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
Introduction to

Signal Processing

Second Edition

Sophocles J. Orfanidis

Rutgers University

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

Copyright © 2010 by Sophocles J. Orfanidis, first edition
Copyright © 2023 by Sophocles J. Orfanidis, second edition

This book was previously published by Pearson Education, Inc.

All rights reserved. No parts of this publication may be reproduced, stored in a retrieval
system, or transmitted in any form or by any means, electronic, mechanical, photocopy-
ing, recording or otherwise, without the prior written permission of the author.

Software tools:
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

Audio files:
Some example audio files were used from the following resources:

   https://uweb.engr.arizona.edu/~429rns/audiofiles/audiofiles.html
   https://uweb.engr.arizona.edu/~429rns/index.htm

   https://dafx.de/DAFX_Book_Page/matlab.html

http://www.ece.rutgers.edu/~orfanidi/intro2sp/2e
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
<table>
<thead>
<tr>
<th>19 DCT, MDCT, and Data Compression</th>
<th>905</th>
</tr>
</thead>
<tbody>
<tr>
<td>19.1 DCT and MDCT Compression Systems, 905</td>
<td></td>
</tr>
<tr>
<td>19.2 Discrete Cosine Transform, 907</td>
<td></td>
</tr>
<tr>
<td>19.3 DCT Compression System, 910</td>
<td></td>
</tr>
<tr>
<td>19.4 MDCT and Time-Domain Aliasing Cancellation, 913</td>
<td></td>
</tr>
<tr>
<td>19.5 Princen-Bradley Windows, 916</td>
<td></td>
</tr>
<tr>
<td>19.6 Derivations and Computer Experiments, 918</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>20 Discrete Wavelet Transforms</th>
<th>928</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.1 Multiresolution Analysis, 928</td>
<td></td>
</tr>
<tr>
<td>20.2 Dilation Equations, 933</td>
<td></td>
</tr>
<tr>
<td>20.3 Wavelet Filter Properties, 939</td>
<td></td>
</tr>
<tr>
<td>20.4 Multiresolution and Filter Banks, 944</td>
<td></td>
</tr>
<tr>
<td>20.5 Discrete Wavelet Transform, 949</td>
<td></td>
</tr>
<tr>
<td>20.6 Multiresolution Decomposition, 962</td>
<td></td>
</tr>
<tr>
<td>20.7 Wavelet Denoising, 963</td>
<td></td>
</tr>
<tr>
<td>20.8 Undecimated Wavelet Transform, 966</td>
<td></td>
</tr>
<tr>
<td>20.9 MATLAB Functions, 975</td>
<td></td>
</tr>
<tr>
<td>20.10 Problems, 977</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>21 Discretization Methods</th>
<th>979</th>
</tr>
</thead>
<tbody>
<tr>
<td>21.1 Continuous-Time Systems, 979</td>
<td></td>
</tr>
<tr>
<td>21.2 Mapping of Initial Conditions, 981</td>
<td></td>
</tr>
<tr>
<td>21.3 Forced Response, 983</td>
<td></td>
</tr>
<tr>
<td>21.4 Solution Procedures, 984</td>
<td></td>
</tr>
<tr>
<td>21.5 Steady-State Sinusoidal Response, 985</td>
<td></td>
</tr>
<tr>
<td>21.6 Continuous-Time Example, 986</td>
<td></td>
</tr>
<tr>
<td>21.7 Discretization Schemes - Summary, 994</td>
<td></td>
</tr>
<tr>
<td>21.8 Forward/Backward Euler, and Trapezoidal Rules, 996</td>
<td></td>
</tr>
<tr>
<td>21.9 Zero-Order and First-Order Holds, 997</td>
<td></td>
</tr>
<tr>
<td>21.10 Sample-by-Sample Processing, 999</td>
<td></td>
</tr>
<tr>
<td>21.11 Initialization Procedures, 1006</td>
<td></td>
</tr>
<tr>
<td>21.12 Forward/Backward Euler, and Trapezoidal Rules, 1014</td>
<td></td>
</tr>
<tr>
<td>21.13 Ideal Sampling, Starred Laplace Transform, z-Transform, 1015</td>
<td></td>
</tr>
<tr>
<td>21.14 Zero-Order Hold, 1018</td>
<td></td>
</tr>
<tr>
<td>21.15 Step Invariance and Impulse Invariance, 1023</td>
<td></td>
</tr>
<tr>
<td>21.16 First-Order Hold, 1025</td>
<td></td>
</tr>
<tr>
<td>21.17 Ramp Invariance, 1027</td>
<td></td>
</tr>
<tr>
<td>21.18 Appendix, 1027</td>
<td></td>
</tr>
<tr>
<td>21.19 MATLAB function - c2d2, 1029</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>22 Control Systems</th>
<th>1030</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.1 Feedback Control Systems, 1030</td>
<td></td>
</tr>
<tr>
<td>22.2 PID Control, 1032</td>
<td></td>
</tr>
<tr>
<td>22.3 Digital Control Systems, 1032</td>
<td></td>
</tr>
<tr>
<td>22.4 Examples, 1035</td>
<td></td>
</tr>
<tr>
<td>22.4.1 Cruise Control, 1035</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>23 Local Polynomial Filters</th>
<th>1058</th>
</tr>
</thead>
<tbody>
<tr>
<td>23.1 Introduction, 1058</td>
<td></td>
</tr>
<tr>
<td>23.2 Local Polynomial Fitting, 1059</td>
<td></td>
</tr>
<tr>
<td>23.3 Exact Design Equations, 1068</td>
<td></td>
</tr>
<tr>
<td>23.4 Geometric Interpretation, 1073</td>
<td></td>
</tr>
<tr>
<td>23.5 Orthogonal Polynomial Bases, 1074</td>
<td></td>
</tr>
<tr>
<td>23.6 Polynomial Predictive and Interpolation Filters, 1075</td>
<td></td>
</tr>
<tr>
<td>23.7 Farrow Realization Structures, 1078</td>
<td></td>
</tr>
<tr>
<td>23.8 Minimum Variance Filters, 1082</td>
<td></td>
</tr>
<tr>
<td>23.9 Predictive Differentiation Filters, 1088</td>
<td></td>
</tr>
<tr>
<td>23.10 Filtering Implementations, 1093</td>
<td></td>
</tr>
<tr>
<td>23.11 Minimum Roughness Weighted Polynomial Filters, 1102</td>
<td></td>
</tr>
<tr>
<td>23.12 Henderson Filters, 1108</td>
<td></td>
</tr>
<tr>
<td>23.13 Hahn Orthogonal Polynomials, 1118</td>
<td></td>
</tr>
<tr>
<td>23.14 Maximally-Flat Filters, 1126</td>
<td></td>
</tr>
<tr>
<td>23.15 Missing Data and Outliers, 1131</td>
<td></td>
</tr>
<tr>
<td>23.16 Weighted Local Polynomial Modeling, 1136</td>
<td></td>
</tr>
<tr>
<td>23.17 Bandwidth Selection with CV and GCV, 1143</td>
<td></td>
</tr>
<tr>
<td>23.18 Local Polynomial Interpolation, 1145</td>
<td></td>
</tr>
<tr>
<td>23.19 Variable and Adaptive Bandwidth, 1150</td>
<td></td>
</tr>
<tr>
<td>23.20 Repeated Observations, 1156</td>
<td></td>
</tr>
<tr>
<td>23.21 Loess Smoothing, 1157</td>
<td></td>
</tr>
<tr>
<td>23.22 Problems, 1159</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>24 Exponential Moving Average Filters</th>
<th>1160</th>
</tr>
</thead>
<tbody>
<tr>
<td>24.1 Mean Tracking, 1160</td>
<td></td>
</tr>
<tr>
<td>24.2 Forecasting and State-Space Models, 1169</td>
<td></td>
</tr>
<tr>
<td>24.3 Higher-Order Polynomial Smoothing Filters, 1170</td>
<td></td>
</tr>
<tr>
<td>24.4 Linear Trend FIR Filters, 1172</td>
<td></td>
</tr>
<tr>
<td>24.5 Higher-Order Exponential Smoothing, 1174</td>
<td></td>
</tr>
<tr>
<td>24.6 Steady-State Exponential Smoothing, 1180</td>
<td></td>
</tr>
<tr>
<td>24.7 Smoothing Parameter Selection, 1186</td>
<td></td>
</tr>
<tr>
<td>24.8 Single, Double, Triple Exponential Smoothing, 1191</td>
<td></td>
</tr>
<tr>
<td>24.9 Tukey's Twicing Operation, 1193</td>
<td></td>
</tr>
<tr>
<td>24.10 Zero-Lag Filters and Twicing, 1194</td>
<td></td>
</tr>
<tr>
<td>24.11 Local Level, Local Slope, Local Acceleration Filters, 1196</td>
<td></td>
</tr>
<tr>
<td>24.12 Basis Transformations and EMA Initialization, 1198</td>
<td></td>
</tr>
<tr>
<td>24.13 Holt's Exponential Smoothing, 1203</td>
<td></td>
</tr>
<tr>
<td>24.14 Problems, 1205</td>
<td></td>
</tr>
</tbody>
</table>
CONTENTS

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 Lo 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.

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. Jeng, 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
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:

\[
X(\Omega) = \int_{-\infty}^{\infty} x(t) e^{-j\Omega t} \, dt
\]

(1.2.1)

where \( \Omega \) is the radian frequency\(^\dagger\) in \([\text{radians/second}]\). The ordinary frequency \( f \) in \([\text{Hertz}] \) or \([\text{cycles/sec}] \) is related to \( \Omega \) by

\[
\Omega = 2\pi f
\]

(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:

\[
x(t) = \int_{-\infty}^{\infty} X(\Omega) e^{j\Omega t} \frac{d\Omega}{2\pi}
\]

(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

\[
x(t) = e^{-\alpha t} e^{j\Omega t} u(t) = e^{s_1 t} u(t)
\]

where \( s_1 = -\alpha + j\Omega \), 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 \( x(t) \):

\[
\begin{align*}
x(t) &\rightarrow \text{linear system } h(t) \rightarrow y(t)
\end{align*}
\]

\(^\dagger\)We use the notation \( \Omega \) to denote the physical frequency in units of \([\text{radians/sec}]\), and reserve the notation \( \omega \) to denote digital frequency in \([\text{radians/sample}]\).
1.2. REVIEW OF ANALOG SIGNALS

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:

\[
Y(\Omega) = H(\Omega) X(\Omega)
\]

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

\[
H(\Omega) = \int_{-\infty}^{\infty} h(t) e^{-j\Omega t} \, dt
\]

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

\[
x(t) = e^{j\Omega t} \quad \text{linear system} \quad H(\Omega) \quad y(t) = H(\Omega) e^{j\Omega t}
\]

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

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

By linear superposition, if the input consists of the sum of two sinusoids of frequencies \( \Omega_1 \) and \( \Omega_2 \) and relative amplitudes \( A_1 \) and \( A_2 \),

\[
x(t) = A_1 e^{j\Omega_1 t} + A_2 e^{j\Omega_2 t}
\]

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

\[
y(t) = A_1 H(\Omega_1) e^{j\Omega_1 t} + A_2 H(\Omega_2) e^{j\Omega_2 t}
\]

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:

![Diagram showing frequency response and sinusoidal inputs and outputs](image)

The input spectrum \( X(\Omega) \) consists of two sharp spectral lines at frequencies \( \Omega_1 \) and \( \Omega_2 \), 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):

\[
Y(\Omega) = H(\Omega) X(\Omega) = H(\Omega) \{2\pi A_1 \delta(\Omega - \Omega_1) + 2\pi A_2 \delta(\Omega - \Omega_2)\}
\]

\[
= 2\pi A_1 H(\Omega_1) \delta(\Omega - \Omega_1) + 2\pi A_2 H(\Omega_2) \delta(\Omega - \Omega_2)
\]

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 \( \Omega_1 \) represents a desired signal and \( \Omega_2 \) an unwanted interference, then a filter may be designed that lets \( \Omega_1 \) pass through, while at the same time it filters out the \( \Omega_2 \) 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 \( T \):

\[
t = nT, \quad n = 0, 1, 2, \ldots
\]

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 \( T \)?

![Diagram illustrating sampling process](image)
1.3. SAMPLING THEOREM

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:

$$f_s = \frac{1}{T}$$  \hspace{1cm} (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.

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, \(f' = f + mf_s, m = 0, \pm 1, \pm 2, \ldots\). 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:

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 \(f_{\text{max}}\), 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 \(f_{\text{max}}\), that is,

$$f_s \geq 2f_{\text{max}}$$  \hspace{1cm} (1.3.2)

or, expressed in terms of the sampling time interval,

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

The minimum sampling rate allowed by the sampling theorem, that is, \(f_s = 2f_{\text{max}}\), 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:

$$\left\{ f_s \right\}_{\text{max}} = 2f_{\text{max}} = \text{Nyquist rate}$$

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

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

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 \(f_{\text{max}}\) and \(f_s\)
depend on the application. Typical sampling rates for some common DSP applications are shown in the following table.

<table>
<thead>
<tr>
<th>application</th>
<th>$f_{\text{max}}$</th>
<th>$f_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>geophysical</td>
<td>500 Hz</td>
<td>1 kHz</td>
</tr>
<tr>
<td>biomedical</td>
<td>1 kHz</td>
<td>2 kHz</td>
</tr>
<tr>
<td>mechanical</td>
<td>2 kHz</td>
<td>4 kHz</td>
</tr>
<tr>
<td>speech</td>
<td>4 kHz</td>
<td>8 kHz</td>
</tr>
<tr>
<td>audio</td>
<td>20 kHz</td>
<td>40 kHz</td>
</tr>
<tr>
<td>video</td>
<td>4 MHz</td>
<td>8 MHz</td>
</tr>
</tbody>
</table>

### 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, $f_{\text{max}}$, must be taken to be at most equal to the Nyquist frequency $f_s/2$, that is, $f_{\text{max}} \leq f_s/2$. This operation is shown in Fig. 1.3.5.

The output of the analog prefilter will then be **bandlimited** to maximum frequency $f_{\text{max}}$ 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.

![Antialiasing prefilter](image)

**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.

![Aliasing in the time domain](image)

**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 $T_{\text{proc}}$ seconds, required for each sample. The time interval $T$ between input samples must be greater than $T_{\text{proc}}$; otherwise, the processor would not be able to keep up with the incoming samples. Thus,

$$T \geq T_{\text{proc}}$$

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

$$2f_{\text{max}} \leq f_s \leq f_{\text{proc}}$$

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 $f_s \geq 2f_{\text{max}}$, can be derived intuitively by considering the sampling of sinusoidal signals only. Figure 1.4.1 shows a sinusoid of frequency $f$,

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

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.

![Sinusoid sampled at rates](image)

**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,\(^1\) 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 $f_s/f$:

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

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

$$\frac{f_s}{f} \geq 2 \text{ samples/cycle} \quad \Rightarrow \quad f_s \geq 2f \quad (1.4.1)$$

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 $f_{\text{max}}$, then by choosing $f_s \geq 2f_{\text{max}}$, 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:

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

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

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

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

\(^1\)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:

$$x(t) = e^{jft} = e^{2\pi jft}$$

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

$$x(nT) = e^{jfnT} = e^{2\pi jfnT}$$

Define also the following family of sinusoids, for $m = 0, \pm 1, \pm 2, \ldots$,

$$x_m(t) = e^{2\pi j(ft + mf_n)t}$$

and their sampled versions,

$$x_m(nT) = e^{2\pi jfnT}e^{2\pi jfnmT} = e^{2\pi jfnT} = x(nT)$$

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

$$e^{2\pi jfnmT} = e^{2\pi jfnm} = 1$$

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

$$x_m(nT) = e^{2\pi jfnT}e^{2\pi jfnmT} = e^{2\pi jfnT} = x(nT)$$

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,

$$f, f \pm f_s, f \pm 2f_s, \ldots, f \pm mf_s, \ldots$$

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.\(^1\) It is obtained by reducing the original $f$ modulo $f_s$, that is,

\(^1\)The only exception is when it falls exactly on the left or right edge of the interval, $f = \pm f_s/2$. 

1. SAMPLING AND RECONSTRUCTION
1.4. SAMPLING OF SINUSOIDS

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\(^1\)

\[ f_a = f \mod(f_s) \]  

(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 \(|f| \leq f_s/2\), 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 \( f_a = f \mod(f_s) \) versus the true frequency \( f \). Observe how the straight line \( f_{true} = f \) 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, \(|f| \leq f_s/2\), 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, \( m = 0, \pm 1, \pm 2, \ldots \), or,

\[ \ldots, -26, -14, -2, 10, 22, 34, 46, \ldots \]

and among these only \( f_a = 10 \mod(12) = 10 - 12 = -2 \) 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.

\(^1\)This differs slightly from a true modulo operation; the latter would bring \( f \) into the right-sided Nyquist interval \([0, f_s]\).

