the Legendre expansion of Eq. (B.2) is the function **legpol** with usage:

```
P = legpol(N,x);      % evaluates Legendre polynomials of orders n=0:N
```

Example B.1: The following MATLAB code illustrates the usage of the **pswf** function and reproduces the data tables and plots from Rhodes [1305] for the case $c = 6$.

```
t0=1; w0=6; c=t0*w0; M=13;

t = linspace(-5,5,501);

[Psi,La,dPsi,Chi] = pswf(t0,w0,M,t);     % calculate the first 14 PSWFs

figure; plot(t,Psi(1:4,:));              % plot first four PSWFs
```

The following graph plots the first four PSWFs, $\psi_n(t)$, $n = 0, 1, 2, 3$.



The following table contains the values of the quantities χ_n, μ_n, λ_n for $n = 0, 1, \dots, 13$, as well the values of the sum, $\psi_n(0)+\psi_n'(0)$, which represents either $\psi_n(0)$ if n is even, or $\psi_n'(0)$ id n is odd. They agree with those in [1305].

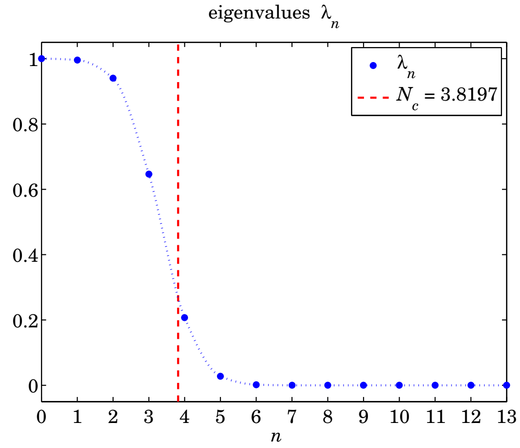
```
  n        chi_n              |mu_n|          psi_n(0)+psi_n'(0)      lambda_n
```

```
---------------------------------------------------------------------------
  0   5.208269160e+00   1.023276504e+00    1.631136822e-03   9.999018826e-01
  1   1.600044275e+01   1.021309607e+00   -5.777995494e-02   9.960616433e-01
  2   2.535647864e+01   9.922435547e-01    3.539393687e-02   9.401733902e-01
  3   3.320419949e+01   8.229938914e-01    4.428586710e-01   6.467919492e-01
  4   4.072019427e+01   4.659781027e-01   -3.045590356e-01   2.073492169e-01
  5   4.977371213e+01   1.693510361e-01   -7.517787765e-01   2.738716624e-02
  6   6.118075690e+01   4.524678973e-02    7.141707743e-01   1.955000734e-03
  7   7.485286653e+01   9.966212672e-03    5.750889706e-01   9.484876556e-05
  8   9.065115937e+01   1.897100693e-03   -1.117460330e+00   3.436783286e-06
  9   1.085154453e+02   3.192404741e-04    3.163806610e-02   9.732115989e-08
 10   1.284188799e+02   4.820488557e-05    1.196006265e+00   2.218980545e-09
 11   1.503474435e+02   6.605196410e-06   -9.118593645e-01   4.166226284e-11
 12   1.742930029e+02   8.286721621e-07   -5.384541132e-01   6.557478591e-13
 13   2.002505133e+02   9.588946460e-08    1.423257002e+00   8.780377122e-15
```

The table was generated by the MATLAB code,

```
n = (0:M)';
Mu = sqrt(2*pi/c*La);                   % calculate |mu_n| from la_n
P = Psi(:,1) + dPsi(:,1);               % interlace psi_n(0) and its derivative psi_n'(0)

fprintf('  n        chi_n             |mu_n|        psi_n(0)+psi_n''(0)    lambda_n\n')
fprintf('---------------------------------------------------------------------------\n')
fprintf('%3d    %1.9e    %1.9e    % 1.9e    %1.9e\n', [n,Chi,Mu,P,La]')
```

The Shannon number is $N_c = 2c/\pi = 3.8197$. We observe in the above table how quickly the eigenvalues λ_n decay towards zero for $n > N_c$. The following figure plots λ_n and the dividing line at N_c (the eigenvalues have been joined by a smooth curve to guide the eye.)

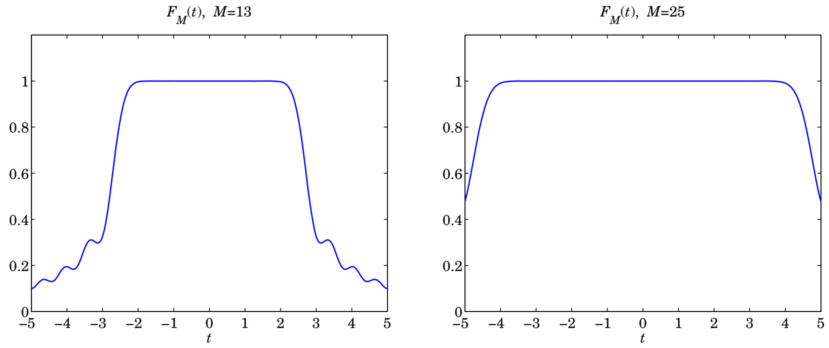


The following MATLAB code tests the relationships (B.32) and (B.33) for $M = 13$ and $M = 25$. The graphs plot the quantity,

$$F_M(t) = \frac{\pi}{\omega_0}\sum_{n=0}^{M}\psi_n^2(t) \tag{B.84}$$

which should tend to unity as M increases and more terms are included in the sum.

```
      F = sum(Psi.^2) * pi/w0;       % test: pi/w0 * \sum_n psi_n(t)^2 = 1
      figure; plot(t,F,'b-')         % gets better as M is increased

      Nc = 2*c/pi;                   % Nc = 3.8197, Shannon number
      Err = abs(sum(La)-Nc);         % Err = 1.3323e-15, for M=13
```



Example B.2: This example tests the relationships (B.31) and (B.35). Setting $t' = 0$ in (B.31), we obtain the limit,

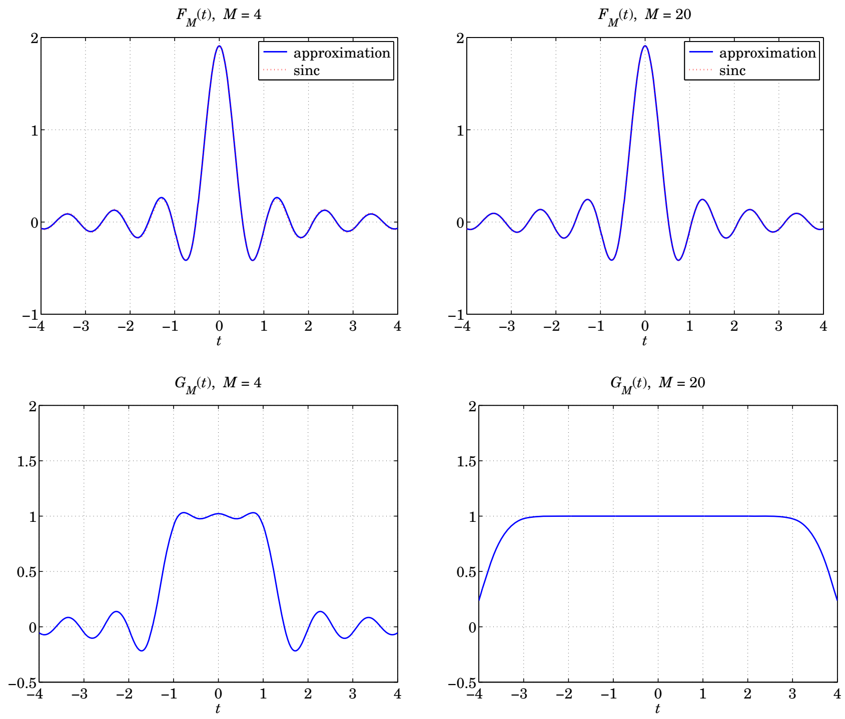
$$\lim_{M\to\infty} F_M(t) = \frac{\sin(\omega_0 t)}{\pi t}\,, \quad \text{where} \quad F_M(t) = \sum_{n=0}^{M}\psi_n(0)\psi_n(t)$$

Similarly we have the limit of (B.35),

$$\lim_{M\to\infty} G_M(t) = 1\,, \quad \text{where} \quad G_M(t) = \frac{2\pi}{\omega_0}\sum_{n=0}^{M}\frac{1}{\mu_n}\psi_n(0)\psi_n(t)$$

The following MATLAB code computes and plots $F_M(t), G_M(t)$ for $M = 4$ and $M = 20$ for the case $t_0 = 1$, $\omega_0 = 6$,

```
      t0 = 1; w0 = 6; c = t0*w0;
      t = linspace(-4,4,801);

      for M = [4,20]
         [Psi,La] = pswf(t0,w0,M,t);
         Psi0 = Psi(:,t==0);

         F = Psi0' * Psi;

         figure; plot(t,F,'b-', t,sinc(w0*t/pi)*w0/pi,'r:');

         n = (0:M)';
         Mu = i.^n .* sqrt(2*pi*La/c);
         G = (2*pi/w0) * Psi0' * diag(1./Mu) * Psi;

         figure; plot(t,G,'b-');
      end
```



The convergence of $F_M(t)$ is very quick and one can barely distinguish the approximation from the exact sinc function on the graphs (see the color graphs in the PDF book file). □

Example B.3: Here we explore the bandlimited approximation (B.47) of a delta function over a limited time interval. Let us denote the approximation by,

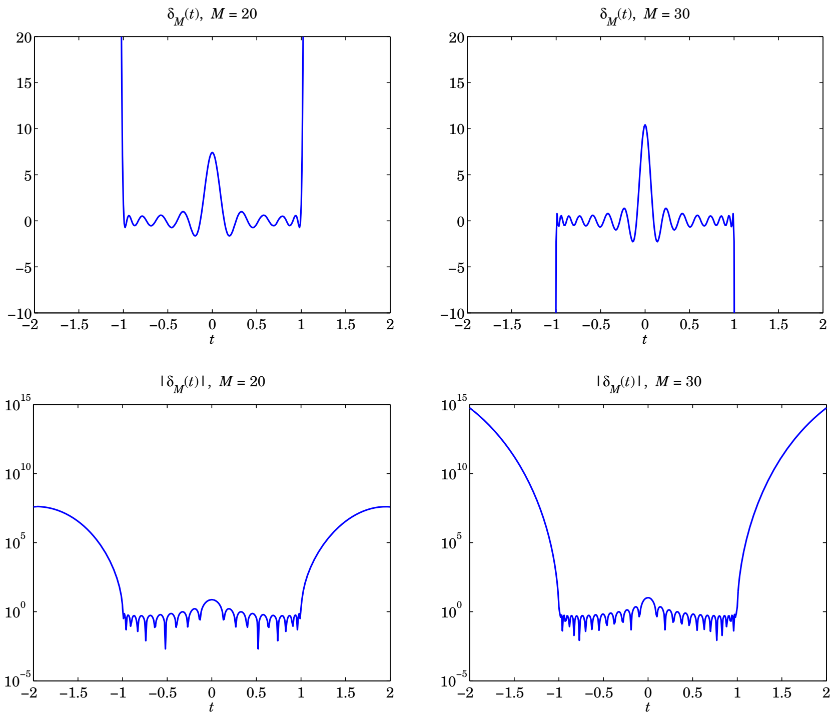
$$\delta_M(t) = \sum_{n=0}^{M}\frac{1}{\lambda_n}\psi_n(0)\psi_n(t) \tag{B.85}$$

The following MATLAB code segment evaluates and plots $\delta_M(t)$ for the following values of the parameters: $t_0 = 1$, $\omega_0 = 4\pi$, and the cases, $M = 20$ and $M = 30$.

```
t0 = 1; w0 = 4*pi;
t = linspace(-2,2,401);

for M = [20,30]
   [Psi,La] = pswf(t0,w0,M,t);
   Psi0 = Psi(:,t==0);

   F = Psi0' * diag(1./La) * Psi;            % the summation in (J.81)

   figure; plot(t,F, 'b-','linewidth',2);
   axis([-2,2,-10,20]); xlabel('\itt');
end
```

Because $\psi_n(t) = (-1)^n\psi_n(-t)$, it follows that $\psi_n(0) = 0$ for odd n and the summation in (B.85) can be restricted to even n's. However, in the MATLAB implementation above it is much simpler to keep all terms. The following graphs plot $\delta_M(t)$ versus t, over the interval $-2t_0 \le t \le 2t_0$.

To display the incredibly large values outside the interval, $[-t_0, t_0]$, we have also plotted the absolute values $|\delta_M(t)|$ using a semi-log scale. Within the $[-t_0, t_0]$ interval, the peak around $t = 0$ approximating $\delta(t)$ becomes narrower with increasing M.



We note that since the maximum frequency is $f_0 = \omega_0/(2\pi) = 2$ in cycles per unit time, the approximation $\delta_M(t)$ clearly oscillates several times faster than f_0 over the $[-t_0, t_0]$ interval, exhibiting a superoscillatory behavior [1327–1340].

The bandlimited Fourier transform of the approximation (B.85) is obtained from the Fourier transform of $\psi_n(t)$,

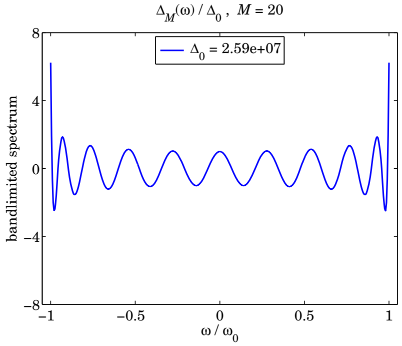
$$\Delta_M(\omega) = \int_{-\infty}^{\infty} e^{-j\omega t}\,\delta_M(t)\,dt = \sum_{n=0}^{M}\frac{1}{\lambda_n}\psi_n(0)\hat{\psi}_n(\omega)$$

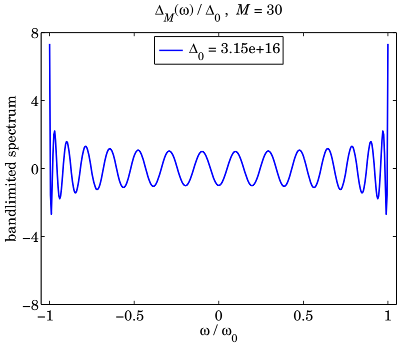
or, using (B.20),

$$
\begin{aligned}
\Delta_M(\omega) &= \sum_{n=0}^{M} \frac{2\pi}{\omega_0\,\lambda_n\,\mu_n}\,\psi_n(0)\,\psi_n\!\left(\frac{\omega t_0}{\omega_0}\right)\cdot\chi_{\omega_0}(\omega) \\
&= \sqrt{\frac{2\pi t_0}{\omega_0}}\,\sum_{n=0}^{M} \frac{1}{i^n\,\lambda_n^{3/2}}\,\psi_n(0)\,\psi_n\!\left(\frac{\omega t_0}{\omega_0}\right)\cdot\chi_{\omega_0}(\omega)
\end{aligned}
\tag{B.86}
$$

The following MATLAB code computes and plots $\Delta_M(\omega)$ over $[-\omega_0,\omega_0]$ for the same parameter values, $t_0 = 1$, $\omega_0 = 4\pi$, and $M = 20, 30$.

```
t0 = 1; w0 = 4*pi;
w = linspace(-1,1,401);      % normalized frequency, w = omega/omega_0

for M = [20,30]
   [Psi,La] = pswf(t0,w0,M,w*t0);
   Psi0 = Psi(:,w==0);
   n = (0:M)';

   D = sqrt(2*pi*t0/w0) * Psi0' * diag(1./i.^n./La.^(3/2)) * Psi;

   D0 = abs(D(w==0));

   figure; plot(w,D/D0, 'b-','linewidth',2);
   axis([-1.05, 1.05, -8, 8]);
end
```





For plotting purposes, the spectrum is normalized to its magnitude at $\omega = 0$, that is, the quantity $\Delta_M(\omega)/\Delta_0$ is plotted, where $\Delta_0 = |\Delta_M(0)|$. The computed values of Δ_0 and hence the values of $\Delta_M(\omega)$ in absolute scales are huge, indeed, we have, $\Delta_0 = 2.5897{\times}10^7$ and $\Delta_0 = 3.1503{\times}10^{16}$ for the two cases of $M = 20$ and $M = 30$, respectively.

We considered this type of example in [46] in our discussion of superresolving optical systems and the design of superdirective aperture antennas.

In the optics context, we must map the variables (t,ω) to the spatial variables (x,k_x), with the optical system introducing bandlimiting in the wavenumber domain k_x. The sharp focusing of the delta-function approximation $\delta_M(x)$ is accompanied by huge values outside a specified "field-of-view" interval $[-x_0,x_0]$ which plays the role of $[-t_0,t_0]$. Such huge values can be blocked with additional aperture stops provided the object fits within this field of view.

In the context of aperture-limited antennas, we must remap the variables (t,ω) onto the spatial variables (k_x,x), so that $\delta_M(t)$ maps into $\delta_M(k_x)$ and becomes the radiation pattern of the antenna as a function of the wavenumber k_x. The Fourier spectrum $\Delta_M(\omega)$ maps onto the aperture-limited field distribution $\Delta_M(x)$ as a function of distance x along the finite antenna, limited to an interval $-a \le x \le a$, where $2a$ is the length of the antenna.

Although in principle one can design an aperture antenna that has high directivity, approximating $\delta(k_x)$, the extremely large values of the aperture distribution $\Delta_M(x)$, make such designs effectively unrealizable in practice.

The role of the bandwidth interval $[-\omega_0,\omega_0]$ is played by the space interval $[-a,a]$, while the role of $[-t_0,t_0]$ is played by the visible region of the wavenumber interval, $[-k_0,k_0]$, where k_0 is the free-space wavenumber, $k_0 = 2\pi/\lambda$. Although the visible radiation pattern is limited to $|k_x| \le k_0$, and emulates a sharp radiation pattern $\delta_M(k_x)$, the rest of the pattern over the invisible region $|k_x| > k_0$ takes on very large values as we saw in this example. These are associated with very large reactive power stored in the vicinity of the antenna.

Further discussion of these issues is presented in [46]. We note also that the superresolution examples of Barnes [1365] can be reproduced by changing the parameters to $\omega_0 = 4$ and $M = [2,4,6,8]$ in the above MATLAB segments, while the supergain design example of Kritikos [1412] is reproduced by the choice, $t_0 = 1, \omega_0 = 6, M = 8$. □

Example B.4: Another superoscillation example from [1334] is constructed as a linear combination of sinusoids of frequencies, $f = 0, 1, 2, 3, 4, 5$, and amplitudes ranging hugely between large positive and large negative values,

$$
y(t) = a_0 + a_1\cos(2\pi t) + a_3\cos(4\pi t) + a_3\cos(6\pi t) + a_4\cos(8\pi t) + a_5\cos(10\pi t)
$$

where,

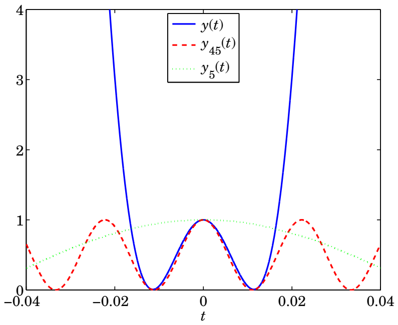
$$
\begin{aligned}
a_0 &= 1\\
a_1 &= +13295000\\
a_2 &= -30802818\\
a_3 &= +26581909\\
a_4 &= -10836909\\
a_5 &= +1762818
\end{aligned}
$$

The maximum frequency present in the signal is, $f_5 = 5$, and one would naively expect that $y(t)$ would not exhibit oscillations that are faster than f_5. However, as can be seen in the left graph below, plotted over the narrow time interval, $|t| \le 0.04$, the signal $y(t)$, plotted in blue, exhibits a fast oscillation with frequency approximately $f = 45$, as can be verified by superimposing the following signal, represented by the dashed red curve.

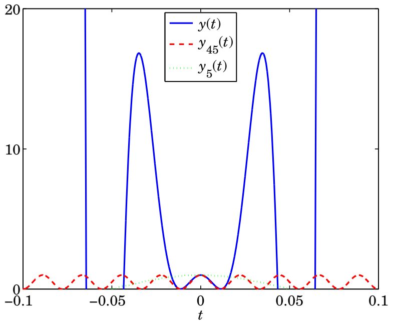
$$
y_{45}(t) = 0.5\cos(90\pi t) + 0.5
$$

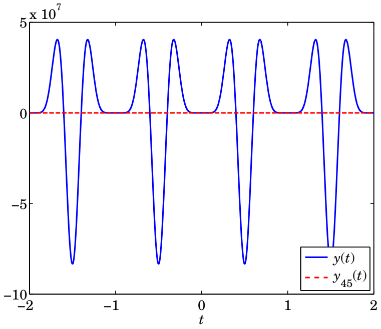
For reference, that fastest component present in $y(t)$,with frequency f_5, is plotted with the green dotted curve,

$$
y_5(t) = \cos(10\pi t)
$$



The left graph below shows the same signals plotted over the wider time range, $|t| \le 0.1$, while the right graph plots the signals of the range, $|t| \le 2$.





The extremely large values outside the narrow superoscillatory regions are typical of superoscillatory signals. More superoscillatory examples exhibiting this behavior are discussed in [46] in the context of designing superdirective apertures and antenna arrays. The MATLAB code for generating the above graphs was as follows,

```
a0 = 1;
a1 =  13295000;
a2 = -30802818;
a3 =  26581909;
a4 = -10836909;
a5 =  1762818;

t = linspace(-0.04,0.04,10001);     % time interval, |t| <= 0.04
% t = linspace(-0.1,0.1,10001);     % time interval, |t| <= 0.10
% t = linspace(-2,2,10001);         % time interval, |t| <= 2.00

y = a0 + a1*cos(2*pi*t) + a2*cos(2*pi*2*t) + a3*cos(2*pi*3*t)...
       + a4*cos(2*pi*4*t) + a5*cos(2*pi*5*t);

y5 = cos(2*pi*5*t);
y45 = 0.5 * cos(2*pi*45*t) + 0.5;

figure; plot(t,y,'b-', t,y45,'r--', t,y5,'g:', 'linewidth',2)
yaxis(0,4,0:4); xlabel('\itt');
```

C MATLAB Toolbox

To save space, all the MATLAB and C functions mentioned in this book, as well as the data files, have been moved into the following zip file,

```
ISP2e-toolbox.zip
```

located in,

```
www.ece.rutgers.edu/~orfanidi/intro2sp/2e
```

References

Texts

[1] B. Gold and C. M. Rader, *Digital Processing of Signals*, McGraw-Hill, New York, 1969.

[2] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Prentice Hall, Englewood Cliffs, NJ, 1989.

[3] A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*, Prentice Hall, Englewood Cliffs, NJ, 1975.

[4] L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*, Prentice Hall, Englewood Cliffs, NJ, 1975.

[5] S. K. Mitra and J. F. Kaiser, eds., *Handbook of Digital Signal Processing*, Wiley, New York, 1993.

[6] T. W. Parks and C. S. Burrus, *Digital Filter Design*, Wiley, New York, 1987.

[7] A. Antoniou, *Digital Filters: Analysis and Design*, 2nd ed., McGraw-Hill, New York, 1993.

[8] D. F. Elliott, ed., *Handbook of Digital Signal Processing*, Academic Press, New York, 1987.

[9] L. R. Rabiner and C. M. Rader, eds., *Digital Signal Processing*, IEEE Press, New York, 1972.

[10] *Selected Papers in Digital Signal Processing, II*, edited by the Digital Signal Processing Committee and IEEE ASSP, IEEE Press, New York, 1976.

[11] *Programs for Digital Signal Processing*, edited by the Digital Signal Processing Committee, IEEE ASSP Society, IEEE Press, New York, 1979.

[12] A. V. Oppenheim, ed., *Applications of Digital Signal Processing*, Prentice Hall, Englewood Cliffs, NJ, 1978.

[13] J. S. Lim and A. V. Oppenheim, eds., *Advanced Topics in Signal Processing*, Prentice Hall, Englewood Cliffs, NJ, 1988.

[14] R. A. Roberts and C. T. Mullis, *Digital Signal Processing*, Addison-Wesley, Reading, MA, 1987.

[15] P. A. Lynn and W. Fuerst, *Introductory Digital Signal Processing with Computer Applications*, Wiley, New York, 1989.

[16] J. G. Proakis and D. G. Manolakis, *Introduction to Digital Signal Processing*, 2nd ed., Macmillan, New York, 1988.

[17] E. C. Ifeachor and B. W. Jervis, *Digital Signal Processing: A Practical Approach*, Addison-Wesley, Reading, MA, 1993.

[18] R. A. Haddad and T. W. Parsons, *Digital Signal Processing: Theory, Applications, and Hardware*, Computer Science Press, W. H. Freeman, New York, 1991.

[19] L. B. Jackson, *Digital Filters and Signal Processing*, Kluwer Academic Publishers, , Norwell, MA 1989.

[20] A. Bateman and W. Yates, *Digital Signal Processing Design*, Computer Science Press, W. H. Freeman, New York, 1991.

[21] S. D. Stearns and D. R. Hush, *Digital Signal Analysis*, 2nd ed., Prentice Hall, Englewood Cliffs, NJ, 1990.

[22] S. D. Stearns and R. A. David, *Signal Processing Algorithms*, Prentice Hall, Englewood Cliffs, NJ, 1988.

[23] D. J. DeFatta, J. G. Lucas, and W. S. Hodgkiss, *Digital Signal Processing: A System Design Approach*, Wiley, New York, 1988.

[24] E. Robinson and S. Treitel, *Geophysical Signal Analysis*, Prentice Hall, Englewood Cliffs, NJ, 1980.

[25] S. M. Kay, *Modern Spectral Estimation: Theory and Application*, Prentice Hall, Englewood Cliffs, NJ, 1988.

[26] S. L. Marple, *Digital Spectral Analysis with Applications*, Prentice Hall, Englewood Cliffs, NJ, 1987.

[27] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*, Prentice Hall, Englewood Cliffs, NJ, 1985.

[28] D. Manolakis, V. K. Ingle, and S. M. Kogon, *Statistical and Adaptive Signal Processing*, Artech House, 2005.

[29] B. Porat, *A Course in Digital Signal Processing*, Wiley, New York, 1996.

[30] A. V. Oppenheim, A. S. Willsky, and I. T. Young, *Signals and Systems*, Prentice Hall, Englewood Cliffs, NJ, 1983.

[31] R. Lyons, *Understanding Digital Signal Processing*, Prentice Hall, Englewood Cliffs, NJ, 2004.

[32] S. K. Mitra, *Digital Signal Processing*, 4/e, McGraw-Hill, New York, 2010.

[33] J. M. Giron-Sierra, *Digital Signal Processing with Matlab Examples*, Vols. 1–3, Springer, Singapore, 2017.

[34] R. A. Losada, “Digital Filters with MATLAB,”, MATLAB Central File Exchange, 2022.
`www.mathworks.com/matlabcentral/fileexchange/19880-digital-filters-with-matlab`

[35] P. H. Scholfield, *The Theory of Proportion in Architecture*, Cambridge Univ. Press, London, 1958.

[36] J. Kappraff, *Connections: The Geometric Bridge Between Art and Science*, McGraw-Hill, New York, 1990.

[37] D. C. Kahaner, C. Moler, and S. Nash, *Numerical Methods and Software*, Prentice Hall, Englewood Cliffs, NJ, 1989.

[38] I. S. Gradshteyn and I. M. Ryzhik, *Table of Integrals, Series, and Products*, Academic Press, New York, 1980.

[39] H. R. Chillingworth, *Complex Variables*, Pergamon, Oxford, 1973.

[40] `http://dlmf.nist.gov/`, F. W. J. Olver, et al., eds. *NIST Digital Library of Mathematical Functions*, 2010, an updated and extended version of the Abramowitz and Stegun Handbook [41].

[41] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions*, Dover Publications, New York, 1965. Available online from: `http://people.math.sfu.ca/~cbm/aands/`, or, `http://www.math.hkbu.edu.hk/support/aands/`.

[42] D. S. G. Pollock, *Handbook of Time Series Analysis, Signal Processing, and Dynamics*, Academic, New York, 1999.

[43] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel, *Time Series Analysis: Forecasting and Control*, 4th ed., Wiley, New York, 2008.

[44] S. J. Orfanidis, *Introduction to Signal Processing*, 1st edition, Prentice Hall, Upper Saddle River, NJ, 1996. Available online from:
`http://www.ece.rutgers.edu/~orfanidi/intro2sp/`.

[45] S. J. Orfanidis, *Applied Optimum Signal Processing*, 2018, online book freely available from:
`https://www.ece.rutgers.edu/~orfanidi/aosp`.

See also, S. J. Orfanidis, *Optimum Signal Processing*, 2nd ed., McGraw-Hill, New York, 1988, and a free online 1996 version available from,
`https://www.ece.rutgers.edu/~orfanidi/osp2e`

[46] S. J. Orfanidis, *Electronagnetic Waves and Antennas*, online book, 2016, available from:
`https://www.ece.rutgers.edu/~orfanidi/ewa/`

Sampling

[47] D. A. Linden, "A Discussion of Sampling Theorems," *Proc. IRE*, **47**, 1219 (1959).

[48] A. J. Jerri, "The Shannon Sampling Theorem—Its Various Extensions and Applications: A Tutorial Review," *Proc. IEEE*, **65**, 1565 (1977).

[49] P. L. Butzer and R. L. Strauss, "Sampling Theory for not Necessarily Band-Limited Functions: A Historical Overview," *SIAM Review*, **34**, 40 (1992).

[50] R. J. Marks II, *Introduction to Shannon Sampling and Interpolation Theory*, Springer-Verlag, New York, 1991.

[51] R. J. Marks II, ed., *Advanced Topics in Shannon Sampling and Interpolation Theory*, Springer-Verlag, New York, 1993.

[52] M. Unser, "Sampling—50 Years After Shannon," *Proc. IEEE*, **88**, 569 (2000).

[53] P. P. Vaidyanathan, "Generalizations of the Sampling Theorem: Seven Decades After Nyquist," *IEEE Trasn. Circuits & Systems–I*, **48**, 1094 (2001).

[54] E. Meijering, "A Chronology of Interpolation: From Ancient Astronomy to Modern Signal and Image Processing," *Proc. IEEE*, **90**, 319 (2002).

[55] J. A. S. Angus, "Modern Sampling: A Tutorial," *J. Audio Eng. Soc.*, **67**, 300 (2019).

A/D & D/A Conversion, Quantization, Dithering, and Noise Shaping

[56] G. B. Clayton, *Data Converters*, Halsted Press, Wiley, New York, 1982.

[57] M. J. Demler, *High-Speed Analog-to-Digital Conversion*, Academic Press, New York, 1991.

[58] G. F. Miner and D. J. Comer, *Physical Data Acquisition for Digital Processing*, Prentice Hall, Englewood Cliffs, NJ, 1992.

[59] D. Seitzer, G. Pretzl, and N. A. Hamdy, *Electronic Analog-to-Digital Converters*, Wiley, New York, 1983.

[60] D. H. Sheingold, ed., *Analog-Digital Conversion Handbook*, 3d ed., Prentice Hall, Englewood Cliffs, NJ, 1986.

[61] A. VanDoren, *Data Acquisition Systems*, Reston Publishing, Reston, VA, 1982.

[62] W. R. Bennett, "Spectra of Quantized Signals," *Bell Syst. Tech. J.*, **27**, 446 (1948).

[63] B. Widrow, "Statistical Analysis of Amplitude-Quantized Sampled-Data Systems," *AIEE Trans. Appl. Ind.*, pt.2, **79**, 555 (1961).

[64] P. F. Swaszek, ed., *Quantization*, Van Nostrand Reinhold, New York, 1985.

[65] A. B. Sripad and D. L. Snyder, "A Necessary and Sufficient Condition for Quantization Errors to Be Uniform and White," *IEEE Trans. Acoust., Speech, Signal Process.*, **ASSP-25**, 442 (1977).

[66] C. W. Barnes, et al., "On the Statistics of Fixed-Point Roundoff Error," *IEEE Trans. Acoust., Speech, Signal Process.*, **ASSP-33**, 595 (1985).

[67] R. M. Gray, "Quantization Noise Spectra," *IEEE Trans. Inform. Theory*, **IT-36**, 1220 (1990), and earlier references therein. Reprinted in Ref. [350], p. 81.

[68] L. G. Roberts, "Picture Coding Using Pseudo-Random Noise," *IRE Trans. Inform. Th.*, **IT-8**, 145 (1962).

[69] L. Schuchman, "Dither Signals and Their Effect on Quantization Noise," *IEEE Trans. Commun.*, **COM-12**, 162 (1964).

[70] N. S. Jayant and P. Noll, *Digital Coding of Waveforms*, Prentice Hall, Englewood Cliffs, NJ 1984.

[71] J. F. Blinn, "Quantization Error and Dithering," *IEEE Comput. Graphics & Appl. Mag.*, (July 1994), p. 78.

[72] S. P. Lipshitz, R. A. Wannamaker, and J. Vanderkooy, "Quantization and Dither: A Theoretical Survey," *J. Audio Eng. Soc.*, **40**, 355 (1992).

[73] J. Vanderkooy and S. P. Lipshitz, "Resolution Below the Least Significant Bit in Digital Systems with Dither," *J. Audio Eng. Soc.*, **32**, 106 (1984).

[74] J. Vanderkooy and S. P. Lipshitz, "Dither in Digital Audio," *J. Audio Eng. Soc.*, **35**, 966 (1987).

[75] J. Vanderkooy and S. P. Lipshitz, "Digital Dither: Signal Processing with Resolution Far Below the Least Significant Bit," *Proc. 7th Int. Conf.: Audio in Digital Times*, Toronto, May 1989, p. 87.

[76] M. A. Gerzon and P. G. Graven, "Optimal Noise Shaping and Dither of Digital Signals," presented at 87th Convention of the AES, New York, October 1989, *AES Preprint 2822*, *J. Audio Eng. Soc.*, (Abstracts) **37**, 1072 (1989).

[77] S. P. Lipshitz, J. Vanderkooy, and R. A. Wannamaker, "Minimally Audible Noise Shaping," *J. Audio Eng. Soc.*, **39**, 836 (1991).

[78] R. A. Wannamaker, "Psychoacoustically Optimal Noise Shaping," *J. Audio Eng. Soc.*, **40**, 611 (1992).

[79] M. A. Gerzon, P. G. Graven, J. R. Stuart, and R. J. Wilson, "Psychoacoustic Noise Shaped Improvements in CD and Other Linear Digital Media," presented at 94th Convention of the AES, Berlin, May 1993, *AES Preprint no. 3501*.

[80] R. van der Waal, A. Oomen, and F. Griffiths, "Performance Comparison of CD, Noise-Shaped CD and DCC," presented at 96th Convention of the AES, Amsterdam, February 1994, *AES Preprint no. 3845*.

[81] J. A. Moorer and J. C. Wen, "Whither Dither: Experience with High-Order Dithering Algorithms in the Studio," presented at 95th Convention of the AES, New York, October 1993, *AES Preprint no. 3747*.

[82] R. A. Wannamaker, "Subtractive and Nonsubtractive Dithering: A Comparative Analysis," presented at 97th Convention of the AES, San Francisco, November 1994, *AES Preprint no. 3920*.

[83] D. Ranada, "Super CD's: Do They Deliver The Goods?," *Stereo Review*, July 1994, p. 61.

[84] M. A. Gerzon and P. G. Craven, "A High-Rate Buried-Data Channel for Audio CD," *J. Audio Eng. Soc.*, **43**, 3 (1995).

[85] A. Oomen, M. Groenewegen, R. van der Waal, and R. Veldhuis, "A Variable-Bit-Rate Buried-Data Channel for Compact Disc," *J. Audio Eng. Soc.*, **43**, 25 (1995).

[86] Trân-Thôńg and B. Liu, "Error Spectrum Shaping in Narrow Band Recursive Filters," *IEEE Trans. Acoust., Speech, Signal Process.*, **ASSP-25**, 200 (1977).

[87] W. E. Higgins and D. C. Munson, "Noise Reduction Strategies for Digital Filters: Error Spectrum Shaping Versus the Optimal Linear State-Space Formulation," *IEEE Trans. Acoust., Speech, Signal Process.*, **ASSP-30**, 963 (1982).

[88] C. T. Mullis and R. A. Roberts, "An Interpretation of Error Spectrum Shaping in Digital Filters," *IEEE Trans. Acoust., Speech, Signal Process.*, **ASSP-30**, 1013 (1982).

[89] J. Dattoro, "The Implementation of Digital Filters for High-Fidelity Audio," *Proc. AES 7th Int. Conf., Audio in Digital Times*, Toronto, 1989, p. 165.

[90] R. Wilson, et al., "Filter Topologies," *J. Audio Eng. Soc.*, **41**, 455 (1993).

[91] U. Zölzer, "Roundoff Error Analysis of Digital Filters," *J. Audio Eng. Soc.*, **42**, 232 (1994).

[92] W. Chen, "Performance of the Cascade and Parallel IIR Filters," presented at 97th Convention of the AES, San Francisco, November 1994, *AES Preprint no. 3901*.

[93] D. W. Horning and R. Chassaing, "IIR Filter Scaling for Real-Time Signal Processing," *IEEE Trans. Educ.*, **34**, 108 (1991).

[94] K. Baudendistel, "Am Improved Method of Scaling for Real-Time Signal Processing Applications," *IEEE Trans. Educ.*, **37**, 281 (1994).

DSP Hardware

[95] E. A. Lee, "Programmable DSP Architectures: Part I," *IEEE ASSP Mag.*, **5**, no. 4, 4 (1988), and "Programmable DSP Architectures: Part II," ibid., **6**, no. 1, 4 (1989).

[96] K-S. Lin, ed., *Digital Signal Processing Applications with the TMS320 Family*, vol. 1, Prentice Hall, Englewood Cliffs, NJ, 1987.

[97] P. E. Papamichalis, ed., *Digital Signal Processing Applications with the TMS320 Family*, vols. 2 and 3, Prentice Hall, Englewood Cliffs, NJ, 1991.

[98] *TMS320C2x, TMS320C3x, TMS320C4x, TMS320C5x, User Guides*, Texas Instruments, Dallas, TX, 1989–93.

[99] R. Chassaing, *Digital Signal Processing with C and the TMS320C30*, Wiley, New York, 1992.

[100] M. El-Sharkawy, *Real Time Digital Signal Processing Applications with Motorola's DSP56000 Family*, Prentice Hall, Englewood Cliffs, NJ, 1990.

[101] M. El-Sharkawy, *Signal Processing, Image Processing and Graphics Applications with Motorola's DSP96002 Processor*, vol. I, Prentice Hall, Englewood Cliffs, NJ, 1994.

[102] V. K. Ingle and J. G. Proakis, *Digital Signal Processing Laboratory Using the ADSP-2101 Microcomputer*, Prentice Hall, Englewood Cliffs, NJ, 1991.

[103] *WE® DSP32 Digital Signal Processor, Application Guide*, AT&T Document Management Organization, 1988.

[104] J. Tow, "Implementation of Digital Filters with the WE®DSP32 Digital Signal Processor," AT&T Application Note, 1988.

[105] S. L. Freeny, J. F. Kaiser, and H. S. McDonald, "Some Applications of Digital Signal Processing in Telecommunications," in Ref. [12], p. 1.

[106] J. R. Boddie, N. Sachs, and J. Tow, "Receiver for TOUCH-TONE Service," *Bell Syst. Tech. J.*, **60**, 1573 (1981).

[107] J. Hartung, S. L. Gay, and G. L. Smith, "Dual-Tone Multifrequency Receiver Using the WE®DSP32 Digital Signal Processor," AT&T Application Note, 1988.

[108] P. Mock, "Add DTMF Generation and Decoding to DSP-μP Designs," *EDN*, March 21, (1985). Reprinted in Ref. [96], p. 543.

[109] A. Mar, ed., *Digital Signal Processing Applications Using the ADSP-2100 Family*, Prentice Hall, Englewood Cliffs, NJ, 1990.

Computer Music

[110] M. V. Mathews, et al., *The Technology of Computer Music*, MIT Press, Cambridge, MA, 1969.

[111] F. R. Moore, *Elements of Computer Music*, Prentice Hall, Englewood Cliffs, NJ, 1990.

[112] C. Roads and J. Strawn, eds., *Foundations of Computer Music*, MIT Press, Cambridge, MA, 1988.

[113] C. Roads, ed., *The Music Machine*, MIT Press, Cambridge, MA, 1989.

[114] C. Dodge and T. A. Jerse, *Computer Music*, Schirmer/Macmillan, New York, 1985.

[115] G. Haus, ed., *Music Processing*, A-R Editions, Inc., Madison, WI, 1993.

[116] B. Vercoe, "Csound: A Manual for the Audio Processing System and Supporting Programs with Tutorials," *MIT Media Lab*, 1993. Csound is available via `ftp` from `cecelia.media.mit.edu` in `pub/Csound` and MSDOS versions from `ftp.maths.bath.ac.uk` in `pub/dream`.

[117] J. M. Chowning, "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation," *J. Audio Eng. Soc.*, **21**, 526, 1973. Reprinted in Ref. [112].

[118] F. R. Moore, "Table Lookup Noise for Sinusoidal Digital Oscillators," *Computer Music J.*, **1**, 26, 1977. Reprinted in Ref. [112].

[119] S. Mehrgardt, "Noise Spectra of Digital Sine-Generators Using the Table-Lookup Method," *IEEE Trans. Acoust., Speech, Signal Process.*, **ASSP-31**, 1037 (1983).

[120] W. M. Hartmann, "Digital Waveform Generation by Fractional Addressing," *J. Acoust. Soc. Am.*, **82**, 1883 (1987).

[121] P. D. Lehrman, "The Computer as a Musical Instrument," *Electronic Musician*, **7**, no. 3, 30 (1991).

[122] M. N. McNabb, "Dreamsong: The Composition," *Computer Music J.*, **5**, no. 4, 1981. Reprinted in Ref. [113].

[123] G. De Poli, “A Tutorial on Digital Sound Synthesis Techniques,” *Computer Music J.*, **7**, no. 4, (1983). Reprinted in Ref. [113].

[124] R. Karplus and A. Strong, “Digital Synthesis of Plucked String and Drum Timbres,” *Computer Music J.*, **7**, 43 (1983). Reprinted in Ref. [113].

[125] D. A. Jaffe and J. O. Smith, “Extensions of the Karplus-Strong Plucked-String Algorithm,” *Computer Music J.*, **7**, 56 (1983). Reprinted in Ref. [113].

[126] C. R. Sullivan, “Extending the Karplus-Strong Algorithm to Synthesize Electric Guitar Timbres with Distortion and Feedback,” *Computer Music J.*, **14**, 26 (1990).

[127] J. O. Smith, “Physical Modeling Using Digital Waveguides,” *Computer Music J.*, **16**, 74 (1992).

[128] G. Mayer-Kress, et al., “Musical Signals from Chua’s Circuit,” *IEEE Trans. Circuits Syst.—II: Analog and Digital Signal Process.*, **40**, 688 (1993).

[129] X. Rodet, “Models of Musical Instruments from Chua’s Circuit with Time Delay,” *IEEE Trans. Circuits Syst.—II: Analog and Digital Signal Process.*, **40**, 696 (1993).

[130] S. Wilkinson, “Model Music,” *Electronic Musician*, **10**, no. 2, 42 (1994).

[131] J. A. Moorer, “Signal Processing Aspects of Computer Music: A Survey,” *Proc. IEEE*, **65**, 1108, (1977). Reprinted in Ref. [158].

Digital Audio Effects

[132] *IEEE ASSP Mag.*, **2**, no. 4, October 1985, Special Issue on Digital Audio.

[133] Y. Ando, *Concert Hall Acoustics*, Springer-Verlag, New York, 1985.

[134] D. Begault, “The Evolution of 3-D Audio,” *MIX*, October 1993, p. 42.

[135] P. J. Bloom, “High-Quality Digital Audio in the Entertainment Industry: An Overview of Achievements and Challenges,” in Ref. [132], p. 2.

[136] B. Blesser and J. M. Kates, “Digital Processing of Audio Signals,” in Ref. [12], p. 29.

[137] N. Brighton and M. Molenda, “Mixing with Delay,” *Electronic Musician*, **9**, no. 7, 88 (1993).

[138] N. Brighton and M. Molenda, “EQ Workshop” *Electronic Musician*, October 1993, p. 105.

[139] D. Cronin, “Examining Audio DSP Algorithms,” *Dr. Dobbs Journal*, **19**, no. 7, 78, (1994).

[140] D. Griesinger, “Practical Processors and Programs for Digital Reverberation,” *Proc. AES 7th Int. Conf., Audio in Digital Times*, Toronto, 1989, p. 187.

[141] G. Hall, “Effects, The Essential Musical Spice,” *Electronic Musician*, **7**, no. 8, 62 (1991).

[142] G. Hall, “Solving the Mysteries of Reverb,” *Electronic Musician*, **7**, no. 6, 80 (1991).

[143] M. Kleiner, B. Dalebäck, and P. Svensson, “Auralization—An Overview,” *J. Audio Eng. Soc.*, **41**, 861 (1993).

[144] P. D. Lehrman, “The Electronic Orchestra, Parts I and II,” *Electronic Musician*, September 1993, p. 41, and October 1993, p. 46.

[145] J. Meyer, “The Sound of the Orchestra,” *J. Audio Eng. Soc.*, **41**, 203 (1993).

[146] J. A. Moorer, “About this Reverberation Business,” *Computer Music J.*, **3**, 13 (1979). Reprinted in Ref. [112].

[147] D. Moulton, “Spectral Management, Parts 1 & 2,” *Home & Studio Recording*, July 1993, p. 22, and August 1993, p. 50.

[148] E. Persoon and C. Vanderbulcke, “Digital Audio: Examples of the Application of the ASP Integrated Signal Processor,” *Philips Tech. Rev.*, **42**, 201 (1986).

[149] J. R. Pierce, *The Science of Musical Sound*, W. H. Freeman and Company, New York, 1992.

[150] K. Pohlmann, *The Principles of Digital Audio*, 2nd ed., H. W. Sams, Carmel, IN, 1989.

[151] K. Pohlmann, ed., *Advanced Digital Audio*, H. W. Sams, Carmel, IN, 1991.

[152] M. R. Schroeder, “Natural Sounding Artificial Reverberation,” *J. Audio Eng, Soc.*, **10**, 219, (1962).

[153] M. R. Schroeder, “Digital Simulation of Sound Transmission in Reverberant Spaces", *J. Acoust. Soc. Am.*, **47**, 424 (1970).

[154] M. R. Schroeder, D. Gottlob, and K. F. Siebrasse, “Comparative Study of European Concert Halls: Correlation of Subjective Preference with Geometric and Acoustic Parameters,” *J. Acoust. Soc. Am.*, **56**, 1195 (1974).

[155] W. L. Sinclair and L. I. Haworth, “Digital Recording in the Professional Industry, parts I and II,” *Electronics & Communications Eng. J.*, June and August 1991.

[156] *SE-50 Stereo Effects Processor*, User Manual, Roland/Boss Corp., 1990.

[157] J. O. Smith, “An Allpass Approach to Digital Phasing and Flanging,” *Proc. Int. Computer Music Conf. (ICMC)*, IRCAM, Paris, Oct. 1984, p. 236.

[158] J. Strawn, ed., *Digital Audio Signal Processing: An Anthology*, W. Kaufmann, Los Altos, CA, 1985.

[159] J. Watkinson, *The Art of Digital Audio*, Focal Press, London, 1988.

[160] P. White, *Creative Recording: Effects and Processors*, Music Maker Books, Cambridgeshire, UK, 1989.

[161] J. D. Reiss and A. P. McPherson, *Audio Effects : Theory, Implementation And Application*, CRC Press, 2014.

[162] A. Uncini, *Digital Audio Processing Fundamentals*, Springer Nature, Switzerland, 2022.

Dynamic Range Control

[163] B. Blesser, “Audio Dynamic Range Compression for Minimum Perceived Distortion,” *IEEE Trans. Audio Electroacoust.*, **AU-17**, 22 (1969).

[164] G. W. McNally, “Dynamic Range Control of Digital Audio Signals,” *J. Audio Eng. Soc.*, **32**, 316 (1984).

[165] G. Davis and R. Jones, *Sound Reinforcement Handbook*, 2nd ed., Yamaha Corp., Hal Leonard Publishing, Milwaukee, WI., 1989.

[166] B. Hurtig, “The Engineer’s Notebook: Twelve Ways to Use Dynamics Processors,” *Electronic Musician*, **7**, no. 3, 66 (1991). See also, B. Hurtig, “Pumping Gain: Understanding Dynamics Processors,” *Electronic Musician*, **7**, no. 3, 56 (1991).

[167] J. M. Eargle, *Handbook of Recording Engineering*, 2nd ed., Van Nostrand Reinhold, New York, 1992.

[168] P. Freudenberg, “All About Dynamics Processors, Parts 1 & 2,” *Home & Studio Recording*, March 1994, p. 18, and April 1994, p. 44.

[169] U. Zölzer, ed., *DAFX - Digital Audio Effects*, Wiley, Chichester, England, 2003. MATLAB and audio files available from:
`http://ant-s4.unibw-hamburg.de/dafx/DAFX_Book_Page/matlab.html`
`https://www.dafx.de/DAFX_Book_Page/matlab.html`

[170] D. Giannoulis, M. Massberg, and J. D. Reiss. “Digital Dynamic Range Compressor Design—A Tutorial and Analysis,” *J. Audio Eng. Soc.*, **60**, 399 (2012).

[171] “Dynamics Processors,” ReneNote 155, Rane Corporation, 2005. Available from,
`https://www.ranecommercial.com/kb_article.php?article=2129`
`https://www.ranecommercial.com/legacy/note155.html`

[172] S. Banerjee, “The Compression Handbook,” 4th ed, Starkey Hearing Research & Technology, Starkey Laboratories, 2017. Available from:
`https://starkeypro.com/pdfs/The_Compression_Handbook.pdf`

[173] Wikipedia article, “Dynamic range compression,” with additional references therein.
`https://en.wikipedia.org/wiki/Dynamic_range_compression`

Biomedical Signal Processing

[174] M. L. Ahlstrom and W. J. Tompkins, “Digital Filters for Real-Time ECG Signal Processing Using Microprocessors,” IEEE Trans. Biomed. Eng., **BME-32**, 708 (1985).

[175] I. I. Christov and I. A. Dotsinsky, “New Approach to the Digital Elimination of 50 Hz Interference from the Electrocardiogram,” *Med. & Biol. Eng. & Comput.*, **26** 431 (1988).

[176] “The Design of Digital Filters for Biomedical Signal Processing,” Part-1, *J. Biomed. Eng.*, **4**, 267 (1982), Part-2, ibid., **5**, 19 (1983), Part-3, ibid., **5**, 91 (1983).

[177] M. Della Corte, O. Cerofolini, and S. Dubini, “Application of Digital Filter to Biomedical Signals,” *Med. & Biol. Eng.*, **12** 374 (1974).

[178] G. M. Friesen, et al., “A Comparison of the Noise Sensitivity of Nine QRS Detection Algorithms,” *IEEE Trans. Biomed. Eng.*, **BME-37**, 85 (1990).

[179] C. Levkov, et al., “Subtraction of 50 Hz Interference from the Electrocardiogram,” *Med. & Biol. Eng. & Comput.*, **22** 371 (1984).

[180] C. L. Levkov, “Fast Integer Coefficient FIR Filters to Remove the AC Interference and the High-Frequency Noise Components in Biological Signals,” *Med. & Biol. Eng. & Comput.*, **27** 330 (1989).

[181] P. A. Lynn, “Online Digital Filters for Biological Signals: Some Fast designs for a Small Computer,” *Med. & Biol. Eng. & Comput.*, **15** 534 (1977).

[182] P. A. Lynn, “Transversal Resonator Digital Filters: Fast and Flexible Online Processors for Biological Signals,” *Med. & Biol. Eng. & Comput.*, **21** 718 (1983).

[183] R. M. Lu and B. M. Steinhaus, “A Simple Digital Filter to Remove Line-Frequency Noise in Implantable Pulse Generators,” *Biomed. Instr. & Technol.*, **27**, 64 (1993).

[184] N. R. Malik, “Microcomputer Realisations of Lynn’s Fast Digital Filtering Designs,” *Med. & Biol. Eng. & Comput.*, **18** 638 (1980). (This reference uses circular addressing to implement delays and refers to its earlier use by J. D. Schoeffler (1971) as a wrap-around queue.)

[185] C. J. Marvell and D. L. Kirk, “Use of a Microprocessor to Simulate Precise Electrocardiograms,” *J. Biomed. Eng.*, **2**, 61 (1980).

[186] V. T. Rhyne, “A Digital System for Enhancing the Fetal Electrocardiogram,” IEEE Trans. Biomed. Eng., **BME-16**, 80 (1969).

[187] J. E. Sheild and D. L. Kirk, “The Use of Digital Filters in Enhancing the Fetal Electrocardiogram,” *J. Biomed. Eng.*, **3**, 44 (1981).

[188] T. P. Taylor and P. W. Macfarlane, “Digital Filtering of the ECG–A Comparison of Low-pass Digital Filters on a Small Computer,” *Med. & Biol. Eng.*, **12** 493 (1974).

[189] N. V. Thakor and D. Moreau, “Design and Analysis of Quantised Coefficient Digital Filters: Application to Biomedical Signal Processing With Microprocessors,” *Med. & Biol. Eng. & Comput.*, **25** 18 (1987).

[190] W. J. Tompkins and J. G. Webster, eds., *Design of Microcomputer-Based Medical Instrumentation*, Prentice Hall, Englewood Cliffs, NJ, 1981.

[191] W. J. Tompkins, ed., *Biomedical Digital Signal Processing*, Prentice Hall, Englewood Cliffs, NJ, 1993.

[192] R. Wariar and C. Eswaran, “Integer Coefficient Bandpass Filter for the Simultaneous Removal of Baseline Wander, 50 and 100 Hz Interference from the ECG,” *Med. & Biol. Eng. & Comput.*, **29** 333 (1991).

Digital TV

[193] A. Acampora, “Wideband Picture Detail Restoration in a Digital NTSC Comb-Filter System,” *RCA Engineer*, **28-5**, 44, Sept./Oct. (1983).

[194] A. A. Acampora, R. M. Bunting, and R. J. Petri, “Noise Reduction in Video Signals by a Digital Fine-Structure Process,” *RCA Engineer*, **28-2**, 48, March/April (1983).

[195] A. A. Acampora, R. M. Bunting, and R. J. Petri, “Noise Reduction in Video Signals Using Pre/Post Signal Processing in a Time Division Multiplexed Component System,” *RCA Review*, **47**, 303 (1986).

[196] M. Annegarn, A. Nillesen, and J. Raven, “Digital Signal Processing in Television Receivers,” *Philips Tech. Rev.*, **42**, 183 (1986).

[197] J. F. Blinn, “NTSC: Nice Technology, Super Color,” *IEEE Comput. Graphics & Appl. Mag.*, (March 1993), p. 17.

[198] J. Isailovic, *Videodisc Systems: Theory and Applications*, Prentice Hall, Englewood Cliffs, NJ, 1987.

[199] H. E. Kallmann, “Transversal Filters,” *Proc. IRE*, **28**, 302 (1940). Perhaps, the earliest reference on FIR and comb filters.

[200] P. Mertz and F. Gray, “A Theory of Scanning and Its Relation to the Characteristics of the Transmitted Signal in Telephotography and Television,” *Bell Syst. Tech. J.*, **13**, 464 (1934).

[201] D. E. Pearson, *Transmission and Display of Pictorial Information*, Wiley, New York, 1975.

[202] J. O. Limb, C. B. Rubinstein, and J. E. Thompson, “Digital Coding of Color Video Signals—A Review,” *IEEE Trans. Commun.*, **COM-25**, 1349 (1977).

[203] C. H. Lu, “Subcarrier Phase Coherence in Noise Reduction of Composite NTSC Signals—Three Approaches and Their Comparison,” *RCA Review*, **47**, 287 (1986).

[204] D. H. Pritchard and J. J. Gibson, “Worldwide Color TV Standards—Similarities and Differences,” *RCA Engineer*, **25-5**, 64, Feb./Mar. (1980).

[205] D. H. Pritchard, “A CCD Comb Filter for Color TV Receiver Picture Enhancement,” *RCA Review*, **41**, 3 (1980).

[206] J. P. Rossi, “Color Decoding a PCM NTSC Television Signal,” *SMPTE J.*, **83**, 489 (1974).

[207] J. P. Rossi, “Digital Television Image Enhancement,” *SMPTE J.*, **84**, 546 (1975).

[208] C. P. Sandbank, ed., *Digital Television*, Wiley, Chichester, England, 1990.

[209] K. B. Benson and D. G. Fink, *HDTV, Advanced Television for the 1990s*, Intertext/Multiscience, McGraw-Hill, New York, 1991.

[210] M. I. Krivocheev and S. N. Baron, “The First Twenty Years of HDTV: 1972-1992,” *SMPTE J.*, **102**, 913 (1993).

[211] Special Issue on All Digital HDTV, *Signal Processing: Image Commun.*, **4**, no. 4–5 (Aug. 1992).

[212] Special Issue on Digital High Definition Television, *Signal Processing: Image Commun.*, **5**, no. 5–6 (Dec. 1993).

[213] R. Hopkins, “Digital Terrestrial HDTV for North America: The Grand Alliance HDTV System,” *IEEE Trans. Consum. Electr.*, **40**, 185 (1994).

Signal Averaging

[214] D. G. Childers, “Biomedical Signal Processing,” in *Selected Topics in Signal Processing*, S. Haykin, ed., Prentice Hall, Englewood Cliffs, NJ, 1989.

[215] A. Cohen, *Biomedical Signal Processing*, vols. 1 and 2, CRC Press, Boca Raton, FL, 1986.

[216] H. G. Goovaerts and O. Rompelman, “Coherent Average Technique: A Tutorial Review,” *J. Biomed. Eng.*, **13**, 275 (1991).

[217] P. Horowitz and W. Hill, *The Art of Electronics*, 2nd ed., Cambridge University Press, Cambridge, 1989.

[218] O. Rompelman and H. H. Ros, “Coherent Averaging Technique: A Tutorial Review, Part 1: Noise Reduction and the Equivalent Filter,” *J. Biomed. Eng.*, **8**, 24 (1986); and “Part 2: Trigger Jitter, Overlapping Responses, and Non-Periodic Stimulation,” ibid., p. 30.

[219] V. Shvartsman, G. Barnes, L. Shvartsman, and N. Flowers, “Multichannel Signal Processing Based on Logic Averaging,” *IEEE Trans. Biomed. Eng.*, **BME-29**, 531 (1982).

[220] C. W. Thomas, M. S. Rzeszotarski, and B. S. Isenstein, “Signal Averaging by Parallel Digital Filters,” *IEEE Trans. Acoust., Speech, Signal Process.*, **ASSP-30**, 338 (1982).

[221] T. H. Wilmshurst, *Signal Recovery from Noise in Electronic Instrumentation*, 2nd ed., Adam Hilger and IOP Publishing, Bristol, England, 1990.

Windows

[222] F. J. Harris, “On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform,” *Proc. IEEE*, **66**, 51 (1978).

[223] N. C. Geçkinli and D. Yavuz, “Some Novel Windows and a Concise Tutorial Comparison of Window Families,” *IEEE Trans. Acoust., Speech, Signal Process.*, **ASSP-26**, 501 (1978).

[224] J. F. Kaiser and R. W. Schafer, “On the Use of the I_0-Sinh Window for Spectrum Analysis,” *IEEE Trans. Acoust., Speech, Signal Process.*, **ASSP-28**, 105 (1980).

[225] A. H. Nuttal, “Some Windows with Very Good Sidelobe Behavior,” *IEEE Trans. Acoust., Speech, Signal Process.*, **ASSP-29**, 84 (1981).

[226] K. M. Prabhu, *Window Functions and Their Applications in Signal Processing*, Taylor & Francis Group, Boca Raton , FL, 2014.

[227] A. W. Doerry, “Catalog of Window Taper Functions for Sidelobe Control,” Sandia National Laboratories, Report SAND2017-4042, April 2017.

[228] C. L. Dolph, “A Current Distribution for Broadside Arrays Which Optimizes the Relationship Between Beam Width and Side-Lobe Level,” *Proc. IRE*, **34**, 335 (1946).

[229] D. Barbiere, “A Method for Calculating the Current Distribution of Tschebyscheff Arrays,” *Proc. IRE*, **40**, 78 (1952).

[230] R. J. Stegen, “Excitation Coefficients and Beamwidths of Tschebyscheff Arrays,” *Proc. IRE*, **41**, 1671 (1953).

[231] A. D. Bresler, “A New Algorithm for Calculating the Current Distributions of Dolph-Chebyshev Arrays,” *IEEE Trans. Antennas Propagat.*, **AP-28**, 951 (1980).

[232] T. T. Taylor, “One Parameter Family of Line Sources Producing $\sin\pi u/\pi u$ Patterns,” Technical Memorandum no.324, Hughes Aircraft Company, Sept. 1953. Available online from: `http://www.ece.rutgers.edu/~orfanidi/ewa/taylor-1953.pdf`

[233] T. T. Taylor, “Design of Line-Source Antennas for Narrow Beamwidth and Low Side Lobes,” *IRE Trans. Antennas Propagat.*, **AP-3**, 16 (1955).

[234] A. Papoulis and M. S. Bertram, “Digital Filtering and Prolate Functions,” *IEEE Trans. Circuit Th.*, **CT-19**, 674 (1972).

[235] A. T. Villeneuve, “Taylor Patterns for Discrete Arrays,” *IEEE Trans. Antennas Propagat.*, **AP-32**, 1089 (1984).

[236] D. Slepian, “Prolate Spheroidal Wave Functions, Fourier Analysis and Uncertainty—V: The Discrete Case,” *Bell Syst. Tech. J.*, **57**, 1371 (1978). Available online from *Bell System Technical Journal*, `http://www.alcatel-lucent.com/bstj/`,
and also, `https://archive.org/details/bstj-archives`

[237] A. T. Walden, “Accurate Approximation of a 0th Order Discrete Prolate Spheroidal Sequence for Filtering and Data Tapering,” *Sig. Process.*, **18**, 341 (1989).

[238] J. W. Adams, “A New Optimal Window,” *IEEE Trans. Acoust., Speech, Signal Process.*, **39**, 1753 (1991).

[239] J. M. Varah, “The Prolate Matrix,” *Lin. Alg. Appl.*, **187**, 269 (1993).

[240] D. B. Percival and A. T. Walden, *Spectral Analysis for Physical Applications*, Cambridge Univ. Press., Cambridge, 1993.

[241] T. Verma, S. Bilbao, and T. H. Y. Meng, “The Digital Prolate Spheroidal Window,” *IEEE Int. Conf. Acoust., Speech, Signal Process.*, **ICASSP-96**, 1351 (1996).

Spectral Analysis and DFT/FFT Algorithms

[242] E. O. Brigham, *The Fast Fourier Transform*, Prentice Hall, Englewood Cliffs, NJ, 1988.

[243] R. W. Ramirez, *The FFT, Fundamentals and Concepts*, Prentice Hall, Englewood Cliffs, NJ, 1985.

[244] C. S. Burrus and T. W. Parks, *DFT/FFT and Convolution Algorithms*, Wiley, New York, 1985.

[245] C. Van Loan, *Computational Frameworks for the Fast Fourier Transform*, SIAM, Philadelphia, 1992.

[246] W. L. Briggs and V. E. Henson, *The DFT: An Owner’s Manual for the Discrete Fourier Transform*, SIAM, Philadelphia, 1995.

[247] J. W. Cooley, P. A. W. Lewis, and P. D. Welch, "The Fast Fourier Transform and Its Applications," *IEEE Trans. Educ.*, **12**, 27 (1969).

[248] G. D. Bergland, "A Guided Tour of the Fast Fourier Transform," IEEE Spectrum, **6**, 41, July 1969.

[249] J. W. Cooley, P. A. W. Lewis, and P. D. Welch, "The Fast Fourier Transform Algorithm: Programming Considerations in the Calculation of Sine, Cosine, and Laplace Transforms," *J. Sound Vib.*, **12**, 315 (1970). Reprinted in Ref. [9].

[250] F. J. Harris, "The Discrete Fourier Transform Applied to Time Domain Signal Processing," *IEEE Commun. Mag.*, May 1982, p. 13.

[251] J. W. Cooley, P. A. W. Lewis, and P. D. Welch, "Historical Notes on the Fast Fourier Transform," *IEEE Trans. Audio Electroacoust.*, **AU-15**, 76 (1967). Reprinted in Ref. [9].

[252] M. T. Heideman, D. H. Johnson, and C. S. Burrus, "Gauss and the History of the Fast Fourier Transform," *IEEE ASSP Mag.*, **4**, no. 4, 14 (1984).

[253] J. W. Cooley, "How the FFT Gained Acceptance," *IEEE Signal Proc. Mag.*, **9**, no. 1, 10 (1992).

[254] P. Kraniauskas, "A Plain Man's Guide to the FFT," *IEEE Signal Proc. Mag.*, **11**, no. 2, 36 (1994).

[255] J. R. Deller, "Tom, Dick, and Mary Discover the DFT," *IEEE Signal Proc. Mag.*, **11**, no. 2, 36 (1994).

[256] P. Duhamel and M. Vetterli, "Fast Fourier Transforms: A Tutorial Review and a State of the Art," *Signal Processing*, **19**, 259 (1990).

[257] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete Cosine Transform," *IEEE Trans. Comput.*, **C-23**, 90 (1974).

[258] H. S. Hou, "A Fast Recursive Algorithm for the Discrete Cosine Transform," *IEEE Trans. Acoust., Speech, Signal Process.*, **ASSP-35**, 1455 (1987).

[259] J. F. Blinn, "What's the Deal with the DCT?," *IEEE Comput. Graphics & Appl. Mag.*, (July 1993), p. 78.

[260] R. C. Singleton, "A Method for Computing the Fast Fourier Transform with Auxiliary Memory and Limited High-Speed Storage," *IEEE Trans. Audio Electroacoust.*, **AU-15**, 91 (1967). Reprinted in Ref. [9].

[261] N. M. Brenner, "Fast Fourier Transform of Externally Stored Data," *IEEE Trans. Audio Electroacoust.*, **AU-17**, 128 (1969).

[262] W. K. Hocking, "Performing Fourier Transforms on Extremely Long Data Streams," *Comput. Phys.*, **3**, 59 (1989).

[263] H. V. Sorensen and C. S. Burrus, "Efficient Computation of the DFT with Only a Subset of Input or Output Points," *IEEE Trans. Acoust., Speech, Signal Process.*, **ASSP-41**, 1184 (1993).

[264] FFTW is perhaps the most efficient, fast, and flexible version of FFT implementations. It is described in, M. Frigo and S. G. Johnson, "The Design and Implementation of FFTW3," *Proc. IEEE*, **93**, 216 (2005), and it is freely available from: `http://www.fftw.org/`

FIR Filter Design

[265] J. F. Kaiser, "Design Methods for Sampled Data Filters," *Proc. 1st Allerton Conf. Circuit System Theory*, p. 221, (1963), and reprinted in Ref. [9], p. 20.

[266] J. F. Kaiser, "Digital Filters," in F. F. Kuo and J. F. Kaiser, eds., *System Analysis by Digital Computer*, Wiley, New York, 1966, p. 228.

[267] J. F. Kaiser, "Nonrecursive Digital Filter Design Using the I_0-Sinh Window Function," *Proc. 1974 IEEE Int. Symp. on Circuits and Systems*, p. 20, (1974), and reprinted in [10], p. 123.

[268] H. D. Helms, "Nonrecursive Digital Filters: Design Methods for Achieving Specifications on Frequency Response," *IEEE Trans. Audio Electroacoust.*, **AU-16**, 336 (1968).

[269] H. D. Helms, "Digital Filters with Equiripple or Minimax Response," *IEEE Trans. Audio Electroacoust.*, **AU-19**, 87 (1971), and reprinted in Ref. [9], p. 131.

[270] R. C. Hansen, "Linear Arrays," in A. W. Rudge, et al., eds., *Handbook of Antenna Design*, vol. 2, 2nd ed., P. Peregrinus and IEE, London, 1986.

[271] T. Saramäki, "Finite Impulse Response Filter Design," in Ref. [5], p. 155.

[272] K. B. Benson and J. Whitaker, *Television and Audio Handbook*, McGraw-Hill, New York, 1990.

[273] P. L. Schuck, "Digital FIR Filters for Loudspeaker Crossover Networks II: Implementation Example," *Proc. AES 7th Int. Conf., Audio in Digital Times*, Toronto, 1989, p. 181.

[274] R. Wilson, et al., "Application of Digital Filters to Loudspeaker Crossover Networks," *J. Audio Eng. Soc.*, **37**, 455 (1989).

[275] K. Steiglitz, T. W. Parks, and J. F. Kaiser, "METEOR: A Constraint-Based FIR Filter Design Program," *IEEE Trans. Acoust., Speech, Signal Process.*, **ASSP-40**, 1901 (1992). This program is available via anonymous `ftp` from `princeton.edu`.

[276] K. Steiglitz and T. W. Parks, "What is the Filter Design Problem?," *Proc. 1986 Princeton Conf. Inform. Sci. Syst.*, p. 604 (1986).

Gibbs Phenomenon

[277] H. Wilbraham, "On a certain periodic function," *Cambridge and Dublin Math. J.*, **3**, 198 (1848).

[278] J. W. Gibbs, Letter in *Nature* **59**, 200 (1898-1899), and, *ibid.*, p.606. Also in Collected Works, Vol. II. New York: Longmans, Green & Co., 1927, p.258.

[279] H. S. Carslaw, "A trigonometrical sum and the Gibbs' phenomenon in Fourier's series," *Amer. J. Math.*, **39**, 185 (1917). See also, H. S. Carslaw, "A historical note on Gibbs' phenomenon in Fourier's series and integrals," *Bull. Am. Math. Soc.*, **31**, 420 (1925).

[280] E. Hewitt and R. E. Hewitt, "The Gibbs-Wilbraham Phenomenon: An Episode in Fourier Analysis," Author(s): Edwin Hewitt and Robert E. Hewitt Source: *Archive Hist. Exact Sciences*, **21**, 129 (1979).

[281] D. Gottlieb and C-W. Shi, "On the Gibbs phenomenon and its resolution," *SIAM Rev.*, **39**, 644 (1997).

[282] `https://en.wikipedia.org/wiki/Fourier_series`
`https://en.wikipedia.org/wiki/Dirichlet_conditions`
`https://en.wikipedia.org/wiki/Gibbs_phenomenon`

Second-Order IIR Filter Design

[283] K. Hirano, S. Nishimura, and S. Mitra, "Design of Digital Notch Filters," *IEEE Trans. Commun.*, **COM-22**, 964 (1974).

[284] M. N. S. Swami and K. S. Thyagarajan, "Digital Bandpass and Bandstop Filters with Variable Center Frequency and Bandwidth," *Proc. IEEE*, **64**, 1632 (1976).

[285] G. W. McNally, "Digital Audio: Recursive Digital Filtering for High Quality Audio Signals," BBC Research Dept. Report, BBC RD 1981/10, Dec. 1981. Available online from: `www.bbc.co.uk/rd/pubs/reports/1981-10.pdf`.

[286] J. A. Moorer, "The Manifold Joys of Conformal Mapping: Applications to Digital Filtering in the Studio," *J. Audio Eng. Soc.*, **31**, 826 (1983). Updated version available online from `www.jamminpower.com`.

[287] S. A. White, "Design of a Digital Biquadratic Peaking or Notch Filter for Digital Audio Equalization," *J. Audio Eng. Soc.*, **34**, 479 (1986).

[288] P. A. Regalia and S. K. Mitra, "Tunable Digital Frequency Response Equalization Filters," *IEEE Trans. Acoust., Speech, Signal Process.*, **ASSP-35**, 118 (1987).

[289] D. J. Shpak, "Analytical Design of Biquadratic Filter Sections for Parametric Filters," *J. Audio Eng. Soc.*, **40**, 876 (1992).

[290] D. C. Massie, "An Engineering Study of the Four-Multiply Normalized Ladder Filter," *J. Audio Eng. Soc.*, **41**, 564 (1993).

[291] F. Harris and E. Brooking, "A Versatile Parametric Filter Using Imbedded All-Pass Sub-Filter to Independently Adjust Bandwidth, Center Frequency, and Boost or Cut," presented at the 95th Convention of the AES, New York, October 1993, *AES Preprint 3757*.

[292] R. Bristow-Johnson, "The Equivalence of Various Methods of Computing Biquad Coefficients for Audio Parametric Equalizers," presented at the 97th Convention of the AES, San Francisco, November 1994, *AES Preprint 3906*.

[293] R. Bristow-Johnson, "Cookbook formulae for audio EQ biquad filter coefficients," available from: `www.musicdsp.org/files/Audio-EQ-Cookbook.txt`.

[294] U. Zölzer and T. Boltze, "Parametric Digital Filter Structures," Presented at the 99th Convention of the AES, New York, October 1995, *AES Preprint 4099*.

[295] U. Zölzer, *Digital Audio Signal Processing*, Wiley, Chichester, UK, 1997.

[296] S. J. Orfanidis, "Digital Parametric Equalizer Design With Prescribed Nyquist-Frequency Gain," *J. Aud. Eng. Soc.*, **45**, 444 (1997).

[297] K. B. Christensen, "A Generalization of the Biquad Parametric Equalizer," Presented at the 115th Convention of the AES, New York, October 2003, *AES Preprint 5916*.

Analog Filter Design

[298] M. E. Van Valkenburg, *Analog Filter Design*, Holt, Rinehart and Winston, New York, 1982.

[299] A. B. Williams and F. J. Taylor, *Electronic Filter Design Handbook*, 2nd ed., McGraw-Hill, New York, 1988.

[300] L. P. Huelsman, *Active and Passive Analog Filter Design: An Introduction*, McGraw-Hill, New York, 1993.

[301] W. J. Thompson, "Chebyshev Polynomials: After the Spelling the Rest Is Easy," *Comput. Phys.*, **8**, 161 (1994).

Elliptic Filter Design

[302] E. I. Zolotarev, "Application of Elliptic Functions to Problems about Functions with Least and Greatest Deviation from Zero," *Zap. Imp. Akad. Nauk. St. Petersburg*, **30** (1877), no. 5. (in Russian); available online from:
`www.math.technion.ac.il/hat/fpapers/zol1.pdf`.

[303] W. Cauer, *Synthesis of Linear Communication Networks*, McGraw-Hill, New York, 1958.

[304] S. Darlington, "Synthesis of Reactance 4-Poles which Produce Prescribed Insertion Loss Characteristics," *J. Math. and Phys.*, **18**, 257 (1939).

[305] A. J. Grossman, "Synthesis of Tchebycheff Parameter Symmetrical Filters," *Proc. IRE*, **45**, 454 (1957).

[306] J. E. Storer, *Passive Network Synthesis*, McGraw-Hill, New York, 1957.

[307] R. W. Daniels, *Approximation Methods for Electronic Filter Design*, McGraw-Hill, New York, 1974.

[308] H. J. Orchard, "Adjusting the Parameters of Elliptic Function Filters," *IEEE Trans. Circuits Syst., I*, **37**, 631 (1990).

[309] H. J. Orchard and A. N. Willson, "Elliptic Functions for Filter Design," *IEEE Trans. Circuits Syst., I*, **44**, 273 (1997).

[310] A. Antoniou, *Digital Filters*, 2nd ed., McGraw-Hill, New York, 1993.

[311] M. D. Lutovac, D. V. Tosic, and B. L. Evans, *Filter Design for Signal Processing*, Prentice Hall, Upper Saddle River, NJ, 2001.

[312] M. Vlcek and R. Unbehauen, "Degree, Ripple, and Transition Width of Elliptic Filters," IEEE Trans. Circ. Syst., **CAS-36**, 469 (1989).

[313] C. G. J. Jacobi, "Fundamenta Nova Theoriae Functionum Ellipticarum," reprinted in *C. G. J. Jacobi's Gesammelte Werke*, vol.1, C. W. Borchardt, ed., Verlag von G. Reimer, Berlin, 1881.

[314] F. Bowman, *Introduction to Elliptic Functions with Applications*, Dover Publications, New York, 1961.

[315] E. H. Neville, *Jacobian Elliptic Functions*, Oxford University Press, 1944.

[316] A. Cayley, *An Elementary Treatise on Elliptic Functions*, Dover Publications, New York, 1961.

[317] D. F. Lawden, *Elliptic Functions and Applications*, Springer-Verlag, New York, 1989.

[318] P. F. Byrd and M. D. Friedman, *Handbook of Elliptic Integrals for Engineers and Scientists*, Springer-Verlag, New York, 1971.

[319] N. I. Akhiezer, *Elements of the Theory of Elliptic Functions*, Translations of Mathematical Monographs, vol.79, American Mathematical Society, Providence, RI, 1990.

[320] R. Hoppe, "Elliptische Integrale und Funktionen nach Jacobi," available online at the web page: `http://www.dfcgen.de/wpapers/elliptic/elliptic.html`

[321] A. G. Constantinides, "Frequency Transformations for Digital Filters," *Elect. Lett.*, **3**, 487 (1967), and *ibid.*, **4**, 115 (1968).

[322] A. G. Constantinides, "Spectral Transformations for Digital Filters," *Proc. IEE*, **117**, 1585 (1970).

[323] M. N. S. Swami and K. S. Thyagarajan, "Digital Bandpass and Bandstop Filters with Variable Center Frequency and Bandwidth," *Proc. IEEE*, **64**, 1632 (1976).

[324] S. K. Mitra, Y. Neuvo, and H. Roivainen, "Design of Recursive Digital Filters with Variable Characteristics," *Int. J. Circ. Th. Appl.*, **18**, 107 (1990).

[325] S. K. Mitra, K. Hirano, and S. Nishimura, "Design of Digital Bandpass/Bandstop Filters with Independent Tuning Characteristics," *Frequenz*, **44**, 117 (1990).

[326] F. Harris and E. Brooking, "A Versatile Parametric Filter Using Imbedded All-Pass Sub-Filter to Independently Adjust Bandwidth, Center Frequency, and Boost or Cut," Presented at the 95th Convention of the AES, New York, October 1993, *AES Preprint 3757*.

High-Order Digital Parametric Equalizer Design

[327] J. A. Moorer, "The Manifold Joys of Conformal Mapping: Applications to Digital Filtering in the Studio," *J. Audio Eng. Soc.*, **31**, 826 (1983). Updated version available online from `www.jamminpower.com`.

[328] F. Keiler and U. Zölzer, "Parametric Second- and Fourth-Order Shelving Filters for Audio Applications," *Proc. IEEE 6th Workshop on Multimedia Signal Processing*, Siena, Italy, Sept., 2004, p.231.

[329] S. J. Orfanidis, "High-Order Digital Parametric Equalizer Design," *J. Audio Eng. Soc.*, **53**, 1026 (2005). `https://www.aes.org/e-lib/browse.cfm?elib=13397`

[330] Available from from the author's web page: `www.ece.rutgers.edu/~orfanidi/hpeq`, and from the JAES supplementary materials page:
`https://www.aes.org/journal/suppmat/hpeq_2005_11.zip`.

[331] R. Bristow-Johnson, Private Communication, June 2005.

[332] R. A. Losada and V. Pellissier, "Designing IIR Filters with a Given 3-dB Point," *IEEE Signal Process. Mag.*, **22**, no.4, 95, July 2005.

[333] J. N. Mourjopoulos, E. D. Kyriakis-Bitzaros, and C. E. Goutis, "Theory and Real-Time Implementation of Time-Varying Digital Audio Filters," *J. Aud. Eng. Soc.*, **38**, 523 (1990).

[334] U. Zölzer, B. Redmer, and J. Bucholz, "Strategies for Switching Digital Audio Filters," Presented at the 95th Convention of the AES, New York, October 1993, *AES Preprint 3714*.