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 \(20 \leq f \leq 60\) kHz will be aliased with \(-20 = 20 - 40 \leq f - f_s \leq 60 - 40 = 20\) 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:

\[ -\sin(14\pi t), \ -\sin(6\pi t), \ \sin(2\pi t), \ \sin(10\pi t), \ \sin(18\pi t) \]

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:

\[ -7, -3, 1, 5, 9 \text{ Hz} \]

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:

\[ f_m = 1 + 4m, \quad m = -2, -1, 0, 1, 2 \]

we can express the five sinusoids as:

\[ x_m(t) = \sin(2\pi f_m t) = \sin(2\pi (1 + 4m) t), \quad m = -2, -1, 0, 1, 2 \]
Replacing \( t = nT = n/f_s = n/4 \) sec, we obtain the sampled signals:

\[
x_m(nT) = \sin(2\pi(1 + 4m)n/4)
\]

which are the same, independently of \( m \). The following figure shows the five sinusoids over the interval \( 0 \leq t \leq 1 \) 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, \( f_{\text{max}} = f_4 = 1.5 \) kHz and the Nyquist rate will be \( 2f_{\text{max}} = 3 \) 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 \( f_{a_1} = f_1 \) and \( f_{a_2} = f_2 \). But \( f_3 \) and \( f_4 \) lie outside the Nyquist interval and they will be aliased with

\[
\begin{align*}
  f_{a_3} &= f_3 \mod(f_s) = 1 \mod(1.5) = 1 - 1.5 = -0.5 \text{ kHz} \\
  f_{a_4} &= f_4 \mod(f_s) = 1.5 \mod(1.5) = 1.5 - 1.5 = 0 \text{ kHz}
\end{align*}
\]

The aliased signal \( x_a(t) \) is obtained from \( x(t) \) by replacing \( f_1, f_2, f_3, f_4 \) by \( f_{a_1}, f_{a_2}, f_{a_3}, f_{a_4} \). Thus, the signal

\[
x(t) = 4 \cos(2\pi f_1t) + 3 \cos(2\pi f_2t) + 2 \cos(2\pi f_3t) + \cos(2\pi f_4t)
\]

will be aliased with

\[
x_a(t) = 4 \cos(2\pi f_{a_1}t) + 3 \cos(2\pi f_{a_2}t) + 2 \cos(2\pi f_{a_3}t) + \cos(2\pi f_{a_4}t)
\]

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 \( 4 \). Similarly, the two spectral lines of strength 2/2 at \( f_3 = \pm 1 \) kHz replicate onto \( f = \pm 0.5 \) kHz and the amplitudes add to give \( 3/2 + 2/2 = 2.5 \). 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:

\[
5 + 2.5e^{2\pi i0.5t} + 2.5e^{-2\pi i0.5t} = 5 + 5 \cos(\pi t)
\]

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 \( x_1(nT) = x_a(nT) = x_2(nT) \).

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:

\[
x(t) = \sin(\pi t) + 2[\sin(3\pi t + 2\pi t) + \sin(3\pi t - 2\pi t)] = 3 \sin(\pi t) + 2 \sin(5\pi t)
\]
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 \( f_{2a} = f_s \mod (f_2) = 2.5 \mod (3) = 2.5 - 3 = -0.5 \). Thus, the given signal will “appear” as:

\[
x_a(t) = 3 \sin (2\pi f_1 t) + 2 \sin (2\pi f_2 t)
\]

\[= 3 \sin (\pi t) + 2 \sin (-\pi t) = 3 \sin (\pi t) - 2 \sin (\pi t)
\]

\[= \sin (\pi t)
\]

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,

\[x_1(t) = 3 \sin (7\pi t) + 2 \sin (5\pi t)
\]

\[x_2(t) = 3 \sin (13\pi t) + 2 \sin (11\pi t)
\]

where we replaced \( \{f_1, f_2\} = \{3.5, 2.5\} \) for \( x_1(t) \), and by \( \{f_1 + 2f_2, f_2 + f_3\} = \{6.5, 5.5\} \) for \( x_2(t) \).

**Example 1.4.6:** Consider a periodic square wave with period \( T_0 = 1 \) sec, defined within its basic period \( 0 \leq t < 1 \) 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, \( m = 1, 3, 5, 7, \ldots \) is given by

\[
x(t) = \sum_{m=1,3,5,\ldots} b_m \sin (2\pi mt) = b_1 \sin (2\pi t) + b_3 \sin (6\pi t) + b_5 \sin (10\pi t) + \cdots
\]

(1.4.4)

where \( b_m = 4/(\pi m) \), \( m = 1, 3, 5, \ldots \). 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 \( f_1 = 1 \) harmonic lies within the Nyquist interval \([-2, 2] \) Hz. For the rate \( f_s = 8 \) Hz, only \( f_1 = 1 \) and \( f_2 = 3 \) Hz lie in \([-4, 4] \) Hz. The following table shows the true frequencies and the corresponding aliased frequencies in the two cases:

<table>
<thead>
<tr>
<th>( f_s )</th>
<th>( f )</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 Hz</td>
<td>( f \mod (4) )</td>
</tr>
<tr>
<td>8 Hz</td>
<td>( f \mod (8) )</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>-1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>-5</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>11</td>
<td>-9</td>
</tr>
<tr>
<td>13</td>
<td>11</td>
</tr>
<tr>
<td>15</td>
<td>\ldots</td>
</tr>
</tbody>
</table>

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

\[
x_a(t) = b_1 \sin (2\pi t) - b_3 \sin (6\pi t) + b_5 \sin (2\pi t) - b_7 \sin (6\pi t) + \cdots
\]

(1.4.5)

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

\[
x_a(t) = (b_1 - b_7 + b_9 - b_{13} + \cdots) \sin (2\pi t) + (b_3 - b_5 + b_{11} - b_{15} + \cdots) \sin (6\pi t)
\]

(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, \( x_a(nT) = A \sin (2\pi n T/4) = A \sin (\pi n T/2) \). 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 \( x_a(nT) = B \sin (3\pi n T/4) + C \sin (3\pi n T/4) \). Demanding the condition \( x_a(nT) = x(nT) \) at \( n = 1, 2 \) gives the two equations

\[
B \sin (\pi/4) + C \sin (3\pi/4) = 1 \quad \Rightarrow \quad B + C = \sqrt{2}
\]

\[
B \sin (\pi/2) + C \sin (3\pi/2) = 1 \quad \Rightarrow \quad B - C = 1
\]

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:
1.4. SAMPLING OF SINUSOIDS

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:

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

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

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

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

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

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

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

The desired coefficients $A, B, C$ are then:

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

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

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

The above results generalize to any sampling rate $f_s = M f$, 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

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

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:

<table>
<thead>
<tr>
<th>coefficients</th>
<th>4 Hz</th>
<th>8 Hz</th>
<th>12 Hz</th>
<th>16 Hz</th>
<th>$\infty$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_1$</td>
<td>1</td>
<td>1.207</td>
<td>1.244</td>
<td>1.257</td>
<td>1.273</td>
</tr>
<tr>
<td>$b_3$</td>
<td>-</td>
<td>0.207</td>
<td>0.333</td>
<td>0.374</td>
<td>0.424</td>
</tr>
<tr>
<td>$b_5$</td>
<td>-</td>
<td>-</td>
<td>0.089</td>
<td>0.167</td>
<td>0.255</td>
</tr>
<tr>
<td>$b_7$</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.050</td>
<td>0.182</td>
</tr>
</tbody>
</table>

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.

Example 1.4.7: A sound wave has the form:

$$x(t) = 2A \cos(10\pi t) + 2B \cos(30\pi t) + 2C \cos(50\pi t) + 2D \cos(60\pi t) + 2E \cos(90\pi t) + 2F \cos(125\pi t)$$

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:

$$x(t) \xrightarrow{\text{analog}} H(f) \xrightarrow{\text{analog}} y(t) \xrightarrow{\text{40 kHz}} \xrightarrow{\text{digital}} \xrightarrow{\text{analog reconstructor}} y_a(t)$$

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:
1.4. SAMPLING OF SINUSOIDS

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:

\[
\begin{align*}
    f_A &= 5 \text{ kHz} & f_C &= 25 \text{ kHz} & f_E &= 45 \text{ kHz} \\
    f_B &= 15 \text{ kHz} & f_D &= 30 \text{ kHz} & f_F &= 62.5 \text{ kHz}
\end{align*}
\]

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:

\[
2A \cos(2\pi f_A t) = A e^{j2\pi f_A t} + A e^{-j2\pi f_A t} \quad A \delta(f - f_A) + A \delta(f + f_A)
\]

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

![Spectrum of x(t)](image)

The sampling process will replicate each of these peaks at multiples of \( f_s = 40 \text{ kHz} \). 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:

\[
\begin{align*}
    f_C &= 25 \Rightarrow f_{C,a} &= f_C \text{ mod } f_s = f_C - f_s = 25 - 40 = -15 \\
    f_D &= 30 \Rightarrow f_{D,a} &= f_D \text{ mod } f_s = f_D - f_s = 30 - 40 = -10 \\
    f_E &= 45 \Rightarrow f_{E,a} &= f_E \text{ mod } f_s = f_E - f_s = 45 - 40 = 5 \\
    f_F &= 62.5 \Rightarrow f_{F,a} &= f_F \text{ mod } f_s = f_F - 2f_s = 62.5 - 2 \times 40 = -17.5
\end{align*}
\]

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:

\[
y_a(t) = 2A \cos(10\pi t t) + 2B \cos(30\pi t t) + 2C \cos(-2\pi 15t) + 2D \cos(-2\pi 10t) + 2E \cos(2\pi 5t) + 2F \cos(-2\pi 17.5t) = (A + E) \cos(10\pi t t) + 2(B + C) \cos(30\pi t t) + 2D \cos(20t t) + 2F \cos(35t t)
\]

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

\[
2C \cos(2\pi f_C t) \rightarrow 2C \cos(2\pi f_{C,a} t)
\]

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 \( f_c/2 = 20 \text{ kHz} \) 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:

![Prefilter Effect](image)

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:

\[
y(t) = 2A|H(f_A)| \cos(10\pi t t) + 2B|H(f_B)| \cos(30\pi t t) + 2C|H(f_C)| \cos(50\pi t t) + 2D|H(f_D)| \cos(60\pi t t) + 2E|H(f_E)| \cos(90\pi t t) + 2F|H(f_F)| \cos(125\pi t t)
\]

(1.4.7)
1.4. SAMPLING OF SINUSOIDS

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:

$$2A \cos(2\pi f_A t) \implies 2A |H(f_A)| \cos(2\pi f_A t + \theta(f_A))$$

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):

$$|H(f_A)| = |H(f_b)| = |H(f_c)| = |H(f_d)| = |H(f_e)| = |H(f_f)| = 0$$

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:

at $f_c$: 60 dB/octave $\times$ 0.322 octaves = 19.3 dB
at $f_d$: 60 dB/octave $\times$ 0.585 octaves = 35.1 dB
at $f_e$: 60 dB/octave $\times$ 1.170 octaves = 70.1 dB
at $f_f$: 60 dB/octave $\times$ 1.644 octaves = 98.6 dB

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:

1. The number of octaves is the number of powers of two, that is, if $f_2 = 2^n f_1 = n = \log_2 (f_2/f_1)$.

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

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

Its spectrum is shown below:

Notice how the audible 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:

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

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 $N = 6N$ 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,
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.
1.4. SAMPLING OF SINUSOIDS

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 ≤ |f| ≤ 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 ≤ f ≤ 100. Any components of the analog input y(t) that lie in that interval would be aliased into the interval \( 60 - 80 \) ≤ f − f_s ≤ 100 − 80, which is the desired audio band −20 ≤ f − f_s ≤ 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 ≤ |f| ≤ 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 × 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 × 47.6 dB = 169.7 dB attenuation. At \( f_s = 62.5 \) kHz, it provides 49.3 dB suppression, as was calculated in Example 1.4.7, that is, |H(\( f_s \))| = \( 10^{-49.3/20} = 1/292 \).

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 \( x(t) = e^{j2\pi ft} \) 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:

\[
\omega = \Omega T = 2\pi f T = \frac{2\pi f}{f_s}
\]

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 \( \omega \), the sampled sinusoid reads simply

\[
x(nT) = e^{2\pi j\frac{\omega}{f_s}}
\]

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

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

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

\[
\omega_a = \omega \mod (2\pi)
\]

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 \( (f + mf_s)T \). Indeed, 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 \( f_a = f \mod (f_s) \). 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 \( f_2 = f_1 + f_s \).
1.4. SAMPLING OF SINUSOIDS

The first wheel covers \( f_1 T = f_1 / f_s = 1/4 \) 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 \( n = 0, 1, 2, 3, \ldots \) 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 \( 1/4 \) of a revolution 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 \( n = 0, 1, 2, 3, \ldots \), 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):

\[
f_\omega = f_2 \ mod(f_s) = 5 \ mod(4) = 5 - 4 = 1
\]

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

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

The fourth wheel is more interesting. It covers \( f_4 T = -3/4 \) 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 \( n = 0, 1, 2, 3, \ldots \), 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 \( f_\omega = f_4 \ mod(f_s) = (-3) \ mod(4) = -3 + 4 = 1 \) 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 \( \omega < \pi \), 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 \( n = 0, 1, 2, 3, \ldots \) 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 \( n = 0, 1, 2, 3, \ldots \) 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 \( \omega_a = \omega \ mod(2\pi) = (5\pi/4 \ mod(2\pi)) = 5\pi/4 - 2\pi = -3\pi/4 \) or frequency \( f_a = (3/8 \ cycle)/(1/4 \ sec) = -1.5 \ Hz \).

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 \( f_a = f \ mod(f_s) = 4 \ mod(4) = 4 - 4 = 0 \).

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:

\[
e^{2\pi j f T n} = e^{2\pi j f f_s n} = e^{i\omega T n} = e^{i\omega n}
\]

being expressed more simply in terms of \( \omega \). 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 \( \omega \) or \( f / f_s \) are the most convenient.
1.5. SPECTRA OF SAMPLED SIGNALS

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:

\[
\hat{X}(f) = \int_{-\infty}^{\infty} \hat{x}(t) e^{-2\pi i ft} \, dt
\]  

(1.5.3)

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

\[
\hat{X}(f) = \sum_{n=-\infty}^{\infty} x(nT) \delta(t - nT) e^{-2\pi i ft} \, dt = \sum_{n=-\infty}^{\infty} x(nT) \int_{-\infty}^{\infty} \delta(t - nT) e^{-2\pi i ft} \, dt
\]

or,

\[
\hat{X}(f) = \sum_{n=-\infty}^{\infty} x(nT) e^{-2\pi i ft n}
\]  

(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)\(^\dagger\) 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 \( e^{-2\pi i ft} \) 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:

\[
x(nT) = \frac{1}{T_s} \int_{-f_s/2}^{f_s/2} \hat{X}(f) e^{2\pi i fT} df = \frac{1}{2\pi} \int_{-\pi}^{\pi} \hat{X}(\omega) e^{int} \, d\omega
\]  

(1.5.5)

\(^\dagger\)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 $\omega = 2\pi f / f_s$.\(^3\)

Eq. (1.5.5) is the inverse DFT and expresses the discrete-time signal $x(nT)$ as a superposition of discrete-time sinusoids $e^{jn\omega}$.\(^2\)

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,

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-2\pi j ft} dt = \sum_{n=-\infty}^{\infty} x(nT) e^{-2\pi j fnT} \cdot T \quad \text{or},$$

$$X(f) \approx T \hat{X}(f) \quad (1.5.6)$$

This approximation becomes exact in the continuous-time limit:

$$X(f) = \lim_{T \to \infty} T \hat{X}(f) \quad (1.5.7)$$

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, $n = 0, 1, 2, \ldots, L - 1$, so that Eq. (1.5.4) is computed approximately by the truncated sum:

$$\hat{X}(f) \approx \hat{X}_L(f) = \sum_{n=0}^{L-1} x(nT) e^{-2\pi j fnT} \quad (1.5.8)$$

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 $z = e^{j\omega} = e^{2\pi j fT}$, we may write Eq. (1.5.4) as the $z$-transform\(^\dagger\)

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

\(^\dagger\)Abusing the notation slightly, we wrote $\hat{X}(\omega)$ for $\hat{X}(f)$.