[335] Y. Ding and D. Rossum, "Filter Morphing of Parametric Equalizers and Shelving Filters for Audio Signal Processing," *J. Aud. Eng. Soc.*, **43**, 821 (1995).

[336] J. Laroche, "Using Resonant Filters for the Synthesis of Time-Varying Sinusoids," Presented at the 105th Convention of the AES, San Francisco, September 1998, *AES Preprint 4782*.

Quantization Effects in Digital Filter Structures

[337] A. H. Gray and J. D. Markel, "Digital Lattice and Ladder Filter Synthesis," *IEEE Trans. Audio Electroacoust.*, **AU-21**, 491 (1973).

[338] A. H. Gray and J. D. Markel, "A Normalized Digital Filter Structure," *IEEE Trans. Acoust., Speech, Signal Process.*, **ASSP-23**, 268 (1975).

[339] A. H. Gray and J. D. Markel, "Roundoff Noise Characteristics of a Class of Orthogonal Polynomial Structures," *IEEE Trans. Acoust., Speech, Signal Process.*, **ASSP-23**, 473 (1975).

[340] J. A. Moorer, "48-Bit Integer Processing Beats 32-Bit Floating Point for Professional Audio Applications," Presented at the 107th Convention of the AES, New York, September 1999, *AES Preprint 5038*.

[341] D. C. Massie, "An Engineering Study of the Four-Multiply Normalized Ladder Filter," *J. Audio Eng. Soc.*, **41**, 564 (1993).

[342] C. T. Mullis and R. A. Roberts, "Synthesis of Minimum Roundoff Noise Fixed-Point Digital Filters," *IEEE Trans. Circuits Syst.*, **CAS-23**, 551 (1976).

[343] S. Y. Hwang, "Minimum Uncorrelated Unit Noise in State-Space Digital Filtering," *IEEE Trans. Acoust., Speech, Signal Process.*, **ASSP-25**, 273 (1977).

[344] C. T. Mullis and R. A. Roberts, *Digital Signal Processing*, Addison-Wesley, Boston, 1987.

[345] C. W. Barnes, "On the Design of Optimal State-Space Realizations of Second-Order Digital Filters," *IEEE Trans. Circuits Syst.*, **CAS-31**, 602 (1984).

[346] B. W. Bomar, "New Second-Order State-Space Structures for Realizing Low Roundoff Noise Digital Filters," *IEEE Trans. Acoust., Speech, Signal Process.*, **ASSP-33**, 106 (1985).

Interpolation, Decimation, Oversampling, and Noise Shaping

[347] R. E. Crochiere and L. R. Rabiner, *Multirate Digital Signal Processing*, Prentice Hall, Englewood Cliffs, NJ, 1983.

[348] R. E. Crochiere and L. R. Rabiner, "Multirate Processing of Digital Signals" in Ref. [13].

[349] P. P. Vaidyanathan, *Multirate Systems and Filter Banks*, Prentice Hall, Englewood-Cliffs, NJ, 1993.

[350] J. C. Candy and G. C. Temes, eds., *Oversampling Delta-Sigma Data Converters*, IEEE Press, Piscataway, NJ, 1992.

[351] J. C. Candy and G. C. Temes, "Oversampling Methods for A/D and D/A Conversion," in Ref. [350].

[352] R. M. Gray, "Oversampled Sigma-Delta Modulation," *IEEE Trans. Commun.*, **COM-35**, 481 (1987). Reprinted in Ref. [350], p. 73.

[353] D. Goedhart, et al., "Digital-to-Analog Conversion in Playing a Compact Disc," *Philips Tech. Rev.*, **40**, 174-179, (1982).

[354] M. W. Hauser, "Principles of Oversampling A/D Conversion," *J. Audio Eng. Soc.*, **39**, 3-26, (1991).

[355] P. J. A. Naus, et al., "A CMOS Stereo 16-bit D/A Converter for Digital Audio," *IEEE J. Solid-State Circuits*, **SC-22**, 390-395, (1987). Reprinted in Ref. [350].

[356] *SONY Semiconductor IC Data Book, A/D, D/A Converters*, 1989 and 1990.

[357] R. Legadec and H. O. Kunz, "A Universal, Digital Sampling Frequency Converter for Digital Audio," *Proc. 1981 IEEE Int. Conf. Acoust., Speech, Signal Process.*, ICASSP-81, Atlanta, GA, p. 595.

[358] T. A. Ramstad, "Digital Methods for Conversion Between Arbitrary Sampling Frequencies," *IEEE Trans. Acoust., Speech, Signal Process.*, **ASSP-32**, 577 (1984).

[359] J. O. Smith and P. Gossett, "A Flexible Sampling-Rate Conversion Method," *Proc. 1984 IEEE Int. Conf. Acoust., Speech, Signal Process.*, ICASSP-84, San Diego, CA, p. 19.4.1. C code is available via `ftp` from `ftp.netcom.com` in directory `pub/thinman/resample.*`.

[360] R. Adams and T. Kwan, "Theory and VLSI Architectures for Asynchronous Sample-Rate Converters," *J. Audio Eng. Soc.*, **41**, 539 (1993).

[361] "SamplePort Stereo Asynchronous Sample Rate Converters, AD1890/AD1891," Data Sheet, Analog Devices, Norwood, MA, 1994.

[362] R. Adams, "Asynchronous Sample-Rate Converters," *Analog Dialogue*, **28**, no. 1, 9 (1994), Analog Devices, Inc., Norwood, MA.

[363] K. Uchimura, et al., “Oversampling A-to-D and D-to-A Converters with Multistage Noise Shaping Modulators,” *IEEE Trans. Acoust., Speech, Signal Process.*, **ASSP-36**, 259 (1988). Reprinted in Ref. [350].

[364] D. B. Ribner, “A Comparison of Modulator Networks for High-Order Oversampled $\Sigma\Delta$ Analog-to-Digital Converters,” *IEEE Trans. Circuits Syst.*, **CAS-38**, 145 (1991).

[365] L. A. Williams, III and B. A. Wooley, “Third-Order Cascaded Sigma-Delta Modulators,” *IEEE Trans. Circuits Syst.*, **CAS-38**, 489 (1991).

[366] R. W. Adams, et al., “Theory and Practical Implementation of a Fifth-Order Sigma-Delta A/D Converter,” *J. Audio Eng. Soc.*, **39**, 515 (1991).

[367] S. Harris, “How to Achieve Optimum Performance from Delta-Sigma A/D and D/A Converters,” *J. Audio Eng. Soc.*, **41**, 782 (1993).

[368] O. Josefsson, “Using Sigma-Delta Converters—Part 1,” *Analog Dialogue*, **28**, no. 1, 26 (1994), Analog Devices, Inc., Norwood, MA, and "Part 2,” ibid., no. 2, 24 (1994).

[369] R. N. J. Veldhuis, M. Breeuwer, and R. G. Van Der Waal, “Subband Coding of Digital Audio Signals,” *Philips J. Res.*, **44**, 329 (1989).

[370] R. N. J. Veldhuis, “Bit Rates in Audio Source Coding,” *IEEE J. Select. Areas Commun.*, **10**, 86 (1992).

[371] G. C. P. Lockoff, “DCC—Digital Compact Cassette,” *IEEE Trans. Consum. Electr.*, **37**, 702 (1991).

[372] G. C. P. Lockoff, “Precision Adaptive Subband Coding (PASC) for the Digital Compact Cassette (DCC),” *IEEE Trans. Consum. Electr.*, **38**, 784 (1992).

[373] A. Hoogendoorn, “Digital Compact Cassette,” *Proc. IEEE*, **82**, 1479 (1994).

[374] T. Yoshida, “The Rewritable MiniDisc System,” *Proc. IEEE*, **82**, 1492 (1994).

STFT and Phase Vocoder

[375] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, 3/e, Pearson, Upper Saddle River, NJ, 2010. Chapter 10 discusses the STFT.

[376] U. Zölzer, ed., *DAFX – Digital Audio Effects*, Wiley, Chichester, England, 2003.
MATLAB and audio files available from:
`http://ant-s4.unibw-hamburg.de/dafx/DAFX_Book_Page/matlab.html`

[377] J. O. Smith III, *Spectral Audio Signal Processing*, W3K Publishing, 2011. Available online from, `https://ccrma.stanford.edu/~jos/sasp/`.

[378] M. S. Puckette, *The Theory and Technique of Electronic Music*, 2006. Available online from, `http://msp.ucsd.edu/techniques.htm`.

[379] W. A. Sethares, *Rhythm and Transforms*, Springer, 2007.
Available from: https//sethares.engr.wisc.edu/RT.html|

[380] J.L. Flanagan and R.M. Golden, “Phase Vocoder,” *Bell Syst. Tech. J.*, **45**45, 1493 (1966).

[381] J. A.. Moorer, “The Use of the Phase Vocoder in Computer Music Applications,” *J. Audio Eng. Soc.*, **26**, 42 (1978).

[382] J. B. Allen and L. R. Rabiner, “A Unified Approach to Short-Time Fourier Analysis and Synthesis,” *Proc. IEEE*, **65**, 1558 (1977).

[383] M. R. Portnoff, “Implementation of the Digital Phase Vocoder Using the Fast Fourier Transform,”*IEEE Trans. Acoust., Speech, Signal Process.*, **24**, 243 (1976). See also, M. R. Portnoff, “Time-Scale Modifications of Speech Based on Short-Time Fourier Analysis,” *ibid.*, **29**, 374 (1981).

[384] M Dolson, “The Phase Vocoder: A Tutorial,” *Computer Music J.*, **10**10, 14 (1986).

[385] J. Laroche and M. Dolson, “Improved Phase Vocoder Time-Scale Modification of Audio,” *IEEE Trans. Speech Audio Proc.*, **7**, 323 (1999). See also, J. Laroche and M. Dolson, “New Phase-Vocoder Techniques Real-Time Pitch-Shifting, Chorusing, Harmonizing, and Other Exotic Audio Modifications,” *J. Audio Eng. Soc.*, **47**, 928 (1999). And also, M. Dolson and J. Laroche, “About This Phasiness Business,” *Proc. Int. Computer Music Conf.*, Ann Arbor, 1997, p. 55. And, also, J. Laroche, “Time and Pitch Scale Modification of Audio Signals,” in *Applications of Digital Signal Processing to Audio and Acoustics*, M. Kahrs and K. Brandenburg, Eds. Kluwer, MA, 1998.

[386] M. S. Puckette, “Phase-Locked Vocoder,” *IEEE ASSP Workshop Appl. Sig. Process. Audio and Acoust.*, 1995. And, also, M. S. Puckette, “On Timbre Stamps and Other Frequency-Domain Filters,” *Proc. ICMA*, 2007, `http://hdl.handle.net/2027/spo.bbp2372.2007.061`

[387] A. De Götzen, Amalia, N. Bernardini, and D. Arfib. “Traditional (?) Implementations of a Phase Vocoder: Tricks of the Trade,” *Proc. COST G-6 Conf. Digital Audio Effects (DAFX-00)*, Verona, Italy. 2000.

[388] D, Barry, D. Dorran, and E. Coyle, “Time and pitch scale modification: a real-time framework and tutorial,” *Proc. 11th Int. Conf. Digital Audio Effets (DAFx-08)*, Espoo, Finland, Sept., 2008.

[389] B. Dias, et al., “Time Stretching & Pitch Shifting with the Web Audio API: Where Are We at?,” *Web Audio Conf. WAC-2016*, April 2016, Atlanta, USA.

[390] J. Drieger and M. Müller, “A Review of Time-Scale Modification of Music Signals,” *Appl. Sci.*, **6**(2), 57 (2016), `https://doi.org/10.3390/app6020057`.

DCT, MDCT, Data Compression

[391] T. Giannakopoulos and A. Pikrakis, *Introduction to Audio Analysis: A MATLAB Approach*, Elsevier, Amsterdam, 2014.

[392] `https://en.wikipedia.org/wiki/Discrete_cosine_transform`

[393] `https://www.mathworks.com/help/images/ref/dct2.html`

[394] N. Ahmed, T. Natarajan, and K. R. Rao, “Discrete Cosine Transform,” *IEEE Trans. Computers*, **C-23**, 90 (1974). See also, N. Ahmed, “How I Came Up With the Discrete Cosine Transform,” *Dig. Sig. Proc.* **1**, 4 (1991).

[395] `https://en.wikipedia.org/wiki/Nasir_Ahmed_(engineer)`

[396] K. R. Rao and P. Yip, *Discrete Cosine Transform: Algorithms, Advantages, Applications*, Academic Press, Boston, 1990. See also, K. R. Rao and P. Yip, (Eds.), *The Transform and Data Compression Handbook*, CRC Press, Boca Raton, 2000. And, also, D. F. Elliott and K. R. Rao, *Fast Transforms: Algorithms, Analyses, Applications*, Academic Press, Orlando, 1982.

[397] Z. Wang and B. R. Hunt, “The Discrete W Transform,” *Appl. Math. Comput.*, **16**, 19 (1985).

[398] G. K. Wallace, “The JPEG still picture compression standard,” *IEEE Trans. Consumer Electr.*, **38**, 18 (1992); and also, *Comm. ACM*, **34** (4), 31 (1991).

[399] https://en.wikipedia.org/wiki/JPEG
https://jpeg.org

[400] D. Austin, "Image Compression: Seeing What's Not There," *AMS Feature Column*, Sept. 2007, http://www.ams.org/publicoutreach/feature-column/fcarc-image-compression.

[401] A. Skodras, C. Christopoulos, and T. Ebrahimi, "The JPEG2000 Still Image Compression Standard," *IEEE Signal Proc. Mag.*, **18** (5), 36, (2001).

[402] J. Lin, "Image Compression – The Mathematics of JPEG 2000," *MSRI Modern Signal Processing*, **46**, 185 (2003).

[403] J. P. Princen and A. B. Bradley, "Analysis/synthesis filter bank based on time domain aliasing cancellation," *IEEE Trans. Acoust., Speech, Signal Proc.*, **ASSP-34**, (5), 1153 (1986).

[404] J. P. Princen, A. W. Johnson, and A. B. Bradley, "Sub-band/transform coding using filter bank designs based on time domain aliasing cancellation," *Proc. IEEE ICASSP 87*, Dallas, TX, April 1987, p. 2161.

[405] Y. Wang, L. Yaroslavsky and M. Vilermo, "On the relationship between MDCT, SDPT and DFT," *WCC 2000 - ICSP 2000. 2000 5th Int. Conf. Signal Proc. Proc.*, 16th World Computer Congress 2000, Beijing, China, 2000, p.44, vol. 1.

[406] M-H Cheng and Y-H Hsu, "Fast IMDCT and MDCT Algorithms—A Matrix Approach," *IEEE Trans. Signal Proc.*, **51**, 221 (2003).

[407] V. Britanak, "A survey of efficient MDCT implementations in MP3 audio coding standard—Retrospective and state-of-the-art," *Signal Processing*, **91**, 624 (2011).

[408] T. Painter and A. Spanias, "Perceptual Coding of Digital Audio," *Proc. IEEE*, **88**, 451 (2000).

[409] M. Bosi and R. E. Goldberg, *Introduction to Digital Audio Coding and Standards*, Kluwer Academic, Boston, 2003.

[410] https://en.wikipedia.org/wiki/Modified_discrete_cosine_transform
https://en.wikipedia.org/wiki/Advanced_Audio_Coding

[411] R. N. Bracewell, "Discrete Hartley Transform," *J. Opt. Soc. Amer.*, **73**, 1832 (1983). See also, R. N. Bracewell, *The Hartley Transform*, Oxford Univ. Press, 1986, and
https://en.wikipedia.org/wiki/Discrete_Hartley_transform

[412] H. S. Malvar, *Signal Processing with Lapped Transforms*, Artech House, Norwood, MA, 1992. See also, H. S. Malvar, "Extended Lapped Transforms: Properties, Applications and Fast Algorithms,,, *IEEE Trans. Signal Process.*, **40**, 2703 (1992).

Wavelets and Applications

[413] I. Daubechies, *Ten Lectures on Wavelets*, SIAM, Philadelphia, PA, 1992.

[414] J. M. Combes, A. Grossmann, and P. Tchamitchian, eds., *Wavelets, Time-Frequency Methods and Phase Space*, Springer-Verlag, Berlin, 1989.

[415] C. K. Chui, *An Introduction to Wavelets*, Academic Press, New York, 1992.

[416] Y. Meyer, *Wavelets, Algorithms and Applications*, SIAM, Philadelphia, 1993.

[417] A. Akansu and R. Haddad, *Multiresolution Signal Decomposition*, Academic Press, New York, 1993.

[418] P. P. Vaidyanathan, *Multirate Systems and Filter Banks*, Prentice Hall, Englewood Cliffs, NJ, 1993.

[419] G. Kaiser, *A Friendly Guide to Wavelets* Birkhäuser, Boston, 1994.

[420] V. Wickerhauser, *Adapted Wavelet Analysis from Theory to Software*, AK Peters, Boston, 1994.

[421] M. Vetterli and J. Kovačević, *Wavelets and Subband Coding*, Prentice Hall, Englewood Cliffs, NJ, 1995.

[422] G. Strang and T. Nguyen, *Wavelets and Filter Banks*, Wellesley-Cambridge Press, Wellesley, MA, 1996.

[423] C. S. Burrus, R. A. Gopinath, and H. Guo, *Introduction to Wavelets and Wavelet Transforms: A Primer*, Prentice Hall, Upper Saddle River, NJ, 1998.

[424] S. Mallat, *A Wavelet Tour of Signal Processing*, Academic, New York, 1998.

[425] A. Antoniadis and G. Oppenheim, eds., *Wavelets and Statistics*, Lecture Notes in Statistics v. 103, Springer-Verlag, New York, 1995.

[426] B. Vidakovic, *Statistical Modeling with Wavelets*, Wiley, New York, 1999.

[427] R. Gençay, F. Selçuk, and B. Whitcher, *An Introduction to Wavelets and Other Filtering Methods in Finance and Economics*, Academic, New York, 2001.

[428] A. Jensen and A. la Cour-Harbo, *Ripples in Mathematics*, Springer, New York, 2001.

[429] S. Jaffard, Y. Meyer, and R. D. Ryan, *Wavelets: Tools for Science and Technology*, SIAM, Philadelphia, 2001.

[430] A. Cohen, *Numerical Analysis of Wavelet Methods*, Elsevier, Amsterdam, 2003.

[431] C. Heil, D. F. Walnut, and I. Daubechies, *Fundamental Papers in Wavelet Theory*, Princeton Univ. Press, Princeton, NJ, 2006.

[432] D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis* Cambridge University Press, Cambridge, 2006.

[433] P. Van Fleet, *Discrete Wavelet Transformations*, Wiley, New York, 2008.

[434] G. P. Nason, *Wavelet Methods in Statistics with R*, Springer, New York, 2008.

[435] G. Strang, "Wavelets and Dilation Equations: A Brief Introduction," *SIAM J. Math. Anal.*, **31**, 614 (1989).

[436] C. Heil and D. Walnut, "Continuous and Discrete Wavelet Transforms," *SIAM Rev.*, **31**, 628 (1989).

[437] L. Cohen, "Time-Frequency Distributions: A Review," *Proc. IEEE,* **77**, 941 (1989).

[438] O. Rioul and M. Vetterli, "Wavelets and Signal Processing," *IEEE SP Mag.*, **8**, no.4, 14, October 1991.

[439] Special issue on Wavelets, *IEEE Trans. Inform. Th.*, **38**, Mar. 1992.

[440] *IEEE Trans. Signal Process.*, Special Issue on Wavelets and Signal Processing, **41**, Dec. 1993.

[441] A. H. Tewfik, M. Kim, and M. Deriche, "Multiscale Signal Processing Techniques: A Review," in N. K. Bose and C. R. Rao, eds., *Handbook of Statistics*, vol. 10, Elsevier, Amsterdam, 1993.

[442] Special Issue on Wavelets, *Proc. IEEE*, **84**, Apr. 1996.

[443] G. Strang, "Wavelet Transforms versus Fourier Transforms," *Bull. (New Series) Am. Math. Soc.*, **28**, 288 (1993).

[444] B. Jawerth and T. Swelden, "An Overview of Wavelet Based Multiresolution Analyses," *SIAM Rev.*, **36**, 377 (1994).

[445] G. Strang, “Wavelets,” *Amer. Scientist*, **82**, 250, May-June 1994.

[446] P. M. Bentley and J. T. E. McDonnell, “Wavelet Transforms: An Introduction,” *Electr. Comm. Eng. J.*, p. 175, Aug. 1994.

[447] A. Graps, “An Introduction to Wavelets,” *IEEE Comput. Sci. Eng. Mag.*, **2**, no. 2, 50, Summer 1995.

[448] J. R. Williams and K. Amaratunga, “Introduction to Wavelets in Engineering,” *Int. J. Numer. Meth. Eng.*, **37**, 2365 (1994).

[449] I. Daubechies, “Where Do Wavelets Come From? A Personal Point of View,” *Proc. IEEE*, **84**, 510 (1996).

[450] W. Sweldens, “Wavelets: What next?,” *Proc. IEEE*, **84**, 680 (1996).

[451] C. Mulcahy, “Plotting and Scheming with Wavelets,” *Math. Mag.*, **69**, 323 (1996).

[452] C. Mulcahy, “Image Compression Using The Haar Wavelet Transform,” *Spelman College Sci. Math. J.*, **1**, 22 (1997).

[453] M . Vetterli, “Wavelets, Approximation, and Compression,” *IEEE SP Mag.*, Sept. 2001, p. 59.

[454] P. P. Vaidyanathan, “Quadrature Mirror Filter Banks, M-band Extensions and Perfect Reconstruction Techniques,” *IEEE ASSP Mag.*, **4**, no. 3, 4, July 1987.

[455] P. P. Vaidyanathan and Z. Doganata, ”The Role of Lossless Systems in Modern Digital Signal Processing: A Tutorial,” *IEEE Trans. Educ.*, **32**, 181 (1989).

[456] P. P. Vaidyanathan, “Multirate Digital Filters, Filter Banks, Polyphase Networks, and Applications: A Tutorial,” *Proc. IEEE*, **78**, 56 (1990).

[457] A. Haar, “Zur Theorie der Orthogonalen Funktionensysteme,” *Math. Annal.*, **69**, 331 (1910). Reprinted in [431].

[458] D. Gabor, “Theory of Communication,” *J. IEE*, **93**, 429 (1946).

[459] D. Esteban and C. Galand, “Application of Quadrature Mirror Filters to Split-Band Voice Coding Schemes,” *Proc. IEEE Int. Conf. Acoust. Speech, Signal Process.,* May 1977, p. 191. Reprinted in [431].

[460] P. J. Burt and E. H. Adelson, “The Laplacian Pyramid as a Compact Image Code,” *IEEE Trans. Commun.*, **31**, 532 (1983). Reprinted in [431].

[461] M. J. T. Smith and T. P. Barnwell III, “A Procedure for Designing Exact Reconstruction Filter Banks for Tree-Structured Sub-Band Coders,” *Proc. IEEE Int. Conf. Acoust., Speech, and Signal Process.*, San Diego, CA, March 1984. Reprinted in [431].

[462] F. Mintzer, “Filters for Distortion-Free Two-Band Multirate Filter Banks,” *IEEE Trans. Acoust., Speech, Signal Process.*, **33**, 626 (1985). Reprinted in [431].

[463] A. Grossmann and J. Morlet, “Decomposition of Hardy Functions into Square Integrable Wavelets of Constant Shape,” *SIAM J. Math. Anal.*, **15**, 723 (1984). Reprinted in [431].

[464] A. Grossmann, J. Morlet, and T. Paul, “Transforms Associated to Square Integrable Group Representations I,” *J. Math. Phys.*, **26**, 2473 (1985). Reprinted in [431].

[465] I. Daubechies, “Orthonormal Bases of Compactly Supported Wavelets,” *Commun. Pure Appl. Math.*, **41** 909 (1988). Reprinted in [431].

[466] G. Battle, “A block spin construction of ondelettes. Part I: Lemarié functions” *Commun. Math. Phys.*, **110**, 601 (1987); and, “Part II: the QFT connection,” *ibid.*, **114**, 93 (1988). Reprinted in [431].

[467] P. G. Lemarié, “Ondelettes á localisation exponentielle,” *J. Math. Pures Appl.*, **67**, 227 (1988).

[468] Y. Meyer, “Wavelets with Compact Support,” Zygmund Lectures, U. Chicago (1987). Reprinted in [431].

[469] S. Mallat, “A Theory for Multiresolution Signal Decomposition: the Wavelet Representation,” *IEEE Trans. Patt. Recogn. Mach. Intell.*, **11**, 674 (1989). Reprinted in [431].

[470] S. Mallat, “Multiresolution Approximations and Wavelet Orthonormal Bases of $L^2(\mathbf{R})$,” *Trans. Amer. Math. Soc.*, **315**, 69 (1989). Reprinted in [431].

[471] A. Cohen, “Ondelettes, Analysis Multirésolutions et Filtres Mirroirs en Quadrature,” *Ann. Inst. H. Poincaré, Anal. Non Linéaire*, **7**, 439 (1990). Reprinted in [431].

[472] A. Grossmann, R. Kronland-Martinet, and J. Morlet, “Reading and Understanding Continuous Wavelet Transforms,” in [414].

[473] M. Holschneider, et al, “A Real Time Algorithm for Signal Analysis with the Help of the Wavelet Transform,” in [414].

[474] I. Daubechies, “The Wavelet Transform, Time-Frequency Localization and Signal Analysis,” *IEEE Trans. Inform. Th.*, **36**, 961 (1990). Reprinted in [431].

[475] M. Holsclmeider, “Wavelet Analysis on the Circle,” *J. Math. Phys.*, **31**, 39 (1990).

[476] G. Beylkin, R. Coifman, and V. Rokhlin, “Fast Wavelet Transforms and Numerical Algorithms I,,, *Commun. Pure Appl. Math.*, **44**, 141 (1991). Reprinted in [431].

[477] W. Lawton, “Tight Frames of Compactly Supported Affine Wavelets,, *J. Math. Phys.*, **31**, 1898 (1990). Reprinted in [431].

[478] W. Lawton, “Necessary and Sufficient Conditions for Constructing Orthonormal Wavelet Bases,” *J. Math. Phys.*, **32**, 57 (1991).

[479] W. Lawton, “Multiresolution Properties of the Wavelet Galerkin Operator,” *J. Math. Phys.*, **32**, 1440 (1991).

[480] I. Daubechies and J. Lagarias, “Two-Scale Difference Equations I. Existence and Global Regularity of Solutions,” *SIAM J. Math. Anal.*, **22**, 1388 (1991); and, “II. Local Regularity, Infinite Products of Matrices and Fractals,” *ibid.*, **24**, 1031 (1992).

[481] A. Cohen, I. Daubechies, and J.-C. Feauveau, “Biorthogonal Bases of Compactly Supported Wavelets,” *Commun. Pure Appl. Math.*, **45**, 485 (1992).

[482] O. Rioul and P. Duhamel, “Fast Algorithms for Discrete and Continuous Wavelet Transforms,” *IEEE Trans. Inform. Th.*, **38**, 569 (1992).

[483] M. Vetterli and C. Herley, “Wavelets and Filter Banks: Theory and Design,” *IEEE Trans. Signal Process.*, **40**, 2207 (1992).

[484] G. G. Walter, “A Sampling Theorem for Wavelet Subspaces,” *IEEE Trans. Inform. Th.*, **38**, 881 (1992).

[485] N. H. Getz, “A Perfectly Invertible, Fast, and Complete Wavelet Transform for Finite Length Sequences: The Discrete Periodic Wavelet Transform,” *SPIE Mathematical Imaging*, vol, 2034, p. 332, (1993).

[486] L. Cohen, “The Scale Representation,” *IEEE Trans. Signal Process.*, **41**, 3275 (1993).

[487] I. Daubechies, “Orthonormal Bases of Compactly Supported Wavelets II, Variations on a Theme,” *SIAM J. Math. Anal.*, **24**, 499 (1993).

[488] O. Rioul, “A Discrete-Time Multiresolution Theory,” *IEEE Trans. Signal Process.*, **41**, 2591 (1993).

[489] X. Xia and Z. Zhang, “On Sampling Theorem, Wavelets, and Wavelet Transforms, *IEEE Trans. Signal Process.*, **41**, 3524 (1993).

[490] W. Sweldens, “The Lifting Scheme: A Custom-Design Construction of Biorthogonal Wavelets,” *Appl. Comput. Harmon. Anal.*,**3**, 186 (1996).

[491] W. Sweldens, “The Lifting Scheme: A Construction of Second Generation Wavelets,” *SIAM J. Math. Anal.*, **29**, 511 (1996).

[492] G. Strang, “Eigenvalues of $(\downarrow 2)H$ and Convergence of the Cascade Algorithm,” *IEEE Trans. Signal Process.*, **44**, 233 (1996).

[493] S. H. Maes, “Fast Quasi-Continuous Wavelet Algorithms for Analysis and Synthesis of One-Dimensional Signals,” *SIAM J. Appl. Math.*, **57**, 1763 (1997).

[494] I. Daubechies and W. Sweldens, “Factoring Wavelet Transforms into Lifting Steps,” *J. Fourier Anal. Appl.*, **4**, 247 (1998).

[495] M. Unser and T. Blu, “Wavelet Theory Demystified,” *IEEE Trans. Signal Process.*, **51**, 470 (2003).

[496] P. Dutilleux, “An Implementation of the Algorithm à Trous to Compute the Wavelet Transform,” in [414].

[497] S. Mallat, “Zero-Crossings of a Wavelet Transform,” *IEEE Trans. Inform. Th.*, **37**, 1019 (1991).

[498] G. Beylkin, “On the Representation of Operators in Bases of Compactly Supported Wavelets,” *SIAM J. Numer. Anal.*, **29**, 1716 (1992).

[499] M. J. Shensa, “The Discrete Wavelet Transform: Wedding the á Trous and Mallat Algorithms,” *IEEE Trans. Signal Process.*, **40**, 2464 (1992).

[500] G. P. Nason and B. W. Silverman, “The Discrete Wavelet Transform in S,” *J. Comput. Graph. Statist.*, **3**, 163 (1994).

[501] G. P. Nason and B. W. Silverman, “The Stationary Wavelet Transform and Some Statistical Applications,” in [425].

[502] R. R. Coifman and D. L. Donoho, “Translation-Invariant Denoising,” in [425].

[503] J. C. Pesquet, H. Krim, and H. Carfantan, “Time-Invariant Orthonormal Wavelet Representations,” *IEEE Trans. Signal Process.*, **44**, 1964 (1996).

[504] J. Liang and T. W. Parks, “A Translation-Invariant Wavelet Representation Algorithm with Applications,” *IEEE Trans. Signal Process.*, **44**, 225 (1996).

[505] M. Lang, et al., “Noise Reduction Using An Undecimated Discrete Wavelet Transform,” *IEEE Signal Process. Lett.*, **3**, 10 (1996).

[506] H. Sari-Sarraf and D. Brzakovic, “A Shift-Invariant Discrete Wavelet Transform,” *IEEE Trans. Signal Process.*, **45**, 2621 (1997).

[507] J. E. Fowler, “The Redundant Discrete Wavelet Transform and Additive Noise, *IEEE Signal Process. Lett.*, **12**, 629 (2005).

[508] A. F. Abdelnour and I. W. Selesnick, “Symmetric Nearly Shift-Invariant Tight Frame Wavelets,” *IEEE Trans. Signal Process.*, **53**, 231 (2005).

[509] J.-L. Starck, J. Fadili, and F. Murtagh, “The Undecimated Wavelet Decomposition and its Reconstruction,” *IEEE Trans. Imag. Process.*, **16**, 297 (2007).

[510] J. D. Johnston, “Transform Coding of Audio Signals Using Perceptual Noise Criteria,” *IEEE J. Selected Areas Commun.*, **6**, 314 (1988).

[511] D. J. LeGall, H. Gaggioni, and C. T. Chen, “Transmission of HDTV Signals Under 140 Mbits/s Using a Subband Decomposition and Discrete Cosine Transform Coding,” in L. Chiariglione, ed., *Signal Processing of HDTV*, Elsevier, Amsterdam, 1988.

[512] JPEG Technical Specification: Revision (DRAFT), Joint Photographic Experts Group, ISO/IEC JTC1/SC2/WG8, CCITT SGVIII, August 1990.

[513] G. K. Wallace, “The JPEG Still Picture Compression Standard,” *Commun. ACM*, **34**, 30 (1991).

[514] D. LeGall, “MPEG: A Video Compression Standard for Multimedia Applications,” *Commun. ACM*, **34** 46 (1991).

[515] N. S. Jayant, “Signal Compression: Technology Targets and Research Directions,” *IEEE J. Sel. Areas Commun.*, **10**, 796 (1992).

[516] M. Antonini, et al., “Image Coding Using Wavelet Transform,” *IEEE Trans. Im. Process.*, **1**, 205 (1992).

[517] R. DeVore, B. Jawerth, and V. Popov, “Compression of Wavelet Decompositions,” *Amer. J. Math.*, **114**, 737 (1992). Reprinted in [431].

[518] R. DeVore, B. Jawerth, and B. Lucier, “Image Compression Through Wavelet Transform Coding,” *IEEE Trans. Inform. Th.*, **38**, 719 (1992).

[519] M. Farge, “Wavelet Transforms and their Applications to Turbulence,” *Ann. Rev. Fluid Mech.*, **24**, 395 (1992).

[520] J. N. Bradley, C. M. Brislawn, and T. Hopper, “The FBI Wavelet/Scalar Quantization Standard for Grey-Scale Fingerprint Image Compression,” *Proc. SPIE*, **1961**, 293 (1993).

[521] C. M. Brislawn, “Fingerprints Go Digital,” *Notices AMS*, **42** no. 11, 1278 (1995).

[522] C. M. Brislawn, et al., “FBI Compression Standard for Digitized Fingerprint Images,” *Proc. SPIE*, **2847**, 344 (1996).

[523] E. J. Stollnitz, T. D. DeRose, and D. H. Salesin, “Wavelets for Computer Graphics: A Primer, part 1,” *IEEE Comput. Graph. Appl.*, **15**, 76 (1995).

[524] G. Pan, “Orthogonal Wavelets with Applications in Electromagnetics,” *IEEE Trans. Magn.*, **32**, 975 (1996).

[525] R. Zaciu, et al., “Image Compression Using an Overcomplete Discrete Wavelet Transform,” *IEEE Trans. Consum. Electron.*, **42**, 500 (1996).

[526] N. Erdol and F. Basbug, “Wavelet Transform Based Adaptive Filters: Analvsis and New Results,” *IEEE Trans. Signal Process.*, **44**, 2163 (1996).

[527] A. Bijaoui, et al., “Wavelets and the Study of the Distant Universe,” *Proc. IEEE*, **84**, 670 (1996).

[528] M. Unser and A. Aldroubi, “A Review of Wavelets in Biomedical Applications,” *Proc. IEEE*, **84**, 626 (1996).

[529] B. K. Alsberg, A. M. Woodward, and D. B. Kell, “An Introduction to Wavelet Transforms for Chemometricians: A Time-Frequency Approach,” *Chemometr. Intell. Lab. Syst.*, **37**, 215 (1997).

[530] B. K. Alsberg, et al., “Wavelet Denoising of Infrared Spectra,” *Analyst*, **122**, 645 (1997).

[531] B. Walczak and D. L. Massart, “Wavelets – Something for Analytical Chemistry?,” *Trends Anal. Cem.*, **15**, 451 (1997).

[532] A. Chambolle, et al., “Nonlinear Wavelet Image Processing: Variational Problems, Compression and Noise Removal Through Wavelet Shrinkage,” *IEEE Trans. Imag. Process.*, **7**, 319 (1998).

[533] A. K-M. Leung, F-T. Chau, and J-B. Gao, “A Review on Applications of Wavelet Transform Techniques in Chemical Analysis: 1989–1997,” *Chemometr. Intell. Lab. Syst.*, **43**, 165 (1998).

[534] G. Strang, “The Discrete Cosine Transform,” *SIAM Rev.*, **41**, 135 (1999).

[535] C. Torrence and G. P. Compo, “A Practical Guide to Wavelet Analysis,” *Bull. Amer. Meteor. Soc.*, **79**, 621 (1998).

[536] J. B. Ramsey, “The Contribution of Wavelets to the Analysis of Economic and Financial Data,” *Phil. Trans. Roy. Soc. Lond. A*, **357**, 2593 (1999).

[537] M. W. Marcellin, et al., “An Overview of JPEG2000,” *Proc. Data Compression Conf.*, Snowbird, Utah, March 2000, p. 523.

[538] ISO/IEC JTC1/SC29/WG1/N1646R, JPEG 2000 Part I Final Committee Draft Version 1.0, Mar. 2000, available from `http://www.jpeg.org/public/fcd15444-1.pdf`.

[539] C. Christopoulos, A. Skodras, and T. Ebrahimi, “The JPEG2000 Still Image Coding System: An Overview,”, *IEEE Trans. Consum. Electron.*, **46**, 1103 (2000).

[540] C-H. Lee, Y-J Wang, and W-L Huang, “A Literature Survey of Wavelets in Power Engineering Applications,” *Proc. Natl. Sci. Counc. ROC(A)*, **24**, 249 (2000).

[541] C.H. Kim and R. Aggarwal, “Wavelet Transforms in Power Systems, Part 1: General Introduction to the Wavelet Transforms,” *Power Eng. J.*, **14**, 81 (2000); and “Part 2: Examples of Application to Actual Power System Transients,”, *ibid.*, **15**, 193 (2000).

[542] S. G. Chang, B. Yu, and M. Vetterli, “Adaptive Wavelet Thresholding for Image Denoising and Compression,” *IEEE Trans. Imag. Process.*, **9**, 1532 (2000).

[543] D. B. H. Tay, “Rationalizing the Coefficients of Popular Biorthogonal Wavelet Filters,” *IEEE Trans. Circ. Syst. Video Tech.*, **10**, 998 (2000).

[544] B. E. Usevitch, “A Tutorial on Modern Lossy Wavelet Image Compression: Foundations of JPEG 2000,” *IEEE SP Mag.*, Sept. 2001, p. 22.

[545] M. D. Adams, “The JPEG-2000 Still Image Compression Standard,” ISO/IEC JTC 1/SC 29/WG1 N 2412, Sept. 2001, Available from `http://www.ece.ubc.ca/~mdadams`.

[546] J-L, Starck and F. Murtagh, “Astronomical Image and Signal Processing Looking at Noise, Information, and Scale,” *IEEE SP Mag.*, p.30, Mar. 2001.

[547] J. B. Ramsey, “Wavelets in Economics and Finance: Past and Future,” *Stud. Nonlin. Dynam. Econometr.*, 2002.

[548] M. Unser and T. Blu, “Mathematical Properties of the JPEG2000 Wavelet Filters,” *IEEE Trans. Imag. Process.*, **12**, 1080 (2003).

[549] F. Truchetet and O. Laligant, “Wavelets in Industrial Applications: A Review,” *Proc. SPIE*, **5607**, 1 (2004).

[550] M. J. Fadili and E. T. Bullmore, “A Comparative Evaluation of Wavelet-Based Methods for Hypothesis Testing of Brain Activation Maps,” *NeuroImage*, **23**, 1112 (2004).

[551] M. N. O. Sadiku, C. M. Akujuobi, and R. C. Garcia, “An Introduction to Wavelets in Electromagnetics,” *IEEE Microwave Mag.*, **6**, no.5, p.63, June 2005.

[552] P. S. Addison, “Wavelet Transforms and the ECG: A Review,” *Physiol. Meas.*, **26**, R155 (2005).

[553] M. Kaboudan, “Computational Forecasting of Wavelet-converted Monthly Sunspot Numbers,” *J. Appl. Statist.*, **33**, 925 (2006).

[554] P. Liò, “Wavelets in Bioinformatics and Computational Biology: State of Art and Perspectives,” *Bionform. Rev.*, **21**, 207 (2007).

[555] P. M. Crowley, “A Guide to Wavelets for Economists,” *J. Econ. Surveys*, **21**, 207 (2007).

[556] J. E. Fowler and B. Pesquet-Popescu, ”An Overview on Wavelets in Source Coding, Communications, and Networks,” *EURASIP J. Imag. Vid. Process.*, vol. 2007, Article ID 60539, (2007).

[557] I. Balasingham and T. A. Ramstad, J. E. Fowler and B. Pesquet-Popescu, ”Are the Wavelet Transforms the Best Filter Banks for Image Compression?” *EURASIP J. Imag. Vid. Process.*, vol. 2008, Article ID 287197, (2008).

[558] F. Truchetet and O. Laligant, “Review of Industrial Applications of Wavelet and Multiresolution-Based Signal and Image Processing,” *J. Electron. Imag.*, **17**, 031102 (2008)

[559] H. Hashish, S. H. Behiry, and N.A. El-Shamy, “Numerical Integration Using Wavelets,” *Appl. Math. Comput.* **211**, 480 (2009).

[560] B. Mandelbrot and J. W. Van Ness, “Fractional Brownian Motions: Fractional Noises and Applications,” *SIAM Rev.*, **10**, 422 (1968).

[561] S. Granger and R. Joyeux, “An Introduction to Long-Memory Time Series Models and Fractional Differencing,” *J. Time Ser. Anal.*, **1**, 15 (1980).

[562] J. R. M. Hosking, “Fractional Differencing,” *Biometrika*, **68**, 165 (1981).

[563] G. Wornell, “A Karhunen-Loève Like Expansion for $1/f$ Processes via Wavelets,” *IEEE Trans. Inform. Th.*, **36**, 859 (1990).

[564] G. Wornell and A. V. Oppenheim, “Wavelet-Based Representations for a Class of Self-Similar Signals with Application to Fractal Modulation,” *IEEE Trans. Inform. Th.*, **38**, 785 (1992).

[565] P. Flandrin, “Wavelet Analysis and Synthesis of Fractional Brownian Motion,” *IEEE Trans. Inform. Th.*, **38**, 910 (1992).

[566] P. Abry, et al., “The Multiscale Nature of Network Traffic,” *IEEE SP Mag.*, **19**, no. 3, 28, May 2002.

[567] R. A. DeVore and B. J. Lucier, “Fast Wavelet Techniques for Near-Optimal Image Processing,” *MILCOM ’92, IEEE Mil. Commun. Conf.*, p.1129, (1992).

[568] D. Donoho, “Unconditional Bases are Optimal Bases for Data Compression and Statistical Estimation,” *Appl. Computat. Harmon. Anal.*, **1**, 100 (1993).

[569] D. L. Donoho and I. M. Johnstone, “Ideal Spatial Adaptation by Wavelet Shrinkage,” *Biometrika*, **81**, 425 (1994).

[570] , D. L. Donoho, “Denoising by Soft Thresholding,” *IEEE Trans. Inform. Th.*, **41**, 613 (1995).

[571] , D. L. Donoho, et al., “Wavelet Shrinkage: Asymptopia?,” *J. Roy. Statist. Soc., Ser. B*, **57**, 301 (1995).

[572] D. L. Donoho and I. M. Johnstone, “Adapting to Unknown Smoothness via Wavelet Shrinkage,” *J. Amer. Statist. Assoc.*, **90**, 1200 (1995). Reprinted in [431].

[573] A. Antoniadis, “Smoothing Noisy Data with Tapered Coiflets Series,” *Scand. J. Statist.*, **23**, 313 (1996).

[574] F. Abramovich and B. W. Silverman, “Wavelet Decomposition Approaches to Statistical Inverse Problems,” *Biometrika*, **85**, 115 (1998).

[575] D. L. Donoho, et al., “Data Compression and Harmonic Analysis,” *IEEE Trans. Inform. Th.*, **44**, 2435 (1998).

[576] F. Abramovich, T. Sapatinas, and B. W. Silverman, “Wavelet Thresholding via Bayesian Approach,” *J. Roy. Statist. Soc., Ser. B.*, **60**, 725 (1998).

[577] B. W. Silverman, “Wavelets in Statistics: Beyond the Standard Assumptions,” *Phil. Trans. Roy. Soc. Lond. A*, **357**, 2459 (1999).

[578] G. P. Nason and R. von Sachs, “Wavelets in Time-Series Analysis,” *Phil. Trans. Roy. Soc. Lond. A*, **357**, 2511 (1999).

[579] F. Abramovich, T. C. Baily, and T. Sapatinas, “Wavelet Analysis and Its Statistical Applications,” *Statistician*, **49**, 1 (2000).

[580] A. Antoniadis, J. Bigot, and T. Sapatinas, “Wavelet Estimators in Nonparametric Regression: A Comparative Simulation Study,” *J. Statist. Softw.*, **6**, 1 (2001).

[581] A. Antoniadis and J. Fan, “Regularization of Wavelet Approximations,” *J. Amer. Statist. Assoc.*, **96**, 939 (2001).

[582] `http://www.cmap.polytechnique.fr/~bacry/LastWave`, LastWave, Emmanuel Bacry.

[583] `http://www.cs.kuleuven.ac.be/~wavelets`, Uytterhoeven, et al., C++ implementation.

[584] `http://www-stat.stanford.edu/~wavelab/` Wavelab.

[585] `http://www.dsp.rice.edu/software/RWT` Rice Wavelet Toolbox.

[586] `http://paos.colorado.edu/research/wavelets`, Torrance and Compo.

[587] `http://www.curvelet.org/`, Curvelets.

[588] `http://www.stats.bris.ac.uk/~wavethresh`, Wavethresh in R.

[589] `http://taco.poly.edu/WaveletSoftware/`, S. Cai and K. Li.

[590] `http://www2.isye.gatech.edu/~brani/wavelet.html`, B. Vidakovic.

[591] `http://www-lmc.imag.fr/SMS/software/GaussianWaveDen/index.html`, A. Antoniadis, J. Bigot, and J. Sapatinas.

[592] `http://www.atmos.washington.edu/~wmtsa/`, Percival and Walden, WMTSA toolbox.

[593] `http://cas.ensmp.fr/~chaplais/UviWave/About_UviWave.html`, Uvi-Wave.

[594] `http://inversioninc.com/wavelet.html`, N. H. Getz, see Ref. [485].

[595] `http://www.math.rutgers.edu/~ojanen/wavekit/`, H. Ojanen, Wavekit.

[596] `http://cam.mathlab.stthomas.edu/wavelets/packages.php`, P. Van Fleet, see [433].

Control Systems

[597] F. T. Ulaby and A. E. Yagle, *Signals and Systems: Theory and Applications*, University of Michigan Free Textbook Initiative, available from: `https://ss2.eecs.umich.edu/`

[598] Z. Gajic, *Linear Dynamic Systems and Signals*, Prentice Hall, 2003.

[599] D. Tilbury, et al., *Control Systems Tutorials for MATLAB & Simulink*, 2011
`https://ctms.engin.umich.edu/CTMS/index.php?aux=Home`

[600] L. Moysis, et al., *An Introduction to Control Theory Applications with Matlab*, 2015, online book available freely from:
`http://ikee.lib.auth.gr/record/270899/files/IntroductionMatlab-2.pdf`

[601] D. I. Wilson, *Advanced Control using MATLAB*, Auckland University of Technology, 2015.

[602] K. J. Åström and P.R. Kumar, “Control: A perspective,”, *Automatica*, **50**, 3 (2014).

[603] J. Bechhoefer, “Feedback for Physicists: a Tutorial Essay on Control,” *Rev. Mod. Phys.*, **77**, 783 (2005).

[604] R. Dorf and R. Bishop, *Modern Control Systems*, 13/e, Pearson, 2017.

[605] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*, 8/e, Pearson, 2019.

[606] J. Golten and A. Verwer, *Control System Design and Simulation*, McGraw-Hill Europe, 1991.

[607] C. L. Phillips and H. T. Nagle, *Digital Control System Analysis and Design*, Prwentica Hall, 1984.

[608] P. B. Deshpande and R. H. Ash, *Elements of Computer Process Control*, Prentice Hall, 1981.

[609] G. F. Franklin, J. D. Powell, *Digital Control of Dynamic Systems*, Addison-Wesley, 1980.

[610] O. Boubaker, “The inverted Pendulum: A fundamental Benchmark in Control Theory and Robotics,” *IEEE Int. Conf. Education and e-Learning Innovations (ICEELI)*, Sousse, Tunisia, July, 2012; see also, O. Boubaker, “The Inverted Pendulum Benchmark in Nonlinear Control Theory: A Survey,” *Int. J. Adv. Robotic Syst.*, **10**, 233 (2013).

[611] I. Kafetzis and L. Moysis, “Inverted Pendulum: A system with innumerable applications,” available from:
`https://ikee.lib.auth.gr/record/288541/files/Inverted pendulum-A system with innumerable applications.pdf`

[612] M. Hehn and R. D’Andrea, “A Flying Inverted Pendulum,” *IEEE Int. Conf. Robotics and Automation*, p. 763, May 2011, Shanghai.

[613] F. Grasser, et al., “JOE: a mobile, inverted pendulum,” *IEEE Trans. Industr. Electr.*, **49**, 107 (2002).

[614] S. W. Nawawi, et al., “Real-time control system for a two-wheeled inverted pendulum mobile robot,” (2010), available from:
`https://www.intechopen.com/download/pdf/12354`

[615] T-P Azevedo Perdicoúlis ans P. Lopes dos Santos, “The secrets of Segway revealed to students: revisiting the inverted pendulum,” *13th APCA Int. Conf. Automatic Control and Soft Computing (CONTROLO)*, p. 43, June 2018, Ponta Delgada, Azores, Portugal.

[616] N. D. Anh, et al., “Vibration control of an inverted pendulum type structure by passive mass-spring-pendulum dynamic vibration absorber,” *J. Sound Vibr.*, **307**, 187 (2007).

[617] D. A. Winter, “Human balance and posture control during standing and walking,” *Gait & Posture*, **3**, no.4, 193 (1995).

[618] P. Morasso, A. Cherif, and J. Zenzeri, “Quiet standing: The Single Inverted Pendulum model is not so bad after all,” `https://doi.org/10.1371/journal.pone.0213870`

[619] A. D. Kuo, “The six determinants of gait and the inverted pendulum analogy: A dynamic walking perspective,” *Human Movement Science*, **26**, 617 (2007).

Local Polynomial Smoothing Filters

[620] G. V. Schiaparelli, “Sul Modo Di Ricavare La Vera Espressione Delle Leggi Della Natura Dalle Curve Empiriche,” *Effemeridi Astronomiche di Milano per l’anno 1866*, p.3–56, reprinted in *Le Opere di G. V. Schiaparelli*, vol.8, Ulrico Hoepli Publisher, Milano, 1930, and Johnson Reprint Corp., New York.

[621] A. Lees, “Interpolation and Extrapolation of Sampled Data,” *IEEE Trans. Inform. Th.*, **2**, 12 (1956).

[622] K. R. Johnson, “Optimum, Linear, Discrete Filtering of Signals Containing a Nonrandom Component,” *IEEE Trans. Inform. Th.*, **2**, 49 (1956).

[623] M. Blum, “An Extension of the Minimum Mean Square Prediction Theory for Sampled Input Signals,” *IEEE Trans. Inform. Th.*, **IT-2**, 176 (1956).

[624] M. Blum, “On the Mean Square Noise Power of an Optimum Linear Discrete Filter Operating on Polynomial plus White Noise Input,” *IEEE Trans. Inform. Th.*, **IT-3**, 225 (1957).

[625] J. D. Musa, “Discrete Smoothing Filters for Correlated Noise,” *Bell Syst. Tech. J.*, **42**, 2121 (1963).

[626] A. Savitzky and M Golay, “Smoothing and Differentiation of Data by Simplified Least Squares Procedures,” *Anal. Chem.*. **36**, 1627 (1964).

[627] M. U. A. Bromba and H. Ziegler, “Efficient Computation of Polynomial Smoothing Digital Filters,” *Anal. Chem.*, **51**, 1760 (1979).

[628] M. U. A. Bromba and H. Ziegler, “Application Hints for Savitzky-Golay Digital Smoothing Filters,” *Anal. Chem.*, **53**, 1583 (1981).

[629] T. H. Edwards and P. D. Wilson, “Digital Least Squares Smoothing of Spectra,” *Appl. Spectrosc.*, **28**, 541 (1974).

[630] T. H. Edwards and P. D. Wilson, “Sampling and Smoothing of Spectra,” *Appl. Spectrosc. Rev.*, **12**, 1 (1976).

[631] C. G. Enke and T. A. Nieman, “Signal-to-Noise Ratio Enhancement by Least-Squares Polynomial Smoothing,” *Anal. Chem.*, **48**, 705A (1976).

[632] H. H. Madden, “Comments on the Savitzky-Golay Convolution Method for Least-Squares Fit Smoothing and Differentiation of Digital Data,” *Anal. Chem.*, **50**, 1383 (1978).

[633] R. A. Leach, C. A. Carter, and J. M. Harris, “Least-Squares Polynomial Filters for Initial Point and Slope Estimation,” *Anal. Chem.*, **56**, 2304 (1984).

[634] P. A. Baedecker, “Comments on Least-Squares Polynomial Filters for Initial Point and Slope Estimation,” *Anal. Chem.*, **57**, 1477 (1985).

[635] J. Steinier, Y. Termonia, and J. Deltour, “Comments on Smoothing and Differentiation of Data by Simplified Least Squares Procedures,” *Anal. Chem.*. **44**, 1627 (1972).

[636] H. Ziegler, “Properties of Digital Smoothing Polynomial (DISPO) Filters,” *Appl. Spectrosc.*, **35**, 88 (1981).

[637] G. R. Phillips and J. M. Harris, “Polynomial Filters for Data Sets with Outlying or Missing Observations: Application to Charged-Coupled-Device- Detected Raman Spectra Contaminated by Cosmic Rays,” *Anal. Chem.*, **62**, 2351 (1990).

[638] M. Kendall, *Time-Series*, 2nd ed., Hafner Press, Macmillan, New York, 1976.

[639] M. Kendall and A. Stuart, *Advanced Theory of Statistics*, vol. 3, 2nd ed., Charles Griffin & Co., London, 1968.

[640] R. W. Hamming, *Digital Filters*, 2nd ed., Prentice Hall, Upper Saddle River, NJ, 1983.

[641] C. S. Williams, *Designing Digital Filters*, Prentice Hall, Upper Saddle River, NJ, 1986.

[642] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C*, 2nd ed., Cambridge Univ. Press, New York, 1992.

[643] J. F. Kaiser and W. A. Reed, “Data Smoothing Using Lowpass Digital Filters,” *Rev. Sci. Instrum.*, **48**, 1447 (1977).

[644] J. F. Kaiser and R. W. Hamming, “Sharpening the Response of a Symmetric Nonrecursive Filter by Multiple Use of the Same Filter,” *IEEE Trans. Acoust., Speech, Signal Process.*, **ASSP-25**, 415 (1975).

[645] J. Luo, et al., “Properties of Savitzky-Golay Digital Differentiators,” *Dig. Sig. Process.*, **15**, 122 (2005).

[646] J. Luo, “Savitzky-Golay Smoothing and Differentiation Filter for Even Number Data,” *Signal Process.*, **85**, 1429 (2005).

[647] S. Hargittai, “Savitzky-Golay Least-Squares Polynomial Filters in ECG Signal Processing,” *Computers Cardiol.*, **32**, 763 (2005).

[648] T. C. Mills, “A Note on Trend Decomposition: The ‘Classical’ Approach Revisited with an Application to Surface Temperature Trends,” *J. Appl. Statist.*, **34**, 963 (2007).

Henderson Filters

[649] E. L. De Forest, “On Some Methods of Interpolation Applicable to the Graduation of Irregular Series, such as Tables of Mortality,” *Ann. Rep. Board of Regents of Smithsonian Institution*, 1871, p.275. Also, *ibid.*, 1873, p.319.

[650] E. L. De Forest, “On Adjustment Formulas,” *The Analyst* (De Moines, Iowa), **4**, 79 (1877), and *ibid.*, p. 107.

[651] E. L. De Forest, “On the Limit of Repeated Adjustments,” *The Analyst* (De Moines, Iowa), **5**, 129 (1878), and *ibid.*, p. 65.

[652] H. H. Wolfenden, “Development of Formulae for Graduation by Linear Compounding, With Special Reference to the Work of Erastus L. De Forest,” *Trans. Actuarial Soc. Am.*, **26**, 81 (1925).

[653] F. R. Macauley, *The Smoothing of Time Series*, Nat. Bureau Econ. Res., NY, 1931.

[654] M. D. Miller, *Elements of Graduation*, Actuarial Soc. Am. and Am. Inst. Actuaries, 1946.

[655] C. A. Spoerl, “Actuarial Science—A Survey of Theoretical Development,” *J. Amer. Statist. Assoc.*, **46**, 334 (1951).

[656] S. M. Stigler, “Mathematical Statistics in the Early States,” *Ann. Statist.*, **6**, 239 (1978).

[657] H. L. Seal, “The Fitting of a Mathematical Graduation Formula: A Historical Review with Illustrations,” *Blätter. Deutsche Gesellschaft für Versicherungsmathematik*, **14**, 237 (1980).

[658] H. L. Seal, “Graduation by Piecewise Cubic Polynomials: A Historical Review,” *Blätter. Deutsche Gesellschaft für Versicherungsmathematik*, **15**, 89 (1981).

[659] J. M. Hoem, “The Reticent Trio: Some Little-Known Early Discoveries in Life Insurance Mathematics by L. H. Opperman, T. N. Thiele, and J. P. Gram,” *Int. Statist. Rev.*, **51**, 213 (1983).

[660] W. F. Sheppard, “Reduction of Errors by Means of Negligible Differences,” *Proc. Fifth Int. Congress of Mathematicians*, **2**, 348 (1912), Cambridge.

[661] W. F. Sheppard, “Fitting Polynomials by Method of Least Squares,” *Proc. London Math. Soc., Ser. 2*, **13**, 97 (1913).

[662] W. F. Sheppard, “Graduation by Reduction of Mean Square Error,” *J. Inst. Actuaries*, **48**, 171 (1914), see also, *ibid.*, **48**, 412 (1914), and **49**, 148 (1915).

[663] R. Henderson, “Note on Graduation by Adjusted Average,” *Trans. Actuarial Soc. Am.*, **18**, 43 (1916).

[664] H. Vaughan, “Further Enquiries into the Summation Method of Graduation,” *J. Inst. Actuaries*, **66**, 463 (1935).

[665] K. Weichselberger, “Über eine Theorie der gleitenden Durchschnitte und verschiedene Anwendungen dieser Theorie,” *Metrica*, **8**, 185 (1964).

[666] I. J. Schoenberg, “Some Analytical Aspects of the Problem of Smoothing,” in *Studies and Essays Presented to R. Courant on his 60th Birthday*, Interscience, NY, 1948.

[667] I. J. Schoenberg, “On Smoothing Operations and Their Generating Functions,” *Bull. Am. Math. Soc.*, **59**, 199 (1953).

[668] T. N. E. Greville, “On Stability of Linear Smoothing Formulas,” *SIAM J. Numer. Anal.*, **3**, 157 (1966).

[669] W. F. Trench, “Stability of a Class of Discrete Minimum Variance Smoothing Formulas,” *SIAM J. Numer. Anal.*, **9**, 307 (1972).

[670] T. N. E. Greville, “On a Problem of E. L. De Forest in Iterated Smoothing,” *SIAM J. Math. Anal.*, **5**, 376 (1974).

[671] O. Borgan, “On the Theory of Moving Average Graduation,” *Scand. Actuarial J.*, p. 83, (1979).

[672] P. B. Kenny and J. Durbin, “Local Trend Estimation and Seasonal Adjustment of Economic and Social Time Series,” *J. Roy. Statist. Soc., Ser. A*, **145**, 1 (1982).

[673] D. London, *Graduation: The Revision of Estimates*, ACTEX publications, Winsted, CT, 1985.

[674] E. S. W. Shiu, “Minimum-R_z Moving-Average Formulas,” *Trans. Soc. Actuaries*, **36**, 489 (1984).

[675] E. S. W. Shiu, “A Survey of Graduation Theory,” in H. H. Panjer, ed., *Actuarial Mathematics*, Proc. Symp. Appl. Math, vol.35, 1986.

[676] E. S. W. Shiu, “Algorithms for MWA Graduation Formulas,” *Actuarial Res. Clearing House*, **2**, 107 (1988).

[677] W. D. Hoskins and P. J. Ponzo, “Some Properties of a Class of Band Matrices,” *Math. Comp.*, **26**, 393 (1972).

[678] A. Eisinberg, P. Pugliese, and N. Salerno, “Vandermonde Matrices on Integer Nodes: The Rectangular Case,” *Numer. Math.*, **87**, 663 (2001).

[679] M. Dow, “Explicit Inverse of Toeplitz and Associated Matrices,” *ANZIAM J.*, **44** (E), 185 (2003).

[680] A. Grey and P. Thomson, “Design of Moving-Average Trend Filters Using Fidelity, Smoothness and Minimum Revisions Criteria,” Res. Rep. CENSUS/SRD/RR-96/1, Statistical Research Division, Bureau of the Census, Washington, DC.

[681] T. Proietti and A. Luati, “Least Squares Regression: Graduation and Filters,” in M. Boumans, ed., *Measurement in Economics: A Handbook*, Academic, London, 2007.

[682] T. Proietti and A. Luati, “Real Time Estimation in Local Polynomial Regression, with Application to Trend-Cycle Analysis,” *Ann. Appl. Statist.*, **2**, 1523 (2008).

[683] A. Luati and T. Proietti, “On the Equivalence of the Weighted Least Squares and the Generalised Least Squares Estimators,” *Compstat 2008—Proc. Comput. Statist.*, P. Brito, ed., Physica-Verlag, Heidelberg, 2008. Available online from `http://mpra.ub.uni-muenchen.de/8910/`

Asymmetric End-Point Filters

[684] T. N. E, Greville, “On Smoothing a Finite Table,” *J. SIAM*, **5**, 137 (1957).

[685] T. N. E, Greville, “Band Matrices and Toeplitz Inverses,” *Lin. Alg. Appl.*, **27**, 199 (1979).

[686] T. N. E, Greville, “Moving-Weighted-Average Smoothing Extended to the Extremities of the Data. I. Theory,” *Scand. Actuarial J.*, p. 39, (1981), and “part II. Methods,”, *ibid.* p.65. See also “Part III. Stability and Optimal Properties,”, *J. Approx. Th.*, **33** 43 (1981).

[687] J. M. Hoem and P. Linnemann, “The Tails in Moving Average Graduation,” *Scand. Actuarial J.*, p. 193, (1988).

Discrete Chebyshev and Hahn Polynomials

[688] P. L. Chebyshev, “Sur l’Interpolation,” reprinted in A. Markoff and N. Sonin, *Oeuvres de P. L. Chebyshev*, vol.1, p. 541, Commissionaires de l’Acádemie Impériale des Sciences, St. Petersbourg, 1899, also Chelsea Publishing Co. , NY, 1961. See also p. 203, 381, 473, 701, and vol.2, p. 219. Available online from `http://www.archive.org/details/uvresdepltcheby00chebgoog`

[689] P. Butzer and F. Jongmans, “P. L. Chebyshev (1821-1894), A Guide to His Life and Work,” *J. Approx. Th.*, **96**, 111 (1999).

[690] C. Jordan, “Sur une Série de Polynomes Dont Chaque Somme Partielle Représente la Meilleure Approximation d’un Degré Donné Suivant la Méthode des Moindres Carrés,” *Proc. London Math. Soc.,* 2nd series, **20**, 297 (1922).

[691] L. Isserlis and V. Romanovsky, “Notes on Certain Expansions in Orthogonal and Semi-Orthogonal Functions,” *Biometrika*, **19**, 87 (1927).

[692] C. Jordan, *Calculus of Finite Differences*, Chelsea Publishing Co. NY, 1939.

[693] G. Szegö, *Orthogonal Polynomials*, Am Math. Soc., Providence, RI, 1939.

[694] P. T. Birge and J. W. Weinberg, “Least Squares Fitting of Data by Means of Polynomials,” *Rev. Mod. Phys.*, **19**, 298 (1947).

[695] M. Weber and A. Erdélyi, “On the Finite Difference Analogue of Rodrigues’ Formula,” *Am. Math. Monthly*, **59**, 163 (1952).

[696] G. E. Forsythe, “Generation and Use of Orthogonal Polynomials for Data-Fitting with a Digital Computer,” *J. Soc. Indust. Appl. Math.*, **5**, 74 (1957).

[697] S. Karlin and J. L. McGregor, “The Hahn Polynomials, Formulas and an Application,” *Scripta Math.*, **26**, 33 (1961).

[698] P. G. Guest, *Numerical Methods of Curve Fitting*, Cambridge Univ. Press, London, 1961.

[699] N. Morrison, *Introduction to Sequential Smoothing and Prediction*, McGraw-Hill, NY, 1969.

[700] B. A. Finlayson, *The Method of Weighted Residuals and Variational Principles*, Academic Press, NY, 1972.

[701] D. E. Clapp, “Adaptive Forecasting with Orthogonal Polynomial Filters,” *AIIE Trans.*, **6**, 359 (1974).

[702] F. B. Hildebrand, *Introduction to Numerical Analysis*, 2/e, McGraw-Hill, New York, 1974, reprinted by Dover Publications, Mineola, NY, 1987.

[703] R. R. Ernst, “Sensitivity Enhancement in Magnetic Resonance,” in *Advances in Magnetic Resonance*, vol. 2, J. S. Waugh, ed., Academic Press, 1966.

[704] C. P. Neuman and D. I. Schonbach, “Discrete (Legendre) Orthogonal Polynomials—A Survey,” *Int. J. Numer. Meth. Eng.*, **8**, 743 (1974).

[705] A. Proctor and P. M. A. Sherwood, “Smoothing of Digital X-ray Photoelectron Spectra by and Extended Sliding Least-Squares Approach,” *Anal. Chem.*, **52** 2315 (1980).

[706] P. D. Willson and S. R. Polo, “Polynomial Filters of any Degree,” *J. Opt. Soc. Am.*, **71**, 599 (1981).

[707] M. U. A. Bromba and H. Ziegler, “On Hilbert Space Design of Least-Weighted- Squares Digital Filters,” *Int. J. Circuit Th. Appl.*, **11**, 7 (1983).

[708] P. Steffen, “On Digital Smoothing Filters: A Brief Review of Closed Form Solutions and Two New Filter Approaches,” *Circ., Syst., and Signal Process.*, fb5, 187 (1986).

[709] H. W. Schüssler and P. Steffen, “Some Advanced Topics in Filter Design,” in Ref. [13].

[710] S. E. Bialkowski, “Generalized Digital Smoothing Filters Made Easy by Matrix Calculations,” *Anal. Chem.*, **61**, 1308 (1989).

[711] P. A. Gorry, “General Least-Squares Smoothing and Differentiation of by the Convolution (Savitzky-Golay) Method,” *Anal. Chem.*, **62**, 570 (1990).

[712] P. A. Gorry, “General Least-Squares Smoothing and Differentiation of Nonuniformly Spaced Data by the Convolution Method,” *Anal. Chem.*, **63**, 534 (1991).

[713] J. E. Kuo and H. Wang, “Multidimensional Least-Squares Smoothing Using Orthogonal Polynomials,” *Anal. Chem.*, **63**, 630 (1991).

[714] G. Y. Pryzva, “Kravchuk Orthogonal Polynomials,” *Ukranian Math. J.*, **44**, 792 (1992).

[715] P. Persson and G. Strang, “Smoothing by Savitzky-Golay and Legendre Filters,” in J. Rosenthal and D. S. Gilliam, eds., *Mathematical Systems Theory in Biology, Communications, Computation, and Finance*, Springer-Verlag, NY, 2003.

[716] W. Gautschi, *Orthogonal Polynomials: Computation and Approximation*, Clarendon Press, Oxford, 2004.

[717] M. E. H. Ismail, *Classical and Quantum Orthogonal Polynomials in One Variable*, Cambridge University Press, Cambridge, (2005).

[718] S. Samadi and A. Nishihara, “Explicit Formula for Predictive FIR Filters and Differentiators Using Hahn Orthogonal Polynomials,” *IEICE Trans. Fundamentals*, **E90**, 1511 (2007).

[719] M. J. Gottlieb, “Concerning Some Polynomials Orthogonal on a Finite or Enumerable Set of Points,” *A. J. Math*, **60**, 453 (1938).

[720] R. E. King and P. N. Paraskevopoulos, “Digital Laguerre Filters,” *Circ. Th. Appl.*, **5**, 81 (1977).

[721] M. R. Teague, “Image Analysis via the General Theory of Moments,” *J. Opt. Soc. Am.*, **70**, 920 (1980).

[722] R. M. Haralick, “Digital Step Edges from Zero Crossing of Second Directional Derivatives,” *IEEE Trans. Patt. Anal. Mach. Intell.*, **PAMI-6**, 58 (1984).

[723] C-S. Liu and H-C. Wang, “A Segmental Probabilistic Model of Speech Using an Orthogonal Polynomial Representation,” *Speech Commun.*, **18** 291 (1996).

[724] P. Meer and I. Weiss, “Smoothed Differentiation Filters for Images,” *J. Vis. Commun. Imag. Process.*, **3**, 58 (1992).

[725] G. Carballo, R. Álvarez-Nodarse, and J. S. Dehesa, “Chebyshev Polynomials in a Speech Recognition Model,” *Appl. Math. Lett.*, **14**, 581 (2001).

[726] R. Mukundan, S. H. Ong, and P. A. Lee, “Image Analysis by Tchebichef Moments,” *IEEE Trans. Image Process.*, **10**, 1357 (2001).

[727] J. Arvesú, J. Coussement, and W. Van Asscheb, “Some Discrete Multiple Orthogonal Polynomials,” *J. Comp. Appl. Math.*, **153**, 19 (2003).

[728] R. Mukundan, “Some Computational Aspects of Discrete Orthonormal Moments,” *IEEE Trans. Image Process.*, **13**, 1055 (2004).

[729] L. Kotoulas and I. Andreadis, “Image Analysis Using Moments,” *Proc. IEEE Int. Conf. Technol. Autom.* (ICTA-05), p.360, (2005).

[730] L. Kotoulas and I. Andreadis, “Fast Computation of Chebyshev Moments,” *IEEE Trans. Circuits Syst. Video Technol.*, **16**, 884 (2006).

[731] K. W. Lee, et al., “Image reconstruction Using Various Discrete Orthogonal Polynomials in Comparison with DCT,” *Appl. Math. Comp.*, **193**, 346 (2007).

[732] H. Zhu, et al., “Image Analysis by Discrete Orthogonal Dual Hahn Moments,” *Patt. Recogn. Lett.* **28**, 1688 (2007).

[733] H. Shu, L. Luo, and J-L Coatrieux, “Moment-Based Approaches in Imaging. Part 1, Basic Features,” *IEEE Eng. Med. Biol. Mag.*, **26**, no.5, 70 (2007).

[734] H. Shu, L. Luo, and J-L Coatrieux, “Moment-Based Approaches in Imaging. Part 2, Invariance,” *IEEE Eng Med Biol Mag.*, **27**, no.1, 81 (2008).

[735] E. Diekema and T. H. Koornwinder, “Differentiation by integration using orthogonal polynomials, a survey,” *, J. Approx.*, **164**, 637 (2012).

Predictive and Fractional-Delay Filters

[736] R. W. Schafer and L. R. Rabiner, “A Digital Signal Processing Approach to Interpolation,” *Proc. IEEE*, **61**, 692 (1973).

[737] H. W. Strube, “Sampled-Data Representation of a Nonuniform Lossless Tube of Continuously Variable Length,” *J. Acoust. Soc. Amer.*, **57**, 256 (1975).

[738] P. Heinonen and Y. Neuvo, “FIR-Median Hybrid Filters with Predictive FIR Substructures,” *IEEE Trans. Acoust., Speech, Signal Process.*, **36**, 892 (1988).

[739] C. W. Farrow, “A Continuously Variable Digital Delay Element,” *Proc. IEEE Int. Symp. Circuits and Systems, ISCAS-88*, p. 2641, (1988).

[740] G-S Liu and C-H Wei, “Programmable Fractional Sample Delay Filter with Lagrange Interpolation,” *Electronics Lett.*, **26**, 1608 (1990).

[741] T. G. Campbell and Y. Neuvo, “Predictive FIR Filters with Low Computational Complexity,” *IEEE Trans. Circ. Syst.*, **38** 1067 (1991).

[742] S. J. Ovaska, “Improving the Velocity Sensing Resolution of Pulse Encoders by FIR Prediction,” *IEEE Trans. Instr. Meas.*, **40**, 657 (1991).

[743] S. J. Ovaska, “Newton-Type Predictors—A Signal Processing Perspective,” *Signal Process.*, **25**, 251 (1991).

[744] G-S Liu and C-H Wei, “A New Variable Fractional Sample Delay Filter with Nonlinear Interpolation,” *IEEE Trans. Circ. Syst.–II*, **39**, 123 (1992).

[745] L. Erup., F. M. Gardner, and R. A. Harris, “Interpolation in Digital Modems—Part II: Implementation and Performance,” *IEEE Trans. Commun.*, **41**, 998 (1993).

[746] T. I. Laakso, et al., “Splitting the Unit Delay—Tools for Fractional Delay Filter Design,” *IEEE Signal Process. Mag.*, **13**, 30, Jan. 1996.

[747] P. J. Kootsookos and R. C. Williamson, “FIR Approximation of Fractional Sample Delay Systems,” *IEEE Trans. Circ. Syst.–II*, **43**, 269 (1996).

[748] O. Vainio, M. Renfors, and T. Saramäki, “Recursive Implementation of FIR Differentiators with Optimum Noise Attenuation,” *IEEE Trans. Instrum. Meas.*, **46**, 1202 (1997).

[749] P. T, Harju, “Polynomial Prediction Using Incomplete Data,” *IEEE Trans. Signal Process.*, **45**, 768 (1997).

[750] S. Tassart and P. Depalle, “Analytical Approximations of Fractional Delays: Lagrange Interpolators and Allpass Filters,” *IEEE Int. Conf. Acoust., Speech, Sig. Process.*, (ICASSP-97), **1** 455 (1997).

[751] S. Väliviita and S. J. Ovaska, “Delayless Recursive Differentiator with Efficient Noise Attenuation for Control Instrumentation,” *Signal Process.*, **69**, 267 (1998).

[752] S-C Pei and C-C Tseng, “A Comb Filter Design Using Fractional-Sample Delay,” *IEEE Trans. Circ. Syst.–II*, **45**, 649 (1998).

[753] S. Väliviita, S. J. Ovaska, and O. Vainio, “Polynomial Predictive Filtering in Control and Instrumentation: A Review,” *IEEE Trans. Industr. Electr.*, **46**, 876 (1999).

[754] E. Meijering, “A Chronology of Interpolation: From Ancient Astronomy to Modern Signal and Image Processing,” *Proc. IEEE*, **90**, 319 (2002).

[755] V. Välimäki, et al. “Discrete-Time Modeling of Musical Instruments,” *Rep. Progr. Phys.*, **69**, 1 (2006).

[756] C. Candan, “An Efficient Filtering Structure for Lagrange Interpolation,” *IEEE Signal Proc. Lett.*, **14**, 17 (2007).

[757] J. Vesma and T. Saramäki, “Polynomial-Based Interpolation Filters—Part I: Filter Synthesis,” *Circ. Syst, Signal Process.*, **26**, 115 (2007).

Maximally Flat Filters

[758] O. Herrmann, “On the Approximation Problem in Nonrecursive Digital Filter Design,” *IEEE Trans. Circ. Th.*, **CT-18**, 411 (1971).

[759] J. A. Miller, “Maximally Flat Nonrecursive Digital Filters,” *Electron. Lett.*, **8**, 157 (1972).

[760] M. F. Fahmy, “Maximally Flat Nonrecursive Digital Filters,” *Int. J. Circ. Th. Appl*, **4**, 311 (1976).

[761] J-P. Thiran, “Recursive Digital Filters with Maximally Flat Group Delay,” *IEEE Trans. Circ. Th.*, **CT-18**, 659 (1971).

[762] M. U. A. Bromba and H. Ziegler, “Explicit Formula for Filter Function of Maximally Flat Nonrecursive Digital Filters,” *Electron. Lett.*, **16**, 905 (1980), and *ibid.*, **18**, 1014 (1982).

[763] H. Baher, “FIR Digital Filters with Simultaneous Conditions on Amplitude and Group Delay,” *Electron. Lett.*, **18**, 296 (1982).

[764] L. R. Rajagopal and S. C. D. Roy, “Design of Maximally-Flat FIR Filters Using the Bernstein Polynomial,,, *IEEE Trans. Circ. Syst.*, **CAS-34**, 1587 (1987).

[765] E. Hermanowicz, “Explicit Formulas for Weighting Coefficients of Maximally Flat Tunable FIR delayers,” *Electr. Lett.*, **28**, 1936 (1992).

[766] I. W. Selesnick and C. S. Burrus, “Maximally Flat Low-Pass FIR Filters with Reduced Delay,” *IEEE Trans. Circ. Syst. II*, **45**, 53 (1998).

[767] I. W. Selesnick and C. S. Burrus, “Generalized Digital Butterworth Filter Design,” *IEEE Trans. Signal Process.*, **46**, 1688 (1998).

[768] S. Samadi, A. Nishihara, and H. Iwakura, “Universal Maximally Flat Lowpass FIR Systems,” *IEEE Trans. Signal Process.*, **48**, 1956 (2000).

[769] R. A. Gopinath, “Lowpass Delay Filters With Flat Magnitude and Group Delay Constraints,” *IEEE Trans. Signal Process.*, **51**, 182 (2003).

[770] Fractional-Delay Systems,” *IEEE Trans. Circ. Syst.–I*, **51**, 2271 (2004).

[771] S. Samadi and A. Nishihara, “The World of Flatness,” *IEEE Circ. Syst. Mag.*, p.38, third quarter 2007.

Local Polynomial Modeling and Loess

[772] E. A. Nadaraya, “On Estimating Regression,” *Th. Prob. Appl.*, **10**, 186 (1964).

[773] G. S. Watson, “Smooth Regression Analysis,” *Sankya, Ser. A*, **26**, 359 (1964).

[774] M. B. Priestley and M. T. Chao, “Non-Parametric Function Fitting,” *J. Roy. Statist. Soc., Ser. B*, **34**, 385 (1972).

[775] C. J. Stone, “Consistent Nonparametric Regression (with discussion),” *Ann. Statist.*, **5**, 595 (1977).

[776] W. S. Cleveland, “Robust Locally Weighted Regression and Smoothing of Scatterplots,” *J. Amer. Statist. Assoc.*, **74**, 829 (1979).

[777] W. S. Cleveland and R. McGill “The Many Faces of a Scatterplot,” *J. Amer. Statist. Assoc.*, **79**, 807 (1984).

[778] . H. Friedman, “A Variable Span Smoother,” Tech. Rep. No. 5, Lab. Comput. Statist., Dept. Statist., Stanford Univ., (1984); see also, J. H. Friedman and W. Stueltze, “Smoothing of Scatterplots,” Dept. Statist., Tech. Rep. Orion 3, (1982).

[779] H-G. Müller, “Smooth Optimum Kernel Estimators of Densities, Regression Curves and Modes,” *Ann. Statist.*, **12**, 766 (1984).

[780] T. Gasser, H-G. Müller, and V. Mammitzsch, “Kernels for Nonparametric Curve Estimation,” *J. Roy. Statist. Soc., Ser. B*, **47**, 238 (1985).

[781] J. A. McDonald and A. B. Owen, “Smoothing with Split Linear Fits,” *Technometrics*, **28**, 195 (1986).

[782] A. B. Tsybakov, “Robust Reconstruction of Functions by the Local-Approximation Method,” *Prob. Inf. Transm.*, **22**, 69 (1986).

[783] W. S. Cleveland and S. J. Devlin, “Locally Weighted Regression: An Approach to Regression Analysis by Local Fitting,” *J. Amer. Statist. Assoc.*, **83**, 596 (1988).

[784] A. Buja, A. Hastie, and R. Tibshirani, “Linear Smoothers and Additive Models (with discussion),” *Ann. Statist.*, **17**, 453 (1989).