\(^\ddagger\)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,

$$x^*(t) = \sum_{n=-\infty}^{\infty} x(nT) \delta(t - nT)$$

$$X^*(s) = \int_{-\infty}^{\infty} x^*(t) e^{-st} dt = \sum_{n=-\infty}^{\infty} x(nT) e^{-sTn}$$

$$X^*(f) = X^*(s) \bigg|_{s=2\pi j} = \sum_{n=-\infty}^{\infty} x(nT) e^{-2\pi j fnT}$$

$$X^*(z) = X^*(s) \bigg|_{z=e^{sT}} = \sum_{n=-\infty}^{\infty} x(nT) z^{-n} \quad (1.5.10)$$

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

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

which consists of the following series of steps going from $X(s)$ to $X^*(s)$, 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,

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

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

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

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 $n$th 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:

$$\hat{x}(t) = x(t) \sum_{n=-\infty}^{\infty} \delta(t - nT) = x(t) s(t) \quad (1.5.14)$$
1.5. SPECTRA OF SAMPLED SIGNALS

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

\[
 s(t) = \sum_{m=-\infty}^{\infty} \delta(t - nT) = \frac{1}{T} \sum_{m=-\infty}^{\infty} e^{2\pi jmf_0 t} \tag{1.5.15}
\]

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

\[
 \hat{x}(t) = x(t)s(t) = \frac{1}{T} \sum_{m=-\infty}^{\infty} x(t) e^{2\pi jmf_0 t}
\]

and using the modulation property of Fourier transforms, which states that if \( X(f) \) is the transform of \( x(t) \) then \( \hat{X}(f-f_c) \) is the transform of \( x(t)e^{2\pi jf_0 t} \), we obtain by taking Fourier transforms of both sides,

\[
 \hat{X}(f) = \frac{1}{T} \sum_{m=-\infty}^{\infty} X(f - mf_0) \tag{1.5.16}
\]

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 \( X(f) \).

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:

\[
 S(f) = \frac{1}{T} \sum_{m=-\infty}^{\infty} \delta(f - mf_0) \tag{1.5.18}
\]

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

\[
 \hat{X}(f) = \frac{1}{T} \sum_{m=-\infty}^{\infty} \int_{-\infty}^{\infty} X(f - f') \delta(f' - mf_0) \, df' = \frac{1}{T} \sum_{m=-\infty}^{\infty} X(f - mf_0)
\]

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

\[
 \hat{X}(f) = \frac{1}{T} \sum_{m=-\infty}^{\infty} x(nT)e^{-2\pi jfn} = \frac{1}{T} \sum_{m=-\infty}^{\infty} X(f - mf_0) \tag{1.5.19}
\]

This is known as the Poisson summation formula. We also see from Fig. 1.5.2 that as \( T \to 0 \), 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 \( f_{\text{max}} \) 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 \( \delta = f_s - 2f_{\text{max}} \), known as the guard band. It follows that the replicas will not overlap if \( \delta > 0 \), or equivalently, \( f_s > 2f_{\text{max}} \). But they will overlap if \( f_s < 2f_{\text{max}} \) 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,

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

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 \( H_{DSP}(f) \), the spectrum \( \hat{X}(f) \) of the input sequence will be reshaped into the output spectrum:

\[
\hat{Y}(f) = H_{DSP}(f) \hat{X}(f)
\]

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)

\[
y(nT) = \int_{-\pi}^{\pi} \hat{Y}(\omega) e^{j\omega nT} d\omega = \int_{-\pi}^{\pi} H_{DSP}(\omega) \hat{X}(\omega) e^{j\omega nT} d\omega
\]

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 \( f_0 \), \( x(t) = e^{j2\pi f_0 t} \). Its Fourier transform is the spectral line \( X(f) = \delta(f - f_0) \). It follows from Eq. (1.5.16) that the sampled sinusoid:

\[
\hat{x}(t) = \sum_{n=-\infty}^{\infty} x(nT) \delta(t - nT) = \sum_{n=-\infty}^{\infty} e^{j2\pi f_0 T n} \delta(t - nT)
\]

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 \( \{f_0 + mf_s, m = 0, \pm 1, \pm 2, \ldots\} \) in accordance with Fig. 1.3.2 and our remarks in Sections 1.4 and 1.3.\( \Box \)

**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:

\[
x(t) = e^{-at} u(t) \\
X(f) = \frac{1}{a + 2\pi f f}
\]

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

\[
\hat{X}(f) = \sum_{n=-\infty}^{\infty} x(nT) e^{-2\pi j f nT} = \sum_{n=0}^{\infty} e^{-at} e^{-2\pi j f nT}
\]

and summing the geometric series, we get

\[
\hat{X}(f) = \frac{1}{1 - e^{-aT} e^{-2\pi j f T}} = \frac{1}{1 - e^{-aT} e^{-2\pi j f T}}
\]

Its magnitude square is

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

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

\[
\frac{1}{T} \sum_{m=-\infty}^{\infty} \frac{1}{a + 2\pi j (f - mf_s)} = \frac{1}{1 - e^{-at} e^{-2\pi j f T}} = \frac{1}{1 - e^{-at} e^{-2\pi j f T}}
\]

The left graph in Fig. 1.5.4 compares the periodic spectrum \( |\hat{X}(f)|^2 \) with the original analog spectrum \( |X(f)|^2 = 1/(a^2 + (2\pi f)^2) \). The spectra are shown in decibels, that is, \( 20 \log_{10} |X(f)| \). The parameter \( a = 0.2 \) 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 \( e^{-x} \approx 1 - x \), valid for small \( x \), or L'Hospital's rule, we obtain

\[
\lim_{T \to 0} T \hat{X}(f) = \lim_{T \to 0} \frac{T}{1 - e^{-at} e^{-2\pi j f T}} = \frac{1}{a + 2\pi j f} = X(f)
\]

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, \( |T \hat{X}_L(f)|^2 \), and with what we can actually compute based on a finite record of samples, \( |T \hat{X}_L(f)|^2 \).

\(^1\)Please see a corrected discussion at the end of this example.
1.5. SPECTRA OF SAMPLED SIGNALS

37

1. SAMPLING AND RECONSTRUCTION

Poisson summation formula. The issue has been discussed in connection with the impulse-invariance filter design method.\(^1\) 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.\(^2\) Indeed, setting \(z = (2\pi fT - j\pi) / 2\), it is not hard to verify that the identity takes on the standard form:

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

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.

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

\[
\hat{X}_L(f) = \sum_{n=0}^{L-1} e^{-j2\pi fn/T} = \frac{1 - e^{-j2\pi fT/L}}{1 - e^{-j2\pi fT}}
\]

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:

\[
\frac{1}{T} \sum_{m=-\infty}^{\infty} a + 2\pi j(f - mT) = \frac{1}{1 - e^{-j2\pi fT}} - \frac{1}{2}
\]

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,

\[
\hat{X}(f) = \sum_{n=0}^{\infty} x(nT)e^{-j2\pi fn/T} = \sum_{n=0}^{\infty} (e^{-j\pi n} - 1/2) \delta(n) e^{-j2\pi fn/T} = \frac{1 - e^{-j2\pi fT}}{1 - e^{-j2\pi fT}} - \frac{1}{2}
\]

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 \( (x(0+) + x(0-))/2 = (1 + 0)/2 = 1/2 \), enables the use of the

![Fig. 1.5.4 Spectra of analog, sampled, and windowed signals.](image)

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 \( \hat{X}(f) \) can also be computed in closed form. Using the finite geometric series:

\[
\sum_{n=0}^{\infty} x^n = \frac{1 - x^L}{1 - x}
\]

we obtain:

\[
\hat{X}_L(f) = \sum_{n=0}^{L-1} e^{-j2\pi fn/T} = \frac{1 - e^{-j2\pi fT/L}}{1 - e^{-j2\pi fT}}
\]

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


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 \( [-f_{pass}, f_{pass}] \) 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 \( H_D(f) \), whose frequency response is the inverse of the response of the prefilter over the passband.

![Fig. 1.5.6 Ideal antialiasing prefilter.](image)
1.5.1 SPECTRA OF SAMPLED SIGNALS

1. SAMPLING AND RECONSTRUCTION

1.5. SPECTRA OF SAMPLED SIGNALS

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 $\alpha_2$ is in dB/octave and is related to $\alpha_{10}$ by:

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

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

$$X(f) = H(f) X(f_0)$$

or, in terms of attenuations in dB:

$$A_X(f) = A(f) + A_{X_0}(f)$$

where $A_X(f) = -20 \log_{10} |X(f)/X(f_0)|$ and similarly for $A_{X_0}(f)$. 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 $A_{X_0}(f)$ 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 $z f_s = \pm 12$ kHz.

*Diagram not shown in text*
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 \( y = 15 \text{ dB} \) 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, \( x = y = 15 \text{ dB} \). 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, \( f_{\text{pass}} = 4 \text{ kHz} \) and \( f_{\text{stop}} = f_s - f_{\text{pass}} = 12 - 4 = 8 \text{ kHz} \). 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 \( A_{\text{stop}} \) dB. The equality \( y = x \) and the requirement that \( x \geq 50 \text{ dB} \) lead to

\[
y = 15 + A_{\text{stop}} = x \geq 50 \implies A_{\text{stop}} \geq 50 - 15 = 35 \text{ dB}
\]

Thus, the specifications of the prefilter are a fairly flat passband over the \( \pm 4 \text{ kHz} \) 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 \( \alpha \text{ dB/octave} \). We have available an off-the-shelf antialiasing prefilter that has a flat passband up to \( f_{\text{max}} \) and attenuates by \( \beta \text{ dB/octave} \) beyond that. It is required that within the \( f_{\max} \) range of interest, the aliased components due to sampling be suppressed by more than \( A \text{ dB} \). Show that the minimum sampling rate that we should use is given by

\[
f_s = f_{\max} + 2^{\alpha/\beta} f_{\max}
\]

where \( y = \alpha + \beta \).

**Solution:** We refer to the following figure, which shows the 0th and \( \pm 1 \)st replicas.

![Diagram showing sampled signal and reconstructed signal](image)

The passband edge is at \( f_{\text{pass}} = f_{\max} \) and the stopband edge at \( f_{\text{stop}} = f_s - f_{\text{pass}} \). 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

\[
A_X(f) = \alpha \log_2 \left( \frac{f}{f_{\max}} \right) + \beta \log_2 \left( \frac{f}{f_{\max}} \right) = y \log_2 \left( \frac{f}{f_{\max}} \right)
\]

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 \( y \text{ dB} \) 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

\[
A_X(f_s - f_{\max}) \geq A \implies y \log_2 \left( \frac{f_s - f_{\max}}{f_{\max}} \right) \geq A
\]

Solving this as an equality gives the minimum acceptable rate

\[
f_s = f_{\max} + 2^A f_{\max}
\]

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 preilters. 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 reconstructor 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.

![Diagram showing reconstructed signal](image)

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

\[
y_a(t) = \sum_{n=-\infty}^{\infty} y(nT) \delta(t - nT)
\]
1.6. ANALOG RECONSTRUCTORS

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:

\[ y_a(t) = \sum_{n=-\infty}^{\infty} y(nT) h(t - nT) \] (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

\[ Y_a(f) = H(f) \hat{Y}(f) \] (1.6.2)

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

\[ \hat{Y}(f) = \sum_{m=-\infty}^{\infty} Y(f - mf_s) \]

### 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,

\[ \hat{Y}(f) = \frac{1}{T} Y(f) \], \quad \text{for} \quad -\frac{f_s}{2} \leq f \leq \frac{f_s}{2} \] (1.6.3)

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

\[ h(t) = \int_{-\infty}^{\infty} H(f) e^{2\pi jft} \, df = \int_{-f_s/2}^{f_s/2} Te^{2\pi jft} \, df \]

or,

\[ h(t) = \sin(\pi t/T) \quad \text{for} \quad -f_s/2 \leq t \leq f_s/2 \]

(ideal reconstructor) (1.6.5)

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 \( y(nT) \):

\[ y(t) = \sum_{n=-\infty}^{\infty} y(nT) h(t - nT) \] (1.6.4)

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

\[ h(t) = \int_{-\infty}^{\infty} H(f) e^{2\pi jft} \, df = \int_{-f_s/2}^{f_s/2} Te^{2\pi jft} \, df \]

or,

\[ h(t) = \sin(\pi t/T) \quad \text{for} \quad -f_s/2 \leq t \leq f_s/2 \]

(ideal reconstructor) (1.6.5)

It is shown in Fig. 1.6.4. Unfortunately, the ideal reconstructor is not realizable. Its impulse response is non-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.
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:

\[
h(t) = u(t) - u(t - T) = \begin{cases} 
1, & \text{if } 0 \leq t \leq T \\
0, & \text{otherwise}
\end{cases}
\]

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 \( s = 2\pi jf \):

\[
H(f) = \frac{1}{2\pi jf} (1 - e^{-2\pi jfT}) = T \frac{\sin(\pi f T)}{\pi f} e^{-\pi jfT}
\]  

(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.

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 \( T \).
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 pre-filters) 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:

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

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):

$$H_{EQ}(f) = \frac{T}{H(f)} e^{\pi f T}, \quad \text{for} \quad -f_s/2 \leq f \leq f_s/2$$

(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 $Y_{EQ}(f) = H_{EQ}(f) \hat{Y}(f)$. The spectrum of the staircase output of the DAC will be $Y_a(f) = H(f) Y_{EQ}(f)$. Therefore, the final reconstructed spectrum at the output of the postfilter will be,

$$Y_{POST}(f) = H_{POST}(f) Y_a(f)$$

$$= H_{POST}(f) H(f) Y_{EQ}(f)$$

$$= H_{POST}(f) H(f) H_{EQ}(f) \hat{Y}(f)$$

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

$$Y_{POST}(f) = H_{POST}(f) Y_a(f)$$

$$= H_{POST}(f) H(f) Y_{EQ}(f)$$

$$= H_{POST}(f) H(f) H_{EQ}(f) \hat{Y}(f)$$

Outside the Nyquist interval, assuming $H_{POST}(f) \approx 0$, 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:
1.6. ANALOG RECONSTRUCTORS

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 \( y(t) = e^{2\pi j f_0 t} \). 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:

\[
\tilde{Y}(f) = \frac{1}{T} \sum_{m=-\infty}^{\infty} \delta(f - f_0 - mf_s)
\]

The spectrum of the reconstructed signal will be:

\[
Y_a(f) = H(f) \tilde{Y}(f) = \frac{1}{T} \sum_{m=-\infty}^{\infty} H(f) \delta(f - f_0 - mf_s)
\]

Taking inverse Fourier transforms, we obtain:

\[
y_a(t) = \frac{1}{T} \sum_{m=-\infty}^{\infty} H(f_m) e^{2\pi jmf_s t} = \frac{1}{T} H(f_0) e^{2\pi jf_0 t} \cdot T e^{2\pi jf_0 t} = e^{2\pi jf_0 t}
\]

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 \( |f_0 + m_0 f_s| < f_s/2 \), where \( m_0 \) is negative if \( f_0 > 0 \). In this case, only the \( m = m_0 \) term will survive the sum giving:

\[
y_a(t) = \frac{1}{T} H(f_m) e^{2\pi jmf_s t} = \frac{1}{T} H(f_0 + m_0 f_s) e^{2\pi jf_0 t}
\]

where \( f_m = f_0 + m_0 f_s \). 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:

\[
y_a(t) = \frac{1}{T} H(f_0) e^{2\pi jf_0 t} \cdot T e^{2\pi jf_0 t} = e^{2\pi jf_0 t}
\]

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.
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:

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

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 \( H(f)H(f) \). Therefore, Eq. (1.6.8) becomes:

\[
y_a(t) = \frac{1}{T} \sum_{m=-\infty}^{\infty} H(f_m)(f_0) e^{2\pi j f_0 t}
\]

But because of the periodicity of \( H(f_0) \), we can replace \( H(f_m) = H(f_0) = T/H(f_0) \), giving:

\[
y_a(t) = \sum_{m=-\infty}^{\infty} H(f_m)/H(f_0) e^{2\pi j f_0 t}
\]

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

**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:

\[
y_a(t) = \sum_{m=-\infty}^{\infty} G(f_m) \cos(2\pi f_0 m + \phi(f_m))
\]

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

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

Note that \( T G(f) \) and \( \phi(f) \) are not quite the magnitude and phase responses of \( H(f) \); those are \( |H(f)| = TG(f) |f| \) and \( \arg H(f) = \phi(f) + \pi (1 - \text{sign} G(f))/2 \). 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 \(-M \leq m \leq M\):

\[
y_a(t) = \sum_{m=-M}^{M} w(m) G(f_m) \cos(2\pi f_m t + \phi(f_m))
\]

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:

\[
w(m) = 0.54 + 0.46 \cos \left( \frac{\pi M}{M} \right), \quad -M \leq m \leq M
\]

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

For the numerical values \( f_0 = 0.125 \text{ kHz}, f_s = 1 \text{ kHz}, M = 15 \), 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 \( 0 \leq t \leq 16 \text{ msec} \), 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:

\[
y_{post}(t) = G(f_0) \cos(2\pi f_0 t + \phi(f_0))
\]

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

Notice how the postfilter output \( y_{post}(t) \) 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 \( y_{post}(t) \), the next two graphs show the cases \( f_0 = 0.25 \) and \( f_0 = 0.5 \) kHz.