[785] B. L. Granovsky and H-G. Müller, “The Optimality of a Class of Polynomial Kernel Functions,” *Stat. Decis.*, **7**, 301 (1989).

[786] W. Härdle, *Applied Nonparametric Regression*, Cambridge Univ. Press, Cambridge, 1990.

[787] A. Hastie and R. Tibshirani, *Generalized Additive Models*, Chapman & Hall, London, 1990.

[788] B. L. Granovsky, H-G. Müller, "Optimizing Kernel Methods: A Unifying Variational Principle," *Int. Stat. Rev.*, **59**, 373 (1991).

[789] N. S. Altman, "An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression," *Amer. Statist.*, **46**, 175 (1992).

[790] I. Fan and I. Gijbels, "Variable Bandwidth and Local Linear Regression Smoothers," *Ann. Statist.*, **20**, 2008 (1992).

[791] W. S. Cleveland and Grosse, "A Package of C and Fortran Routines for Fitting Local Regression Models," 1992. Available from: `http://www.netlib.org/a/dloess`.

[792] W. S. Cleveland, *Visualizing Data*, Hobart Press, Summit, NJ, 1993.

[793] I. Fan, "Local Linear Regression Smoothers and Their Minimax Efficiencies," *Ann. Statist.*, **21**, 196 (1993).

[794] A. Hastie and C. Loader, "Local Regression: Automatic Kernel Carpentry," *Statist. Sci.*, **8**, 120 (1993).

[795] M. C. Jones, S. J. Davies, and B. U. Park, "Versions of Kernel-Type Regression Estimators," *J. Amer. Statist. Assoc.*, **89**, 825 (1994).

[796] I. Fan and I. Gijbels, "Data-Driven Bandwidth Selection in Local Polynomial Fitting: Variable Bandwidth and Spatial Adaptation," *J. Roy. Statist. Soc., Ser. B*, **57**, 371 (1995).

[797] D. Ruppert, S. J. Sheather, and M. P. Wand, "An Effective Bandwidth Selector for Local Least Squares Regression," *J. A. Statist. Assoc.*, **90**, 125 (1995).

[798] M. P. Wand and M. C. Jones, *Kernel Smoothing*, Chapman & Hall, London, 1995.

[799] W. S. Cleveland and C. Loader, "Smoothing by Local Regression: Principles and Methods," in W. Härdle and M. G. Schimek, eds., *Statistical Theory and Computational Aspects of Smoothing*, Physica-Verlag, Heidelberg, May 1996.

[800] M. C. Jones, J. S. Marron, and S. J. Sheaver, "A Brief Survey of Bandwidth Selection for Density Estimation," *J. Amer. Statist. Assoc.*, **91**, 401 (1996).

[801] B. Seifert and T. Gasser, "Finite Sample Variance of Local Polynomials: Analysis and Solutions," *J. Amer. Statist. Assoc.*, **91**, 267 (1996).

[802] J. S. Simonoff, *Smoothing Methods in Statistics*, Springer-Verlag, New York, 1996.

[803] I. Fan and I. Gijbels, *Local Polynomial Modelling and Its Applications*, Chapman & Hall, London, 1996.

[804] A. Goldenshluger and A. Nemirovski, "On Spatial Adaptive Estimation of Nonparametric Regression," *Math. Meth. Stat.*, **6**, 135 (1997).

[805] A. W. Bowman and A. Azzalini, *Applied Smoothing Techniques for Data Analysis*, Oxford Univ. Press, New York, 1997.

[806] C. M. Hurvich and J. S. Simonoff, "Smoothing Parameter Selection in Nonparametric Regression Using an Improved AIC Criterion," *J. Roy. Statist. Soc., Ser. B*, **60**, 271 (1998).

[807] C. R. Loader, "Bandwidth Selection: Classical or Plug-In?," *Ann. Statist.*, **27**, 415 (1999).

[808] C. Loader, *Local Regression and Likelihood*, Springer-Verlag, New York, 1999.

[809] V. Katkovnik, "A New method for Varying Adaptive Bandwidth Selection," *IEEE Trans. Signal Process.*, **47**, 2567 (1999).

[810] I. Horová, "Some Remarks on Kernels," *J. Comp. Anal. Appl.*, **2**, 253 (2000).

[811] W. R. Schucany, "An Overview of Curve Estimators for the First Graduate Course in Nonparametric Statistics," *Statist. Sci.*, **19**, 663 (2004).

[812] C. Loader, "Smoothing: Local Regression Techniques," in J. Gentle, W. Härdle, and Y. Mori, eds., *Handbook of Computational Statistics*, Springer-Verlag, Heidelberg, 2004.

[813] V. Katkovnik, K. Egiazarian, and J. Astola, *Local Approximation Techniques in Signal and Image Processing*, SPIE Publications, Bellingham, WA, 2006.

[814] Data available from: `http://www.netlib.org/a/dloess`. Original source: N. D. Brinkman, "Ethanol - A Single-Cylinder Engine Study of Efficiency and Exhaust Emissions," *SAE Transactions*, **90**, 1410 (1981).

[815] Data available from `http://fedc.wiwi.hu-berlin.de/databases.php`, (MD*Base collection). Original source: Ref. [786] and G. Schmidt, R. Mattern, and F. Schüller, EEC Res. Program on Biomechanics of Impacts, Final report, Phase III, Project 65, Inst. für Rechtsmedizin, Univ. Heidelberg, Germany.

Exponential Moving Average Filters

[816] R. G. Brown, *Smoothing, Forecasting and Prediction of Discrete-Time Series*, Prentice Hall, Englewood-Cliffs, NJ, 1962.

[817] D. C. Montgomery and L. A. Johnson, *Forecasting and Time Series Analysis*, McGraw-Hill, New York, 1976.

[818] C. D. Lewis, *Industrial and Business Forecasting Methods*, Butterworth Scientific, London, 1982.

[819] B. Abraham and J. Ledolter, *Statistical Methods for Forecasting*, Wiley, New York, 1983.

[820] S. Makridakis, et al., *The Forecasting Accuracy of Major Time Series Models*, Wiley, New York, 1983.

[821] S. Makridakis, S. C. Wheelwright, and R. J. Hyndman, *Forecasting, Methods and Applications*, 3/e, Wiley, New York, 1998.

[822] C. Chatfield, *Time Series Forecasting*, Chapman & Hall/CRC Press, Boca Raton, FL, 2001.

[823] R. J. Hyndman, A. B. Koehler, J. K. Ord, and R. D. Snyder, *Forecasting with Exponential Smoothing*, Springer-Verlag, Berlin, 2008.

[824] C. C. Holt, "Forecasting Seasonals and Trends by Exponentially Weighted Moving Averages," Office of Naval Research memorandum (ONR 52), 1957, reprinted in *Int. J. Forecast.*, **20**, 5 (2004); see also, *ibid.*, **20**, 11 (2004).

[825] P. R. Winters, "Forecasting Sales by Exponentially Weighted Moving Averages," *Manag. Sci.*, **6**, 324 (1960).

[826] J. F. Muth, "Optimal Properties of Exponentially Weighted Forecasts," *J. Amer. Statist. Assoc.*, **55**, 299 (1960).

[827] R. G. Brown and R. F. Meyer, "The Fundamental Theorem of Exponential Smoothing," *Oper. Res.*, **9**, 673 (1961).

[828] D. A. D'Esopo, "A Note on Forecasting by the Exponential Smoothing Operator," *Oper. Res.*, **9**, 686 (1961).

[829] D. R. Cox, "Prediction by Exponentially Weighted Moving Averages and Related Methods," *J. Roy. Statist. Soc., Ser. B*, **23**, 414 (1961).

[830] R. H. Morris and C. R. Glassey, "The Dynamics and Statistics of Exponential Smoothing Operators," *Oper. Res.*, **11**, 561 (1963).

[831] H. Theil and S. Wage, “Some Observations on Adaptive Forecasting,” *Manag. Sci.*, **10**, 198 (1964).

[832] P. J. Harrison, “Short-Term Sales Forecasting,” *Appl. Statist.*, **14**, 102 (1965).

[833] P. J. Harrison, “Exponential Smoothing and Short-Term Sales Forecasting,” *Manag. Sci.*, **13**, 821 (1967).

[834] W. G. Gilchrist, “Methods of Estimation Involving Discounting,” *J. Roy. Satist. Soc., Ser. B*, **29**, 355 (1967).

[835] C. C. Pegels, “Exponential Forecasting: Some New Variations,” *Manag. Sci.*, **15**, 311 (1969).

[836] A. C. Watts, “On Exponential Smoothing of Discrete Time Series,” *IEEE TRans. Inform. Th.*, **16**, 630 (1970).

[837] K. O. Cogger, “The Optimality of General-Order Exponential Smoothing,” *Oper. Res.*, **22**, 858 (1974).

[838] S. D. Roberts and D. C. Whybark, “Adaptive Forecasting Techniques,” *Int. J. Prod. Res.*, **12**, 635 (1974).

[839] M. L. Goodman, “A New Look at Higher-Order Exponential Smoothing for Forecasting,” *Oper. Res.*, **22**, 880 (1974).

[840] D. E. Clapp, “Adaptive Forecasting with Orthogonal Polynomial Models,” *AIIE Trans.*, **6**, 359 (1974).

[841] J. W. Tukey, *Exploratory Data Analysis*, Addison-Wesley, Reading, MA, 1977.

[842] J. F. Kaiser and R. W. Hamming, “Sharpening the Response of a Symmetric Nonrecursive Filter by the Multiple Use of the same Filter,” *IEEE Trans. Acoust., Speech, Signal Process.*, **ASSP-25**, 415 (1977).

[843] E. Mckenzie, “The Monitoring of Exponentially Weighted Forecasts,” *J. Oper. Res. Soc.*, **29**, 449 (1978).

[844] C. Chatfield, “The Holt-Winters Forecasting Procedure,” *Appl. Statist.*, **27**, 264 (1978).

[845] R. Fildes, “Quantitative Forecasting—The State of the Art: Extrapolative Methods,” *J. Oper. Res. Soc.*, **30**, 691 (1979).

[846] S. Ekern, “Adaptive Exponential Smoothing Revisited,” *J. Oper. Res. Soc.*, **32**, 775 (1981).

[847] S. A. Roberts, “A General Class of Holt-Winters Type Forecasting Models,” *Manag. Sci.*, **28**, 808 (1982).

[848] E. J. Muth, “The Discrete Laguerre Polynomials and their Use in Exponential Smoothing,” *IIE Trans.*, **15**, 166 (1983).

[849] E. S. Gardner, Jr., “Exponential Smoothing: The State of the Art,” *J. Forecast.*, **4**, 1 (1985).

[850] B. Abraham and J. Ledolter, “Forecast Functions Implied by Autoregressive Integrated Moving Average Models and Other Related Forecast Procedures,” *Int. Statist. Rev.*, **54**, 51 (1986).

[851] D. J. Dalrymple, “Sales Forecasting Practices: Results from a United States Survey,” *Int. J. Forecast.*, **3**, 379 (1987).

[852] C. Chatfield and M. Yar, “Holt-Winters Forecasting: Some Practical Issues,” *Statistician*, **37**, 129 (1988).

[853] E. Yashchin, “Estimating the Current Mean of a Process Subject to Abrupt Changes,” *Technometrics*, **37**, 311 (1995).

[854] S. Satchell and A. Timmermann, “On the Optimality of Adaptive Expectations: Muth Revisited,” *Int. J. Forecast.*, **11**, 407 (1995).

[855] H. Winklhofer, A. Diamantopoulos, and S. F. Witt, “Forecasting practice: A Review of the Empirical Literature and an Agenda for Future Research,” *Int. J. Forecast.*, **12**, 193 (1996).

[856] S. Makridakis and M. Hibon, “The M3-Competition: Results, Conclusions and Implications,” *Int. J. Forecast.*, **16**, 451 (2000).

[857] C. Chatfield, et al., “A New Look at Models for Exponential Smoothing,” *Statistician*, **50**, 147 (2001).

[858] A. Chen and E. A. Elsayed, “Design and Performance Analysis of the Exponentially Weighted Moving Average Mean Estimate for Processes Subject to Random Step Changes,” *Technometrics*, **44**, 379 (2002).

[859] D. J. Robb and E. A. Silver, “Using Composite Moving Averages to Forecast Sales,” *J. Oper. Res. Soc.*, **53**, 1281 (2002).

[860] J. W. Taylor, “Smooth Transition Exponential Smoothing,” *J. Forecast.*, **23**, 385 (2004).

[861] E. S. Gardner, Jr., “Exponential Smoothing: The State of the Art—Part II,” *Int. J. Forecast.*, **22**, 239 (2006).

[862] B. Billah, et al., “Exponential Smoothing Model Selection for Forecasting,” *Int. J. Forecast.*, **22**, 239 (2006).

[863] J. G. De Gooijer and R. J. Hyndman, “25 Years of Time Series Forecasting,” *Int. J. Forecast.*, **22**, 443 (2006).

Filtering Methods and Technical Analysis in Financial Markets

[864] S. B. Achelis, *Technical Analysis from A to Z*, 2nd ed., McGraw-Hill, NY, 2001.
available online, `https://www.metastock.com/customer/resources/taaz/`

[865] J. W. Wilder, *New Concepts in Technical Trading Systems*, Trend Research, Greensboro, NC, 1978.

[866] “Surviving The Test of Time With J. Welles Wilder,” interview by B. Twomey, *Tech. Anal. Stocks & Commod.*, **27**, no.3, 58 (2009).

[867] T. S. Chande and S. Kroll, *The New Technical Trader*, Wiley, NY, 1994.

[868] J. F. Ehlers, *Rocket Science for Traders*, Wiley, NY, 2001.

[869] J. F. Ehlers, *Cybernetic Analysis for Stocks and Futures*, Wiley, NY, 2004.

[870] P. J. Kaufman, *New Trading Systems and Methods*, 4/e, Wiley, 2005.

[871] D. K. Mak, *Mathematical Techniques in Financial Market Trading*, World Scientific, Singapore, 2006.

[872] *Technical Analysis*, PDF book, 2011, Creative Commons Attribution-Share, available from: `https://www.mrao.cam.ac.uk/~mph/Technical_Analysis.pdf`

[873] International Federation of Technical Analysts, `www.ifta.org`

[874] V. Zakamulin, *Market Timing with Moving Averages*, Palgrave Macmillan, 2017. See also by same author, “Moving Averages for Market Timing,”, Oct. 2016. Available at SSRN: `https://ssrn.com/abstract=2854180`

[875] D. Penn, “The Titans Of Technical Analysis,” *Tech. Anal. Stocks & Commod.*, **20**, no.10, 32 (2002).

[876] A. W. Lo and J. Hasanhodzic, *The Heretics of Finance*, Bloomberg Press, NY, 2009.

[877] M. Carr and A. Hestla, “Technical Analysis Adapts and Thrives,” *Tech. Anal. Stocks & Commod.*, **29**, no.4, 46 (2011).

[878] J. K. Hutson, “Good Trix”, *Tech. Anal. Stocks & Commod.*, **1**, no.5, 105, (1983); *ibid.*, **2**, no.2, 91, (1984). See also, D. Penn, “TRIX”, *Tech. Anal. Stocks & Commod.*, **29**, no.9, 197, (2003).

[879] R. Barrons Roosevelt, “Metaphors For Trading,” *Tech. Anal. Stocks & Commod.*, **16**, no.2, 67 (1998).

[880] T. S. Chande, “Adapting Moving Averages to Market Volatility,” *Tech. Anal. Stocks & Commod.*, **10**, no.3, 108 (1992).

[881] P. G. Mulloy, “Smoothing Data with Faster Moving Averages,” *Tech. Anal. Stocks & Commod.*, **12**, no.1, 11 (1994).

[882] P. G. Mulloy, “Smoothing Data with Less Lag,” *Tech. Anal. Stocks & Commod.*, **12**, no.2, 72 (1994).

[883] T. S. Chande, “Forecasting Tomorrow’s Trading Day,” *Tech. Anal. Stocks & Commod.*, **10**, no.5, 220 (1992).

[884] P. E. Lafferty, “The End Point Moving Average,” *Tech. Anal. Stocks & Commod.*, **13**, no.10, 413 (1995).

[885] D. Kraska, “The End Point Moving Average,”, Letters to *Tech. Anal. Stocks & Commod.*, **14**, Feb. (1996).

[886] J. F. Ehlers, “Zero-Lag Data Smoothers,” *Tech. Anal. Stocks & Commod.*, **20**, no.7, 26 (2002). See also, J. F. Ehlers and R. Way, “Zero Lag (Well, Almost),” *ibid.*, **28**, 30, Nov. (2010).

[887] W. Rafter, “The Moving Trend,” *Tech. Anal. Stocks & Commod.*, **21**, no.1, 38 (2003).

[888] D. Meyers, “Surfing the Linear Regression Curve with Bond Futures,” *Tech. Anal. Stocks & Commod.*, **16**, no.5, 209 (1998).

[889] B. Star, “Confirming Price Trend,” *Tech. Anal. Stocks & Commod.*, **25**, no.13, 72 (2007).

[890] P. E. Lafferty, “How Smooth is Your Data Smoother?,” *Tech. Anal. Stocks & Commod.*, **17**, no.6, 251 (1999).

[891] T. Tillson, “Smoothing Techniques For More Accurate Signals,” *Tech. Anal. Stocks & Commod.*, **16**, no.1, 33 (1998).

[892] J. Sharp, “More Responsive Moving Averages,” *Tech. Anal. Stocks & Commod.*, **18**, no.1, 56 (2000).

[893] A. Hull, “How to reduce lag in a moving average,” `https://alanhull.com/hull-moving-average`.

[894] B. Star, “Detecting Trend Direction and Strength,” *Tech. Anal. Stocks & Commod.*, **20**, no.1, 22 (2007).

[895] S. Evens, “Momentum And Relative Strength Index,” *Tech. Anal. Stocks & Commod.*, **17**, no.8, 367 (1999).

[896] S. Evens, “Stochastics,” *Tech. Anal. Stocks & Commod.*, **17**, no.9, 392 (1999).

[897] P. Roberts, “Moving Averages: The Heart of Trend Analysis,” *Alchemist*, **33**, 12 (2003), Lond. Bullion Market Assoc., available online from: `www.lbma.org.uk`.

[898] K. Edgeley “Oscillators Go with the Flow,” *Alchemist*, **37**, 17 (2005), Lond. Bullion Market Assoc., available online from: `www.lbma.org.uk`.

[899] D. Penn, “Moving Average Trios,” *Tech. Anal. Stocks & Commod.*, **25**, no.9, 54 (2007).

[900] B. Star, “Trade the Price Swings,” *Tech. Anal. Stocks & Commod.*, **21**, no.12, 68 (2003).

[901] A. Sabodin, “An MACD Trading System,” *Tech. Anal. Stocks & Commod.*, **26**, no.3, 12 (2008).

[902] C. K. Langford, “Three Common Tools, One Protocol,” *Tech. Anal. Stocks & Commod.*, **26**, no.10, 48 (2008).

[903] H. Seyedinajad, “The RSI Miracle,” *Tech. Anal. Stocks & Commod.*, **27**, no.1, 12 (2009).

[904] M. Alves, “Join the Band: Applying Hysteresis to Moving Averages,” *Tech. Anal. Stocks & Commod.*, **27**, no.1, 36 (2009).

[905] E. Donie, “An MACD Parallax View,” *Tech. Anal. Stocks & Commod.*, **27**, no.4, 12 (2009).

[906] R. Singh and A. Kumar, “Intelligent Stock Trading Technique using Technical Analysis,” *Int. J. Mgt. Bus. Studies*, **1**, 46 (2011).

[907] J. Bollinger, “Using Bollinger Bands,” *Tech. Anal. Stocks & Commod.*, **10**, no.2, 47 (1992).

[908] S. Evens, “Bollinger Bands,” *Tech. Anal. Stocks & Commod.*, **17**, no.3, 116 (1999).

[909] S. Vervoort, “Smoothing the Bollinger %b,” *Tech. Anal. Stocks & Commod.*, **28**, no.5, 40 (2010); and Part 2, *ibid.*, **28**, no.6, 48 (2010).

[910] J. Gopalakrishnan and B. Faber, “Interview: System Trading Made Easy With John Bollinger,” *Tech. Anal. Stocks & Commod.*, **30**, no.3, 36 (2012).

[911] A. Mustapha, “Bollinger Bands & RSI: A Magical Combo,” *Tech. Anal. Stocks & Commod.*, **34**, no.6, 18 (2016).

[912] M. Widner, “Signaling Change with Projection Bands,” *Tech. Anal. Stocks & Commod.*, **13**, no.7, 275 (1995).

[913] J. Andersen, “Standard Error Bands,” *Tech. Anal. Stocks & Commod.*, **14**, no.9, 375 (1996).

[914] S. Evens, “Keltner Channels,” *Tech. Anal. Stocks & Commod.*, **17**, no.12, 533 (1999).

[915] D. Penn, “Donchian Breakouts,” *Tech. Anal. Stocks & Commod.*, **20**, no.2, 34 (2002); and, “Building a Better Breakout,”, *ibid.*, **21**, no.10, 74 (2003).

[916] B. Star, “Trade Breakouts And Retracements With TMV,” *Tech. Anal. Stocks & Commod.*, **30**, no.2, 13 (2012).

[917] F. Bertrand, “RSI Bands,” *Tech. Anal. Stocks & Commod.*, **26**, no.4, 44 (2008).

[918] S. Lim, T. T.. Hisarli, and N, S. He, “Profitability of a Combined Signal Approach: Bollinger Bands and the ADX,” *IFTA J.*, p.23, 2014 edition, `https://ifta.org/publications/journal/`.

[919] P, Aan, “Parabolic Stop/Reversal,” *Tech. Anal. Stocks & Commod.*, **7**, no.11, 411 (1989).

[920] T. Hartle, “The Parabolic Trading System,”, *Tech. Anal. Stocks & Commod.*, **11**, no.11, 477 (1993).

[921] D. Meyers, “Modifying the Parabolic Stop And Reversal,” *Tech. Anal. Stocks & Commod.*, **14**, no.4, 152 (1995).

[922] J. Sweeney, “Parabolics,” *Tech. Anal. Stocks & Commod.*, **15**, no.7, 329 (1997).

[923] R. Teseo, “Stay in the Market with Stop-And-Reverse,” *Tech. Anal. Stocks & Commod.*, **20**, no.4, 76 (2002).

[924] K. Agostino and B. Dolan, “Make the Trend Your Friend in Forex,” *Tech. Anal. Stocks & Commod.*, **22**, no.9, 14 (2004).

[925] D. Sepiashvili, “The Self-Adjusting RSI,” *Tech. Anal. Stocks & Commod.*, **24**, no.2, 20 (2006).

[926] G. Siligardos, "Leader Of The MACD," *Tech. Anal. Stocks & Commod.*, **26**, no.7, 24 (2008).

[927] M. J. Pring, "The Special K, Part 1," *Tech. Anal. Stocks & Commod.*, **26**, no.12, 44 (2008); and Part 2, *ibid.*, **27**, no.1, 28 (2009); see also, *ibid.*, "Identifying Trends With The KST Indicator," **10**, no.10, 420 (1992).

[928] P. Konner, "Combining RSI with RSI," *Tech. Anal. Stocks & Commod.*, **29**, no.1, 16 (2011).

[929] A. A. Merrill, *Filtered Waves, Basic Theory: A Tool for Stock Market Analysis*, Analysis Press, Chappaqua, NY, 1977.

[930] S. Raftopoulos, "Zigzag Validity,", *Tech. Anal. Stocks & Commod.*, **20**, no.8, 28 (2002).

[931] S. Raftopoulos, "The Zigzag Trend Indicator,", *Tech. Anal. Stocks & Commod.*, **21**, no.11, 26 (2003).

[932] W. Cringan, "Zigzag Targets,", *Tech. Anal. Stocks & Commod.*, **21**, no.2, 24 (2003).

[933] N. J. Brown, "Zigzag and One Rank Compared,", *Tech. Anal. Stocks & Commod.*, **22**, no.6, 24 (2004).

[934] G. Siligardos, "Filtering Price Movement,", *Tech. Anal. Stocks & Commod.*, **33**, no.5, 12 (2015).

[935] *Fidelity's Technical Indicator Guide:*
`https://www.fidelity.com/learning-center/trading-investing/technical-analysis/technical-indicator-guide/overview`

OANDA Technical Indicator Guide and Tutorials:
`https://www.oanda.com/forex-trading/learn/forex-indicators`
`https://www.oanda.com/forex-trading/learn/technical-analysis-for-traders`

[936] *TradingView Wiki:*
`https://www.tradingview.com/wiki`

[937] A. Raudys, V. Lenciauskas, and E. Malcius, "Moving Averages for Financial Data Smoothing," in T. Skersys, R. Butleris, and R. Butkiene (Eds.), *Proceedings Information and Software Technologies*, 19th Int. Conf., ICIST 2013; paper available online from,
`pdfs.semanticscholar.org/257b/837649d8b50662b3fe2c21fce825a1c184e5.pdf`

[938] C. W. Gross and J. E. Sohl, "Improving Smoothing Models with an Enhanced Initialization scheme," *J. Bus. Forecasting*, **8**, 13 (1989).

[939] J. R. Taylor, *Introduction to Error Analysis*, Oxford University Press, University Science Books, Mill Valley, CA.

Markowitz Portfolios

[940] H. Markowitz, "Portfolio Selection," *J. Finance*, **7**, 77 (1962).

[941] W. F. Sharpe, "Capital asset prices: A theory of market equilibrium under conditions of risk," *J. Finance*, **19**, 425 (1964).

[942] H. Markowitz, *Mean-Variance Analysis in Portfolio Choice and Capital Markets*, Wiley (2000).

[943] R. Merton, "An analytic derivation of the efficient portfolio frontier," *J. Financial Quant. Anal.* **7**, 1851 (1972).

[944] H. M. Markowitz, "Foundations of Portfolio Theory," *J. Finance*, **46**, 469 (1991).

[945] W. F. Sharpe, "Capital Asset Prices with and without Negative Holdings," *J. Finance*, **46**, 489 (1991).

[946] H. M. Markowitz, "The General Mean-Variance Portfolio Selection Problem [and Discussion]," *Phil. Trans.: Phys. Sci. Eng.*, **347**, 543 (1994).

[947] H. M. Markowitz, "The Early History of Portfolio Theory: 1600-1960," *Financial Analysts J.*, **55**, no.4, p.5, 1999.

[948] K. V. Fernando, "Practical Portfolio Optimization," Numerical Algorithms Group, Tech. Report, `https://www.nag.co.uk/doc/techrep/Pdf/tr2_00.pdf`

[949] P. A. Forsyth, "An Introduction to Computational Finance Without Agonizing Pain," 2007, available online from, `https://cs.uwaterloo.ca/~paforsyt/agon.pdf`

[950] H. Ahmadi and D. Sitdhirasdr, "Portfolio Optimization is One Multiplication, the Rest is Arithmetic," *J. Appl. Fin. & Banking*, **6** 81 (2016);
`http://www.scienpress.com/download.asp?ID=1729`

[951] J. B. Heaton, N. G. Polson, and J. H.Witte, "Deep learning for finance: deep portfolios," *Appl. Stoch. Models Bus. Ind.*, **33**, 3 (2017); with discussions, *ibid.*,, p.13, and p.16, and rejoinder, p.19.

Spline Smoothing

[952] `http://pages.cs.wisc.edu/~deboor/bib/`, extensive online spline bibliography.

[953] G. Wahba, *Spline Models for Observational Data*, SIAM Publications, Philadelphia, 1990.

[954] P. J. Green and B. W. Silverman, *Nonparametric Regression and Generalized Linear Mofdels: A Roughness Penalty Approach*, Chapman & Hall, London, 1994.

[955] R. L. Eubank, *Spline Smoothing and Nonparametric Regression*, Marcel Dekker, New York, 1988.

[956] I. M. Gelfand and S. V. Fomin, *Calculus of Variations*, Dover Publications, Mineola, NY, 2000; reprint of 1963 Prentice Hall edition.

[957] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis*, Springer-Verlag, New York, 1980.

[958] J. L. Walsh, J. H. Ahlberg, and E. N. Nilson, "Best Approximation Properties of the Spline Fit," *J. Math. Mech.*, **11**, 225 (1962).

[959] I. J. Schoenberg, "Spline Functions and the Problem of Graduation," *Proc. of the Nat. Acad. Sci.*, **52**, no.4, 947 (1964).

[960] C. H. Reinsch, "Smoothing by Spline Functions," *Numer. Mathematik*, **10**, 177 (1967), and "Smoothing by Spline Functions. II," *ibid.*, **16**, 451 (1971).

[961] P. M. Anselone and P. J. Laurent, "A General Method for the Construction of Interpolating or Smoothing Spline-Functions," *Numer. Math.*, **12**, 66 (1968).

[962] D. Kershaw, "The Explicit Inverses of Two Commonly Occurring Matrices," *Math. Comp.*, **23**. 189 (1969).

[963] A. M. Erisman and W. F. Tinney, "On Computing Certain Elements of the Inverse of a Sparse Matrix," *Commun. ACM*, **18**, 177 (1975).

[964] S. Wold, "Spline Functions in Data Analysis," *Technometrics*, **16**, 1 (1974).

[965] L. L. Horowitz, "The Effects of Spline Interpolation on Power Spectral Density," *IEEE Trans. Acoust., Speech, Signal Process.*, **ASSP-22**, 22 (1974).

[966] L. D'Hooge, J. De Kerf, and M. J. Goovaerts, "Adjustment of Mortality Tables by Means of Smoothing Splines," *Bulletin de l'Association Royale des Actuaires Belge*, **71**, 78 (1976).

[967] D. L. Jupp, "B-Splines for Smoothing and Differentiating Data Sequences," *Math. Geol.*, **8**, 243 (1976).

[968] C.S. Duris, "Discrete Interpolating and Smoothing Spline Functions," *SIAM J. Numer. Anal.*, **14**, 686 (1977), and "Fortran Routines for Discrete Cubic Spline Interpolation and Smoothing," *ACM Trans. Math. Softw.*, **6**, 92 (1980).

[969] H. S. Hou and H. C. Andrews, "Cubic Splines for Image Interpolation and Digital Filtering," *IEEE Trans. Acoust., Speech, Signal Process.*, **ASSP-26**, 508 (1978).

[970] G. H. Golub, M. Heath, and G. Wahba, "Generalized Cross-Validation as a Method for Choosing a Good Ridge Parameter," *Technometrics*, **21**, 215 (1979).

[971] P. Craven and G. Wahba, "Smoothing by Spline Functions, Estimating the Correct Degree of Smoothing by the Method of Generalized Cross-Validation," *Numer. Math.*, **31**, 377 (1979).

[972] P. L. Smith, "Splines as a Useful and Convenient Statistical Tool," *Amer. Statist.*, **33**, 57 (1979).

[973] R. G. Keys, "Cubic Convolution Interpolation for Digital Image Processing," *IEEE Trans. Acoust., Speech, Signal Process.*, **ASSP-29**, 1153 (1981).

[974] J. McCutcheon, "Some Remarks on Splines,", *Trans. Fac. Actuaries*, **37**, 421 (1981).

[975] C. L. Vaughan, "Smoothing and Differentiation of Displacement-Time Data: Application of Splines and Digital Filtering," *Int. J. Bio-Med. Comput.*, **13**, 375 (1982).

[976] E. J. Wegman and I. W. Wright, "Splines in Statistics," *J. Amer. Statist. Assoc.*, **78**, 351 (1983).

[977] B. K. P. Horn, "The Curve of Least Energy," *ACM Trans. Math. Softw.*, **9**, 441 (1983).

[978] B. W. Silverman, "A Fast and Efficient Cross-Validation Method for Smoothing Parameter Choice in Spline Regression," *J. Amer. Statist. Assoc.*, **79**, 584 (1984).

[979] M. F. Hutchison and F. R. de Hoog, "Smoothing Noisy Data with Spline Functions," *Numer. Math.*, **47**, 99 (1985).

[980] B. W. Silverman, "Some Aspects of the Spline Smoothing Approach to Non-Parametric Regression Curve Fitting," *J. Roy. Statist. Soc., Ser. B*, **47**, 1 (1985).

[981] P. H. C. Eilers and B. D. Marx, "Flexible Smoothing with B-Splines and Penalties," *Statist. Sci.*, **11**, 89 (1989).

[982] K. F. Üstüner and L. A. Ferrari, "Discrete Splines and Spline Filters," *IEEE Trans. Circ. Syst.—II*, **39**, 417 (1992).

[983] M. A. A. Moussa and M. Y. Cheema, "Non-Parametric Regression in Curve Fitting," *Statistician*, **41** 209 (1992).

[984] M. Unser, A. Aldroubi, and M. Eden, "B-Spline Signal Processing: Part I—Theory," *IEEE Trans. Signal Process.*, **41**, 821 (1993), and "Part II—Efficient Design and Applications," *ibid.*, p. 834.

[985] R. L. Eubank, "A Simple Smoothing Spline," *Amer. Statist.*, **48**, 103 (1994).

[986] D. Nychka, "Splines as Local Smoothers," *Ann. Statist.*, **23**, 1175 (1995).

[987] M. Unser, "Splines, A Perfect Fit for Signal and Image Processing," *IEEE Sig. Process. Mag.*, **16**, no.6, 22, (1999).

[988] R. Champion, C. T. Lenard, and T. M. Mills, "A Variational Approach to Splines," *ANZIAM J.*, **42**, 119 (2000).

[989] V. Solo, "A Simple Derivation of the Smoothing Spline," *Amer. Statist.*, **54**, 40 (2000).

[990] S. Sun, M. B. Egerstedt, and C. F. Martin, "Control Theoretic Smoothing Splines," *IEEE Trans. Autom. Contr.*, **45**, 2271 (2000).

[991] H. Bachau, et al., "Applications of B-Splines in Atomic and Molecular Physics," *Rep. Prog. Phys.*, **64**, 1815 (2001).

[992] S. A. Dyer and J. S. Dyer, "Cubic-Spline Interpolation, Part 1,", *IEEE Instr. & Meas. Mag.*, March 2001, p. 44, and "Part 2,", *ibid.*, June 2001, p.34.

[993] J. D. Carew, et al., "Optimal Spline Smoothing of fMRI Time Series by Generalized Cross-Validation," *NeuroImage*, **18**, 950 (2003).

[994] A. K. Chaniotis and D. Poulikakos, "High Order Interpolation and Differentiation Using B-Splines," *J. Comput. Phys.*, **197**, 253 (2004).

[995] P. H. C. Eilers, "Fast Computation of Trends in Scatterplots," *Kwantitatieve Meth.*, **71**, 38 (2004).

[996] T. C. M. Lee, "Improved Smoothing Spline Regression by Combining Estimates of Different Smoothness," *Statist. Prob. Lett.*, **67**, 133 (2004).

[997] M. Unser and T. Blu, "Cardinal Exponential Splines: Part I—Theory and Filtering Algorithms," *IEEE Trans. Signal Process.*, **53**, 1425 (2005), and M. Unser, "Cardinal Exponential Splines: Part II—Think Analog, Act Digital," *ibid.*, p. 1439.

[998] H. L. Weinert, "A Fast Compact Algorithm for Cubic Spline Smoothing," *Comput. Statist. Data Anal.*, **53**, 932 (2009).

[999] G. Kimeldorf and G. Wahba, "A Correspondence Between Bayesian Estimation on Stochastic Processes and Smoothing by Splines," *Ann. Math. Statist.*, **41**, 495 (1970).

[1000] G. Kimeldorf and G. Wahba, "Some Results on Tschebycheffian Spline Functions," *J. Math. Anal. Appl.*, **33**, 82 (1971).

[1001] G. Wahba, "Improper Priors, Spline Smoothing and the Problem of Guarding Against Model Errors in Regression," *J. Roy. Statist. Soc., Ser. B*, **40**, 364 (1978).

[1002] H. L. Weinert and G. S. Sidhu, "A Stochastic Framework for Recursive Computation of Spline Functions: Part II, Interpolating Splines," *IEEE Trans. Inform. Th.*, **24**, 45 (1978).

[1003] H. L. Weinert, R. H. Byrd, and G. S. Sidhu, "A Stochastic Framework for Recursive Computation of Spline Functions: Part II, Smoothing Splines," *J. Optim. Th. Appl.*, **30**, 255 (1980).

[1004] W. E. Wecker and C. F. Ansley, "The Signal Extraction Approach to Nonlinear Regression and Spline Smoothing," *J. Amer. Statist. Assoc.*, **78**, 81 (1983).

[1005] R. Kohn and C. F. Ansley, "A New Algorithm for Spline Smoothing Based on Smoothing a Stochastic Process," *SIAM J. Stat. Comput.*, **8**, 33 (1987).

[1006] R. Kohn and C. F. Ansley, "A Fast Algorithm for Signal Extraction, Influence and Cross-Validation in State Space Models, *Biometrika*, **76**, 65 (1989).

Whittaker-Henderson Smoothing

[1007] A. Hald, "T. N. Thiele's Contributions to Statistics," *Int. Statist. Rev.*, **49**, 1 (1981), with references to Thiele's works therein.

[1008] S. L. Lauritzen, "Time Series Analysis in 1880: A Discussion of Contributions Made by T. N. Thiele," *Int. Statist. Rev.*, **49**, 319 (1981). Reprinted in S. L. Lauritzen, ed., *Thiele: Pioneer in Statistics*, Oxford Univ. Press, Oxford, New York, 2002.

[1009] G. Bohlmann, “Ein Ausgleichungsproblem,” *Nachrichten Gesellschaft Wissenschaften zu Göttingen, Mathematische-Physikalische Klasse*, no.3, p.260, (1899).

[1010] E. Whittaker, “On a New Method of Graduation,” *Proc. Edinburgh Math. Soc.*, **41**, 63 (1923).

[1011] E. Whittaker, “On the Theory of Graduation,” *Proc. Roy. Soc. Edinburgh*, **44**, 77 (1924).

[1012] E. Whittaker and G. Robinson, *The Calculus of Observations*, Blackie & Son, London, 1924.

[1013] R. Henderson, “A New Method of Graduation,” *Trans. Actuarial Soc. Am.*, **25**, 29 (1924).

[1014] R. Henderson, “Further Remarks on Graduation,” *Trans. Actuarial Soc. Am.*, **26**, 52 (1925).

[1015] A. C. Aitken, “On the Theory of Graduation,” *Proc. Roy. Soc. Edinburgh*, **46**, 36 (1925).

[1016] A. W. Joseph, “The Whittaker-Henderson Method of Graduation,” *J. Inst. Actuaries*, **78**, 99 (1952).

[1017] C. E. V. Leser, “A Simple Method of Trend Construction,” *J. Roy. Statist. Soc., Ser. B*, **23**, 91 (1961).

[1018] A. W. Joseph, “Subsidiary Sequences for Solving Leser’s Least-Squares Graduation Equations,” *J. Roy. Statist. Soc., Ser. B*, **24**, 112 (1962).

[1019] G. S. Kimeldorf and D. A. Jones, “Bayesian Graduation,” *Trans. Soc. Actuaries*, **19**, Pt.1, 66 (1967).

[1020] R. J. Shiller, “A Distributed Lag Estimator Derived from Smoothness Priors,” *Econometrica*, **41**, 775 (1973).

[1021] B. D. Cameron, et al., “Some Results of Graduation of Mortality Rates by the Whittaker-Henderson and Spline Fitting Methods,” *Bulletin de l’Association Royale des Actuaires Belge*, **71**, 48 (1976).

[1022] G. Taylor, “A Bayesian Interpretation of Whittaker-Henderson Graduation,” *Insurance: Math. & Econ.*, **11**, 7 (1992).

[1023] R. J. Verrall, “A State Space Formulation of Whittaker Graduation, with Extensions,” *Insurance: Math. & Econ.*, **13**, 7 (1993).

[1024] D. R. Schuette, “A Linear Programming Approach to Graduation”, *Trans. Soc. Actuaries*, **30**, 407 (1978); with Discussions, *ibid.*, pp. 433, 436, 440, 442, 443.

[1025] F. Y. Chan, et al., “Properties and modifications of Whittaker-Henderson graduation,” *Scand. Actuarial J.*, **1982**, 57 (1982).