The case \( f_0 = 0.5 \text{ kHz} \) corresponds to the Nyquist frequency \( f_s/2 \), having the maximum amplitude and phase distortion. In all cases, however, \( y_{post}(t) \) 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:

\[
\frac{H(f_m)}{H(f_0)} = \frac{\sin(\pi f_m T)}{\pi f_m T} \frac{\sin(\pi f_0 T)}{\sin(\pi f_0 T)} e^{-j\pi f_m} e^{-j\pi f_0 t}
\]

\[
= \frac{\sin(\pi f_0 T + \pi m M)}{\sin(\pi f_0 T)} \frac{f_0}{f_m} e^{-j\pi m} (-1)^m = \frac{f_0}{f_m}
\]
where we used the property \( \cos(x + \pi m) = (-1)^m \cos x \). Thus,

\[
y_a(t) = \sum_{m=-M}^{M} w(m) \frac{f_c}{f_m} \cos(2\pi f_m t)
\]

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.
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 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 \( H_{\text{PRE}}(f) \) 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 \( H_{\text{DSP}}(f) \), so that \( \hat{Y}(f) = H_{\text{DSP}}(f) \hat{X}(f) \).

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

\[
\hat{X}(f) = \frac{1}{f_s} \sum_{m=-\infty}^{\infty} X(f - mf_s) = \frac{1}{f_s} \left[ X(f) + \text{replicas} \right]
\]

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

\[
Y_a(f) = H_{\text{POST}}(f) Y(f) = H_{\text{POST}}(f) H_{\text{DAC}}(f) \hat{Y}(f) = H_{\text{POST}}(f) H_{\text{DAC}}(f) H_{\text{DSP}}(f) \hat{X}(f) = H_{\text{POST}}(f) H_{\text{DAC}}(f) H_{\text{DSP}}(f) \frac{1}{f_s} \left[ X(f) + \text{replicas} \right] = H_{\text{POST}}(f) H_{\text{DAC}}(f) H_{\text{DSP}}(f) \frac{1}{f_s} \left[ H_{\text{PRE}}(f) X_a(f) + \text{replicas} \right]
\]
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 \(H_{\text{PRE}}(f)\) 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

\[
X_a(f) = H_{\text{DSP}}(f)X_a(f), \quad \text{for} \quad |f| \leq \frac{f_s}{2}
\]

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 \(H_{\text{DSP}}(f)\) 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,

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

The shifted-sinc functions, \(f_s(t) = h(n-nT)\), may be thought of as a basis for 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. 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.

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, that \(x(nT) = x_a(nT)\). Repeat the above questions if the sampling rate is \(f_s = 10\) Hz.

3. The signal \(x(t) = \cos(3\pi t) + 4 \sin(2\pi t)\sin(3\pi t)\), 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.

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.

5. The analog signal \(x(t) = \sin(6\pi t) \left[1 + 2 \cos(4\pi t)\right]\), 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.

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.]

7. Consider the periodic triangular waveform with period \(T_s = 1\) 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 \(x_{\text{rec}}(t)\) that will appear at the output of the reconstructor will have the form:

\[
x_{\text{rec}}(t) = A \sin(2\pi f_1 t) + B \sin(2\pi f_2 t)
\]

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.](image-url)

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 \(0 \leq t \leq 2\) 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 \( f_s = 10 \text{ kHz} \).

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

Repeat this part, but with sampling rate \( f_s = 4 \text{ Hz} \). 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, \( H(f) = 1 \).

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 \( x(t) \) given in Eq. (1.15.1).

Then, prove its Fourier transform expression (1.15.18).

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

\[
\hat{X}(f) = \sum_{n=0}^{\infty} x(nT) e^{-2\pi j f n T} \implies x(nT) = \frac{1}{f_s} \int_{-\infty}^{\infty} \hat{X}(f) e^{2\pi j f n T} df
\]

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:

\[
\hat{X}(f) = \frac{1}{2T} \sum_{n=-\infty}^{\infty} \left[ \delta (f - f_0 - m f_s) + \delta (f + f_0 + m f_s) \right]
\]

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: \( x(t) = e^{-at} e^{j2\pi f_0 t} \). Let \( x(nT) = e^{-at} e^{j2\pi f_0 T n} \) be its samples, and let \( x_a(t) = x(nT) \), \( n = 0, 1, \ldots, L-1 \) be its windowed version to length \( L \). Show that the magnitude spectra of the analog, sampled, and windowed signals are given by:

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

\[
|\hat{X}(f)|^2 = \frac{1}{1 - 2e^{-2\tau} \cos(2\pi (f - f_0) T) + e^{-2\tau}}
\]

\[
|\hat{X}_w(f)|^2 = \frac{1}{1 - 2e^{-2\tau T} \cos(2\pi (f - f_0) T) + e^{-2\tau T}}
\]

Show the limits:

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

For \( f_s = 2 \text{ Hz} \), plot on another graph, the three spectra \( |X(f)|^2 \), \( |\hat{X}(f)|^2 \), and \( |\hat{X}_w(f)|^2 \), over the range \( 0 \leq f \leq 3 \text{ Hz} \).

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 \( \alpha \text{ dB} \) per decade. We have available an off-the-shelf antialiasing prefilter that has a flat passband up to \( f_{max} \) and attenuates by \( \beta \text{ dB} \) 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 \( A \text{ dB} \). Show that the minimum sampling rate that we should use is given by

\[
f_s = f_{max} + 10^{\alpha/(\alpha+\beta)}
\]

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

\[
|X_{in}(f)| = \frac{1}{\sqrt{1 + (f/f_0)^3}}
\]

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

\[
|H(f)| = \frac{1}{\sqrt{1 + (f/f_0)^3}}
\]

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 \( f_s = 70 \text{ kHz} \). 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 \( N \text{th} \) 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 \( A_{pass} = 1 \text{ dB} \) in the passband and the total suppression of the spectral images to be greater than \( A = 60 \text{ dB} \).
1.9 PROBLEMS

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 \( f_{\text{max}} = 20 \) 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 \( 50 \leq f_s \leq 120 \) 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:

\[
|X_a(f)| = \frac{1}{\sqrt{1 + (f/f_a)^{2N_a}}}
\]

where \( f_a \) and \( N_a \) are given. The highest frequency of interest is \( f_{\text{max}} = 2f_a \). 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 \( 0 \leq f \leq f_{\text{max}} \) 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 \( N \)th 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 \( 0 \leq f \leq f_{\text{max}} \) 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

\[
f_s = f_{\text{max}} \left[ 1 + 10^{A/20N(1+N_0)} \right]
\]

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 \( f_0 \) is given and is equal to some value above \( 2f_{\text{max}} \). 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 \( p(t) \).

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:
1.9. PROBLEMS

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

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

1.25 The sum of sinusoids

\[ y(t) = A_1 e^{2\pi j f_1 t} + A_2 e^{2\pi j f_2 t} \]

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-equilizing 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

\[ y_a(t) = \sum_{m=-\infty}^{\infty} 
\left( A_1 m e^{2\pi j f_1 m t} + A_2 m e^{2\pi j f_2 m t} \right) \]

where \( A_{1m} = A_1 f_1 / f_{1m} \), \( A_{2m} = A_2 f_2 / f_{2m} \), and \( f_{1m} = f_1 + m f_s \), \( f_{2m} = f_2 + m f_s \). What would a final postfilter do to each of these terms?

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, \( x(nT)_Q \), 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.

![Analog to digital conversion](image)

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 \( x(nT)_Q \), 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:

\[ Q = \frac{R}{2^B} \tag{2.1.1} \]

This equation can also be written in the form:

\[ \frac{R}{Q} = 2^B \tag{2.1.2} \]
which gives the number of quantization levels.

![Diagram of signal quantization](image)

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 $2^3 = 8$ 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,

$$e(nT) = x_Q(nT) - x(nT) \quad (2.1.3)$$

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

$$-\frac{Q}{2} \leq e \leq \frac{Q}{2} \quad (2.1.4)$$

Therefore, the maximum error is $e_{\text{max}} = Q/2$ 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:

$$\bar{e} = \frac{1}{Q} \int_{-Q/2}^{Q/2} e \, de = 0, \quad \text{and} \quad \bar{e^2} = \frac{1}{Q} \int_{-Q/2}^{Q/2} e^2 \, de = \frac{Q^2}{12} \quad (2.1.5)$$

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:

$$e_{\text{rms}} = \sqrt{\bar{e^2}} = \frac{Q}{\sqrt{12}} \cdot \frac{R}{\sqrt{2}} 2^{-B} \quad (2.1.6)$$

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:

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

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:

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

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:

$$20 \log_{10} \left( \frac{R}{Q} \right) = 20 \log_{10} (2^B) = B \cdot 20 \log_{10} 2, \quad \text{or,} \quad \text{SNR} = 20 \log_{10} \left( \frac{R}{Q} \right) = 6B \text{ dB (6-dB-per-bit-rule)} \quad (2.1.7)$$

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.

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

† 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.
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 $B$, $e_{rms} = Q / \sqrt{2T} = R^{-\delta} / \sqrt{2T}$ and solve for $B$:

$$B = \log_2 \left[ \frac{R}{e_{rms} \sqrt{T}} \right] = \log_2 \left[ \frac{10}{50 \cdot 10^{-5} \sqrt{12}} \right] = 15.82$$

which is rounded to $B = 16$ bits, corresponding to $2^B = 65536$ quantization levels. With this value of $B$, we find $e_{rms} = R^{-\delta} / \sqrt{T} = 44$ microvolts. The bit rate will be $B_{fs} = 16 \cdot 44 = 704$ 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 $e_{rms} = R^{-\delta} / \sqrt{T} = 11$ millivolts. The bit rate in this case is $B_{fs} = 8 \cdot 8 = 64$ 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

$$x_0(n) = x(n) + e(n)$$

we may think of the quantized signal $x_0(n)$ 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:

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

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

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

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

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

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 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 $\sigma_e^2$ 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

$$S_{ee}(f) = \frac{\sigma_e^2}{f_s}$$

for \( \frac{f_s}{2} \leq f \leq \frac{f_s}{2} \)

\(1\)In units of digital frequency \( \omega = 2\pi f/f_s \), it is \( S_{ee}(\omega) = \sigma_e^2/2\pi \).
2.2. OVERSAMPLING AND NOISE SHAPING

2.2. OVERSAMPLING AND NOISE SHAPING

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, they are done together because quantization error will be less accurate, but there will be many more of them and their effect will average out to recover the lost accuracy.

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 \( H_{NS}(f) \). 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:

\[
x_Q(n) = x(n) + \varepsilon(n)
\]

(2.2.2)

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 \( H_{NS}(f) \):

\[
S_{\varepsilon\varepsilon}(f) = |H_{NS}(f)|^2 S_{\varepsilon e}(f) = \frac{\sigma_e^2}{f_s} |H_{NS}(f)|^2
\]

(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:

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

(2.2.4)

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

\[
S_{\varepsilon e}(f) \Delta f = \sigma_e^2 \frac{\Delta f}{f_s} = \sigma_e^2 \frac{f_b - f_a}{f_s}
\]

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

\[
\sigma_e^2 \frac{f_s}{fs} \sigma_e^2 f_s = \sigma_e^2
\]

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 \( H_{NS}(f) \). 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:

\[
x_Q(n) = x(n) + \varepsilon(n)
\]

(2.2.2)

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 \( H_{NS}(f) \):

\[
S_{\varepsilon\varepsilon}(f) = |H_{NS}(f)|^2 S_{\varepsilon e}(f) = \frac{\sigma_e^2}{f_s} |H_{NS}(f)|^2
\]

(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:

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

(2.2.4)

Fig. 2.2.2 Model of noise shaping quantizer.

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 \( H_{NS}(f) \). 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:

\[
x_Q(n) = x(n) + \varepsilon(n)
\]

(2.2.2)

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 \( H_{NS}(f) \):

\[
S_{\varepsilon\varepsilon}(f) = |H_{NS}(f)|^2 S_{\varepsilon e}(f) = \frac{\sigma_e^2}{f_s} |H_{NS}(f)|^2
\]

(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:

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

(2.2.4)

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

\[
S_{\varepsilon e}(f) \Delta f = \sigma_e^2 \frac{\Delta f}{f_s} = \sigma_e^2 \frac{f_b - f_a}{f_s}
\]

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

\[
\sigma_e^2 \frac{f_s}{fs} \sigma_e^2 f_s = \sigma_e^2
\]

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 \( H_{NS}(f) \). 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:

\[
x_Q(n) = x(n) + \varepsilon(n)
\]

(2.2.2)

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 \( H_{NS}(f) \):

\[
S_{\varepsilon\varepsilon}(f) = |H_{NS}(f)|^2 S_{\varepsilon e}(f) = \frac{\sigma_e^2}{f_s} |H_{NS}(f)|^2
\]

(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:

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

(2.2.4)
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:

$$L = \frac{\sigma_e^2}{\sigma_e} = \frac{2^{2(B-B')}}{2^{2\Delta B}}$$

or, equivalently,

$$\Delta B = 0.5 \log_2 L \quad (2.2.6)$$

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.

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 $[-f_s/2, f_s/2]$:

$$\sigma_e^2 = \frac{\sigma_e^2}{f_s} \int_{-f_s/2}^{f_s/2} |H_{NS}(f)|^2 \, df$$

(2.2.7)

Note that it reduces to Eq. (2.2.5) if there is no noise shaping, that is, $H_{NS}(f) = 1$. We will see in Section 14.7 that a typical $p$th order noise shaping filter operating at the high rate $f'_s$ has magnitude response:

$$|H_{NS}(f)|^2 = \left| 2 \sin \left( \frac{\pi f}{f'_s} \right) \right|^{2p}$$

$$\text{for} \quad -\frac{f_s}{2} \leq f \leq \frac{f_s}{2} \quad (2.2.8)$$

For small frequencies $f$, we may use the approximation, $\sin x \approx x$, to obtain:

$$|H_{NS}(f)|^2 = \left( \frac{2\pi f}{f'_s} \right)^{2p}$$

$$\text{for} \quad |f| \ll f'_s/2 \quad (2.2.9)$$

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:

$$\sigma_e^2 = \frac{\sigma_e^2}{f_s} \int_{-f_s/2}^{f_s/2} \left( \frac{2\pi f}{f'_s} \right)^{2p} \, df = \sigma_e^2 \frac{\pi^{2p}}{2p+1} \left( \frac{f_s}{f'_s} \right)^{2p+1} \left( \frac{f'_s}{f_s} \right)^{2p+1}$$

Using $\sigma_e^2/\sigma_e^2 = 2^{2(B-B')} = 2^{-2\Delta B}$, we obtain:

$$2^{-2\Delta B} = \frac{\pi^{2p}}{2p+1} \left( \frac{f'_s}{f_s} \right)^{2p+1}$$

Solving for $\Delta B$, we find the gain in bits:

$$\Delta B = (p + 0.5) \log_2 L - 0.5 \log_2 \left( \frac{\pi^{2p}}{2p+1} \right) \quad (2.2.10)$$

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 $\Delta B$ versus oversampling ratio $L$ for various quantizer orders.

<table>
<thead>
<tr>
<th>$p$</th>
<th>$L$</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\Delta B = 0.5 \log_2 L$</td>
<td>1.0</td>
<td>1.5</td>
<td>2.0</td>
<td>2.5</td>
<td>3.0</td>
<td>3.5</td>
</tr>
<tr>
<td>1</td>
<td>$\Delta B = 1.5 \log_2 L - 0.86$</td>
<td>2.1</td>
<td>3.6</td>
<td>5.1</td>
<td>6.6</td>
<td>8.1</td>
<td>9.6</td>
</tr>
<tr>
<td>2</td>
<td>$\Delta B = 2.5 \log_2 L - 2.14$</td>
<td>2.9</td>
<td>5.4</td>
<td>7.9</td>
<td>10.4</td>
<td>12.9</td>
<td>15.4</td>
</tr>
<tr>
<td>3</td>
<td>$\Delta B = 3.5 \log_2 L - 3.55$</td>
<td>3.5</td>
<td>7.0</td>
<td>10.5</td>
<td>14.0</td>
<td>17.5</td>
<td>21.0</td>
</tr>
<tr>
<td>4</td>
<td>$\Delta B = 4.5 \log_2 L - 5.02$</td>
<td>4.0</td>
<td>8.5</td>
<td>13.0</td>
<td>17.5</td>
<td>22.0</td>
<td>26.5</td>
</tr>
<tr>
<td>5</td>
<td>$\Delta B = 5.5 \log_2 L - 6.53$</td>
<td>4.5</td>
<td>10.0</td>
<td>15.5</td>
<td>21.0</td>
<td>26.5</td>
<td>32.0</td>
</tr>
</tbody>
</table>

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].
2.3. D/A CONVERTERS

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.](image)

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 \( B \)-bit, 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 D/A with full-scale range \( R \), as shown in Fig. 2.3.1. Given \( B \) input bits of zeros and ones, \( b = [b_1, b_2, \ldots, b_B] \), 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.](image)

The manner in which the \( B \) bits \([b_1, b_2, \ldots, b_B] \) 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:

\[
x_Q = R(b_12^{-1} + b_22^{-2} + \cdots + b_B2^{-B})
\]

(2.3.1)

The minimum level is \( x_Q = 0 \) and is reached when all the bits are zero, \( b = [0, 0, \ldots, 0] \). The smallest nonzero level is \( x_Q = Q = R2^{-B} \) and corresponds to the least significant bit (LSB) pattern \( b = [0, 0, \ldots, 0, 1] \). The most significant bit (MSB) pattern is \( b = [1, 0, 0, \ldots, 0] \) and corresponds to the output value \( x_Q = R/2 \). The maximum level is reached when all bits are one, that is, \( b = [1, 1, \ldots, 1] \) 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

\[
2^{-1} + 2^{-2} + \cdots + 2^{-B} = 2^{-1}(1 + 2^{-1} + 2^{-2} + \cdots + 2^{-(B-1)})
= 2^{-1} \left( \frac{1 - 2^{-B}}{1 - 2^{-1}} \right) = 1 - 2^{-B}
\]

Eq. (2.3.1) can be written also in terms of the quantization width \( Q \), as follows:

\[
x_Q = R2^{-B}(b_12^{B-1} + b_22^{B-2} + \cdots + b_{B-1}2^1 + b_B)
\]

or,

\[
x_Q = Qm
\]

(2.3.2)
where \( m \) is the integer whose binary representation is \((b_1 b_2 \cdots b_B)\), that is,
\[
m = b_1 2^{B-1} + b_2 2^{B-2} + \cdots + b_B 2^0
\]

As the integer \( m \) takes on the \( 2^B \) values \( m = 0, 1, 2, \ldots, 2^B - 1 \), 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:
\[
x_Q = R (b_1 2^{-1} + b_2 2^{-2} + \cdots + b_B 2^{-B}) - 0.5
\]

The minimum and maximum attainable levels are obtained by shifting the corresponding natural binary values by \( R/2 \):
\[
x_Q = 0 - \frac{R}{2} = -\frac{R}{2} \quad \text{and} \quad x_Q = (R - Q) - \frac{R}{2} = \frac{R}{2} - Q
\]

The analog value \( x_Q \) can also be expressed in terms of \( Q \), as in Eq. (2.3.2). In this case we have:
\[
\boxed{x_Q = Q m'} \quad (2.3.4)
\]
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
\[
m' = -2^{B-1}, -2, -1, 0, 1, 2, \ldots, 2^{B-1} - 1
\]

One unnatural property of the offset binary code is that the level \( x_Q = 0 \) is represented by the nonzero bit pattern \( b = [1, 0, \ldots, 0] \). 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
\[
x_Q = R (\overline{b_1} 2^{-1} + b_2 2^{-2} + \cdots + b_B 2^{-B}) - 0.5
\]

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 but 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.

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
\[
m_{2c} = \overline{m} + 1 = \overline{0011} + (0001) = (1100) + (0000) = (1101) = -3
\]
which, according to the fifth column of the table, corresponds to the two’s complement

<table>
<thead>
<tr>
<th>Converter type</th>
<th>I/O relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>natural binary</td>
<td>( x_Q = R (b_1 2^{-1} + b_2 2^{-2} + \cdots + b_B 2^{-B}) )</td>
</tr>
<tr>
<td>offset binary</td>
<td>( x_Q = R (b_1 2^{-1} + b_2 2^{-2} + \cdots + b_B 2^{-B} - 0.5) )</td>
</tr>
<tr>
<td>two’s complement</td>
<td>( x_Q = R (\overline{b_1} 2^{-1} + b_2 2^{-2} + \cdots + b_B 2^{-B} - 0.5) )</td>
</tr>
</tbody>
</table>

Table 2.3.1 Converter types.

<table>
<thead>
<tr>
<th>( b_1 ) ( b_2 ) ( b_3 ) ( b_4 )</th>
<th>natural binary</th>
<th>offset binary</th>
<th>2’s C</th>
</tr>
</thead>
<tbody>
<tr>
<td>( m ) ( x_Q = Qm ) ( m' )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1 1 15 9.375 7 4.375 0 1 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1 0 14 8.750 6 3.750 0 1 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0 1 13 8.125 5 3.125 0 1 0 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0 0 12 7.500 4 2.500 0 1 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1 1 11 6.875 3 1.875 0 0 1 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1 0 10 6.250 2 1.250 0 0 1 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0 1 9 5.625 1 0.625 0 0 0 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0 0 8 5.000 0 0.000 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1 1 7 4.375 -1 -0.625 1 1 1 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1 0 6 3.750 -2 -1.250 1 1 1 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 1 5 3.125 -3 -1.875 1 1 0 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 0 4 2.500 -4 -2.500 1 1 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1 1 3 1.875 -5 -3.125 1 0 1 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1 0 2 1.250 -6 -3.750 1 0 1 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 1 1 0.625 -7 -4.375 1 0 0 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 0 0 0.000 -8 -5.000 1 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.3.2 Converter codes for \( B = 4 \) bits, \( R = 10 \) volts.
2.3. D/A CONVERTERS

The following C routine dac.c simulates the operation of the bipolar two’s complement converter. Its inputs are the $B$ bits $[b_1, b_2, \ldots, b_B]$, and the full-scale range $R$, and its output is the analog value $x_Q$ computed by Eq. (2.3.5).

```c
/* dac.c - bipolar two’s complement D/A converter */

double dac(b, B, R);
int *b, B;
double R;
{
    int i;
    double dac = 0;
    b[0] = 1 - b[0];
    for (i = B - 1; i >= 0; i--)
        dac = 0.5 * (dac + b[i]);
    dac = R * (dac - 0.5);
    return dac;
}
```

Its usage is:

```c
int *b;
B = 4;
b = (int *) calloc(B, sizeof(int));
```

The array $b[i]$ can also be allocated dynamically for any desired value of $B$ using `calloc`. The main program must include the lines:

```c
int *b;
B = 4;
b = (int *) calloc(B, sizeof(int));
```

The internal for-loop in dac.c implements a variant of Hörner’s rule for evaluating a polynomial. The result is the binary sum $\sum b_i 2^{-i}$ 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 $[b_1, b_2, \ldots, b_B]$, 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.](image)

The conversion algorithm is as follows. Initially all $B$ bits are cleared to zero, $b = [0, 0, \ldots, 0]$, 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.
2.4. A/D CONVERTERS

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 \( b = [b_1, b_2, \ldots, b_B] \), which can be sent to the output.

At each test, the SAR bit vector \( b \) 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:

\[
\text{for each } x \text{ to be converted, do:} \\
\text{initialize } b = [0, 0, 0, 0] \\
\text{for } i = 1, 2, \ldots, B \text{ do:} \\
\quad b_i = 1 \\
\quad x_Q = \text{dac}(b, B, R) \\
\quad \text{if } (x \geq x_Q) \\
\qquad C = 1 \\
\quad \text{else} \\
\qquad C = 0 \\
\quad b_i = C
\]

Therefore, \( C \) becomes a \textit{serial representation} of the bit vector \( b \). The algorithm imitates the operations shown in Fig. 2.4.2. It can be written more compactly as follows:

\[
\text{for each } x \text{ to be converted, do:} \\
\text{initialize } b = [0, 0, 0, 0] \\
\text{for } i = 1, 2, \ldots, B \text{ do:} \\
\quad b_i = 1 \\
\quad x_Q = \text{dac}(b, B, R) \\
\quad b_i = u(x - x_Q)
\]

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 \( \text{dac} \)). 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 \textit{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 \( C = u(x - x_Q) \).

<table>
<thead>
<tr>
<th>test</th>
<th>( b_1 b_2 b_3 b_4 )</th>
<th>( x_Q )</th>
<th>( C = u(x - x_Q) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b_1 )</td>
<td>1 0 0 0</td>
<td>0.000</td>
<td>1</td>
</tr>
<tr>
<td>( b_2 )</td>
<td>1 1 0 0</td>
<td>2.500</td>
<td>1</td>
</tr>
<tr>
<td>( b_3 )</td>
<td>1 1 0 0</td>
<td>3.750</td>
<td>0</td>
</tr>
<tr>
<td>( b_4 )</td>
<td>1 1 0 1</td>
<td>3.125</td>
<td>1</td>
</tr>
<tr>
<td>( b_5 )</td>
<td>0 1 0 1</td>
<td>1.875</td>
<td>0</td>
</tr>
</tbody>
</table>

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, \( b_1 \) 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

<table>
<thead>
<tr>
<th>test</th>
<th>( b_1 b_2 b_3 b_4 )</th>
<th>( x_Q )</th>
<th>( C = u(x - x_Q) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b_1 )</td>
<td>1 0 0 0</td>
<td>0.000</td>
<td>0</td>
</tr>
<tr>
<td>( b_2 )</td>
<td>0 1 0 0</td>
<td>-2.500</td>
<td>1</td>
</tr>
<tr>
<td>( b_3 )</td>
<td>0 1 1 0</td>
<td>-1.250</td>
<td>0</td>
</tr>
<tr>
<td>( b_4 )</td>
<td>0 1 0 1</td>
<td>-1.875</td>
<td>1</td>
</tr>
<tr>
<td>( b_5 )</td>
<td>0 1 0 0</td>
<td>-1.875</td>
<td>0</td>
</tr>
</tbody>
</table>

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 \textit{rounding} to the nearest level, we must shift \( x \) by half the spacing between levels, that is, use:

\[
y = x + \frac{1}{2}Q
\]

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
2. QUANTIZATION

Example 2.4.3: The two's complement rounded 4-bit representations of \( x = 3.5 \) and \( x = -1.5 \) are:

\[
\begin{align*}
\text{for } x = 3.5 & \Rightarrow x_0 = 3.750 & \Rightarrow b_\text{ = [0,1,1,0]} \\
\text{for } x = -1.5 & \Rightarrow x_0 = -1.250 & \Rightarrow b_\text{ = [1,1,1,0]}
\end{align*}
\]

They are obtained from the offset binary by complementing the MSB. The quantized values \( x_0 \) 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 \( A = 3 \text{ volts} \) and \( f = 0.04 \text{ cycles/sample} \). The sinusoid is evaluated at the ten sampling times \( n = 0, 1, \ldots, 9 \) and \( x(n) \) is quantized using a 4-bit successive approximation ADC with full-scale range \( R = 10 \text{ volts} \). The following table shows the sample and quantized values \( x_0(n) \) and the offset binary and two's complement binary codes representing them.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( x(n) )</th>
<th>( x_0(n) )</th>
<th>2's C</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3.000</td>
<td>3.125</td>
<td>0101</td>
<td>1101</td>
</tr>
<tr>
<td>1</td>
<td>2.906</td>
<td>3.125</td>
<td>0101</td>
<td>1101</td>
</tr>
<tr>
<td>2</td>
<td>2.629</td>
<td>2.500</td>
<td>0100</td>
<td>1100</td>
</tr>
<tr>
<td>3</td>
<td>2.187</td>
<td>1.875</td>
<td>0111</td>
<td>1011</td>
</tr>
<tr>
<td>4</td>
<td>1.607</td>
<td>1.875</td>
<td>0111</td>
<td>1011</td>
</tr>
<tr>
<td>5</td>
<td>0.927</td>
<td>0.625</td>
<td>0001</td>
<td>1001</td>
</tr>
<tr>
<td>6</td>
<td>0.188</td>
<td>0.000</td>
<td>0000</td>
<td>1000</td>
</tr>
<tr>
<td>7</td>
<td>-0.562</td>
<td>-0.625</td>
<td>1111</td>
<td>0111</td>
</tr>
<tr>
<td>8</td>
<td>-1.277</td>
<td>-1.250</td>
<td>1110</td>
<td>0110</td>
</tr>
<tr>
<td>9</td>
<td>-1.912</td>
<td>-1.875</td>
<td>1101</td>
<td>0101</td>
</tr>
</tbody>
</table>

The 2's complement and offset binary codes differ only in their MSB. The quantized values they represent are the same. □

The following routine \texttt{adc.c} simulates the operation of a bipolar two's complement successive approximation ADC. It makes successive calls to \texttt{dac.c} to determine each bit.

```c
/* 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);
    y = x + Q/2;
    for (i = 0; i < B; i++)
        b[i] = 0;
    b[0] = 1 - u(y);
```
2.4. A/D CONVERTERS

Example 2.4.6: This example illustrates the statistical properties of the quantization error. Consider $L$ samples of the noisy sinusoidal signal:

$$x(n) = A \cos(2\pi f_0 n + \phi) + v(n), \quad n = 0, 1, \ldots, L - 1$$

where $\phi$ is a random phase distributed uniformly in the interval $[0, 2\pi]$ and $v(n)$ is white noise of variance $\sigma_v^2$. 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:

$$e(n) = x_Q(n) - x(n), \quad n = 0, 1, \ldots, L - 1$$

According to the standard statistical model discussed in Section 2.1, the quantization noise samples $e(n)$ should be distributed uniformly over the interval $-Q/2 \leq e \leq Q/2$. 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:

$$\sigma_e^2 = \frac{1}{L} \sum_{n=0}^{L-1} e^2(n)$$

$$R_{ae}(k) = \frac{1}{L} \sum_{n=0}^{L-1-k} e(n) e(n+k)$$

We can also compute the autocorrelation of $x(n)$ itself:

$$R_{xx}(k) = \frac{1}{L} \sum_{n=0}^{L-1-k} x(n) x(n+k)$$

where in the last three equations, $k$ ranges over a few lags $0 \leq k \leq M$, with $M$ typically being much less than $L - 1$. Note also that $\sigma_e^2 = R_{ae}(0)$. 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
- $f_0 = 1/1000 \approx 0.001$ 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:

$$v = 0.5R(u - 0.5)$$

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:

```c
for (i=1; i<B; i++) {
    loop starts with i = 1
    turn ith bit ON
    compute DAC output
    test and correct bit
    }
```
for \((n=0; n<L; n++)\)
\[v[n] = 0.5 * R * (\text{ran}(&\text{iseed}) - 0.5);\]

where the initial seed\(^1\) 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 \(\sigma_e = Q/\sqrt{T^2} = 1/\sqrt{T^2} = 0.289\). 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 \([-Q/2, Q/2]\) 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.

\[\begin{align*}
\text{Histogram of } e
\end{align*}\]

The next two figures show the standard normalized correlation functions:

\[\rho_{ee}(k) = \frac{R_{ee}(k)}{R_{ee}(0)} \quad \rho_{ea}(k) = \frac{R_{ea}(k)}{\sqrt{R_{ee}(0)R_{ax}(0)}} \quad \rho_{ax}(k) = \frac{R_{ax}(k)}{R_{ax}(0)}\]

computed for lags \(k = 0, 1, \ldots, M\) using Eqs. (2.4.3–2.4.5).

\[\begin{align*}
\text{Normalized Correlations} \\
\text{\hspace{1cm}} \rho_{aa}(k) \\
\text{\hspace{1cm}} \rho_{ee}(k) \\
\text{Anticorrelation} \\
\text{\hspace{1cm}} \rho_{ax}(k)
\end{align*}\]

Theoretically, \(\rho_{ee}(k)\) should be \(\delta(k)\) and \(\rho_{ea}(k)\) should be zero. Using the results of Problem 2.20, the theoretical expression for \(\rho_{ax}(k)\) will be, for the particular numerical values of the parameters:

\[\rho_{ax}(k) = 0.6 \cos(2\pi f_0 k) + 0.4 \delta(k)\]

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.

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 \(B \leq 12\) 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.

**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

\[\begin{align*}
\text{Fig. 2.5.1 Analog dither.}
\end{align*}\]
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.

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:

\[
p(v) = \frac{1}{\sqrt{2\pi\sigma_v^2}} e^{-v^2/2\sigma_v^2}
\]

(2.5.4)

with the recommended value for the variance:

\[
\sigma_v^2 = \frac{1}{4} Q^2
\]

(2.5.5)

which corresponds to the rms value \( v_{\text{rms}} = Q/2 \), or half-LSB. It follows from Eq. (2.5.3) that the total error variance will be:

\[
\sigma_e^2 = \frac{1}{12} Q^2 + \frac{1}{4} Q^2 = 4 \cdot \frac{1}{12} Q^2 = \frac{1}{3} Q^2
\]

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:

\[-\frac{1}{2} Q \leq v \leq \frac{1}{2} Q\]

The corresponding pdf and variance are in this case:

\[
p(v) = \begin{cases} 
\frac{1}{Q}, & \text{if } -\frac{1}{2} Q \leq v \leq \frac{1}{2} Q \\
0, & \text{otherwise}
\end{cases}
\] and \( \sigma_v^2 = \frac{1}{12} Q^2 \) (2.5.6)

Therefore, the total error variance will be:

\[
\sigma_e^2 = \frac{1}{12} Q^2 + \frac{1}{2} Q^2 = 2 \cdot \frac{1}{12} Q^2 = \frac{1}{6} Q^2
\]

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:

\[
p(v) = \begin{cases} 
\frac{Q - |v|}{Q^2}, & \text{if } -Q \leq v \leq Q \\
0, & \text{otherwise}
\end{cases}
\] and \( \sigma_v^2 = \frac{1}{6} Q^2 \) (2.5.7)

and, the total error variance will be:

\[
\sigma_e^2 = \frac{1}{12} Q^2 + \frac{1}{6} Q^2 = 3 \cdot \frac{1}{12} Q^2 = \frac{1}{4} Q^2
\]

In summary, the total error variance in the three cases and the undithered case \( (v = 0) \) will be:

\[
\sigma_e^2 = \begin{cases} 
Q^2/12, & \text{undithered} \\
2Q^2/12, & \text{rectangular dither} \\
3Q^2/12, & \text{triangular dither} \\
4Q^2/12, & \text{Gaussian dither}
\end{cases}
\] (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 \texttt{ran}. For example,

\[
v = Q(u - 0.5)
\]

where \( u \) is the random number returned by \texttt{ran}, that is, uniformly distributed over the interval \([0, 1)\). The shifting and scaling of \( u \) imply that \( v \) will be uniformly distributed within \([-Q/2 \leq v < Q/2] \).