[1026] F. Y. Chan, et al., “A generalization of Whittaker-Henderson graduation,” *Trans. Actuarial Soc. Am.*, **36**, 183 (1984).

[1027] F. Y. Chan, et al., “Applications of linear and quadratic programming to some cases of the Whittaker-Henderson graduation method,” *Scand. Actuarial J.*, **1986**, 141 (1986).

[1028] G. Mosheiov and A. Raveh, “On Trend Estimation of Time Series: A Simple Linear Programming Approach,” *J. Oper. Res. Soc.*, **48**, 90 (1997).

[1029] R. J. Brooks, et al., “Cross-validatory graduation,” *Insurance: Math. Econ.*, **7**, 59 (1988).

[1030] P. H. C. Eilers, “A Perfect Smoother,” *Anal. Chem.*, **75**, 3631 (2003).

[1031] W. E. Diewert and T. J. Wales, “A ‘New’ Approach to the Smoothing Problem,” in M. T. Belongia and J. M. Binner, eds., *Money, Measurement and Computation*, Palgrave Macmillan, New York, 2006.

[1032] H.L. Weinert, “Efficient Computation for Whittaker-Henderson Smoothing,” *Comput. Statist. Data Anal.*, **52**, 959 (2007).

[1033] T. Alexandrov, et al. “A Review of Some Modern Approaches to the Problem of Trend Extraction,” US Census, Statistics Report No. 2008-3, available online from `http://www.census.gov/srd/papers/pdf/rrs2008-03.pdf`.

[1034] A. S. Nocon and W. F. Scott, “An extension of the Whittaker-Henderson method of graduation,” *Scand. Actuarial J.*, **2012**, 70 (2012).

[1035] J. Vondrák, “A Contribution to the Problem of Smoothing Observational Data,” *Bull. Astron. Inst. Czech.*, **20**, 349 (1969).

[1036] J. Vondrák, “Problem of Smoothing Observational Data II,” *Bull. Astron. Inst. Czech.*, **28**,84 (1977).

[1037] J. Vondrák and A. Čepek, “Combined Smoothing Method and its Use in Combining Earth Orientation Parameters Measured by Space Techniques,” *Astron. Astrophys. Suppl. Ser.*, **147**, 347 (2000).

[1038] D. W. Zheng, et al., “Filtering GPS Time-Series using a Vondrak Filter and Cross-Validation,” *J. Geodesy*, **79**, 363 (2005).

[1039] Z-W Li, et al., “Least Squares-Based Filter for Remote Sensing Image Noise Reduction,” *IEEE Trans. Geosci. Rem. Sens.*, **46**, 2044 (2008).

[1040] Z-W Li, et al., “Filtering Method for SAR Interferograms with Strong Noise,” *Int. J. Remote Sens.*, **27**, 2991 (2006).

Hodrick-Prescott and Bandpass Filters

[1041] R. J. Hodrick and E. C. Prescott, “Postwar U.S. Business Cycles: An Empirical Investigation,” *J. Money, Credit & Banking*, **29**, 1 (1997); earlier version: Carnegie-Mellon Univ., Discussion Paper No. 451, (1980).

[1042] M. Unser, A. Aldroubi, and M. Eden, “Recursive Regularization Filters: Design, Properties, and Applications,” *IEEE Trans. Patt. Anal. Mach. Intell.*, **13**, 272 (1991).

[1043] A. C. Harvey and A. Jaeger, “Detrending, Stylized Facts and the Business Cycle,” *J. Appl. Econometr.*, **8**, 231 (1993).

[1044] R. G. King and S. T. Rebelo, “Low Frequency Filtering and Real Business Cycles,” *J. Econ. Dynam. Contr.*, **17**, 207 (1993), and appendix available online from `http://www.kellogg.northwestern.edu/faculty/rebelo/htm/LFF-Appendix.pdf`.

[1045] T. Cogley and J. M. Nason, “Effects of the Hodrick-Prescott Filter on Trend and Difference Stationary Time Series. Implications for Business Cycle Research,” *J. Econ. Dynam. Contr.*, **19**, 253 (1995).

[1046] J. Ehlgen, “Distortionary Effects of the Optimal Hodrick-Prescott Filter,” *Econ. Lett.*, **61**, 345 (1998).

[1047] U. Woitech, “A Note on the Baxter-King Filter,” Dept. Econ., Univ. Glasgow, Working Paper, No. 9813, 1998, `http://www.gla.ac.uk/media/media_22357_en.pdf`.

[1048] M. Baxter and R. G. King, “Measuring Business Cycles: Approximate Band-Pass Filters for Economic Time Series,” *Rev. Econ. Stat.*, **81**, 575 (1999).

[1049] Y. Wen and B. Zeng, “A Simple Nonlinear Filter for Economic Time Series Analysis,” *Econ. Lett.*, **64**, 151 (1999).

[1050] M. Bianchi, M. Boyle, and D. Hollingsworth, “A Comparison of Methods for Trend Estimation,” *Appl. Econ. Lett.*, **6**, 103 (1999).

[1051] P. Young and D. Pedregal, “Recursive and En-Bloc Approaches to Signal Extraction,” *J. Appl. Statist.*, **26**, 103 (1999).

[1052] J. J. Reeves, et al., “The Hodrick-Prescott Filter, a Generalization, and a New Procedure for Extracting an Empirical Cycle from a Series,” *Stud. Nonlin. Dynam. Econometr.*, **4**, 1 (2000).

[1053] D. S. G. Pollock, “Trend Estimation and De-Trending via Rational Square-Wave Filters,” *J. Econometr.*, **99**, 317 (2000).

[1054] V. Gómez, “The Use of Butterworth Filters for Trend and Cycle Estimation in Economic Time Series,” *J. Bus. Econ. Statist.*, **19**, 365 (2001).

[1055] T. M. Pedersen, “The Hodrick-Prescott Filter, the Slutzky Effect, and the Distortionary Effect of Filters,” *J. Econ. Dynam. Contr.*, **25**, 1081 (2001).

[1056] E. Slutzky, “The Summation of Random Causes as the Source of Cyclic Processes,” *Econometrica*, **37**, 105 (1937).

[1057] V. M. Guerrero, R. Juarez, and P. Poncela, “Data Graduation Based on Statistical Time Series Methods,” *Statist. Probab. Lett.*, **52**, 169 (2001).

[1058] M. O. Ravn and H. Uhlig, “On Adjusting the Hodrick-Prescott Filter for the Frequency of Observations,” *Rev. Econ. Statist.*, **84**, 371 (2002).

[1059] C. J. Murray, “Cyclical Properties of Baxter-King Filtered Time Series,” *Rev. Econ. Statist.*, **85**, 472 (2003).

[1060] A. C. Harvey and T. M . Trimbur, “General Model-Based Filters for Extracting Cycles and Trends in Economic Time Series,” *Rev. Econ. Statist.*, **85**, 244 (2003).

[1061] L. J. Christiano + T. J. Fitzgerald, “The Band Pass Filter,” *Int. Econ. Rev.*, **44**, 435 (2003).

[1062] A. Iacobucci and A. Noullez, “A Frequency Selective Filter for Short-Length Time Series,” *Comput. Econ.*, **25**, 75 (2005).

[1063] A. Guay and P. St.-Amant, “Do the Hodrick-Prescott and Baxter-King Filters Provide a Good Approximation of Business Cycles?,” *Ann. Économie Statist.*, No. 77, p. 133, Jan-Mar. 2005.

[1064] T. M. Trimbur, “Detrending Economic Time Series: A Bayesian Generalization of the Hodrick-Prescott Filter,” *J. Forecast.*, **25**, 247 (2006).

[1065] A. Maravall, A. del Río, “Temporal Aggregation, Systematic Sampling, and the Hodrick-Prescott Filter,” *Comput. Statist. Data Anal.*, **52**, 975 (2007).

[1066] V. M. Guerrero, “Estimating Trends with Percentage of Smoothness Chosen by the User,” *Int. Statist. Rev.*, **76**, 187 (2008).

[1067] T. McElroy, “Exact Formulas for the Hodrick-Prescott Filter,” *Econometr. J.*, **11**, 209 (2008).

[1068] D. E. Giles, “Constructing confidence bands for the Hodrick-Prescott filter,” *Appl. Econ. Letters*, **20**, 480 (2013).

[1069] D. S. G. Pollock, “Econometric Filters,” *Comput. Econ.*, **48**, 669 (2016).

L_1 Trend Filtering

[1070] S-J. Kim, et al., “ℓ_1 Trend Filtering,” *SIAM Rev.*, **51**, 339 (2009).

[1071] A. Moghtaderi, P. Borgnat, and P. Flandrin, “Trend Filtering: Empirical Mode Decompositions Versus ℓ_1 and Hodrick-Prescott,” *Adv. Adaptive Data Anal.*, **3**, 41 (2011).

[1072] B. Wahlberg, C. R. Rojas, and M. Annergren, “On ℓ_1 Mean and Variance Filtering,” *2011 Conf. Record 45th Asilomar Conf. Signals, Systems and Computers*, (ASILOMAR), IEEE, p. 1913, (2011).

[1073] R. J. Tibshirani, “Adaptive piecewise polynomial estimation via trend filtering,” *Ann. Stat.*. **42**, 285 (2014).

[1074] Y-X Wang, et al., “Trend Filtering on Graphs,” *Proc. 18th Int. Conf. Artif. Intell. Stat.* (AISTATS), p. 1042, May 2015.

[1075] A. Ramdas and R. J. Tibshirani, “Fast and Flexible ADMM Algorithms for Trend Filtering,” *J. Comput. Graph. Stat.*, **25**, 839 (2016).

[1076] H. Yamada and L. Jin, “Japan’s output gap estimation and ℓ_1 trend filtering,” *Empir. Econ.*, **45**, 81 (2013).

[1077] H. Yamada, “Estimating the trend in US real GDP using the ℓ_1 trend filtering,” *Appl. Econ. Letters*, 2016, p. 1.

[1078] H. Yamada and G. Yoon, “Selecting the tuning parameter of the ℓ_1 trend filter,” *Studies Nonlin. Dynam. Econometr.*, **20**, 97 (2016).

[1079] S. Selvin, et al., “ℓ_1 Trend Filter for Image Denoising,” *Procedia Comp. Sci.*, **93**, 495 (2016).

[1080] J. Ottersten, B. Wahlberg, and C. R. Rojas, “Accurate Changing Point Detection for ℓ_1 Mean Filtering,” *IEEE Sig. Process. Lett.*, **23**, 297 (2016).

Regularization

[1081] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3/e, Johns Hopkins University Press, Baltimore, 1996.

[1082] D. S. Watkins, *Fundamentals of Matrix Computations*, 2/e, Wiley, New York, 2002.

[1083] A. Björck, *Numerical Methods for Least Squares Problems*, SIAM Press, Philadelphia, 1996.

[1084] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge Univ. Press, Cambridge, 2004. Available online from:
`http://sites.google.com/site/ingridteles02/Book-ConvexOptimization.pdf`.

[1085] A. N. Tikhonov and V. Y. Arsenin, *Solution of Ill-Posed Problems*, Winston, Washington DC, 1977.

[1086] A. N. Tikhonov, et al., *Numerical Methods for the Solution of Ill-Posed Problems*, Springer, New York, 1995.

[1087] A. E. Hoerl and R. W. Kennard, “Ridge Regression: Biased Estimation for Nonorthogonal Problems,” *Technometrics*, **12**, 55 (1970).

[1088] V. V. Ivanov, *Theory of Approximate Methods and Their Application to the Numerical Solution of Singular Integral Equations*, Nordhoff International, 1976.

[1089] V. A. Morozov, *Methods for Solving Incorrectly Posed Problems*, Springer-Verlag, New York, 1984.

[1090] N. Aronszajn, “Theory of Reproducing Kernels,” *Trans. Amer. Math. Soc.*, **68**, 337 (1950).

[1091] M. Foster, “An Application of the Wiener-Kolmogorov Smoothing Theory to Matrix Inversion,” *J. SIAM*, **9**, 387 (1961).

[1092] D. L. Phillips, “A Technique for the Numerical Solution of Certain Integral Equations of the First Kind,” *J. ACM*, **9**, 84 (1962).

[1093] M. A. Aizerman, E. M. Braverman, and L. I. Rozoner, “Theoretical Foundations of the Potential Function Method in Pattern Recognition Learning,” *Autom. Remote Contr.*, **25**, 821 (1964).

[1094] J. Callum, “Numerical Differentiation and Regularization,” *SIAM J. Numer. Anal.*, **8**, 254 (1971).

[1095] L. Eld’en, “An Algorithm for the Regularization of Ill-Conditioned, Banded Least Squares Problems,” *SIAM J. Statist. Comput.*, **5**, 237 (1984).

[1096] A. Neumaier, “Solving Ill-Conditioned and Singular Linear Systems: A Tutorial on Regularization,” *SIAM Rev.*, **40**, 636 (1988).

[1097] M. Bertero, C. De Mol, and E. R. Pikes, “Linear Inverse Problems with Discrete Data: I: General Formulation and Singular System Analysis,” *Inv. Prob.*, **1**, 301 (1985).; and “II. Stability and Regularisation,” *ibid.*, **4**, 573 (1988).

[1098] M. Bertero, T. Poggio, and V. Torre, “Ill-Posed Problems in Early Vision,” *Proc. IEEE*, **76**, 869 (1988).

[1099] T. Poggio and F. Girosi, “Networks for Approximation and Learning,” *Proc. IEEE*, **78**, 1481 (1990).

[1100] A. M. Thompson, J. W. Kay, and D. M. Titterington, “Noise Estimation in Signal Restoration Using Regularization,” *Biometrika*, **78**, 475 (1991).

[1101] C. Cortes and V. Vapnik, “Support Vector Networks,” *Mach. Learn.*, **20**, 1 (1995).

[1102] F. Girosi, M. Jones, and T. Poggio, “Regularization Theory and Neural Networks Architectures,” *Neural Comput.*, **7**, 219 (1995).

[1103] A. J. Smola, B. Schölkopf, and K-R. Müller, “The Connection Between Regularization Operators and Support Vector Kernels,” *Neural Net.*, **11**, 637 (1998).

[1104] V. N. Vapnik, *Statistical Learning Theory*, Wiley, New York, 1998.

[1105] W. Fu, “Penalized Regressions: The Bridge versus the Lasso,” *J. Comput. Graph. Statist.*, **7**, 397 (1998). 1998.

[1106] V. Cherkassky and F. Mulier, *Learning from Data: Concepts, Theory, and Methods*, Wiley, New York, 1998.

[1107] F. Girosi, “An Equivalence Between Sparse Approximation and Support Vector Machines,” *Neural Comput.*, **10**, 1455 (1998).

[1108] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*, Cambridge Univ. Press, Cambridge, 2000.

[1109] T. Evgeniou, M. Pontil, and T. Poggio, “Regularization Networks and Support Vector Machines,” *Adv. Comput. Math.*, **13** 1 (2000).

[1110] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer-Verlag, New York, 2001.

[1111] F. Cucker and S. Smale, “On the Mathematical Foundations of Learning,” *Bull. AMS*, **39**, 1 (2001).

[1112] L. Tenorio, “Statistical Regularization of Inverse Problems,” *SIAM Rev.*, **43**, 347 (2001).

[1113] K-R. Müller, et al., “An Introduction to Kernel-Based Learning Algorithms,” *IEEE TRans. Neural Net.*, **12**, 181 (2001).

[1114] B. Schölkopf, R. Herbrich, and A. J. Smola, “A Generalized Representer Theorem,” *Proc. 14th Ann. Conf. Comput. Learn. Th.*, p.416, (2001).

[1115] T. Evgeniou, et al., “Regularization and Statistical Learning Theory for Data Analysis,” *Comput. Statist. Data Anal.*, **38**, 421 (2002).

[1116] F. Cucker and S. Smale, “Best Choices for Regularization Parameters in Learning Theory: On the Bias-Variance Problem,” *Found. Comput. Math.*, **2**, 413 (2002).

[1117] B. Schölkopf and A. Smola. *Learning with Kernels*, MIT Press, Cambridge, MA, 2002.

[1118] J. A. K. Suykens, et al., *Least Squares Support Vector Machines*, World Scientific, Singapore, 2002.

[1119] Z. Chen and S. Haykin, “On Different Facets of Regularization Theory,” *Neural Comput.*, **14**, 2791 (2002).

[1120] T. Poggio and S. Smale, “The Mathematics of Learning: Dealing with Data,” *Notices AMS*, **50**, no.5, 537 (2003).

[1121] M. Martínez-Ramón and C. Christodoulou, *Support Vector machines for Antenna Array Processing and Electromagnetics*, Morgan & Claypool, 2006.

[1122] M. Martínez-Ramón, et al., “Kernel Antenna Array Processing,”, *IEEE Trans. Antennas Propagat.*, **55**, 642 (2007).

[1123] M. Filippone, et al. “A Survey of Kernel and Spectral Methods for Clustering,” *Patt. Recogn.*, **41**, 176 (2008).

[1124] W. Liu, P. P. Pokharel, and J. C. Principe, “The Kernel Least-Mean-Square Algorithm,” *IEEE Trans. Signal Process.*, **56**, 543 (2008).

[1125] G. H. Golub, M. Heath, and G. Wahba, “Generalized Cross-Validation as a Method for Choosing a Good Ridge Parameter,” *Technometrics*, **21**, 215 (1979).

L_1 Regularization and Sparsity

[1126] O. J. Karst, “Linear Curve Fitting Using Least Deviations,” *J. Amer. Statist. Assoc.*, **53**, 118 (1958).

[1127] E. J. Schlossmacher, “An Iterative Technique for Absolute Deviations Curve Fitting,” *J. Amer. Statist. Assoc.*, **68**, 857 (1973).

[1128] V. A. Sposito, W. J. Kennedy and, J. E. Gentle, “Algorithm AS 110: L_p Norm Fit of a Straight Line,” *J. Roy. Statist. Soc., Series C*, **26**, 114 (1977).

[1129] R. H. Byrd, D. A. Pyne, “Convergence of the iteratively reweighted least squares algorithm for robust regression,” Tech. Report, 313, Dept. Math. Sci., Johns Hopkins University, Baltimore, MD, 1979

[1130] C. S. Burrus, 2012, “Iterative Reweighted least-squares,” *OpenStax-CNX web site*, `http://cnx. org/content/m45285/1.12`.

[1131] S. C. Narula and J. F. Wellington, “The Minimum Sum of Absolute Errors Regression: A State of the Art Survey,” *Int. Statist. Review*, **50**, 317 (1982).

[1132] R. Yarlagadda, J. B. Bednar, and T. L. Watt, “Fast algorithms for l_p deconvolution,” *IEEE Trans. Signal Process.*, **33**, 174 (1985). See also, J. A. Scales and S. Treitel, “On the connection between IRLS and Gauss’ method for l_1 inversion: Comments on ‘Fast algorithms for l_p deconvolution’,” *ibid.*, **35**, 581 (1987).

[1133] J. A. Scales, A. Gersztenkorn, and S. Treitel, “Fast l_p solution of large, sparse, linear systems: Application to seismic travel time tomography,” *J. Comput. Phys.*, **75**, 314 (1988).

[1134] G. Darche, “Iterative L^1 deconvolution,” Stanford Exploration Project, Annual Report 61, Jan. 1989; available from: `http://sepwww.stanford.edu/public/docs/sep61`.

[1135] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D*, **60**, 259 (1992).

[1136] K. Natarajan, "Sparse approximate solutions to linear systems," *SIAM J. Comput.*, **24**, 227 (1995).

[1137] R. Tibshirani, "Regression shrinkage and selection via the Lasso," *J. Roy. Statist. Soc., Ser. B*, **58**, 267 (1996).

[1138] F. Gorodnitsky and B. Rao, "Sparse signal reconstruction from limited data using FOCUSS: A reweighted norm minimization algorithm," *IEEE Trans. Signal Process.*, **45**, 600 (1997).

[1139] M. R. Osborne, B. Presnell, and B. A. Turlach, "On the LASSO and Its Dual," *J. Comput. Graph. Stat.*, **9**, 319 (2000).

[1140] S. S. Chen, D. L. Donoho, and M. A. Saunders, "Atomic decomposition by basis pursuit," *SIAM Rev.*, **43**, 129 (2001).

[1141] B. Efron, et al., "Least Angle Regression," *Ann. Statist.*, **32**, 407 (2004).

[1142] I. Daubechies, M. Defrise, and C. D. Mol, "An iterative thresholding algorithm for linear inverse problems with a sparsity constraint," *Comm. Pure Appl. Math.,* **57** 1413 (2004).

[1143] R. Tibshirani, et al., "Sparsity and smoothness via the fused Lasso," *J. Roy. Statist. Soc., Ser. B*, **67**, 91 (2005).

[1144] J-J. Fuchs, "Recovery of exact sparse representations in the presence of bounded noise." *IEEE Trans. Inform. Th.*, **51**, 3601 (2005); and, "On Sparse Representations in Arbitrary Redundant Bases," *ibid.*, **50**, 1341 (2004).

[1145] J. A. Tropp, "Just Relax: Convex Programming Methods for Identifying Sparse Signals in Noise," *IEEE Trans. Inform. Th.*, **52**, 1030 (2006).

[1146] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *J. Roy. Statist. Soc., Ser. B*, **67**, 301 (2005).

[1147] D. L. Donoho, "For most large underdetermined systems of linear equations the minimal ℓ_1-norm solution is also the sparsest solution," *Comm. Pure Appl. Math.*, **59**, 797 (2006).

[1148] E. J. Candès and T. Tao, "Decoding by linear programming," *IEEE Trans, Inform. Th.*, **51**, 4203 (2005).

[1149] E. J. Candès, J. K. Romberg, and T. Tao, "Stable signal recovery from incomplete and inaccurate measurements," *Comm. Pure Appl. Math.*, **59** 1207 (2006).

[1150] E. J. Candès, J. K. Romberg, "ℓ_1-MAGIC: Recovery of Sparse Signals via Convex Programming," User's Guide, 2006, available online from:
`https://statweb.stanford.edu/~candes/l1magic/downloads/l1magic.pdf`

[1151] D. L. Donoho, "Compressed Sensing," *IEEE Trans, Inform. Th.*, **52**, 1289 (2006).

[1152] H. Zou, T. Hastie, and R. Tibshirani, "Sparse Principal Component Analysis," *J. Comput. Graph. Stat.*, **15**, 265 (2006).

[1153] M. Aharon, M. Elad, and A. Bruckstein, "K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation," *IEEE Trans. Signal Process.*, **54**, 4311 (2006).

[1154] S-J Kim, et al., "An Interior-Point Method for Large-Scale ℓ_1-Regularized Least Squares," *IEEE J. Selected Topics Sig. Process.*, **1**, 606 (2007).

[1155] A. d'Aspremont, et al., "A direct formulation for sparse PCA using semidefinite programming," *SIAM Rev.*, **49**, 434 (2007).

[1156] M. A. T. Figueiredo, R. D. Nowak, and S. J. Wright, "Gradient projection for sparse reconstruction: Application to compressed sensing and other inverse problems," *IEEE J. Selected Topics Sig. Process.*, **1**, 586 (2007).

[1157] M. Lobo, M. Fazel, and S. Boyd, "Portfolio optimization with linear and fixed transaction costs," *Ann. Oper. Res.*, **152**, 341 (2007).

[1158] E. J. Candès and T. Tao, "The Dantzig Selector: Statistical Estimation When p Is Much Larger than n," *Ann. Statist.*, **35**, 2313 (2007); with Discussions, *ibid.*, p. 2352, 2358, 2365, 2370, 2373, 2385, 2392.

[1159] E. J. Candès, M. Wakin, and S. Boyd, "Enhancing sparsity by reweighted ℓ_1 minimization," *J. Fourier Anal. Appl.*, **14**, 877 (2008).

[1160] R. G. Baraniuk, et al., "A simple proof of the restricted isometry property for random matrices," *Constructive Approx.* **28**, 253 (2008).

[1161] E. J. Candès, "The restricted isometry property and its implications for compressed sensing," *Comptes Rendus Mathematique*, **346**, 589 (2008).

[1162] E. J. Candès and M. Wakin, "An introduction to compressive sampling," *IEEE Sig. Process. Mag.*, **25**(2), 21 (2008).

[1163] A. M. Bruckstein, D. L. Donoho, and M. Elad, "From Sparse Solutions of Systems of Equations to Sparse Modeling of Signals and Images," *SIAM Rev.*, **51**, 34 (2009).

[1164] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM J. Imaging Sci.*, **2**, 183 (2009).

[1165] H. Mohimani, M. Babaie-Zadeh, and C. Jutten, "A fast approach for overcomplete sparse decomposition based on smoothed L_0 norm," *IEEE Trans. Signal Process.*, **57**, 289 (2009).

[1166] R. E. Carrillo and K. E. Barner, "Iteratively re-weighted least squares for sparse signal reconstruction from noisy measurements," *43rd IEEE Conf. Inform. Sci. Syst.*, CISS 2009, p. 448.

[1167] A. Cohen, W. Dahmen, and R. DeVore, "Compressed sensing and best k-term approximation," *J. Amer. Math. Soc.*, **22**, 211 (2009).

[1168] E. J. Candès and Y. Plan, "Near-ideal model selection by ℓ_1 minimization," *Ann. Statist.*, **37**, 2145 (2009).

[1169] M. J. Wainwright, "Sharp Thresholds for High-Dimensional and Noisy Sparsity Recovery Using ℓ_1-Constrained Quadratic Programming (Lasso)," *IEEE Trans. Inform. Th.*, **55**, 2183 (2009).

[1170] I. Daubechies, M. Fornasier, and I. Loris, "Accelerated Projected Gradient Method for Linear Inverse Problems with Sparsity Constraints," *J. Fourier Anal. Appl.*, **14**, 764 (2008).

[1171] I. Daubechies, et al., "Iteratively reweighted least squares minimization for sparse recovery," *Comm. Pure Appl. Math.*, **63**, 1 (2010).

[1172] D. Wipf and S. Nagarajan, "Iterative reweighted ℓ_1 and ℓ_2 methods for finding sparse solutions," *IEEE J. Selected Topics Sig. Process.*, **4**, 317 (2010).

[1173] E. Van Den Berg, et al., "Algorithm 890: Sparco: A testing framework for sparse reconstruction," *ACM Trans. Math. Softw.*, **35**, 29 (2009). Sparco web site:
`http://www.cs.ubc.ca/labs/scl/sparco/`

[1174] M. Elad, *Sparse and Redundant Representations: From Theory to Applications in Signal and Image Processing*, Springer, 2010.

[1175] P. Bühlmann and S. van de Geer, *Statistics for High-Dimensional Data*, Springer, 2011.

[1176] J. Yang, and Y. Zhang, "Alternating direction algorithms for ℓ_1-problems in compressive sensing," *SIAM J. Sci. Comp.*, **33**, 250 (2011). YALL1 package available from: `http://yall1.blogs.rice.edu/`

[1177] E. J. Candès, et al. "Robust Principal Component Analysis?," *J. Assoc. Comput. Mach.*, **58**, 11 (2011).

[1178] D. Hardoon and J. Shawe-Taylor, "Sparse canonical correlation analysis," *Mach. Learn.* **83**, 331 (2011).

[1179] Z. Ma, "Sparse principal component analysis and iterative thresholding," *Ann. Stat.*, **41**, 772 (2013).

[1180] M. V. Afonso, J. M. Bioucas-Dias, and M. A. T. Figueiredo, "Fast Image Recovery Using Variable Splitting and Constrained Optimization," *IEEE Trans. Image Process.*, **19**, 2345 (2010); and "An Augmented Lagrangian Approach to the Constrained Optimization Formulation of Imaging Inverse Problems," *ibid.*, **20**, 68 (2011). SALSA software available from: `http://cascais.lx.it.pt/~mafonso/salsa.html`

[1181] S. Boyd, et al., "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, **3**(1), 3(1), 1 (2011); see also, `http://stanford.edu/~boyd/admm.html`.

[1182] N. Parikh and S. Boyd, "Proximal Algorithms," *Foundations and Trends in Optimization*, **1**, 123 (2013).

[1183] F. Bach, et al., "Optimization with Sparsity-Inducing Penalties," *Foundations and Trends in Machine Learning*, **4**(1), 1 (2012).

[1184] Y-B Zhao and D. Li, "Reweighted ℓ_1-minimization for sparse solutions to underdetermined linear systems," *SIAM J. Optim.*, **22**, 1065 (2012).

[1185] J. Mairal and B. Yu, "Complexity analysis of the lasso regularization path," *arXiv*, arXiv preprint: 1205.0079 (2012).

[1186] I. Selesnick, 2012, "Introduction to Sparsity in Signal Processing," *OpenStax-CNX web site*, `https://cnx.org/content/m43545/latest`, including MATLAB examples using SALSA [1180].

[1187] S. Foucart and H. Rauhut, *A Mathematical Introduction to Compressive Sensing*, Birkhäuser, 2013.

[1188] J. P. Brooks, J. H. Dulá, and E. L. Boone, "A Pure L_1-norm Principal Component Analysis," *Comput. Stat. Data Anal.*, **61**, 83 (2013).

[1189] R. C. Aster, B. Borchers, and C. H. Thurber, *Parameter Estimation and Inverse Problems*, 2/e, Academic Press, 2013.

[1190] R. J. Tibshirani, "The lasso problem and uniqueness," *Electr. J. Statist.*, **7**, 1456 (2013).

[1191] D. Ba, et al., "Convergence and Stability of Iteratively Re-weighted Least Squares Algorithms," *IEEE Trans. Signal Process.*, **62**, 183 (2014).

[1192] I. Rish and G. Grabarnik, *Sparse Modeling: Theory, Algorithms, and Applications,* Chapman and Hall/CRC, 2014.

[1193] T. Hastie, R. Tibshirani, and M. Wainwright, *Statistical Learning with Sparsity: The Lasso and Generalizations*, CRC Press, 2015.

[1194] C. F.Mecklenbräuker, P. Gerstoft, and E. Zöchmann, "c-LASSO and its dual for sparse signal estimation from array data," *Sig. Process.*, **130**, 204 (2017).

[1195] `http://dsp.rice.edu/cs`, Compressive Sensing Resources.

[1196] MATLAB packages for solving the L_1 regularization and related problems:

Mathworks	`https://www.mathworks.com/help/stats/lasso-and-elastic-net.html` `https://www.mathworks.com/help/stats/lasso.html`
ADMM	`http://stanford.edu/~boyd/admm.html`
CVX	`http://cvxr.com/cvx/`
FISTA	`http://ie.technion.ac.il/~becka/papers/rstls_package.zip`
Homotopy	`http://www.ece.ucr.edu/~sasif/homotopy/`
L1-MAGIC	`https://statweb.stanford.edu/~candes/l1magic/`
LARS	`https://publish.illinois.edu/xiaohuichen/code/lars/` `https://sourceforge.net/projects/sparsemodels/files/LARS/`
NESTA	`https://statweb.stanford.edu/~candes/nesta/`
REGTOOLS	`http://www.imm.dtu.dk/~pcha/Regutools/`
SALSA	`http://cascais.lx.it.pt/~mafonso/salsa.html`
SOL	`http://web.stanford.edu/group/SOL/software.html`
Sparco	`http://www.cs.ubc.ca/labs/scl/sparco/`
SpaRSA	`http://www.lx.it.pt/~mtf/SpaRSA/`
Sparselab	`https://sparselab.stanford.edu/`
SPGL1	`http://www.cs.ubc.ca/labs/scl/spgl1/`
TwIST	`http://www.lx.it.pt/~bioucas/TwIST/TwIST.htm`
YALL1	`http://yall1.blogs.rice.edu/`

X-11 Seasonal Adjustment Method

[1197] J. Shiskin, A. Young, and J. Musgrave, "The X-11 Variant of the Census Method II Seasonal Adjustment Program," US Census Bureau, Technical Paper 15, (1967), available from [1201].

[1198] E. B, Dagum, "The X-11-ARIMA Seasonal Adjustment Method," Statistics, Canada, (1980), available from [1201].

[1199] `http://www.census.gov/srd/www/x12a/`, US Census Bureau X-12-ARIMA Seasonal Adjustment Program.

[1200] `http://www.census.gov/srd/www/sapaper/sapaper.html`, US Census Bureau Seasonal Adjustment Papers.

[1201] `http://www.census.gov/srd/www/sapaper/historicpapers.html`, Historical Papers on X-11 and Seasonal Adjustment.

[1202] K. F. Wallis, "Seasonal Adjustment and Relations Between Variables," *J. Amer. Statist. Assoc.*, **69**, 18 (1974).

[1203] K. F. Wallis, "Seasonal Adjustment and Revision of Current Data: Linear Filters for the X-11 Method," *J. Roy. Statist. Soc., Ser. A*, **145**, 74 (1982).

[1204] W. R. Bell and S. C. Hillmer, "Issues Involved with Seasonal Adjustment of Economic Time Series," *J. Bus. Econ. Statist.*, **2**, 291 (1984). Available on line from `http://www.census.gov/srd/papers/pdf/rr84-09.pdf`.

[1205] W. R. Bell and B. C. Monsell, "X-11 Symmetric Linear Filters and their Transfer Functions," US Census Bureau, SRD Research Report, No. RR-92/15, (1992). Available online from the web site [1200].

[1206] E. B. Dagum, N. Chhab, and K. Chiu, "Derivation and Properties of the X11ARIMA and Census X11 Linear Filters," *J. Official Statist.*, **12**, 329 (1996).

[1207] J. C. Musgrave, "A Set of Weights to End all End Weights," Working paper, US Dept. Commerce, (1964), available online from [1201].

[1208] M. Doherty, "The Surrogate Henderson Filters in X-11", *Aust. N. Z. J. Stat.*, **43**, 385 (2001), originally circulated in 1996.

[1209] D. F. Findley, et al., "New Capabilities and Methods of the X-12-ARIMA Seasonal-Adjustment Program," *J. Bus. Econ. Statist.*, **16**, 127 (1998), with Comments, p.153.

[1210] D. Ladiray and B. Quenneville, *Seasonal Adjustment with the X-11 Method*, Lecture Notes in Statistics No. 158, Springer-Verlag, New York, 2001. Available online from the web site [1200] (in French and Spanish.)

[1211] A. G. Gray and P. J. Thomson, "On a Family of Finite Moving-Average Trend Filters for the Ends of Series," *J. Forecasting*, **21**, 125 (2002).

[1212] B. Quenneville, D. Ladiray, and B. Lefrançois, "A Note on Musgrave Asymmetrical Trend-Cycle Filters," *Int. J. Forecast.*, **19**, 727 (2003).

[1213] D. F. Findley and D. E. K. Martin, "Frequency Domain Analysis of SEATS and X-11/X-12-ARIMA Seasonal Adjustment Filters for Short and Moderate-Length Time Series," *J. Off, Statist.*, **22**, 1 (2006).

[1214] C. E. V. Leser, "Estimation of Quasi-Linear Trend and Seasonal Variation," *J. Amer. Statist. Assoc.*, **58**, 1033 (1963).

[1215] H. Akaike, "Seasonal Adjustment by a Bayesian Modeling," *J. Time Ser. Anal.*, **1**, 1 (1980).

[1216] E. Schlicht, "A Seasonal Adjustment Principle and a Seasonal Adjustment Method Derived from this Principle," *J. Amer. Statist. Assoc.*, **76**, 374 (1981).

[1217] F. Eicker, "Trend-Seasonal Decomposition of Time Series as Whittaker-Henderson Graduation," *Statistics*, **19**, 313 (1988).

Model-Based Seasonal Adjustment

[1218] E. J. Hannan, "The Estimation of Seasonal Variation in Economic Time Series," *J. Amer. Statist. Assoc.*, **58**, 31 (1963).

[1219] E. J. Hannan, "The Estimation of Changing Seasonal Pattern," *J. Amer. Statist. Assoc.*, **59**, 1063 (1964).

[1220] M. Nerlove, "Spectral Analysis of Seasonal Adjustment Procedures," *Econometrica*, **32**, 241 (1964).

[1221] J. P. Burman, "Moving Seasonal Adjustment of Economic Time Series," *J. Roy. Statist. Soc., Ser. A*, **128**, 534 (1965).

[1222] D. M. Grether and M. Nerlove, "Some Properties of "Optimal" Seasonal Adjustment," *Econometrica*, **38**, 682 (1970).

[1223] G. E. P. Box, S. Hillmer, and G. C. Tiao, "Analysis and Modeling of Seasonal Time Series," (1978), available online from [1201].

[1224] J. P. Burman, "Seasonal Adjustment by Signal Extraction," *J. Roy. Statist. Soc., Ser. A*, **143**, 321 (1980).

[1225] S. C. Hillmer and G. C. Tiao, "An ARIMA-Model-Based Approach to Seasonal Adjustment," *J. Amer. Statist. Assoc.*, **77**, 63 (1982).

[1226] W. S. Cleveland, A. E, Freeny, and T. E. Graedel, "The Seasonal Component of Atmospheric CO_2: Information from New Approaches to the Decomposition of Seasonal Time Series," *J. Geoph. Res.*, **88**, 10934 (1983).

[1227] P. Burridge and K. F. Wallis, "Unobserved-Components Models for Seasonal Ajustment Filters," *J. Bus. Econ. Statist.*, **2**, 350 (1984).

[1228] G. Kitagawa and W. Gersch, "A Smoothness Priors-State Space Modeling of Time Series with Trend and Seasonality," *J. Amer. Statist. Assoc.*, **79**, 378 (1984).

[1229] R. B. Cleveland, et al., "STL: A Seasonal-Trend Decomposition Procedure Based on Loess," *J. Official Statist.*, **6**, 3 (1990).

[1230] G. Kitagawa and W. Gersch, *Smoothness Priors Analysis of Time Series*, Springer, New York, 1996.

[1231] V. Gómez and A. Maravall, "Programs TRAMO and SEATS. Instructions for the User (with some updates)," Working Paper 9628, (Servicio de Estudios, Banco de España, 1996).

[1232] C. Planas, "The Analysis of Seasonality In Economic Statistics: A Survey of Recent Developments," *Qüestió*, **22**, 157 (1998).

[1233] V. Gómez and A. Maravall, "Seasonal Adjustment and Signal Extraction in Economic Time Series," chapter 8, in *A Course in Time Series Analysis*, D. Peña, G. C. Tiao, and R. S. Tsay, eds., Wiley, New York, 2001. Available online from `http://bde.es/servicio/software/tramo/sasex.pdf`.

[1234] J. A.D. Aston, et al., "New ARIMA Models for Seasonal Time Series and Their Application to Seasonal Adjustment and Forecasting," US Census Bureau, (2007), available online from [1200].

Unobserved Components Models

[1235] E. J. Hannan, "Measurement of a Wandering Signal Amid Noise," *J. Appl. Prob.*, **4**, 90 (1967).

[1236] E. L. Sobel, "Prediction of a Noise-Distorted, Multivariate, Non-Stationary Signal," *J. Appl. Prob.*, **4**, 330 (1967).

[1237] W. P. Cleveland and G. C. Tiao, "Decomposition of Seasonal Time Series: A Model for the Census X-11 Program," *J. Amer. Statist. Assoc.*, **71**, 581 (1976).

[1238] D. A. Pierce, "Signal Extraction Error in Nonstationary Time Series," *Ann. Statist.*, **7**, 1303 (1979).

[1239] W. Bell, "Signal Extraction for Nonstationary Time Series," *Ann. Statist.*, **12**, 646 (1984), with correction, *ibid.*, **19**, 2280 (1991).

[1240] A. Maravall, "A Note on Minimum Mean Squared Error Estimation of Signals with Unit Roots," *J. Econ. Dynam. & Contr.*, **12**, 589 (1988).

[1241] W. R. Bell and E. K. Martin, "Computation of Asymmetric Signal Extraction Filters and Mean Squared Error for ARIMA Component Models," *J. Time Ser. Anal.*, **25**, 603 (2004). Available online from [1200].

[1242] S. Beveridge and C. Nelson, "A New Approach to Decomposition of Economic Time Series into Permanent and Transitory Components with Particular Attention to Measurement of the Business Cycle," *J. Monet. Econ.*, **7**, 151 (1981).