Triangular dither can be generated just as simply by noting that the triangular pdf is the convolution of two rectangular ones and therefore can be obtained as the sum of two independent rectangular random numbers, that is,

\[
v = v_1 + v_2
\]

where \( v_1, v_2 \) are generated from two independent uniform \( u_1, u_2 \), by

\[
v_1 = Q(u_1 - 0.5) \quad \text{and} \quad v_2 = Q(u_2 - 0.5)
\]

**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 \( f_0 = 0.025 \text{ cycles/sample} \), which corresponds to \( 1/f_0 = 40 \text{ samples per cycle} \). At an audio rate of \( f_s = 40 \text{ kHz} \), 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 \( R = 8 \text{ volts} \). 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 \( y_0(n) \) were generated by the following loop:

```c
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 \( v_1 \) and \( v_2 \) are independent of each other because each call to \texttt{ran} updates the seed to a new value.

The following graphs show the undithered sinusoid \( x(n) \) and its quantized version \( x_0(n) \), together with its Fourier spectrum \( |X_0(f)| \) plotted over the right half of the Nyquist interval, that is, \( 0 \leq f \leq 0.5 \), in units of cycles/sample.

![Undithered Spectrum](image)

The spectrum of \( x_0(n) \) 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 \( y_0(n) \), together with its spectrum \( |Y_0(f)| \).

![Dithered Spectrum](image)

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 \( x_0(n) \) and \( y_0(n) \) were generated for \( 0 \leq n \leq N_{\text{tot}} - 1 \), with \( N_{\text{tot}} = 1000 \). Then, they were windowed using
2.5. ANALOG AND DIGITAL DITHER

a length-$N_{\text{tot}}$ Hamming window, that is,

$$y_Q(n) = w(n)y_1(n), \quad n = 0, 1, \ldots, N_{\text{tot}} - 1$$

where,

$$w(n) = 0.54 - 0.46 \cos \left( \frac{2\pi n}{N_{\text{tot}} - 1} \right), \quad n = 0, 1, \ldots, N_{\text{tot}} - 1$$

And, their DTFT

$$Y_Q(f) = \sum_{n=0}^{N_{\text{tot}}-1} y_Q(n)e^{-2\pi fn}$$

was evaluated at 200 equally spaced frequencies $f$ over the interval $0 \leq f \leq 0.5$ [cycles/sample], that is, $f_t = 0.5i/200, i = 0, 1, \ldots, 199$.

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:

$$f = 0.01, 0.05, 0.07, 0.13, 0.17, 0.19, 0.23, 0.25, \ldots$$

which are the aliased versions of the following out-of-band odd harmonics:

$$33f_0, 35f_0, 31f_0, 29f_0, 39f_0, 27f_0, 41f_0, 25f_0, \ldots$$

The beneficial effect of dithering works, of course, for any value of $f_0$. □
2.6. PROBLEMS

2.2 Consider the signal \( x(n) = 5 \sin(2\pi f n/1000) \), 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 \( n = 0, 1, \ldots, 19 \), 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 \(-Q < e \leq 0\). Assume a uniform probability density \( p(e) \) over this interval, that is,

\[
p(e) = \begin{cases} \frac{1}{Q} & \text{if } -Q < e \leq 0 \\ 0 & \text{otherwise} \end{cases}
\]

Determine the mean \( m_e \) and variance \( \sigma_e^2 \) 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 \( f_s \) 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 \( f_s \) in Hz.

2.9 Show that the two's complement expression defined in Eq. (2.3.5) can be written in the alternative form:

\[ x_Q = R \left( -b_1 2^{-1} + b_2 2^{-2} + \cdots + b_N 2^{-N} \right) \]

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:

\[
\begin{align*}
\text{initialize } & p = 0 \\
\text{for } i = M, M-1, \ldots, 0 \text{ do:} & \\
& p = ap + b_i \\
\text{end do} \\
\text{end for}
\end{align*}
\]

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 \( \text{pol.c} \) 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:

\[
\text{double \ pol} (M, \ b, \ a) \\
\text{int \ M; \ order \ of \ polynomial} \\
\text{double *b, \ a; \ b \ is \ (M+1)-dimensional}
\]

2.12 Consider the following variation of Hörner’s algorithm:

\[
\begin{align*}
\text{initialize } & q_{M-1} = b_M \\
\text{for } i = M-1, M-2, \ldots, 1 \text{ do:} & \\
& q_{i-1} = a q_i + b_i \\
& p = a q_0 + b_0 \\
\text{end for}
\end{align*}
\]

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 \( \{q_0, q_1, \ldots, q_{M-1}\} \) 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 \( \text{dac} \) 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 \( \text{dac} \) 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:

\[
\begin{align*}
\text{initialize } & p = 0 \\
\text{for } i = M, M-1, \ldots, 1 \text{ do:} & \\
& p = a (p + b_i) \\
\text{end for}
\end{align*}
\]

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.
   b. 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.
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:

![Diagram of two ADCs](image)

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 $\text{adc}$ and $\text{dac}$ as building blocks, write a routine that implements this block diagram. Test your routine on the signal:

$$x(n) = A \cos(2\pi f_0 n), \quad n = 0, 1, \ldots, L - 1$$

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 $B = B_1 + B_2 + B_3$?
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/68K, 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:

\[
\{x_0, x_1, x_2, \ldots, x_n, \ldots\} \xrightarrow{H} \{y_0, y_1, y_2, \ldots, y_n, \ldots\}
\]

that is, \( x_0 \rightarrow y_0, x_1 \rightarrow y_1, x_2 \rightarrow y_2, \) 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:

\[
x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \end{bmatrix} \xrightarrow{H} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \end{bmatrix} = y
\]

Thus, the I/O rule maps the input vector \( x \) into the output vector \( y \) according to some functional mapping:

\[
y = H[x]
\]  

(3.1.1)

For linear systems, this mapping becomes a linear transformation by a matrix \( H \), \( y = Hx \). 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.](image)

**Example 3.1.1:** \( y(n) = 2x(n) \). It corresponds to simple scaling of the input:

\[
\{x_0, x_1, x_2, x_3, x_4, \ldots\} \xrightarrow{H} \{2x_0, 2x_1, 2x_2, 2x_3, 2x_4, \ldots\}
\]

**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.

\[\text{For brevity, we denoted } \{x(0), x(1), x(2), \ldots\} \text{ by subscripts } \{x_0, x_1, x_2, \ldots\}.\]
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:

\[
\begin{bmatrix}
y_0 \\
y_1 \\
y_2 \\
y_3 \\
y_4 \\
y_5 \\
\end{bmatrix}
= 
\begin{bmatrix}
2 & 0 & 0 & 0 \\
3 & 2 & 0 & 0 \\
4 & 3 & 2 & 0 \\
0 & 4 & 3 & 2 \\
0 & 0 & 4 & 3 \\
0 & 0 & 0 & 4 \\
\end{bmatrix}
\begin{bmatrix}
x_0 \\
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5 \\
\end{bmatrix}
= H x
\]

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:

\[
\begin{bmatrix}
y_0 \\
y_1 \\
y_2 \\
y_3 \\
y_4 \\
y_5 \\
y_6 \\
\end{bmatrix}
= 
\begin{bmatrix}
2 & 0 & 0 & 0 & 0 \\
3 & 2 & 0 & 0 & 0 \\
4 & 3 & 2 & 0 & 0 \\
0 & 4 & 3 & 2 & 0 \\
0 & 0 & 4 & 3 & 2 \\
0 & 0 & 0 & 4 & 0 \\
\end{bmatrix}
\begin{bmatrix}
x_0 \\
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5 \\
x_6 \\
\end{bmatrix}
= H x
\]

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:

\[
y(n) = 2x(n) + 3w_1(n) + 4w_2(n)
\]

\[
w_2(n+1) = w_1(n)
\]

\[
w_1(n+1) = x(n)
\]

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 \(n\)th time step; thus, at time \(n+1\) we have:

\[
y(n+1) = 2x(n+1) + 3w_1(n+1) + 4w_2(n+1)
\]

\[
w_2(n+2) = w_1(n+1)
\]

\[
w_1(n+2) = x(n+1)
\]

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:

\[
\begin{align*}
\text{for each new input sample } x \text{ do: } \\
y := 2x + 3w_1 + 4w_2 \\
w_2 := w_1 \\
w_1 := x
\end{align*}
\]

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:

\[
y(n) = g(x(n), s(n)) \quad \text{(output equation)}
\]

\[
s(n+1) = f(x(n), s(n)) \quad \text{(state updating equation)}
\]

where \(s(n)\) is an internal state vector of appropriate dimension, like \(s(n) = \begin{bmatrix} w_1(n) \\ w_2(n) \end{bmatrix}\) of the previous example. The I/O algorithm calculates both the output \(y(n)\) and the next state \(s(n+1)\) from the knowledge of the present input \(x(n)\) and the present state \(s(n)\). It can be rephrased in the repetitive algorithmic form:

\[
\text{for each new input sample } x \text{ do: } \\
y := g(x, s) \\
s := f(x, s)
\]

State-space realizations of LTI systems are described by functions \(f\) and \(g\) that are linear functions of their arguments, that is, \(f(x,s) = As + Bx, g(x,s) = Cs + Dx\), where \(A, B, C, D\) have appropriate dimensions. In particular, for the above example we have:

\[
y := 2x + 3w_1 + 4w_2 = [3, 4] \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + 2x = [3, 4]s + 2x = g(x,s)
\]

\[
s = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} := \begin{bmatrix} x \\ w_1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} x = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ x \end{bmatrix} \equiv f(x,s)
\]

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 \(y(n-1)\) and output sample \(x(n-1)\).

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:

\[
\text{for each new input sample } x \text{ do: } \\
y := 0.5w_1 + 2x + 3v_1 \\
w_1 := y \\
v_1 := x
\]

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.
3.1. INPUT/OUTPUT RULES

An alternative I/O computational rule for Example 3.1.5, corresponding to the so-called canonical realization of the system, is as follows:

\[
\text{for each new input sample } x \text{ do:} \\
w_0 := x + 0.5 w_1 \\
y' := 2w_0 + 3w_1 \\
w_1 := w_0
\]

It uses the auxiliary quantities \( \{w_0, w_1\} \).

Example 3.1.7: \( y(n) = \frac{1}{5} \{ x(n+2) + x(n+1) + x(n) + x(n-1) + x(n-2) \} \). 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: \( y(n) = \frac{1}{n} \{ x(0) + x(1) + \cdots + x(n-1) \} \). 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:

\[
\{x_0, x_1, x_2, x_3, x_4, x_5, x_6, \ldots \} \overset{H}{\rightarrow} \{x_0, x_2, x_4, x_6, \ldots \}
\]

Example 3.1.16: \( y(n) = \begin{cases} 
\frac{x(n/2)}{2}, & \text{if } n \text{ is even} \\
0, & \text{if } n \text{ is odd}
\end{cases} \). 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:

\[
\{x_0, x_1, x_2, x_3, x_4, \ldots \} \overset{H}{\rightarrow} \{x_0, 0, x_1, 0, x_2, 0, x_3, 0, \ldots \}
\]

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

\[
x(n) = a_1 x_1(n) + a_2 x_2(n)
\]

is given by the linear combination of outputs

\[
y(n) = a_1 y_1(n) + a_2 y_2(n)
\]

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.

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

$$a_1x_1^2(n) + a_2x_2^2(n) \neq (a_1x_1(n) + a_2x_2(n))^2$$

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.

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 $n - D$:

$$y(n - D) = (n - D)x(n - D) = nx(n - D) = y_D(n)$$

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

$$y(n - D) = x(2(n - D)) = x(2n - 2D) \neq x(2n - D) = y_D(n)$$

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:

$$\{x_0, x_1, x_2, x_3, x_4, x_5, x_6, \ldots\} \overset{\text{H}}{\rightarrow} \{x_0, x_2, x_4, x_6, \ldots\}$$

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,

$$\delta(n) \overset{\text{H}}{\rightarrow} h(n)$$
3.3. IMPULSE RESPONSE

![Impulse response of an LTI system.](image)

Fig. 3.3.1 Impulse response of an LTI system.

or, in terms of the sample values:

\[ \{1, 0, 0, 0, \ldots\} \xrightarrow{H} \{h_0, h_1, h_2, h_3, \ldots\} \]

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,

\[ \delta(n) + \delta(n - 1) + \delta(n - 2) \xrightarrow{H} h(n) + h(n - 1) + h(n - 2) \]

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:

\[ x(0) h(n) + x(1) h(n - 1) + x(2) h(n - 2) \]

as shown in Fig. 3.3.3. In general, an arbitrary input sequence \( \{x(0), x(1), x(2), \ldots\} \) 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,

\[ y(n) = x(0) h(n) + x(1) h(n - 1) + x(2) h(n - 2) + x(3) h(n - 3) + \cdots \] (3.3.1)

or written more compactly:

\[ y(n) = \sum_m x(m) h(n - m) \]

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:

\[ y(n) = \sum_m h(m) x(n - m) \]

(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.
3.4. FIR AND IIR FILTERS

An FIR filter has impulse response \( h(n) \) that extends only over a finite time interval, say \( 0 \leq n \leq M \), and is identically zero beyond that:

\[
\{h_0, h_1, h_2, \ldots, h_M, 0, 0, 0, \ldots\}
\]

\( M \) is referred to as the filter order. The length of the impulse response vector \( h = [h_0, h_1, \ldots, h_M] \) is:

\[
L_h = M + 1
\]

![FIR and IIR impulse responses.](image)

The impulse response coefficients \( \{h_0, h_1, \ldots, h_M\} \) 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 \( 0 \leq m \leq M \) are present. Therefore, Eq. (3.3.3) is simplified to the finite-sum form:

\[
y(n) = \sum_{m=0}^{M} h(m)x(n-m) \quad \text{(FIR filtering equation)} \tag{3.4.1}
\]

or, explicitly

\[
y(n) = h_0 x(n) + h_1 x(n-1) + h_2 x(n-2) + \cdots + h_M x(n-M)
\]  

Thus, the I/O equation is obtained as a weighted sum of the present input sample \( x(n) \) and the past \( M \) samples \( x(n-1), x(n-2), \ldots, x(n-M) \).

**Example 3.4.1:** Second-order FIR filters are characterized by three impulse response coefficients \( h = [h_0, h_1, h_2] \) 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 \( h = [2, 3, 4] \).

**Example 3.4.2:** Similarly, third-order FIR filters are characterized by four weights \( h = [h_0, h_1, h_2, h_3] \) 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 \( h \) 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) \( h = [h_0, h_1, h_2, h_3] = [2, 3, 5, 2] \)

(b) \( h = [h_0, h_1, h_2, h_3, h_4] = [1, 0, 0, 0, -1] \)

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 \( h \) are equivalent. \( \square \)

An IIR filter, on the other hand, has an impulse response \( h(n) \) of infinite duration, defined over the infinite interval \( 0 \leq n < \infty \). Eq. (3.3.3) now has an infinite number of terms:

\[
y(n) = \sum_{m=0}^{\infty} h(m)x(n-m)
\]

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 \( \{h_0, h_1, h_2, \ldots\} \) 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:

\[
h(n) = u(n) = \begin{cases} 
1 & \text{if } n \geq 0 \\
0 & \text{if } n \leq 0 
\end{cases}
\]

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

\[
y(n) = \sum_{m=0}^{n} h(m)x(n-m) = \sum_{m=0}^{n} x(n-m)
\]
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 \( x(n) = \delta(n) \).

\[\square\]

**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

\[
\begin{align*}
h(0) &= ah(-1) + \delta(0) = a \cdot 0 + 1 = 1 \\
h(1) &= ah(0) + \delta(1) = a \cdot 1 + 0 = a \\
h(2) &= ah(1) + \delta(2) = a \cdot a + 0 = a^2 \\
h(3) &= ah(2) + \delta(3) = a \cdot a^2 + 0 = a^3
\end{align*}
\]

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

\[
y(n) = x(n) + ax(n - 1) + a^2x(n - 2) + a^3x(n - 3) + \cdots
\]

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 \( h(n) \).

\[\square\]

**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.

\[\square\]

**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

\[
h(n) = \begin{cases} 2, & \text{if } n = 0 \\ 4(0.5)^{n-1}, & \text{if } n \geq 1 \end{cases}
\]

**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

\[
y_n = h_0 x_n + h_1 x_{n-1} + h_2 x_{n-2} + h_3 x_{n-3} + h_4 x_{n-4} + \cdots
\]

and for the previous output

\[
y_{n-1} = 2x_n + 4x_{n-1} + 2[x_{n-2} + 0.5x_{n-3} + 0.5^2x_{n-4} + \cdots]
\]

Multiplying by 0.5, we have

\[
0.5y_{n-1} = x_{n-1} + 2[x_{n-2} + 0.5x_{n-3} + 0.5^2x_{n-4} + \cdots]
\]

Subtracting it from \( y_n \), we find the I/O difference equation

\[
y_n - 0.5y_{n-1} = 2x_n + 3x_{n-1}
\]
3.4. FIR AND IIR FILTERS

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. \( \square \)

**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:

\[
\begin{align*}
h(0) &= 0.25h(-2) + \delta(0) = 0.25 \cdot 0 + 1 = 1 \\
h(1) &= 0.25h(-1) + \delta(1) = 0.25 \cdot 0 + 0 = 0 \\
h(2) &= 0.25h(0) + \delta(2) = 0.25 \cdot 1 + 0 = 0.25 = (0.5)^2 \\
h(3) &= 0.25h(1) + \delta(3) = 0.25 \cdot 0 + 0 = 0 \\
h(4) &= 0.25h(2) + \delta(4) = 0.25 \cdot 0.25 + 0 = (0.25)^2 = (0.5)^4
\end{align*}
\]

And, in general, for \( n \geq 0 \)

\[
h(n) = \begin{cases} 
(0.5)^n, & \text{if } n = \text{even} \\
0, & \text{if } n = \text{odd}
\end{cases}
\]

Equivalently, we can write:

\[
h = \{ 1, 0, (0.5)^2, 0, (0.5)^4, 0, (0.5)^6, 0, (0.5)^8, 0, \ldots \}
\]

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 \( x(n) \). \( \square \)

**Example 3.4.9:** Determine the I/O difference equation of the IIR filter that has the following causal periodic impulse response:

\[
h(n) = \{ 2, 3, 4, 5, 2, 3, 4, 5, 2, 3, 4, 5, \ldots \}
\]

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

\[
h(n-4) = \{ 0, 0, 0, 0, 2, 3, 4, 5, 2, 3, 4, 5, \ldots \}
\]

Subtracting it from \( h(n) \), we get

\[
h(n) - h(n-4) = \{ 2, 3, 4, 5, 0, 0, 0, 0, 0, 0, 0, 0, \ldots \}
\]

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. \( \square \)

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:

\[
h(n) = \sum_{i=1}^{M} a_i h(n-i) + \sum_{i=0}^{L} b_i \delta(n-i)
\]

or, written explicitly

\[
h_n = a_1 h_{n-1} + a_2 h_{n-2} + \cdots + a_M h_{n-M} + b_0 \delta_n + b_1 \delta_{n-1} + \cdots + b_L \delta_{n-L}
\]

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,

\[
y(n) = \sum_{i=1}^{M} a_i y(n-i) + \sum_{i=0}^{L} b_i x(n-i)
\]
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.

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

\[ y(n) = \sum_{m=-\infty}^{\infty} h(m)x(n-m) \] (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 \( x(n+1), x(n+2), \ldots \), 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:

\[ y_n = \cdots + h_{-2} x_{n+2} + h_{-1} x_{n+1} + h_0 x_n + h_1 x_{n-1} + h_2 x_{n-2} + \cdots \]

which shows that to compute the output \( y(n) \) at the current time \( n \), one needs to know the future input samples \( x(n+1), x(n+2), \ldots \), 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:

\[ y_n = \cdots + h_{-2} x_{n+2} + h_{-1} x_{n+1} + h_0 x_n + h_1 x_{n-1} + h_2 x_{n-2} + \cdots \]

which shows that to compute the output \( y(n) \) at the current time \( n \), one needs to know the future input samples \( x(n+1), x(n+2), \ldots \), which are not yet available for processing.
3.5. CAUSALITY AND STABILITY

\[ y(n) = \sum_{m=-\infty}^{\infty} h(m)x(n-m) \]  
(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

\[ y_D(n) = \sum_{m=-\infty}^{\infty} h_D(m)x(n-m) \]  
(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 \(-2 \leq n \leq 2\). The corresponding I/O convolutional equation (3.5.2) becomes

\[ y(n) = \sum_{m=-2}^{2} h(m)x(n-m) = \frac{1}{5} \sum_{m=-2}^{2} x(n-m) \]

\[ = \frac{1}{5} [x(n+2)+x(n+1)+x(n)+x(n-1)+x(n-2)] \]

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

\[ y_2(n) = y(n-2) = \frac{1}{5} [x(n+2)+x(n+1)+x(n-2)+x(n-3)+x(n-4)] \]

This filtering equation must be thought of as smoothing the middle sample \( x(n-2) \) and not the current sample \( x(n) \). \( \square \)

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 \(|y(n)| \leq B\) if its input is bounded, say \(|x(n)| \leq A\). 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**:

\[ \sum_{n=-\infty}^{\infty} |h(n)| < \infty \]  
(3.5.4)

**Example 3.5.2:** Consider the following four examples of \( h(n) \):

\[ h(n) = (0.5)^n u(n) \]  