[1243] V. Gómez and A. Maravall, "Estimation, Prediction, and Interpolation for Nonstationary Series with the Kalman Filter," *J. Amer. Statist. Assoc.* **89**, 611 (1994).

[1244] P. Young, "Data-Based Mechanistic Modelling of Environmental, Ecological, Economic, and Engineering Systems," *Environ. Model. & Soft.*, **13**, 105 (1998).

[1245] V. Gómez, "Three Equivalent Methods for Filtering Finite Nonstationary Time Series," *J. Bus. Econ. Stat.*, **17**, 109 (1999).

[1246] A. C. Harvey and S. J. Koopman, "Signal Extraction and the Formulation of Unobserved Components Modelsm" *Econometr. J.*, **3**, 84 (2000).

[1247] R. Kaiser and A. Maravall, *Measuring Business Cycles in Economic Time Series*, Lecture Notes in Statistics, **154**, Springer-Verlag, New York, 2001. Available online from `http://www.bde.es/servicio/software/tramo/mhpfilter.pdf`.

[1248] E. Ghysels and D. R. Osborn, *The Econometric Analysis of Seasonal Time Series*, Cambridge Univ. Press, Cambridge, 2001.

[1249] D. S. G. Pollock, "Filters for Short Non-Stationary Sequences," *J. Forecast.*, **20**, 341 (2001).

[1250] R. Kaiser and A. Maravall, "Combining Filter Design with Model-Based Filtering (with an Application to Business Cycle Estimation)," *Int. J. Forecast.*, **21** 691 (2005).

[1251] A. Harvey and G. De Rossi, "Signal Extraction," in *Palgrave Handbook of Econometrics*, vol 1, K. Patterson and T. C. Mills, eds., 2006, Palgrave MacMillan, New York, 2006.

[1252] A. Harvey, "Forecasting with Unobserved Components Time Series Models," *Handbook of Economic Forecasting*, G. Elliot, C. Granger, and A. Timmermann, eds., North Holland, 2006.

[1253] D. S. G. Pollock, "Econometric Methods of Signal Extraction," *Comput. Statist. Data Anal.*, **50**, 2268 (2006).

[1254] M. Bujosa, A. Garcia-Ferrer, and P. C. Young, "Linear Dynamic Harmonic Regression," *Comput. Statist. Data Anal.*, **52**, 999 (2007).

[1255] T. McElroy, "Matrix Formulas for Nonstationary ARIMA Signal Extraction," *Econometr. Th.*, **24**, 988 (2008).

[1256] M. Wildi, *Real-Time Signal Extraction*, Springer, New York, 2008. Available online from `www.idp.zhaw.ch/fileadmin/user_upload/engineering/_Institute_und_Zentren/IDP/sonderthemen/sef/signalextraction/papers/IDP-WP-08Sep-01.pdf`.

Random Number Generators

[1257] D. E. Knuth, *The Art of Computer Programming*, vol. 2, 2nd ed., Addison-Wesley, Reading, MA, 1981.

[1258] P. Bratley, B. L. Fox, and L. Schrage, *A Guide to Simulation*, Springer-Verlag, New York, 1983.

[1259] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C*, 2nd ed., Cambridge Univ. Press, New York, 1992.

[1260] S. K. Park and K. W. Miller, "Random Number Generators: Good Ones Are Hard to Find," *Comm. ACM*, **31**, 1192 (1988).

[1261] B. D. Ripley, "Computer Generation of Random Variables: A Tutorial," *Int. Stat. Rev.*, **51**, 301 (1983).

[1262] D. J. Best, "Some Easily Programmed Pseudo-Random Normal Generators," *Austr. Comput. J.*, **11**, 60 (1979).

[1263] C. A. Whitney, "Generating and Testing Pseudorandom Numbers," *BYTE*, (October 1984), p. 128.

[1264] S. Kirkpatrick and E. Stoll, "A Very Fast Shift-Register Sequence Random Number Generator," *J. Computational Phys.*, **40**, 517 (1981).

[1265] G. Marsaglia, "A Current View of Random Number Generators," in *Proceedings, Computer Science and Statistics, 16th Symposium on the Interface*, L. Billard, ed., Elsevier, North-Holland, 1985.

[1266] G. Marsaglia, "Toward a Universal Random Number Generator," *Statist. & Prob. Lett.*, **8**, 35 (1990).

[1267] G. Marsaglia and A. Zaman, "A New Class of Random Number Generators," *Annals Appl. Prob.*, **1**, 462 (1991).

[1268] G. Marsaglia and A. Zaman, "Some Portable Very-Long Random Number Generators," *Comput. Phys.*, **8**, 117 (1994).

[1269] P. Lewis, A. Goodman, and J. Miller, "A Pseudo-Random Number Generator for the System/360," *IBM Syst. J.*, **8**, 136 (1969).

[1270] L. Schrage, "A More Portable Fortran Random Number Generator," *ACM Trans. Math. Soft.*, **5**, 132 (1979).

[1271] C. Bays and S. D. Durham, "Improving a Poor Random Number Generator," *ACM Trans. Math. Soft.*, **2**, 59 (1976).

[1272] G. Marsaglia and T. A. Bray, "On-line Random Number Generators and their Use in Combinations," *Comm. ACM*, **11**, 757 (1968).

[1273] B. A. Wichmann and I. D. Hill, "An Efficient and Portable Pseudo-Random Number Generator," *Appl. Stat.*, **31**, 188 (1982).

[1274] B. A. Wichmann and D. Hill, "Building a Random Number Generator," *BYTE*, March 1987, p. 127.

[1275] D. Lorrain, "A Panoply of Stochastic Cannons," in Ref. [113], p. 351.

[1276] Suggested by Dr. N. Moayeri, private communication, 1993.

[1277] W. A. Gardner, *Introduction to Random Processes*, 2nd ed., McGraw-Hill, New York, 1989.

$1/f$ Noise

[1278] D. A. Bell, *Noise and the Solid State*, Halsted Press, (Wiley), New York, 1985.

[1279] M. Gardner, "White and Brown Music, Fractal Curves and One-Over-f Fluctuations," *Sci. Amer.*, **288**, 16 (1978).

[1280] M. S. Keshner, "$1/f$ Noise," *Proc. IEEE*, **70**, 212 (1982).

[1281] B. B. Mandelbrot, *The Fractal Geometry of Nature*, W. H. Freeman, New York, 1983.

[1282] W. H. Press, "Flicker Noise in Astronomy and Elsewhere," *Comments Astrophys.*, **7**, no. 4, 103 (1978).

[1283] M. Schroeder, *Fractals, Chaos, Power Laws*, W. H. Freeman, New York, 1991.

[1284] R. F. Voss and J. Clark, "$1/f$ Noise in Music and Speech," *Nature*, **258**, no. 5533, 317 (1975).

[1285] R. F. Voss and J. Clark, "$1/f$ Noise in Music: Music from $1/f$ Noise," *J. Acoust. Soc. Amer.*, **63**, 258 (1978).

[1286] B. J. West and M. Shlesinger, "The Noise in Natural Phenomena," *Amer. Scientist*, **78**, 40, Jan-Feb, (1990).

[1287] A. van der Ziel, *Noise in Solid State Devices and Circuits*, Wiley, New York, 1986.

[1288] M. Reichbach, “Modeling $1/f$ Noise by DSP,” Graduate Special Problems Report, ECE Department, Rutgers University, Fall 1993.

[1289] R. Bristow-Johnson, reported in `comp.dsp` newsgroup, Jan. 1995, and private communication.

Prolate Spheroidal Wave Functions

[1290] D. Slepian and H. O. Pollak, "Prolate Spheroidal Wave Functions, Fourier Analysis, and Uncertainty—I", *Bell Syst. Tech. J.*, **40**, no.1, 43 (1961), available online from:
`https://archive.org/details/bstj40-1-43`

[1291] H. J. Landau and H. O. Pollak, “Prolate Spheroidal Wave Functions, Fourier Analysis and Uncertainty—II,” *Bell Syst. Tech. J.*, **40**, no.1, 65 (1961), available online from:
`https://archive.org/details/bstj40-1-65`

[1292] H. J. Landau and H. O. Pollak, “Prolate Spheroidal Wave Functions, Fourier Analysis and Uncertainty—III: The Dimension of the Space of Essentially Time- and Band-Limited Signals,” *Bell Syst. Tech. J.*, **41**, no.4, 1295 (1962), available online from:
`https://archive.org/details/bstj41-4-1295`

[1293] D. Slepian, “Prolate Spheroidal Wave Functions, Fourier Analysis and Uncertainty—IV: Extensions to Many Dimensions; Generalized Prolate Spheroidal Functions,” *Bell Syst. Tech. J.*, **43**, no.6, 3009 (1964), available online from: `https://archive.org/details/bstj43-6-3009`

[1294] D. Slepian, “Prolate Spheroidal Wave Functions, Fourier Analysis and Uncertainty—V: The Discrete Case,” *Bell Syst. Tech. J.*, **57**, 1371 (1978). Available online from:
`https://archive.org/details/bstj57-5-1371`

[1295] D. Slepian, “Some Comments on Fourier Analysis, Uncertainty and Modeling,” *SIAM Rev.*, **25**, 379 (1983).

[1296] D. Slepian and E. Sonnenblick, “Eigenvalues Associated with Prolate Spheroidal Wave Functions of Zero Order,” *Bell Syst. Tech. J.*, **44**, no.8, 1745 (1965), available online from:
`https://archive.org/details/bstj44-8-1745`

[1297] D. Slepian, “On Bandwidth,” *Proc. IEEE*, **64**, 292 (1976).

[1298] H. J. Landau, “An Overview of Time and Frequency Limiting,” in *Fourier Techniques and Applications*, J. F. Price (ed.), Plenum, New York, 1985.

[1299] H. J. Landau, “Extrapolating a Band-Limited Function from its Samples Taken in a Finite Interval,” *IEEE Trans. Inform. Th.*, **IT-32**, 464 (1986).

[1300] B. R. Frieden, “Evaluation, Design and Extrapolation Methods for Optical Signals, Based on Use of Prolate Functions,” *Progr. Optics*, **9**, 311 (1971).

[1301] C. J. Bouwkamp, “On spheroidal wave functions of order zero,” *J. Math. and Physics*, **26**, 79 (1947).

[1302] C. J. Bouwkamp, “On the theory of spheroidal wave functions of order zero,” *Nederl. Akad. Wetensch., Proc.*, p.931 (1950), online:
`http://www.dwc.knaw.nl/DL/publications/PU00018840.pdf`

[1303] C. J. Bouwkamp, “Theoretical and numerical treatment of diffraction through a circular aperture,” *IEEE Trans. Antennas Propagat.*, **18**, 152 (1970).

[1304] C. Flammer, *Spheroidal Wave Functions*, Stanford Univ. Press, Stanford, CA, 1957.

[1305] D. R. Rhodes, “On the Spheroidal Functions,” *J. Res. Nat. Bureau of Standards - B. Mathematical Sciences*, **74B**, no. 3, 187 (1970), available online from:
`http://nvlpubs.nist.gov/nistpubs/jres/74B/jresv74Bn3p187_A1b.pdf`

[1306] F. W. J. Olver, et al, (Eds.), “NIST Handbook of Mathematical Functions,” Chapter 30, NIST and Cambridge University Press, 2010, available online from: `http://dlmf.nist.gov/30`

[1307] D. B. Hodge, “Eigenvalues and Eigenfunctions of the Spheroidal Wave Equation,” *J. Math. Phys.*, **11**, 2308 (1970).

[1308] T. A. Beu and R. I. Câmpeanu, “Prolate Radial Spheroidal Wave Functions,” *Computer Phys. Commun.*, **30**, 177 (1983).

[1309] M. B. Kozin, V. V. Volkov, and D. I. Svergun, “A Compact Algorithm for Evaluating Linear Prolate Functions,” *IEEE Trans. Signal Proc.*, **45**, 1075 (1997).

[1310] Le-Wei Li, et al., “Computations of spheroidal harmonics with complex arguments: A review with an algorithm,” *Phys. Rev. E*, **58**, 6792 (1998).

[1311] H. Xiao, V. Rokhlin, and N. Yarvin, “Prolate spheroidal wavefunctions, quadrature and interpolation,” *Inverse Problems*, **17**, 805 (2001).

[1312] F. A. Grünbaum and L. Miranian, “Magic of the prolate spheroidal functions in various setups,” *Proc. SPIE 4478, Wavelets: Applications in Signal and Image Processing IX*, p.151, Dec. 2001.

[1313] B. Larsson, T. Levitina, and E. J. Brändas, “On prolate spheroidal wave functions for signal processing,” *Int. J. Quantum Chem.*, **85**, 392 (2001).

[1314] K. Khare and N. George, “Sampling theory approach to prolate spheroidal wavefunctions,” *J. Phys. A: Math. Gen.*, **36**, 10011 (2003).

[1315] P. E. Falloon, P. C. Abbott, and J. B. Wang, “Theory and computation of spheroidal wavefunctions,” *J. Phys. A: Math. Gen.*, **36**, 5477 (2003).

[1316] I. C. Moore and M. Cada, “Prolate spheroidal wave functions, an introduction to the Slepian series and its properties,” *Appl. Comput. Harmon. Anal.*, **16**, 208 (2004).

[1317] J. P. Boyd, “Algorithm 840: Computation of Grid Points, Quadrature Weights and Derivatives for Spectral Element Methods Using Prolate Spheroidal Wave Functions-Prolate Elements,” *ACM Trans. Math. Software*, **31**, no. 1, 149 (2005).

[1318] G. Walter and T. Soleski, “A new friendly method of computing prolate spheroidal wave functions and wavelets,” *Appl. Comput. Harmon. Anal.*, **19** 432 (2005).

[1319] P. Kirby, “Calculation of spheroidal wave functions,” *Computer Phys. Commun.*, **175**, 465 (2006).

[1320] A. Karoui and T. Moumni, “New efficient methods of computing the prolate spheroidal wave functions and their corresponding eigenvalues,” *Appl. Comput. Harmon. Anal.*, **24**, 269 (2008).

[1321] A. Karoui, “Unidimensional and bidimensional prolate spheroidal wave functions and applications,” *J. Franklin Inst.*, **348**, 1668 (2011).

[1322] J. A. Hogan and J. D. Lakey, *Duration and Bandwidth Limiting: Prolate Functions, Sampling, and Applications*, Springer Birkhäuser, New York, 2012.

[1323] A. Osipov, V. Rokhlin, and H. Xiao, *Prolate Spheroidal Wave Functions of Order Zero*, Springer, New York, 2013.

[1324] R. Adelman, N. A. Gumerov, and R. Duraiswami, “Software for Computing the Spheroidal Wave Functions Using Arbitrary Precision Arithmetic,” *arXiv:1408.0074* [cs.MS], 2014, available online from: `http://arxiv.org/abs/1408.0074`

[1325] N. C. Gallagher, Jr. and G. L. Wise, “A Representation for Band-Limited Functions,” *Proc. IEEE*, **63**, 1624 (1975).

[1326] H-P Chang, T. K. Sarkar, and O. M. C. Pereira-Filho, “Antenna Pattern Synthesis Utilizing Spherical Bessel Functions,” *IEEE Trans. Antennas Propagat.*, **48**, 853 (2000).

Superoscillations

[1327] F. Bond and C. Cahn, “On the sampling the zeros of bandwidth limited signals,” *IRE Ttans. Inform. Th.*, **4**, 110 (1958).

[1328] M. V. Berry, “Faster than Fourier,” in *Quantum Coherence and Reality, In Celebration of the 60th Birthday of Yakir Aharonov*, J. S. Anandan and J. L. Safko, eds., International Conference on Fundamental Aspects of Quantum Theory, p.55, World Scientific, 1994.

[1329] W. Qiao, “A simple model of Aharonov-Berry’s superoscillations,” *J. Phys. A: Math. Gen.*, **29**, 2257 (1996).

[1330] M. S. Calder and A. Kempf, “Analysis of superoscillatory wave functions,” *J. Math. Phys.*, **46**, 012101 (2005).

[1331] P. J. S. G. Ferreira and A. Kempf, “Superoscillations: Faster Than the Nyquist Rate,” *IEEE Trans. Signal Proc.*, **54**, 3732 (2006).

[1332] M. V. Berry and S. Popescu, “Evolution of quantum superoscillations and optical super-resolution without evanescent waves, *J. Phys. A: Math. Gen.*, **39**, 6965 (2006).

[1333] P. J. S. G. Ferreira, A. Kempf, and M. J. C. S. Reis, “Construction of Aharonov-Berry’s superoscillations,” *J. Phys. A: Math. Theor.*, **40**, 5141 (2007).

[1334] F. M. Huang, Y. Chen, F. J. Garcia de Abajo, and N. I. Zheludev “Optical super-resolution through super-oscillations,” *J. Optics A*, **9**, S285 (2007)

[1335] M. R. Dennis, A. C. Hamilton, and J. Courtial, “Superoscillation in speckle patterns,” *Opt. Lett.*, **33**, 2976 (2008).

[1336] Y. Aharonov, et al., “Some mathematical properties of superoscillations,” *J. Phys. A: Math. Theor.*, **44** 365304 (2011).

[1337] E. Katzav and M. Schwartz, “Yield-Optimized Superoscillations,” *IEEE Trans. Signal Proc.*, **61**, 3113 (2013).

[1338] D. G. Lee and P. J. S. G. Ferreira, “Direct Construction of Superoscillations," *IEEE Trans. Signal Proc.*, **62**, 3125 (2014).

[1339] D. G. Lee and P. J. S. G. Ferreira, “Superoscillations With Optimum Energy Concentration," *IEEE Trans. Signal Proc.*, **62**, 4857 (2014).

[1340] M. V. Berry, “Superoscillations, Endfire and Supergain,” Ch.21 in D .C. Struppa and J.M. Tollaksen, Eds., *Quantum Theory: A Two-Time Success Story*, Springer-Verlag Italia 2014.

Superresolution, Restoration, Degrees of Freedom

[1341] J. Lindberg, “Mathematical concepts of optical superresolution,” *J. Opt.*, **14**, 083001 (2012).

[1342] H. Wolter, “On Basic Analogies and Principal Differences Between Optical and Electronic Information,” *Progr. Optics*, **1**, 157 (1961).

[1343] J. L. Harris, “Diffraction and resolving power,” *J. Opt. Soc. Am.*, **54**, 931 (1964).

[1344] M. Bertero and E. R. Pike, “Resolution in diffraction-limited imaging, a singular value analysis,” *Opt. Acta*, **29**, 727 (1982); and, *ibid.*, **29**, 1599 (1982).

[1345] M. Bertero and C, de Mol, “Super-resolution by data inversion,” *Progr. Optics*, **36**, 129 (1996).

[1346] M. Bertero and P. Boccacci, *Introduction to Inverse Problems in Imaging*, CRC press, 1998.

[1347] M. Bertero and P. Boccacci, “Super-resolution in computational imaging,” *Micron*, **34**, 265 (2003).

[1348] A. J. den Dekker and A. van den Bos, “Resolution: a survey,” *J. Opt. Soc. Am.*, **A-14**, 547 (1997).

[1349] N. I. Zheludev, “What diffraction limit?,” *Nature Mat.*, **7**, 420 (2008).

[1350] F. M. Huang and N. I. Zheludev, “Super-Resolution without Evanescent Waves,” *Nano Lett.*, **9**, 1249 (2009).

[1351] E. Rogers, et al., “A super-oscillatory lens optical microscope for subwavelength imaging,” *Nature Mat.*, **11**, 432 (2012).

[1352] H. J. Hyvärinen, et al., “Limitations of superoscillation filters in microscopy applications,” *Opt. Lett.*, **37**, 903 (2012).

[1353] A. M. H.Wong and G. V. Eleftheriades, “Adaptation of Schelkunoff’s Superdirective Antenna Theory for the Realization of Superoscillatory Antenna Arrays,” *IEEE Ant. Wireless Propag. Lett.*, **9**, 315 (2010).

[1354] K. G. Makris and D. Psaltis, “Superoscillatory diffraction-free beams,” *Opt. Lett.*, **36**, 4335 (2011).

[1355] A. M. H. Wong and G. V. Eleftheriades, “Temporal Pulse Compression Beyond the Fourier Transform Limit,” *IEEE Trans. Microwave Theory Tech.*, **59** 2173 (2011).

[1356] A. M. H. Wong and G. V. Eleftheriades, “Advances in Imaging Beyond the Diffraction Limit,” *Photonics J., IEEE*, **4**, 586 (2012).

[1357] E. Greenfield, et al., “Experimental generation of arbitrarily shaped diffractionless superoscillatory optical beams,” *Opt. Express*, **21**, 13425 (2013).

[1358] A. M. H. Wong and G. V. Eleftheriades, “Superoscillations without Sidebands: Power-Efficient Sub-Diffraction Imaging with Propagating Waves,” *Scientific Reports*, **5**, no.8449, 1 (2015).

[1359] R. W. Gerchberg, “Super-resolution through error energy reduction,” *Opt. Acta*, **21**, 709 (1974).

[1360] A. Papoulis, “A new algorithm in spectral analysis and band-limited extrapolation,” *IEEE Trans. Circ. Syst.,* **22**, 735 (1975).

[1361] C. Pask, “Simple optical theory of super-resolution,” *J. Opt. Soc. Am.*, **66**, 68 (1976).

[1362] G. A. Viano, “On the extrapolation of optical image data,” *J. Math. Phys.*, **17**, 1160 (1976).

[1363] M. Bertero, C. de Mol, and G. A. Viano, “On the problems of object restoration and image extrapolation in optics,” *J. Math. Phys.*, **20**, 509 (1979).

[1364] C. K. Rushforth and R. L. Frost, “Comparison of some algorithms for reconstructing space-limited images,” *J. Opt. Soc. Am.*, **70**, 1539 (1980).

[1365] C. W. Barnes, “Object Restoration in a Diffraction-Limited Imaging System,” *J. Opt. Soc. Am.*, **56**, 575 (1966).

[1366] B. R. Frieden, “Band-Unlimited Reconstruction of Optical Objects and Spectra,” *J. Opt. Soc. Am.*, **57**, 1013 (1967).

[1367] R. B. Frieden, “On arbitrarily perfect imagery with a finite aperture,” em J. Mod. Opt., **16**, 795 (1969).

[1368] W. Lukosz, “Optical Systems with Resolving Powers Exceeding the Classical Limit,” *J. Opt. Soc. Am.*, **56**, 1463 (1966); see also, part II, *ibid.*, **57**, 932 (1967);

[1369] C. K. Rushford and R. W. Harris, “Restoration, Resolution, and Noise,” *J. Opt. Soc. Am.*, **58**, 539 (1968).

[1370] J. A. Bucklew and B. E. A. Saleh, “Theorem for high-resolution high-contrast image synthesis,” *J. Opt. Soc. Am.*, **A-2**, 1233 (1985).

[1371] D. L. Donoho and P. B. Stark, “Uncertainty principles and signal recovery,” *SIAM J. Math.*, **49**, 906 (1989).

[1372] R. Piestun and D. A. B. Miller, “Electromagnetic degrees of freedom of an optical system,” *J. Opt. Soc. Am.*, **A-17**, 892 (2000).

[1373] M. I. Kolobov and C. Fabre, “Quantum Limits on Optical Resolution,” *Phys. Rev. Lett.*, **85** 3789 (2000).

[1374] V. N. Beskrovnyy and M. I. Kolobov, “Quantum limits of super-resolution in reconstruction of optical objects,” *Phys. Rev.*, **A-71**, 043802 (2005).

[1375] M. I. Kolobov and V. N. Beskrovnyy, “Quantum theory of super-resolution for optical systems with circular apertures,” *Opt. Commun.*, **264**, 9 (2006).

[1376] V. N. Beskrovnyy and M. I. Kolobov, “Quantum-statistical analysis of superresolution for optical systems with circular symmetry,” *Phys. Rev.*, **A-78**, 043824 (2008);

[1377] C. L. Matson and D. W. Tyler, “Primary and secondary superresolution by data inversion,” *Opt. Express*, **14**, 456 (2006).

[1378] M. Z. Nashed, “Operator-Theoretic and Computational Approaches to Ill-Posed Problems with Applications to Antenna Theory,” *IEEE Trans. Antennas Propagat.*, **AP-29**, 220 (1981).

[1379] F. Yaman, V. G. Yakhno, and R. Potthast, “A Survey on Inverse Problems for Applied Sciences,” *Mathematical Problems in Engineering*, vol. 2013 (2013), Article ID 976837, available online from: `http://www.hindawi.com/journals/mpe/2013/976837/`

[1380] G. C. Sherman, “Integral-Transform Formulation of Diffraction Theory,” *J. Opt. Soc. Am.*, **57**, 1490 (1967).

[1381] J. R. Shewell and E. Wold, “Inverse Diffraction and a New Reciprocity Theorem,” *J. Opt. Soc. Am.*,, **58**, 1596 (1968).

[1382] J. J. Stamnes, “Focusing of two-dimensional waves,” *J. Opt. Soc. Am.*, **71**, 15 (1981).

[1383] J. J. Stamnes, “Focusing of a perfect wave and the Airy pattern formula,” *Opt. Commun.*, **37**, 311 (1981).

[1384] J. J. Stamnes, *Waves in Focal Regions*, Taylor & Francis Group, New York, 1986.

[1385] M. Nieto-Vesperinas, *Scattering and Diffraction in Physical Optics*, Wiley, New York, 1991.

[1386] J. J. Stamnes and H. A. Eide, “Exact and approximate solutions for focusing of two-dimensional waves,” *J. Opt. Soc. Am.*, **A-15**, 1285 (1998), and p.1292, and p.1308.

[1387] R. Merlin, “Radiationless Electromagnetic Interference: Evanescent-Field Lenses and Perfect Focusing,” *Science*, **217**, 927 (2007).

[1388] A. Grbic and R. Merlin, “Near-Field Focusing Plates and Their Design,” *IEEE Trans. Antennas Propagat.*, **56**, 3159 (2008).

[1389] L. E. Helseth, “The almost perfect lens and focusing of evanescent waves,” *Opt. Commun.*, **281**, 1981 (2008).

[1390] L. E. Helseth, “Focusing of evanescent vector waves,” *Opt. Commun.*, **283**, 29 (2010).

[1391] M. F. Imani and A. Grbic, “An analytical investigation of near-field plates,” *Metamaterials*, **4**, 104 (2010).

[1392] A. Grbic, et al., “Near-Field Plates: Metamaterial Surfaces/Arrays for Subwavelength Focusing and Probing,” *Proc. IEEE*, **99**, 1806 (2011).

[1393] M. F. Imani and A. Grbic, “Planar Near-Field Plates,” *IEEE Trans. Antennas Propagat.*, **61**, 5425 (2013).

Superdirectivity

[1394] C. W. Oseen, “Die Einsteinsche Nadelstichstrahlung und die Maxwellschen Gleichungen,’ *Ann. Physik*, **69**, 202 (1922).

[1395] C. J. Bouwkamp and N. G. de Bruijn, “The problem of optimum antenna current distribution,” *Philips Res. Rep.*, **1**, 135 (1945/46); available online from,
`https://pure.tue.nl/ws/files/4289258/597471.pdf`

[1396] P. M. Woodward, “A method of calculating the field over a plane aperture required to produce a given polar diagram,” *J. IEE*, Part IIIA, **93**, 1554 (1946).

[1397] P. M. Woodward and J. D. Lawson, “The theoretical precision with which an arbitrary radiation-pattern may be obtained from a source of finite size,” *J. IEE*, Part III, **95**, 363 (1948).

[1398] T. T. Taylor, “Design of Line-Source Antennas for Narrow Beamwidth and Low Side Lobes,” *IRE Trans. Antennas Propagat.*, **AP-3**, 16 (1955).

[1399] T. T. Taylor, “Design of Circular Apertures for Narrow Beamwidth and Low Side Lobes,” *IRE Trans. Antennas Propagat.*, **AP-8**, 17 (1960).

[1400] R. Hansen, “Tables of Taylor distributions for circular aperture antennas,” *IRE Trans. Antennas Propagat.*, **AP-8**, 23 (1960).

[1401] R. C. Rudduck and D. C. F. Wu, “Directive Gain of Circular Taylor Patterns,” *Radio Sci.*, **6**, 1117 (1971).

[1402] R. C. Hansen, “A one-parameter circular aperture distribution with narrow beamwidth and low side lobes,” *IEEE Trans. Antennas Propagat.*, **AP-24**, 477 (1976).

[1403] A. C. Ludwig, “Low Sidelobe Aperture Distributions for Blocked and Unblocked Circular Apertures,” *IEEE Trans. Antennas Propagat.*, **AP-30**, 933 (1982).

[1404] N. Yaru, “A Note on Super-Gain Antenna Arrays,” *Proc. IRE*, **39**, 1081 (1951).

[1405] D. R. Rhodes, “The Optimum Line Source for the Best Mean-Square Approximation to a Given Radiation Pattern,” *IEEE Trans. Antennas Propagat.*, **AP-11**, 440 (1963).

[1406] T. S. Fong, “Eigenelements of the Finite Fourier Transform and their Application to Antenna Pattern Synthesis,” Report No. P67-93, Hughes Aircraft Co., Culver City, Ca, 1966; available online from,
`http://www.dtic.mil/cgi-bin/GetTRDoc?AD=AD0655791`

[1407] T. S. Fong, "On the Problem of Optimum Antenna Aperture Distribution," *J. Franklin Inst.*, **283**, 235 (1967).

[1408] D. R. Rhodes, "On an Optimum Line Source for Maximum Directivity," *IEEE Trans. Antennas Propagat.*, **AP-19**, 485 (1971).

[1409] D. R. Rhodes, "On a class of optimum aperture distributions for pattern shaping," *IEEE Trans. Antennas Propagat.*, **20**, 262 (1972).

[1410] D. R. Rhodes, "On the quality factor of strip and line source antennas and its relationship to superdirectivity ratio," *IEEE Trans. Antennas Propagat.*, **20**, 318 (1972).

[1411] H. N. Kritikos and M. R. Dresp, "Aperture Synthesis and Supergain," *Proc. IEEE*, **52**, 1052 (1964).

[1412] H. N. Kritikos, M. R. Dresp, and K. C. Lang, "Interference Suppression Studies, Studies of Antenna Side-lobe Reduction," Moore School of Electrical Engineering, University of Pennsylvania, Oct. 1964. Available online from:
`http://www.dtic.mil/cgi-bin/GetTRDoc?AD=AD0609076`

[1413] G. J. Buck and J. J. Gustincic, "Resolution Limitations of a Finite Aperture," *IEEE Trans. Antennas Propagat.*, **AP-15**, 376 (1967).

[1414] R. L. Kirlin, "Optimum resolution gain with prolate spheroidal wave functions," *J. Opt. Soc. Am.*, **64**, 404 (1974).

Neural Networks

[1415] Neural Network, Wikipedia, including history of neural networks,
`https://en.wikipedia.org/wiki/Neural_network`
see also, article on Backpropagation
`https://en.wikipedia.org/wiki/Backpropagation`

[1416] M. T. Hagan, et al, *Neural Network Design*, available freely from,
`http://hagan.okstate.edu/nnd.html`

[1417] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016, available from, `http://www.deeplearningbook.org`.

[1418] M. Nielsen, *Neural Networks and Deep Learning*, 2019, free online book,
`neuralnetworksanddeeplearning.com`

[1419] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*, online book, 2022, `d2l.ai/index.html`

[1420] A. Krizhevsky, I. Sutskever, G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," Advances in Neural Information Processing Systems 25 (NIPS), 2012.

[1421] S. Mallat, "Understanding deep convolutional networks," *Phil. Trans. R. Soc.*, **A374**, 20150203 (2016).

[1422] Shallow Neural Networks Bibliography, The Mathworks, 2022
`www.mathworks.com/help/deeplearning/ug/shallow-neural-networks-bibliography.html`

[1423] Bibliography on "Convolutional Neural Networks, Design, Implementation Issues," `www.visionbib.com/bibliography/pattern652imnn1.html` see also, "Annotated Computer Vision Bibliography," `www.visionbib.com/bibliography/contents.html`

[1424] L. Fausett, *Fundamentals of Neural Networks*, Prentice Hall, 1994.

[1425] S. Haykin, *Neural Networks*, IEEE Press, 1994.

[1426] F-A. Luo and R. Unbehauen, *Applied Neural Networks for Signal Processing*, Cambridge Univ. Press, 1997.

[1427] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by backpropagating errors," *Nature*, **323**, 533 (1986). See also, "Learning Internal Representations by Error Propagation," in D. E. Rumelhart and J. L. McClelland, eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1, MIT Press, Cambridge, MA, 1986. For the historical roots of backpropagation see, P. Werbos, *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*, Wiley, 1994, and also P. Werbos, "Backpropagation Through Time: What It Does and How to Do It," *Proc. IEEE*, **78**, 1550 (1990), and also, Y. Le Cun, "Une procedure d'apprentissage pour reseau a seuil assymetrique," *Cognitiva*, **85**, 599 (1985), and, D. B. Parker, "Learning-logic: Casting the cortex of the human brain in silicon," Technical Report TR-47, Center for Computational Research in Economics and Management Science, MIT, 1985.

[1428] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Math. Control Signals Syst.*, **2**, 303 (1989).

[1429] K. Hornik, M. Stinchcombe, H. White "Multilayer feedforward networks are universal approximators," *Neural Networks*, **2**, 359 (1989); see also K. Hornik, "Approximation capabilities of multilayer feedforward networks," *ibid*, **4**, 251 (1991); and also *ibid.*, **6**, 1069 (1993).

[1430] B. Widrow and R. Winter, "Neural nets for adaptive filtering and adaptive pattern recognition," *Computer*, **21**, no.3, 25 (1988). See also, B. Widrow and M.A. Lehr, "30 years of adaptive neural networks: perceptron, Madaline, and backpropagation,", *Proc. IEEE*, **78**, 1415 (1990).

[1431] A. Lapedes and R. Farber, "How neural nets work," *Neural Information Processing Systems*, 1987. See also, "Nonlinear signal processing using neural networks: Prediction and system modelling," Technical Report LA-UR-87-2662, Los Alamos Lab., 1987.

[1432] A. S. Weigend, B. A. Huberman, and D, E. Rumelhart, "Predicting the Future: A Connectionist Approach," *Int. J. Neural Syst.*, **1**, 193 (1990).

[1433] N. A. Gershenfeld and A. S. Weigend, "The future of time series: learning and understanding," In A. S. Weigend N. A. Gershenfeld (Eds.), *Time Series Prediction: Forecasting the Future and Understanding the Past*, Addison-Wesley, Reading, 1993.

[1434] G. Zhang, B. E. Patuwo, and M. Y. Hu, "Forecasting with artificial neural networks: The state of the art," *Int. J. Forecasting*, **14**, 35 (1998).

[1435] M. S. Gashler and S. C. Ashmore, "Modeling time series data with deep Fourier neural networks," *Neurocomputing*, **188**, 3 (2016).

[1436] P. Lara-Benitez, M. Carranza-Garcia, and J. C. Riquelme, "An Experimental Review on Deep Learning Architectures for Time Series Forecasting," *Int. J. Neural Syst.*, **31**, 21300001 (2021).