(stable and causal)

\[ h(n) = -(0.5)^n u(-n-1) \]  
(unstable and anticausal)

\[ h(n) = 2^n u(n) \]  
( unstable and causal)

\[ h(n) = -2^n u(-n-1) \]  
(stable and anticausal)

In the two causal cases, the presence of the unit step \( u(n) \) causes \( h(n) \) to be nonzero only for \( n \geq 0 \), whereas in the anticausal cases, the presence of \( u(-n-1) \) makes \( h(n) \) nonzero only for \(-n-1 \geq 0 \) or \( n+1 \leq 0 \) or \( n \leq -1 \). 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:

\[ \sum_{n=-\infty}^{\infty} |h(n)| = \sum_{n=0}^{\infty} (0.5)^n = \frac{1}{1-0.5} < \infty \]

\[ \sum_{n=-\infty}^{\infty} |h(n)| = \sum_{n=-1}^{\infty} (0.5)^n = \sum_{m=1}^{\infty} 2^m = \infty \]

\[ \sum_{n=-\infty}^{\infty} |h(n)| = \sum_{n=0}^{\infty} 2^n = \infty \]

\[ \sum_{n=-\infty}^{\infty} |h(n)| = \sum_{n=-1}^{\infty} 2^n = \sum_{m=1}^{\infty} (0.5)^m = \frac{0.5}{1-0.5} < \infty \]

where in the first and fourth cases, we used the infinite geometric series formulas:

\[ \sum_{m=0}^{\infty} x^m = \frac{1}{1-x} \quad \text{and} \quad \sum_{m=1}^{\infty} x^m = \frac{x}{1-x} \]
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, $H(z) = \frac{1}{1-0.5z^{-1}}$, and therefore, cannot be distinguished on the basis of just the transfer function. Similarly, cases three and four have the common transfer function $H(z) = \frac{1}{1-2z^{-1}}$. □

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

$$
H(z) = \frac{1}{H(z)}
$$

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.\(^1\) 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. $y(n) = 2^{x(n)}$

d. $y(n) = n x(n-3) + 3x(n)$

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. $y(n) = 3x(n-2) + 2x(n-1) + 4x(n-3)$

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. $h(n) = (0.9j)^n u(n) + (-0.9j)^n u(n)$

\(^1\)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.
3.6 PROBLEMS

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:

\[
h(n) = \begin{cases} 
5, & \text{if } n = 0 \\
6(0.8)^{n-1}, & \text{if } n \geq 1 
\end{cases}
\]

Working with the convolutional filtering equation, derive the difference equation satisfied by \( y(n) \).

3.7 To understand the role played by the first two values \( h(0) \) and \( h(1) \), redo Problem 3.6 starting with the more general expression for \( h(n) \):

\[
h(n) = \begin{cases} 
c_0, & \text{for } n = 0 \\
c_1a^{n-1}, & \text{for } n \geq 1 
\end{cases}
\]

which has \( h(0) = c_0 \) and \( h(1) = c_1 \). First, determine the difference equation satisfied by \( h(n) \) for all \( n \geq 0 \). 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:

\[
h_n + a_1h_{n-1} + a_2h_{n-2} + \cdots + a_Mh_{n-M} = 0, \quad \text{for } n \geq M
\]

where \( \{a_1, a_2, \ldots, a_M\} \) are the coefficients of the polynomial whose roots are the (complex) numbers \( \{p_1, p_2, \ldots, p_M\} \), 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 \( n \geq M \) necessary.

3.9 A causal linear time-invariant filter has impulse response:

\[
h_n = C_0\delta(n) + C_1p_1^n + C_2p_2^n + \cdots + C_Mp_M^n, \quad n \geq 0
\]

Show that it satisfies the same difference equation as in the previous problem, but with the restriction \( n \geq M + 1 \).

3.10 A causal linear time-invariant filter has impulse response:

\[
h_n = C_1p_1^n + C_2p_2^n, \quad n \geq 0
\]

Working in the time domain, show that the difference equation satisfied by \( h_n \) for all \( n \geq 0 \) 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:

\[
h_n = C_0\delta(n) + C_1p_1^n + C_2p_2^n, \quad n \geq 0
\]

Show that the difference equation satisfied by \( h_n \) for all \( n \geq 0 \) 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:

\[
h_n = C_1p_1^n + C_2p_2^n + C_Mp_M^n, \quad n \geq 0
\]

Working in the time domain, show that the difference equation satisfied by \( h_n \) for all \( n \geq 0 \) 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 \( \{a_1, a_2, a_3, b_0, b_1, b_2\} \) 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. \( h(n) = (0.5)^nu(n) + 4(0.8)^nu(n) \)

b. \( h(n) = (0.5)^nu(n) + (-0.5)^nu(n) \)

c. \( h(n) = [3(0.4)^n + 4(0.5)^n - 7(-0.5)^n]u(n) \)

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, \( |x(n)| \leq B \), then the output is bounded by \( |y(n)| \leq AB \).

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 \( x(n) = \text{sign}(h(-n)) \) 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 \( y_D(n) = \text{delayed version of} \ y(n) \), 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:

\[
Y(z) = H(z)X(z) = [z^{-D}H(z)][z^D X(z)]
\]

3.18 Prove the inequality (3.5.6). Is the right-hand side finite? Does it get smaller as \( D \) gets larger?

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:

\[
T_L = LT
\]

(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:

\[
L = T_L f_s
\]

(4.1.2)

The \( L \) collected signal samples, say \( x(n) \), \( n = 0, 1, \ldots, L - 1 \), can be thought of as a block:

\[
x = [x_0, x_1, \ldots, x_{L-1}]
\]

(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)

\[
y(n) = \sum_m h(m)x(n - m) = \sum_m x(m)h(n - m)
\]

(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:

\[
y(n) = \sum_{i+j=n} h(i)x(j) \quad \text{(convolution table form)}
\]

(4.1.5)

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 \( h(n) = 0, 1, \ldots, M \). It may be represented as a block:

\[
h = [h_0, h_1, \ldots, h_M]
\]

(4.1.6)

Its length (i.e., the number of filter coefficients) is one more than its order:

\[
L_h = M + 1
\]

(4.1.7)

The convolution of the length-\( L \) input \( x \) of Eq. (4.1.3) with the order-\( M \) filter \( h \) 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:

\[
0 \leq m \leq M
\]

(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,

\[
0 \leq n - m \leq L - 1
\]

(4.1.9)

To determine the range of values of the output index \( n \), we rewrite (4.1.9) in the form:

\[
m \leq n \leq L - 1 + m
\]

and use (4.1.8) to extend the limits to:

\[
0 \leq m \leq n \leq L - 1 + m \leq L - 1 + M, \quad \text{or,}
\]

\[
0 \leq n \leq L - 1 + M
\]

(4.1.10)

This is the index range of the output sequence \( y(n) \). Therefore, it is represented by a block

\[
y = [y_0, y_1, \ldots, y_{L-1+M}]
\]

(4.1.11)

with length

\[
L_y = L + M
\]

(4.1.12)

Thus, \( y \) is longer than the input \( x \) 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

\footnote{More correctly, \( T_L = (L - 1)T \), but for large \( L \), Eq. (4.1.1) is simpler. See also Section 9.1.}
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:

$$L_y = L_x + L_h - 1$$

(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

$$m = \min(n, M)$$

(4.1.16)

Therefore, in the case of an order-$M$ FIR filter and a length-$L$ input, the direct form of convolution is given as follows:

$$y(n) = \sum_{m=\max(0,n-L+1)}^{\min(n,M)} h(m)x(n-m)$$

(4.1.17)

for $n = 0, 1, \ldots, L + M - 1$. Sometimes, we will indicate this convolutional operation by the compact notation:

$$y = h \ast x$$

As an example, consider the case of an order-3 filter and a length-5 input signal. The filter, input, and output blocks are

$$h = [h_0, h_1, h_2, h_3]$$

$$x = [x_0, x_1, x_2, x_3, x_4]$$

$$y = h \ast x = [y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7]$$

Using the values in Eq. (4.1.17), we find all the output samples:

$$y_0 = h_0x_0$$

$$y_1 = h_0x_1 + h_1x_0$$

$$y_2 = h_0x_2 + h_1x_1 + h_2x_0$$

$$y_3 = h_0x_3 + h_1x_2 + h_2x_1 + h_3x_0$$

$$y_4 = h_0x_4 + h_1x_3 + h_2x_2 + h_3x_1$$

$$y_5 = h_1x_4 + h_2x_3 + h_3x_2$$

$$y_6 = h_2x_4 + h_3x_3$$

$$y_7 = h_3x_4$$

### 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

$$y_5 = \sum_{i+j=5} h_i x_j = h_1x_4 + h_2x_3 + h_3x_2$$
The required computations can be arranged in a table [24] as shown in Fig. 4.1.2 with the filter \( h \) written vertically and the input block \( x \) horizontally.\(^7\)

\[
\begin{array}{cccccc}
\hline
x_0 & x_1 & x_2 & x_3 & x_4 \\
\hline
h_0 & h_0x_0 & h_0x_1 & h_0x_2 & h_0x_3 & h_0x_4 \\
h_1 & h_1x_0 & h_1x_1 & h_1x_2 & h_1x_3 & h_1x_4 \\
h_2 & h_2x_0 & h_2x_1 & h_2x_2 & h_2x_3 & h_2x_4 \\
h_3 & h_3x_0 & h_3x_1 & h_3x_2 & h_3x_3 & h_3x_4 \\
\end{array}
\]

Convolution table

Fig. 4.1.2 Convolution table.

The \( n \)th row of the table is filled by multiplying the \( x \) 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 \( n \)th 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 \( h_2x_4 \), 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:

\[
h = [1, 2, -1, 1], \quad x = [1, 1, 2, 1, 2, 1, 1]
\]

**Solution:** The convolution table, with \( h \) arranged vertically and \( x \) horizontally, is

<table>
<thead>
<tr>
<th>( h )</th>
<th>1 1 2 1 2 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 1 2 1 2 1 1</td>
</tr>
<tr>
<td>2</td>
<td>2 2 4 2 4 2 2</td>
</tr>
<tr>
<td>-1</td>
<td>-1 -1 -2 -1 -2 -1</td>
</tr>
<tr>
<td>1</td>
<td>1 1 2 1 2 1 1</td>
</tr>
</tbody>
</table>

Folding the table, we get

\[
y = [1, 3, 3, 5, 3, 7, 4, 3, 3, 0, 1]
\]

Note that there are \( L_y = L + M = 8 + 3 = 11 \) output samples.\(^8\)

\(^7\)It can also be arranged the other way, with the filter horizontally and the input vertically.

\(^8\)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 \( h = [h_0, h_1, h_2, h_3] \) in the example of Eq. (4.1.18). The input signal

\[
x = [x_0, x_1, x_2, x_3, x_4]
\]

can be written as a linear combination of delayed impulses:

\[
x = x_0[1, 0, 0, 0, 0] + x_1[0, 1, 0, 0, 0] + x_2[0, 0, 1, 0, 0] + x_3[0, 0, 0, 1, 0] + x_4[0, 0, 0, 0, 1]
\]

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:

\[
x = x_0[1, 0, 0, 0, 0] + x_1[0, 1, 0, 0, 0] + h_{x_2,[0, 0, h_x1, h_x2, h_x3, h_x4, 0, 0, 0, 0]}
\]

\[
y = x_0h_0, x_1h_1 + x_2h_2 + x_3h_3 + \ldots + x_4h_4
\]

The result is the same as Eq. (4.1.18). Indeed, the indicated linear combinations in the right-hand side give:

\[
y = [y_0, y_1, y_2, \ldots, y_7]
\]

For computational purposes, the LTI form can be represented pictorially in a table form, as shown in Fig. 4.1.3. The impulse response \( h \) is written horizontally, and the input \( x \) vertically.

The rows of the table correspond to the successive delays (right shifts) of the \( h \) sequence—the \( m \)th row corresponds to delay by \( m \) units. Each row is scaled by the
4.1. BLOCK PROCESSING METHODS

Fig. 4.1.3 LTI form of convolution.

The corresponding input sample, that is, the $m$th 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:

<table>
<thead>
<tr>
<th>$h_0$</th>
<th>$h_1$</th>
<th>$h_2$</th>
<th>$h_3$</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_0$</td>
<td>$x_0 h_0$</td>
<td>$x_1 h_1$</td>
<td>$x_2 h_2$</td>
<td>$x_3 h_3$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$x_1$</td>
<td>0</td>
<td>$x_1 h_0$</td>
<td>$x_1 h_1$</td>
<td>$x_1 h_2$</td>
<td>$x_1 h_3$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$x_2$</td>
<td>0</td>
<td>0</td>
<td>$x_2 h_0$</td>
<td>$x_2 h_1$</td>
<td>$x_2 h_2$</td>
<td>$x_2 h_3$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$x_3$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$x_3 h_0$</td>
<td>$x_3 h_1$</td>
<td>$x_3 h_2$</td>
<td>$x_3 h_3$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$x_4$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$x_4 h_0$</td>
<td>$x_4 h_1$</td>
<td>$x_4 h_2$</td>
<td>$x_4 h_3$</td>
<td>0</td>
</tr>
</tbody>
</table>

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:

$$y(n) = \min(n,M-1) \sum_{m=\max(0,n-M)}^{m=\min(M,n)} x(m) h(n-m)$$

(4.1.19)

for $n = 0, 1, \ldots, L + M - 1$.

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.

Example 4.1.3: Calculate the convolution of Example 4.1.1 using the matrix form.

The convolutional equations (4.1.16) or (4.1.19) can also be written in the linear matrix form:

$$y = H x$$

(4.1.20)

where $H$ is built out of the filter’s impulse response $h$. Because the output vector $y$ has length $L + M$ and the input vector $x$ 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

$$y = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} h_0 & 0 & 0 & 0 & 0 & 0 & 0 \\ h_1 & h_0 & 0 & 0 & 0 & 0 & 0 \\ h_2 & h_1 & h_0 & 0 & 0 & 0 & 0 \\ h_3 & h_2 & h_1 & h_0 & 0 & 0 & 0 \\ h_4 & h_3 & h_2 & h_1 & h_0 & 0 & 0 \\ h_5 & h_4 & h_3 & h_2 & h_1 & h_0 & 0 \\ h_6 & h_5 & h_4 & h_3 & h_2 & h_1 & h_0 \end{bmatrix} x = H x$$

Note that the columns of $H$ are the successively delayed replicas of the impulse response vector $h$. 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.
### 4.1. BLOCK PROCESSING METHODS

**Solution:** Because $L_y = 11$ and $L_x = 8$, the filter matrix will be $11 \times 8$ dimensional. We have:

\[
H_x = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 1 \\ 5 \\ 2 \\ 3 \\ 7 \\ 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 5 \\ 7 \\ 4 \\ 3 \\ 0 \\ 0 \end{bmatrix}
\]

which agrees with Example 4.1.1.

There is also an alternative matrix form written as follows:

\[
\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} x_0 & 0 & 0 & 0 \\ x_1 & x_0 & 0 & 0 \\ x_2 & x_1 & x_0 & 0 \\ x_3 & x_2 & x_1 & x_0 \\ x_4 & x_3 & x_2 & x_1 \\ x_5 & 0 & x_4 & x_3 \\ x_6 & 0 & 0 & x_4 \\ x_7 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix}
\]

Instead of a filter matrix $H$ acting on the input data vector $x$, it has a data matrix acting on the filter vector $h$. It can be written compactly in the form:

\[
y = Xh
\]

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 $n$th 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:

\[
x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}
\]

\[
h = \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 5 \\ 7 \end{bmatrix}
\]

\[
Xh = \begin{bmatrix} 1 \end{bmatrix}
\]

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 $h$ 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.](image-url)

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 $M \leq n \leq L - 1$, the filter remains completely over the nonzero portion of the input data, and the outputs are given by the form

\[
y_n = h_0 x_n + h_1 x_{n-1} + \cdots + h_M x_{n-M}
\]
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 \( h \) as being stationary and the input block \( x \) 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:

- \( 0 \leq n < M \) (input-on transients)
- \( M \leq n \leq L - 1 \) (steady state)
- \( L - 1 < n \leq L - 1 + M \) (input-off transients)

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.](image)

For the input-on transients, the restriction \( 0 \leq n < M < L - 1 \) implies the following summation limits:

\[
\max(0, n - L + 1) = 0, \quad \min(n, M) = n
\]

For the steady-state range, \( M \leq n \leq L - 1 \), we have:

\[
\max(0, n - L + 1) = 0, \quad \min(n, M) = M
\]

And, for the input-off transients, \( M < L - 1 < n \leq L - 1 + M \), we have:

\[
\max(0, n - L + 1) = n - L + 1, \quad \min(n, M) = M
\]

Therefore, Eq. (4.1.16) takes the following different forms depending on the value of the output index \( n \):

\[
y_n = \begin{cases} 
\sum_{m=0}^{n} h_m x_{n-m}, & \text{if } 0 \leq n < M, \\
\sum_{m=0}^{M} h_m x_{n-m}, & \text{if } M \leq n \leq L - 1, \\
\sum_{m=n-L+1}^{M} h_m x_{n-m}, & \text{if } L - 1 < n \leq L - 1 + M,
\end{cases}
\]

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:

\[
y(n) = \sum_{m=0}^{M} h(m) x(n-m)
\]

(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

\[
y_1 = h_0 x_1 + h_1 x_0 - h_2 x_{-1} - h_3 x_{-2}
\]

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

\[
y_6 = h_0 x_6 + h_1 x_5 + h_2 x_{-1} + h_3 x_{-2} + h_4 x_{-3} + h_5 x_{-4}
\]
because \( x_n \) was assumed to have length 5, and therefore \( x_0 = x_5 = 0 \). 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, \( 0 \leq n < \infty \), 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:

\[
y_n = \sum_{m=\max(0,n-L+1)}^{\min(n, M)} h_m x_{n-m}, \quad \text{if} \quad M = \infty, L < \infty
\]

\[
y_n = \sum_{m=0}^{\min(n, M)} h_m x_{n-m}, \quad \text{if} \quad M < \infty, L = \infty
\]

\[
y_n = \sum_{m=0}^{n} h_m x_{n-m}, \quad \text{if} \quad M = \infty, L = \infty
\]

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

\[
y(n) = \sum_{m=0}^{n} (0.75)^m (-1)^{n-m} = (-1)^n \sum_{m=0}^{n} (-0.75)^m
\]

\[
= (-1)^n \frac{1 - (-0.75)^{n+1}}{1 + 0.75} = \frac{4}{7} (-1)^n + \frac{3}{7} (0.75)^n
\]

where in the second term, we wrote \((-1)^n (-0.75)^m = (0.75)^m\). 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,

\[
y(n) \to H(1) \quad \text{and} \quad y(n) \to (-1)^n H(-1)
\]

where in this example,

\[
H(z) = \frac{1}{1 - 0.75 z^{-1}} \quad \Rightarrow \quad H(1) = \frac{1}{1 - 0.75} = 4, \quad H(-1) = \frac{1}{1 + 0.75} = \frac{4}{7}
\]

In part (c), the input is finite with length \( L = 25 \). Therefore,

\[
y_n = \sum_{m=\max(0,n-L+1)}^{\min(n, M)} h_m x_{n-m} = \sum_{m=\max(0,n-L+1)}^{n} (0.75)^m
\]

We must distinguish two subranges in the output index: For \( 0 \leq n \leq 24 \), we have

\[
y_n = \sum_{m=0}^{n} (0.75)^m = \frac{1 - (0.75)^{n+1}}{1 - 0.75} = 4 - 3(0.75)^n
\]
4.1. BLOCK PROCESSING METHODS

and for $25 \leq n < \infty$,

$$y_n = \sum_{m=n-24}^{n} (0.75)^m (0.75)^{n-24} \frac{1 - (0.75)^{n-(n-24)+1}}{1 - 0.75}$$

$$= (0.75)^{n-24} \frac{1 - (0.75)^{25}}{1 - 0.75}$$

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 $0 \leq n \leq 24$, the output "paces" up and during the input-off period $n \geq 25$, 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:

![Graphs showing output signals for cases (a-c)]

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:

$$y(n) = 0.75 y(n-1) + x(n)$$

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 $n \geq 0$: $y(n) = 0.75 y(n-1) + 1$

For $n = 0$, it gives $y(0) = 0.75 y(-1) + 1 = 0.75 \cdot 0 + 1 = 1$, which agrees with the expression $y(n) = 4 - 3(0.75)^n$ evaluated at $n = 0$. For $n \geq 1$, we have, starting with the right-hand side:

$$0.75 y(n-1) + 1 = 0.75 \left[ 4 - 3(0.75)^{n-1} \right] + 1 = 4 - 3(0.75)^n = y(n)$$

In case (b), we have $x(n) = (-1)^n u(n)$ and the difference equation becomes for $n \geq 1$:

$$0.75 y(n-1) + x(n) = 0.75 \left[ \frac{4}{7} (-1)^{n-1} + \frac{3}{7} (0.75)^{n-1} \right] + (-1)^n$$

$$= -0.75 \frac{4}{7} (-1)^n + \frac{3}{7} (0.75)^n + (-1)^n = \frac{4}{7} (-1)^n + \frac{3}{7} (0.75)^n = y(n)$$

In case (c), we have the difference equations:

$$y(n) = 0.75 y(n-1) + 1, \quad \text{for } 0 \leq n \leq 24$$

and

$$y(n) = 0.75 y(n-1), \quad \text{for } n \geq 25$$

The first value at $n = 25$ will be $y(25) = 0.75 y(24)$, and therefore, it requires knowledge of the "initial" value $y(24)$. If that is known, then the solution of the homogeneous equation will be $y(n) = (0.75)^{n-24} y(24)$, for $n \geq 25$. 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:

$$-M_1 \leq n \leq M_2$$

and the input signal $x(n)$ over the interval:

$$-L_1 \leq n \leq L_2 - 1$$

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,

\[-M_1 \leq m \leq M_2\]  \hspace{1cm} \[-L_1 \leq n - m \leq L_2 - 1\]

From these it follows that the output index \(n\) must vary over the range:

\[-M_1 - L_1 \leq n \leq L_2 + M_2 - 1\]

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:

\[
\max(-M_1, n - L_2 + 1) \leq m \leq \min(n + L_1, M_2)
\]

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):