[1437] Neural network software platforms and libraries

MATLAB, The Mathworks, `www.mathworks.com`

PyTorch, `pytorch.org`

TensorFlow, `www.tensorflow.org`

Keras, Xception, InceptionV3, `keras.io` theano, `pypi.org/project/Theano`

OpenNN, `www.opennn.net`

FANN, `leenissen.dk/fann/wp`

DyNet, `github.com/clab/dynet`

Chainer, `chainer.org`

Apache MXNet, `mxnet.apache.org`

Waffles, `waffles.sourceforge.net`

ImageNet, `www.image-net.org`

CIFAR datasets, `www.cs.toronto.edu/~kriz/cifar.html`

MNIST handwritten digit database`yann.lecun.com/exdb/mnist`

Neural network software, Wikipedia,
`en.wikipedia.org/wiki/Neural_network_software`

Top 27 Artificial Neural Network Software in 2022
`www.predictiveanalyticstoday.com/top-artificial-neural-network-software`

Top 10 Libraries In C/C++ For Machine Learning,
`analyticsindiamag.com/top-10-libraries-in-c-c-for-machine-learning`

Index

- 1% time constant, 245, 300
- 3-bit parity problem, 1382, 1384
- 3-dB cutoff frequency, 507, 708, 711, 712
- 3-dB width, 255, 271, 515, 536, 713, 739, 741, 742, 746, 811, 812
- 6 dB per bit rule, 64
- 60 Hz noise, 261, 704, 738
- 60 dB time constant, 245, 806, 813, 821

- A/D converter, 53, 62
  - delta-sigma, 67, 688
  - flash or parallel, 84
  - subranging, 84, 93
  - successive approximation, 76, 84
- accumulation-distribution, 1255
- activation functions, 1378
- adaptive signal processing, 95, 99, 813, 832
- ADC, *see* A/D converter
- aliased frequency, 11
- aliasing, 5, 8, 10, 34, 432
  - examples of, 11–25
  - in rotational motion, 26
- alternating-step response, 249
- amplitude modulation, 796
- analog
  - Bessel filters, 667
  - Butterworth filters, 58, 538, 665, 675
  - Chebyshev filters, 559
  - dither, 84
  - frequency response, 3
  - impulse response, 2
  - linear filtering, 3
  - linear system, 2
  - postfilter, 45
  - prefilter, 38
  - reconstructor, 10, 42, 623
- antenna tracking system, 1037
- anti-image postfilter, *see* postfilter
- antialiasing prefilter, *see* prefilter
- attenuation, 21, 39, 47
  - dB per decade, 39
  - dB per octave, 21, 40
  - of window sidelobes, 364
- audio effects processor, 266, 801, 829
- autocorrelation, 395

- backpropagation algorithm, 1379
- backward Euler, 996, 1014
- bandpass filters, 550
- bands, 1242
  - Bolinger, 1242
  - fixed-width, 1242
  - Keltner, 1242
  - projection, 1242
  - standard error, 1242
  - Starc, 1242
- bandstop filters, 555
- bandwidth selection, 1143
- biasing, 416
- bilinear transformation, 504, 996
  - for comb filters, 534, 741
- biomedical signal processing, 738
- bit reversal, 442
- block convolution, 141, 449
- block diagram, 111
  - for canonical form, 284
  - for cascade form, 295
  - for direct form, 277
  - for FIR filter, 161, 163
  - for second-order section, 282
  - for transposed form, 188, 328
- block processing, 95, 120, 121
  - overlap-add method, 141, 453
  - overlap-save method, 453
- Bolinger bands, 1242
- BPDN, basis pursuit denoising, 1307
- Butterworth moving average filters, 1228

- C routines:
  - `I0`, modified Bessel function, 494
  - `adc`, A/D conversion, 80
  - `allpass`, allpass reverb, 816
  - `bitrev`, bit reversal, 444
  - `blockcon`, overlap-add, 143
  - `can2`, canonical form, 287
  - `can3`, canonical form, 311
  - `can`, canonical form, 285
  - `cas2can`, cascade to canonical, 308
  - `cas`, cascade form, 297
  - `ccan2`, circular canonical, 320
  - `ccan`, circular canonical, 314
  - `ccas2`, circular cascade, 321
  - `ccas`, circular cascade, 319
  - `cdelay2`, circular delay line, 183
  - `cdelay`, circular delay line, 180
  - `cfir1`, circular FIR filtering, 178
  - `cfir2`, circular FIR filtering, 182
  - `cfir`, circular FIR filtering, 176
  - `conv`, convolution, 137
  - `csos2`, circular SOS, 321
  - `csos`, circular SOS, 318
  - `dac`, D/A conversion, 75
  - `delay`, delay operation, 158
  - `delta`, delta function, 140
  - `dftmerge`, DFT merge, 445
  - `dir2`, direct form, 280
  - `dir`, direct form, 278
  - `dot`, dot product, 168
  - `fft`, FFT, 443
  - `fir2`, FIR filtering, 169
  - `fir3`, FIR filtering, 171
  - `fir`, FIR filtering, 167
  - `gdelay2`, generalized delay, 785
  - `gran`, Gaussian random generator, 1393
  - `ifft`, inverse FFT, 446
  - `lowpass`, lowpass reverb, 819
  - `modwrap`, modulo-N reduction, 426
  - `plain`, plain reverb, 816
  - `ran1f`, $1/f$ noise generator, 1400
  - `ranh`, hold random generator, 1395
  - `ranl`, linearly interpolated, 1396
  - `ran`, uniform random generator, 1392
  - `shuffle`, shuffling in FFT, 443
  - `sine`, sinusoidal wavetable, 793
  - `sos`, second-order section, 294
  - `square`, square wavetable, 794
  - `swap`, swapping in FFT, 444
  - `tap2`, circular tap outputs, 183
  - `tapi2`, interpolated delay, 808
  - `tapi`, interpolated delay, 807
  - `tap`, circular tap outputs, 181
  - `trapez`, trapezoidal wavetable, 794
  - `u`, unit-step function, 81
  - `wavgeni`, wavetable generator, 792
  - `wavgenr`, wavetable generator, 792
  - `wavgen`, wavetable generator, 791
  - `wrap2`, circular index wrapping, 182
  - `wrap`, circular pointer wrapping, 176
- canonical form, 230, 281
  - difference equations of, 284
  - sample processing algorithm, 285, 777
- capital asset line, 1264
- capital asset pricing model, CAPM, 1268
- capital market line, 1268
- CAPM, 1268
- cascade form, 294
  - coefficient matrices, 296
  - internal states, 296
  - pipelining of, 313
  - processing time, 312
  - sample processing algorithm, 297
- cascade to canonical, 301, 307
- causality, 111, 402
  - and finitely anticausal filters, 112, 113
  - and interpolation filters, 112
  - and inverse filters, 112, 116
  - and off-line processing, 113
  - and real-time processing, 113
  - and smoothing filters, 112
  - and stability, 115, 267
  - in z-domain, 193, 199, 233
- census X-11 decomposition filters, 1358
- central limit theorem, 1393
- Chaikin money flow, 1255
- Chaikin oscillator, 1255
- Chaikin volatility, 1255
- Chande momentum oscillator, CMO, 1252
- channels, 1242
- Chebyshev window, 378, 390
- chorusing, 809
- chrominance signals, 751
- circular
  - addressing, 121, 169, 172, 314
  - buffer, 172
  - canonical form, 315
  - cascade form, 319
  - convolution, 449
  - delay, 172, 180, 183
  - direct form, 176
  - pointer, 173
  - pointer index, 174
  - state vector, 174, 175, 181, 183
  - wrapping, 176, 177, 180, 182
- classical seasonal decomposition, 1344
- classical spectral analysis, 399
- coefficient quantization, 353
- COLA, *see* constant-overlap-add property
- comb filters, 259, 263, 326, 534, 746
  - 3-dB width, 746
  - complementarity property, 738
  - design, 534, 746
  - for digital reverb, 805
  - for noise reduction, 264
  - for signal enhancement, 264
  - in digital audio effects, 801
  - in digital TV, 749
  - sample processing algorithm of, 803
- commodity channel index, CCI, 1255
- complementarity, 471, 486, 490, 513, 738, 746, 756
- complex poles, 2, 301, 769
- compressors, *see* dynamics processors
- computational resolution, *see* resolution
- computer music
  - amplitude modulation, 796
  - frequency modulation, 783
  - musical instrument models, 783, 824
  - physical modeling, 783, 824
  - wavetable generators, 783
- constant-overlap-add property, 886

- control systems, 1030
  - antenna tracking system, 1037
  - cruise control, 1035
  - digital control systems, 1032
  - feedback control, 1030
  - inverted pendulum, 1048
  - PID control, 1032
  - thermostat control, 1053
- convolution, 111, 120, 121
  - circular, 449
  - direct form of, 104, 121, 122
  - fast convolution example, 457
  - fast via FFT, 449
  - flip-and-slide form of, 130
  - for discrete-time systems, 104
  - in z-domain, 191
  - in continuous-time, 3, 43
  - in frequency domain, 207
  - LTI form of, 104, 121, 126
  - matched filtering example, 460
  - matrix form of, 128
  - of finite sequences, 122
  - of infinite sequences, 133
  - overlap-add method, 141, 453
  - overlap-save method, 453
  - table form of, 121, 124
- coupled form generator, 771
- cross validation, 1143
- cruise control, 1035
- cutoff frequency, 507
- CVX package, 1264, 1290, 1307, 1370

- D/A converter, 46, 53, 71
  - and reconstruction, 46
  - delta-sigma requantizer, 688, 694
  - natural binary, 72
  - offset binary, 73
  - two's complement, 73
- DAC, *see* D/A converter
- DC gain, 138, 249
- DCC compact cassette, 494
- DCT, 905
  - compression system, 910
  - inverse DCT, 910
  - normalized DCT matrix, 909
- decibels, dB, *see* attenuation
- decimation, 622, 676
  - and delta-sigma quantizer, 689
  - and oversampling, 23, 676
  - averaging decimators, 679, 692
  - downsampled spectrum, 677
  - filter design, 678
  - filter specifications, 678
  - hold decimators, 679, 692
  - ideal, 678
  - Kaiser designs, 678, 692
  - multistage, 679
  - prefilter specifications, 680
  - sample processing algorithm, 679
- deconvolution, 264, 452
  - of noisy data, 266
- delay, 101, 154, 181
  - circular, 172, 180, 183
  - linear, wrapped, 172
  - linearly interpolated, 807
  - tapped, 160
  - update, 158, 173, 177
- delta-sigma quantizer, 67, 688
  - decimator filter, 689
  - first-order model of, 689, 700
  - MASH architectures, 702
  - requantizer, 688, 694
  - second-order model of, 322, 701
- DEMA filters, 733
- detrended price oscillator, 1256
- DFS, *see* discrete Fourier series
- DFT, 360, 410
  - N-point DFT of length-L signal, 410
  - and DTFT, 410
  - and periodic signals, 432
  - biasing in computation of, 416
  - frequencies, 410
  - matrix, 418
  - matrix form of, 418
  - merging, 436, 445
  - modulo-N reduction, 421
  - negative frequencies, 370, 372, 414, 433
  - time wrapping, 421
  - twiddle factor, 419
  - zero padding, 413
- DFT frequencies, 410, 412
- difference equation, 111, 234, 277
- differentiation filters, 1088
- differentiator filter, 469, 499
- digital audio effects, 182, 266, 800
  - chorusing, 809
  - comb filters, 801
  - delays, 801
  - dynamics processors, 829
  - echoes, 801
  - flanging, 806
  - linearly interpolated delay, 807
  - multitap delays, 825
  - phasing, 810
  - reverberation, 813
  - stereo imaging, 806, 846
- digital reverberation, *see* reverberation
- digital signal processor, 1, 53
- digital TV, 749
- direct form
  - difference equations of, 234, 277
  - for FIR filter, 160, 161, 163
  - for IIR filter, 227, 275
  - internal states, 276
  - sample processing algorithm, 278, 774
- direct form I, *see* direct form
- direct form II, *see* canonical form
- directional movement system, 1253
- discrete cosine transform, 360, 466, *see* DCT
- discrete Fourier series, 18, 432
- discrete Fourier transform, *see* DFT
- discrete time
  - convolution, 104
  - filter, 53
  - Fourier transform, *see* DTFT
  - linear filtering, 55
  - linear system, 96
  - system, 95
  - system I/O rules, 96
- discrete-time Fourier transform, 1104
- discretization methods, 979
  - backward Euler, 996, 1014
  - bilinear transformation, 996
  - continuous-time systems, 979
  - first-order hold, 999, 1025
  - forced response, 983
  - forward Euler, 996, 1014
  - impulse invariance, 1024
  - initialization procedures, 981, 1006
  - observability matrix, 1007
  - ramp invariance, 1027
  - realizations, 999
  - sample processing, 999
  - sinusoidal response, 985
  - staircase reconstructor, 1022
  - starred Laplace transform, 1015
  - state-space realizations, 1003
  - step invariance, 1023
  - summary, 994
  - trapezoidal, 996, 1014
  - triangular hold, 1025
  - zero-order hold, 997, 1018
- dither, 66, 84
  - and noise shaping, 688
  - example of, 87
  - Gaussian, 85
  - non-subtractive, 85
  - rectangular, 85
  - subtractive, 90
  - triangular, 85, 87
- Dolph-Chebyshev window, 483
- Donchian channels, 1242
- downsampler, 677
- DPSS window, 377, 388
- DRC, *see* dynamics processors
- DSP chips, 55, 169, 310, 355
  - examples of, 178
  - instruction time, 170
  - MACs, 169, 310, 313, 355
  - MFLOPS, 170, 313
  - MIPS, 171, 312, 313
  - processing time, 171, 178
  - quantization effects in, 355
  - wordlengths, 355
- DSP system
  - building blocks of, 154
  - components of, 53
  - with oversampling, 71, 680
- DTFT, 30, 202, 362, 1417
  - computation by Hörner's rule, 373
  - and DFT, 410
  - and unit circle, 373, 412
  - and windowing, 362
  - at single frequency, 372
  - computation, 372
  - geometric interpretation of, 205
  - magnitude and phase, 207
  - negative frequencies, 370, 372, 414, 433
  - of length-L signal, 372
  - of rectangular window, 362
  - of windowed sinusoids, 365
  - periodicity of, 30
- DTMF, 374, 768
- dual-tone multi-frequency, *see* DTMF
- dynamic momentum index, DMI, 1256
- dynamic range control, *see* dynamics processors
- dynamics processors, 829
  - adaptive processing, 832
  - attack time constants, 833
  - compression/expansion ratio, 830
  - compressor/expander model, 830
  - compressors, 829
  - duckers, 837
  - envelope detector, 830
  - expanders, 834
  - gain processors, 830, 831
  - gain smoothing, 833
  - level detectors, 830
  - limiters, 834
  - noise gates, 835
  - release time constants, 833

- ECG processing, 738, 740–742
- efficient frontier, 1261, 1266
- elliptic filters, 576
  - analog, 593
  - degree equation, 590
  - digital, 605
  - elliptic rational function, 587
  - frequency transformations, 609
  - frequency-shifted realizations, 614
  - Jacobian elliptic functions, 580
  - Landen transformations, 591
  - MATLAB functions, 611
- EMA, 708
  - improved, 710
- EMA filters, 733
- EMA initialization, 1198, 1225
- EMA, exponential moving average, 1160
- encircled energy, 1418
- envelopes, 1242
- equalizer

  - channel, 265
  - comb, 534
  - for multistage interpolation, 668
  - for staircase DAC, 661
  - graphic, 488, 523
  - matched Nyquist-frequency gain, 532
  - parametric, 254, 523, 849
  - postfilter, 47, 664
  - prefilter, 39
  - room acoustics, 265
  - shelving, 530, 532
- error spectrum shaping, 355, 688
- exact LPSM filters, 1068
- expanders, *see* dynamics processors
- exponential smoother, 710, 1160
- exponentially weighted moving average, 708
- exponentially-weighted moving average, 1160

- fast convolution, 449
- fast Fourier transform, *see* FFT
- feedback system, 328
- FFT, 360, 436
  - bit reversal, 442
  - computational cost, 438
  - decimation in time, 438
  - merging, 436, 441, 445
  - of real-valued signals, 466
  - shuffling, 441, 443
- Fibonacci sequence, 270
- filter
  - as state machine, 95
  - equivalent descriptions of, 111, 224
  - group delay, 241
  - internal state of a, 95
  - magnitude response, 240
  - phase delay, 241
  - phase response, 240
  - Q-factor, 739, 812
- filter banks, 494
- filter design
  - analog filters, 538
  - by pole/zero placement, 207
  - comb filters, 259, 534, 746
  - first-order, 252
  - for noise reduction, 704
  - for sampling rate conversion, 681
  - for signal enhancement, 704
  - frequency sampling, 498, 662, 666
  - notch filters, 258, 259, 742, 811
  - of FIR filters, 469
  - of IIR filters, 504
  - of Savitzky-Golay smoothers, 1058
  - parametric equalizers, 523
  - parametric resonators, 254
  - peaking filters, 519
  - second-order, 254, 258
  - window method, 469
- filter lag, 721
- filter scaling, 355
- filtering, 3, 35, 55
  - adaptive, 95
  - by block processing, 95, 120
  - by sample processing, 95, 120
  - in direct form, 105, 106
  - in LTI form, 104
  - of random signals, 401
- filtering methods in financial markets, 1209
- FIR averager, 692, 714
- FIR filter, 95, 104, 105, 120
  - block diagram, 161, 163
  - circular buffer, 176, 178, 182, 184
  - coefficients, 105
  - difference equations for, 234
  - direct form of, 159
  - in steady-state form, 132
  - length, 105
  - linear phase property of, 242
  - order, 105
  - sample processing algorithm for, 159, 161, 163, 167
  - taps, 105
  - weights, 105
  - window method, 469
- FIR filter design, 469
  - approximation properties, 475
  - examples, 486–494
  - frequency sampling, 498, 662, 666
  - Gibbs phenomenon, 475
  - Hamming window, 477
  - ideal filters, 469
  - Kaiser window, 481
  - linear phase property, 474
  - Meteor design program, 499
  - Parks-McClellan method, 499
  - rectangular window, 472
  - window method, 469
- first-order hold, 999, 1025
- first-order IIR smoother, 708
- fixed-width bands, 1242
- flanging, 806
- forecast oscillator, 1256
- forecasting and state-space models, 1169
- forward Euler, 996, 1014
- Fourier transform, 2
  - discrete-time, DTFT, 30
  - of sampled signal, 30
- frequency
  - aliased or perceived, 11, 26
  - and unit circle, 207
  - biasing, 416
  - DFT, 410
  - digital, 26, 203
  - leakage, 31, 362, 368
  - modulation, 783, 798
  - negative, 28, 370, 372, 414, 433
  - Nyquist, 6

  - resolution, 360, 366
  - response, 3, 111, 203
  - spectrum, 2, 202
  - units used in DSP, 28
- frequency sampling design, 498, 662, 666

- generalized cross-validation, 1143
- generalized double EMA, GDEMA, 1231
- generalized efficient frontier, 1270
- generator, *see* waveform generators
- geometric series
  - finite, 37, 72, 134, 197, 247, 717, 803
  - infinite, 17, 36, 114, 193, 194, 197, 198, 425, 708, 773, 804, 815, 818
- Gibbs phenomenon, 475
- graphic equalizers, 488, 523
- group delay, 241, 721
- guard band, 34

- Hahn orthogonal polynomials, 1118
- Hamming window, 51, 366, 386, 477, 630, 654
- hardware realizations, 169, 310
- Henderson filters, 1082, 1108
- hermitian property, 207
- higher-order exponential smoothing, 1174
- higher-order polynomial smoothing, 1170
- highpass filters, 547
- Hilbert transformer, 469
- Hodrick-Prescott filters, 1272, 1279
- hold interpolators, 647
- Holt's exponential smoothing, 1203
- Hull moving average, 1231
- Hörner's rule, 91, 373

- I/O rules, 96
- IIR filter, 95, 104, 106
  - circular buffer, 315, 319
  - difference equations for, 106, 110, 234
  - transfer function of, 233
- IIR filter design, 504
  - analog filters, 538
  - bandpass filters, 550
  - bandstop filters, 555
  - bilinear transformation, 504
  - Butterworth, 538
  - Chebyshev filters, 559–571
  - comb filters, 534
  - first-order filters, 507
  - higher-order filters, 536
  - highpass filters, 547
  - lowpass filters, 543
  - notch filters, 514
  - parametric equalizers, 523
  - peaking filters, 519
  - second-order filters, 514
  - specifications, 537
- ILRS, integrated linear regression slope, 1212
- impulse invariance, 1024
- impulse response, 2, 102, 111, 224, 225, 233, 256, 264, 266, 268
- indicator function, 1410
- input-off transients, 131, 161, 823
- input-on transients, 131, 161, 823
- instantaneous gradient, 1162
- instruction time, 170, 171, 178, 311
- integrated linear regression slope, 1212
- interference
  - 60 Hz noise, 261, 264
  - periodic, 264
- internal state, *see* state
- interpolation, 622
  - 4-fold, 626, 651
  - computational rate, 631
  - cutoff frequency, 629
  - DAC equalization, 661
  - filter design, 628
  - filter specifications, 625, 635
  - ideal, 636
  - in direct form, 628
  - in polyphase form, 622, 630, 632, 655
  - Kaiser designs, 638
  - linear and hold, 647
  - multistage designs, 639, 657
  - multistage equalization, 668
  - postfilter equalization, 664
  - postfilter specifications, 624, 664
  - sample processing algorithm, 635
- interpolation filters, 1075
- interpolation vs. smoothing splines, 1271
- inverse z-transforms, 208
- inverse DFT, 429, 430, 446
- inverse DTFT, 30, 203, 241, 242, 364, 429, 1417
  - and Fourier series, 30, 204
- inverse FFT, 446
- inverse filters, 264
  - stable and causal, 267
- inverse Fourier transform, 2, 430
- inverse STFT, 886
- inverted pendulum, 1048
- IRLS, iterative reweighted least-squares, 1307
- ISTFT, 886
- iterative reweighted least-squares, IRLS, 1290, 1307

- Kaiser window, 366, 387
  - for filter design, 481
  - for interpolator designs, 638
  - for spectral analysis, 366, 495
  - parameters, 484, 496
  - resolution, 495
- Karplus-Strong string algorithm, 824
- KBD windows, 918
- Keltner bands, 1242
- kernel machines, 1283
- Krawtchouk polynomials, 1126

- L0 trend indicator, 1239

- L1 trend filtering, 1289
- Landen transformations, 591
- Laplace transform, 2
  - starred, 32, 1015
- LASSO, least absolute shrinkage and selection operator, 1307
- lattice realizations, 331
  - filtering, 344
  - frequency response, 346
  - MATLAB functions, 343
  - normalized lattice, 341
  - quantization effects, 353
  - rearranged lattice, 339
  - stability test, 346
  - standard lattice, 333
- lattice/ladder realizations, 331
- leakage, 362, 368
- leaky ReLU, 1378
- Legendre polynomials, 1407
- limiters, *see* dynamics processors
- linear filtering, 3, 35, 55, 240
- linear interpolators, 647
- linear phase property, 242, 474, 707
- linear regression, 1217
- linear regression indicator, 1213
- linear regression slope indicator, 1213
- linear superposition, 3, 240
- linear system, 2
  - in matrix form, 96
  - state space realization, 98
- Linear time-invariant, *see* LTI system
- linear trend FIR filters, 1172
- linearity, 100, 240
- linearly interpolated delay, 807
- LMS algorithm, 1162
- loan / mortgage amortization, 217
- local level filters, 1213, 1233
- local polynomial fitting, 1059
- local polynomial interpolation, 1145
- local polynomial modeling, 1136
- local polynomial smoothing filters, 1058
- local slope filters, 1213, 1233
- loess smoothing, 1157
- logistic function, 1378
- loudspeaker crossover filters, 488
- lowpass differentiator, 499
- lowpass filters, 543
- lowpass Hilbert transformer, 499
- LPSM filters, 1058
- LTI system, 95
  - anticausal, 111
  - causal, 111
  - double-sided, 111
  - equivalent descriptions of, 111, 224
  - FIR, 95
  - frequency response, 2
  - IIR, 95
  - impulse response, 2, 102
- luminance signal, 751

- MAC, 169, 171, 310, 313, 355
- magnitude response, 207, 240
- marginal stability, 202, 205, 250, 402
- Market indicators:
  - **accdist**, accumulation/distribution line, 1252
  - **atr**, average true range, 1246
  - **bbands**, Bolinger bands, 1246
  - **bma**, Butterworth moving average, 1230
  - **cci**, commodity channel index, 1252
  - **chosc**, Chaikin oscillator, 1252
  - **chvol**, Chaikin volatility, 1252
  - **cmflow**, Chaikin money flow, 1252
  - **cmo**, Chande momentum oscillator, 1252
  - **delay**, d-fold delay, 1235
  - **dema**, double EMA, 1217
  - **dirmov**, directional movement system, 1252
  - **dmi**, dynamic momentum index, 1252
  - **donch**, Donchian channels, 1246
  - **dpo**, detrended price oscillator, 1252
  - **ehma**, exponential Hull moving average, 1235
  - **fbands**, fixed-width bands, 1246
  - **forosc**, forecast oscillator, 1252
  - **gdema**, generalized DEMA, 1235
  - **hma**, Hull moving average, 1235
  - **ilrs**, integrated linear regression slope, 1212
  - **kbands**, Keltner bands, 1246
  - **l0trend**, L_0 trend indicator, 1240
  - **lreg**, linear regression indicators, level, slope, R-square, standard-errors, 1220
  - **mom**, momentum, price rate of change, 1252
  - **ohlcyy**, OHLC chart with left/right y-axes, 1221
  - **ohlc**, open-high-low-close bar chart, 1221
  - **pbands**, projection bands & oscillator, 1246
  - **pma2**, quadratic PMA, 1214
  - **pmaimp2**, PMA2 impulse response, 1214
  - **pmaimp**, PMA impulse response, 1214
  - **pma**, predictive moving average, 1214
  - **pnvi**, positive/negative volume indices, 1252
  - **prosc**, price oscillator and MACD, 1252
  - **psar**, parabolic SAR, 1250
  - **r2crit**, R-square critical values, 1219
  - **rsi**, relative strength index, 1252
  - **sebands**, standard-error bands, 1246
  - **sema**, single EMA, 1217
  - **shma**, simple Hull moving average, 1235
  - **sma**, simple moving average, 1212
  - **stbands**, Starc bands, 1246
  - **stdev**, length-N standard deviation, 1242
  - **stoch**, stochastic, percent-K, percent-D, 1252
  - **t3**, Tillson's T3 indicator, 1235
  - **tcrit**, t-distribution critical values, 1219
  - **tdistr**, cumulative t-distribution, 1219
  - **tema**, triple EMA, 1217
  - **tma**, triangular moving average, 1212
  - **trix**, TRIX oscillator, 1252

  - **vema**, variable-length EMA, 1252
  - **vhfilt**, Vertical horizontal filter, 1252
  - **wema**, Wilder's EMA, 1228
  - **wma**, weighted moving average, 1212
  - **yylim**, adjust left/right y-axes limits, 1221
  - **zema**, zero-lag EMA, 1235
  - **zigzag**, zigzag indicator, 1238
- market portfolio, 1268
- Markowitz portfolio theory, 1259
- maximally-flat filters, 1126
- MDCT, 905, 913
  - data compression system, 906
  - inverse MDCT, 914
  - KBD window, 918
  - Princen-Bradley windows, 916
  - window construction, 918
- MFLOPS, 170, 313
- minimum roughness filters, 1102
- minimum variance filters, 1082
- MIPS, 171, 312, 313
- missing data and outliers, 1131
- modified Bessel function, 483, 494
- modified DCT, 905, *see* MDCT
- modulo addressing, *see* circular
- modulo-N reduction, 421
- moments in smoothing filters, 1086
- momentum, 1251
- momentum, price rate of change, 1253
- moving average convergence divergence, MACD, 1254
- moving average filter, *see* FIR filter
- moving average filters, 1209
  - Butterworth, BMA, 1228
  - EMA, exponential, 1209
  - initialization, 1222
  - predictive, PMA, 1212
  - reduced lag, 1231
  - SMA, simple, 1209
  - TMA, triangular, 1209
  - WMA, weighted, 1209
- multi-delay audio effects, 825
- multiplier/accumulator, *see* MAC
- multirate filter banks, 494
- multirate signal processing, 99, 622
- multistage equalization, 668
- multistage interpolators, 639
- multitap delay audio effects, 825
- Musgrave asymmetric filters, 1362
- musical instrument models, 783, 824

- NAR, nonlinear autoregressive models, 1385
- negative frequencies, 28, 370, 372, 414, 433
- neural networks, 1376
- neural networks for time series prediction, 1385
- noise gates, *see* dynamics processors
- noise reduction, 264, 704
  - comb filters for, 738
  - filter design for, 704
  - FIR averager, 710
  - first-order IIR smoother, 708
  - noise reduction ratio, 706
  - notch filters for, 738
  - SNR in, 706
  - time constant vs. group delay, 720, 724
  - transient response, 707
- noise reduction ratio, 402, 706
- noise shaping, 66, 688
  - dithered, 688
  - error spectrum shaping, 355, 688, 771
  - quantizer, 69, 688
  - quantizer comparisons, 70
  - requantizer, 688, 694
- non-rectangular windows, 366
- nonlinear signal processing, 99
- notch and comb filters, 1320
- notch and comb filters with fractional delay, 1326
- notch filters, 258, 259, 514, 811
  - 3-dB width, 811
  - for ECG processing, 738, 740, 742
  - phasing audio effects, 810
  - Q-factor, 812
- notch polynomial, 260
- NRR, *see* noise reduction ratio
- NRR, noise reduction ratio, 402
- Nyquist
  - frequency, 6
  - interval, 6, 203
  - rate, 6

- OLA, *see* overlap-add reconstruction
- optimum mean-variance portfolios, 1259
- orthogonal polynomial bases, 1074
- oscillator, *see* waveform generator
- oscillators, 1251
- overlap-add method, 141, 453
- overlap-add reconstruction, 886
- overlap-save method, 453
- oversampling, 23, 66, 622, 625
  - and decimation, 23, 676
  - and interpolation, 622
  - and noise shaping, 66
  - and quantization, 68
  - digital filter, 625
  - DSP system, 71, 680
  - in CD and DAT players, 71, 623
  - in postfiltering, 53, 624, 664
  - in prefiltering, 24, 39
  - ratio, 68

- parabolic SAR, 1242
- parallel form, 222, 229, 494
  - processing time, 313
- parametric equalizers, 254, 523
  - 2nd order, 523
  - bandwidth definitions, 864
  - high-order, 849

  high-order analog designs, 879
  high-order Butterworth, 857
  high-order Chebyshev type-1, 858
  high-order Chebyshev type-2, 859
  high-order elliptic, 861
  MATLAB functions, 880
  order determination, 863
  realizations, 867
parametric resonators, 254, 519
parametric spectral analysis, 368
Parks-McClellan method, 499
Parseval's equation, 204
partial fraction expansion, 208, 243
  of z-transforms, 208
  remove/restore method, 209
peaking filters, 519
Pell's sequence, 271
perceptual coding, 494, 688
periodic interference, 264, 704
periodic signal extraction, 1319
periodogram, 395
  averaging, 399
  averaging and smoothing, 368
  modified periodogram, 398
  smoothing, 400
PF, PFE, *see* partial fraction expansion
phase delay, 241, 721
phase response, 207, 240
phase vocoder, 891
phasing, 810
physical modeling of instruments, 783, 824
physical resolution, *see* resolution
PID control, 1032
pipelining of cascade form, 313
pitch-scale modification, 896
PMA filters, 733
Poisson summation formula, 34
pole/zero designs
  comb filters, 259
  first-order filters, 252
  notch filters, 258, 259
  parametric resonators, 254
  second-order filters, 254, 258
pole/zero pattern, 111, 205
polynomial interpolation filters, 1075
polynomial predictive filters, 1075
polyphase filters, 622, 627, 630, 632
portfolio with inequality constraints, 1263
portfolio with multiple constraints, 1269
positive/negative volume indices, 1255
postfilter, 45, 53, 624
  and oversampling, 49, 624
  Bessel, 667
  Butterworth, 665, 675
  equalized digitally, 47, 51, 664
  specifications of, 46, 664
power spectrum, 395
predictive differentiation filters, 1088
predictive filters, 1075
predictive moving average filters, 1212
prefilter, 35, 38, 53, 623
  and oversampling, 24, 39, 680
  attenuation, 39
  Butterworth, 58
  equalized digitally, 39
  ideal, 7, 38
  practical, 18, 38
  specifications of, 38, 680
price oscillator, 1254
Princen-Bradley windows, 916
processing time, 8, 171, 178, 311
projection bands, 1242
prolate spheroidal wave functions, 1406
PSWF, prolate spheroidal wave functions, 1406

Q-factor, 515, 739, 742, 748, 812
quantization, 62
  and dithering, 66
  and oversampling, 68
  by rounding, 63, 78
  by truncation, 63, 77
  error, 63
  granulation noise in, 66
  noise, 65
  noise shaping, 66
  process, 62
  signal-to-noise ratio, 64
  width, 62
quantization effects
  coefficient quantization, 353
  coupled form, 771
  error spectrum shaping, 355, 771
  poles near unit circle, 771
  roundoff error, 302, 353, 355
quantizer
  6 dB per bit rule, 64
  delta-sigma, 67
  dynamic range of a, 64
  error spectrum shaping, 688
  error, average, 64
  error, maximum, 64
  error, root-mean-square, 64
  full-scale range of, 62
  noise model of a, 65, 82
  noise shaping, 69, 688
  noise shaping comparisons, 70
  number of levels of a, 62
  probabilistic interpretation of a, 64
  resolution, 62
  uniform, 62

R-square indicator, 1217
ramp invariance, 1027
random number generators, 810, 1389
  $1/f$ noise, 1399, 1400, 1404, 1405
  Gaussian, 1389, 1393

  general interpolator, 1399, 1404
  hold interpolator, 1394, 1395
  linear congruential, LCG, 1390
  linearly interpolated, 810, 1396
  low-frequency, 810, 1394
  portable, 1391
  uniform, 1389, 1392
random signals, 395
  autocorrelation, 395
  filtering of, 401
  power spectrum, 395
random walk, 402
rate compressor, *see* downsampler
rate expander, *see* upsampler
reconstructor, 10, 42, 623
  and D/A conversion, 46
  ideal, 10, 43, 624
  ideal impulse response of, 44
  in interpolation, 44
  staircase, 42, 44, 623
  truncated impulse response of, 44, 629
rectangular window, 51, 361
  in FIR filter design, 472
  in spectral analysis, 362
  sidelobes, 364
  width, 363
reduced-lag moving average filters, 1231
region of convergence, 193, 195, 199
  and causality, 199
  and stability, 200
regularization and kernel machines, 1283
regularization filters, 1277
relative strength index, RSI, 1252
ReLU, rectified linear unit, 1378
resolution, 360, 366
  and leakage, 362, 368
  and mainlobe width, 363, 366, 367
  and windowing, 360
  computational, 414
  of Kaiser window, 495
  of non-rectangular windows, 367
  physical, 362, 363, 367, 414
  uncertainty principle, 362
resonators, *see* parametric resonators
reverberation, 182, 264, 813
  60 dB time constant, 806, 813
  allpass, 816
  and comb filters, 805
  early reflections, predelay, 813
  gated and reversed, 814
  late reflections, 813
  lowpass, 818, 819
  lowpass, time constants, 821, 823
  plain, 805, 814, 816
  plain, time constants, 821, 823
  sample processing algorithm, 815
  Schroeder's reverberator, 817
risk aversion, 1263
risk premium, 1268
RLS algorithm, 1163
ROC, *see* region of convergence
roots of unity, 222, 412, 739, 780, 805, 821
  and DFT, 412
roundoff error, 302, 353, 355

sample processing algorithm, 111, 120, 154
  for allpass reverb, 815
  for canonical form, 231, 285
  for cascade form, 297
  for circular canonical form, 315
  for circular cascade form, 319
  for circular FIR filter, 176, 184
  for decimation, 679
  for direct form, 227, 278
  for FIR filtering, 159, 161, 163, 167, 168
  for interpolation, 635
  for lowpass reverb, 819
  for multi-delay, 826
  for multitap delay, 828
  for parallel form, 230
  for ringing delay, 826
  for sample rate conversion, 685
  for Schroeder's reverb processor, 817
  for transposed form, 232
  for variable notch filters, 812
sampled signal, 29
  flat-top sampled, 29, 60
  ideally sampled, 29
sampling
  aliasing due to, 5
  and Poisson summation formula, 34
  as modulation, 33
  frequencies introduced by, 5
  function, 33, 699
  guard band, 34
  hardware limits in, 8
  ideal and practical, 29
  Nyquist frequency, 6
  Nyquist rate, 6
  of periodic signals and DFT, 432
  of sinusoids, 9, 35, 50
  process, 4
  rate, 5, 7
  spectrum replication due to, 5
  theorem, 5, 9, 44
  time interval, 4
sampling rate conversion, 622, 681, 700
  filter design for, 681
  sample processing algorithm, 685
sampling theorem, 1417
Saramäki windows, 483
Savitzky-Golay smoothing filters, 1058
scaling, 355
Schroeder's reverberator, 817
Schur-Cohn Stability Test, 346
seasonal decomposition filters, 1342

- seasonal moving-average filters, 1352
- seasonal Whittaker-Henderson decomposition, 1368
- second-order section, 275, 282, 294, 318, 321
  - circular buffer form of, 318, 321
- security market line, 1268
- Shannon number, 1408, 1417
- Sharpe ratio, 1265
- shelving filters, 530, 532
- short-time Fourier transform, 882, *see* STFT
- sigma-delta, *see* delta-sigma
- signal averaging, 1336
- signal enhancement, 264, 704
  - comb filters, 738, 747
  - comb filters for TV, 749
  - filter design for, 704
  - noise reduction ratio, 706
  - notch filters, 738
  - of periodic signals, 747
  - SNR in, 706
  - transient response, 707
- signal extraction, periodic, 1319
- signal generators, *see* waveform generators
- signal-to-noise ratio, 706
- sinc kernel, 1407
- single, double, triple EMA, 1191, 1215
- sinusoidal generators, 767
- sinusoidal response, 3, 239, 243
  - of FIR filters, 252
- SMA filters, 733
- smoothing filters, 710, 1058
  - exponential, 710, 1160
  - in spectroscopy, 1058, 1086
  - least-squares, 1058
  - moment constraints, 1086
  - polynomial data smoothing, 1058
  - Savitzky-Golay, 1058
- smoothing parameter selection, 1186
- smoothing splines, 1271
- SNR, *see* signal-to-noise ratio
- SOS, *see* second-order section
- sparse seasonal Whittaker-Henderson decomposition, 1370
- sparse Whittaker-Henderson methods, 1289
- spatial arrays, 393
- spectrum, 2, 202
  - analysis, 360, 368
  - and z-transform, 31, 202
  - and pole/zero pattern, 205
  - and time windowing, 31
  - and unit circle, 202
  - computation, 31, 360
  - DTFT, DFT, 360
  - hermitian property of, 207
  - numerical approximation of, 31
  - of analog signal, 2, 4
  - of oversampled signal, 624
  - of periodic signal, 779
  - of sampled signal, 30, 202, 623
  - of video signal, 750
  - of windowed sinusoids, 365
  - parametric methods, 368
  - periodicity of, 30, 35, 203
  - power, 395
  - replicating transformation, 745
  - replication, 5, 7, 10, 32, 677, 699
  - statistical reliability of, 368
- spherical Bessel functions, 1407
- SRC, *see* sampling rate conversion
- stability, 111, 114, 402
  - and causality, 115, 267
  - and inverse filters, 116, 264
  - condition, 114
  - in z-domain, 193, 200, 233, 244
  - marginal, 202, 205, 250
  - of multitap delays, 829
- standard-error bands, 1242
- Starc bands, 1242
- state
  - circular, 173–175, 181, 183
  - initialization, 159
  - internal, 98, 129, 154, 158, 173, 227, 276
  - machine, 95
- state space realization, 98, 120, 355
- steady state, 3
  - frequency response, 3
  - sinusoidal response, 3
- steady state response, 131, 133, 139, 239, 243
  - alternating-step response, 249
  - DC gain, 138, 249
  - for FIR filters, 131
  - for IIR filters, 239
  - unit-step response, 249
- steady-state EMA, 1180
- steepest descent, 1162
- step invariance, 1023
- STFT, 882
  - Bartlett window, 887
  - COLA property, 886
  - computation, 888
  - constant-overlap-add property, 886
  - definition, 883
  - hanning window, 887
  - inverse STFT, 886
  - overlap-add reconstruction, 886
  - phase vocoder, 891
  - phase vocoder model, 892
  - pitch-scale modification, 896
  - spectrogram, 884
  - STFT signal processing system, 887
  - time-scale modification, 891
- stochastic oscillator, 1254
- stock's beta, 1268
- sunspot time series, 403
- sunspot time series prediction, 1384
- superdirectivity ratio, Taylor's, 1418
- superoscillations, 1415

- tangency portfolio, 1265
- tapped delay line, *see* delay
- TDAC, 913, 915, 917
- technical analysis in financial markets, 1209
- thermostat control, 1053
- thricing, 1193
- Tikhonov regularization, 1414
- Tillson's T3 indicator, 1231
- time constant, 244, 300, 716, 806, 821, 823
- time delay, 101
- time invariance, 100, 101
- time window, *see* windowing
- time-bandwidth product, 1407
- time-bandwidth tradeoffs, 729
- time-domain aliasing cancellation, *see* TDAC
- time-scale modification, 891
- time-series forecast indicator, 1213
- transfer function, 111, 225, 233, 277
- transient response, 131, 133, 139, 242
  - in noise reduction filters, 707
  - in reverberation, 814, 823
  - input-off transients, 131, 161, 168
  - input-on transients, 131, 161
  - of FIR filters, 131, 252
  - of IIR filters, 242
  - time constant, 244
- transposed form, 188, 328
  - for FIR filter, 188
  - for IIR filter, 231
- transposed realization, 287
- transversal filter, *see* FIR filter
- trapezoidal, bilinear, Tustin transformation, 996, 1014
- TRIX oscillator, 1257
- Tukey's twicing operation, 1193
- twicing, 1193
- twicing and zero-lag filters, 1194
- twiddle factor, 419
- two mutual fund theorem, 1262

- unilateral z-transform, 215
- unit circle
  - and DTFT, 373, 412
- unit-step response, 249
- upsampler, 628

- variable and adaptive bandwidth, 1150
- variable-length EMA. VEMA, 1257
- vertical detail enhancement, 755, 759
- vertical horizontal filter, VHF, 1253
- video signal, 704, 751
- video spectrum, 750
- Vondrak filters, 1272

- waveform generators, 110, 767, 772
  - periodic, 110, 264, 772
  - sinusoidal, 767
- wavelets, 494
  - a trous operation, 946
  - analysis and synthesis filter banks, 946
  - analysis and synthesis with UWT, 967
  - decimated and undecimated filter banks, 967
  - denoising, 963
  - dilation equations, 933
  - discrete wavelet transform, 949
  - DWT in convolutional form, 960
  - DWT in matrix form, 952
  - fast DWT, 957
  - Haar & Daubechies scaling functions, 929
  - inverse DWT, 954
  - inverse UWT, 969
  - Mallat's algorithm, 944
  - MATLAB functions, 975
  - multiresolution analysis, 928
  - multiresolution and filter banks, 944
  - multiresolution decomposition, 931, 962
  - orthogonal DWT transformation, 958
  - periodized DWT, 954
  - refinement equations, 933
  - scaling and wavelet filters, 935, 939
  - symmlets, 936
  - UWT denoising, 974
  - UWT matrices, 969
  - UWT multiresolution decomposition, 972
  - UWT, undecimated wavelet transform, 966
  - visushrink method, 965
- wavetable
  - amplitude modulation, 796
  - circular, 767, 781
  - frequency modulation, 798
  - fundamental frequency of, 784
  - generator, 182, 767, 781, 787, 793
  - in computer music, 782, 783
  - linearly interpolated, 787
  - offset index, 785
  - oscillator, 793
  - synthesis, 182, 767, 782, 783, 793
  - waveshaping, 783, 824
- weighted local polynomial modeling, 1136
- weighted polynomial filters, 1102
- WEMA, Wilder's EMA, 1227
- Whittaker-Henderson smoothing, 1272
- windowing, 31, 360, 361, 367
  - Chebyshev window, 378, 390
  - DPSS window, 377, 388
  - Hamming window, 51, 366, 367, 386, 477
  - in frequency domain, 362
  - Kaiser window, 366, 387, 481
  - leakage, 31, 366, 368
  - rectangular, 361
  - rectangular window, 382
  - sidelobes, 364, 366
  - width, 366
  - window parameters, 374
- wrapping in DFT, 421

z-transforms, 31, 190
    and causality, 199
    and stability, 200
    basic properties of, 190
    modulation property of, 223
    pole/zero pattern of, 205
    region of convergence, 193, 195, 199
zero padding in DFT, 413, 450
zero-lag EMA, 1231
zero-lag filters, 723, 1194
zero-order hold, 997, 1018
zigzag indicator, 1237