```c
#include <stdlib.h>

#define max(a, b) (((a) > (b)) ? (a) : (b))
#define min(a, b) (((a) < (b)) ? (a) : (b))

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

    for (n = 0; n < L+M; n++)
    {
        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`:

```c
double *h, *x, *y;

h = (double *) calloc(M+1, sizeof(double)); \textit{\(M+1\)-dimensional}

x = (double *) calloc(L, sizeof(double)); \textit{\(L\)-dimensional}

y = (double *) calloc(L+M, sizeof(double)); \textit{\((L+M)\)-dimensional}
```

In some C implementations,\(^1\) the include-file *stdlib.h* contains the definitions of the two macros \texttt{max} and \texttt{min}; otherwise they must be added in the above routine:

\(^1\)For example, Microsoft and Borland C.

---

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:

\[
y_{dc} = \sum_{m} h(m)
\]  \hspace{1cm} (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:

\[
y(n) = G[x(n) + x(n - 1) + x(n - 2) + \cdots + x(n - 14)]
\]

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:

\[
h_n = \begin{cases} 
G, & \text{for } 0 \leq n \leq 14 \\
0, & \text{otherwise}
\end{cases}
\]

The DC gain will be:

\[
y_{dc} = \sum_{m=0}^{14} h(m) = \sum_{m=0}^{14} G = 15G = 1.5
\]

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:

```c
for (n=0; n<K; n++)
    if (n%K < K/2)
        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`:

```c
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. \(\square\)

**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) \(h(n) = \begin{cases} \ b a^n, & \text{for } 0 \leq n \leq M \\ 0, & \text{otherwise} \end{cases} \)

Take \(M = 14\), \(a = 0.75\), and \(b = 1 - a = 0.25\). Its DC gain is almost unity:

\[
y_{dc} = \sum_{m=0}^{M} h(m) = b \sum_{m=0}^{M} a^m = (1 - a) \frac{1 - a^{M+1}}{1 - a} = 1 - a^{M+1}
\]

or, \(y_{dc} = 1 - (0.75)^{15} = 0.987\).

(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 \(H(z)\):

\[
h = [0.2, -1, 2, -2, 1, -0.2] = \frac{1}{5} [1, -5, 10, -10, 5, -1]
\]

The square wave input of the previous example is fed into the two filters. The resulting output signals, computed by two calls to \texttt{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 \textit{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. \(\square\)

**Example 4.1.9:** To demonstrate the concepts of impulse response, linearity, and time invariance, consider an FIR filter with impulse response

\[
h(n) = (0.95)^n, \quad \text{for } 0 \leq n \leq 24
\]

and an input signal

\[
x(n) = \delta(n) + 2\delta(n - 40) + 2\delta(n - 70) + \delta(n - 80), \quad n = 0, 1, \ldots, 120
\]

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,

\[
y(n) = h(n) + 2h(n - 40) + 2h(n - 70) + h(n - 80) \quad (4.1.25)
\]

The input signal can be generated with the aid of the following C routine that implements a delta function \(\delta(n)\):

```c
/* 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
4.1. BLOCK PROCESSING METHODS

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 $L_f = L + M = 121 + 24 = 145$, and can be generated by single call to $\text{conv}$:

    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. \qed

4.1.10 Overlap-Add Block Convolution Method

In the above examples, the entire input signal was passed to $\text{conv}$ 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 $x_0$, $x_1$, $x_2$, ..., is convolved with the order-$M$ filter $h$ producing the outputs blocks:

$$y_0 = h \ast x_0$$
$$y_1 = h \ast x_1$$
$$y_2 = h \ast x_2$$

and so on. The resulting blocks are pieced together according to their absolute timing. Block $y_0$ starts at absolute time $n = 0$; block $y_1$ starts at $n = L$ because the corresponding input block $x_1$ starts then; block $y_2$ 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).

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

$$x = \begin{bmatrix} 1, 1, 2 \end{bmatrix}, \begin{bmatrix} 1, 2, 1 \end{bmatrix}, \begin{bmatrix} 2, 1, 0 \end{bmatrix}$$

where we padded an extra zero at the end of $x_2$ to get a length-3 block. Convolving each block separately with $h = \begin{bmatrix} 1, 2, -1, 1 \end{bmatrix}$ gives:

$$y_0 = h \ast x_0 = \begin{bmatrix} 1, 3, 3, -1, 2 \end{bmatrix}$$
$$y_1 = h \ast x_1 = \begin{bmatrix} 1, 4, 5, 3, 0, 2 \end{bmatrix}$$
$$y_2 = h \ast x_2 = \begin{bmatrix} 1, 3, 1, 0, 1 \end{bmatrix}$$

These convolutions can be done by separately folding the three convolution subtables:

<table>
<thead>
<tr>
<th>$h \ast x$</th>
<th>block 0</th>
<th>block 1</th>
<th>block 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h$</td>
<td>1 1 1 2</td>
<td>1 2 2 4</td>
<td>1 2 2 4</td>
</tr>
<tr>
<td>$x_0$</td>
<td>1 1 1 2</td>
<td>1 2 2 4</td>
<td>1 2 2 4</td>
</tr>
<tr>
<td>$x_1$</td>
<td>-1 -1 -2</td>
<td>-1 -2 -2</td>
<td>-1 -2 -2</td>
</tr>
<tr>
<td>$x_2$</td>
<td>1 1 1 2</td>
<td>1 2 2 4</td>
<td>1 2 2 4</td>
</tr>
</tbody>
</table>

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:

$$y = \begin{bmatrix} 1, 3, 3, 4, -1, 2 \end{bmatrix}$$
$$y_0 = \begin{bmatrix} 1, 3, 3, 4, -1, 2 \end{bmatrix}$$
$$y_1 = \begin{bmatrix} 1, 4, 5, 3, 0, 2 \end{bmatrix}$$
$$y_2 = \begin{bmatrix} 1, 3, 1, 0, 1 \end{bmatrix}$$

$$y = \begin{bmatrix} 1, 3, 3, 5, 3, 7, 4, 3, 3, 0, 1 \end{bmatrix}$$

Fig. 4.1.6 Overlap-add block convolution method.
4.1. BLOCK PROCESSING METHODS

which agrees with Example 4.1.1.

The method can be implemented by the following algorithm, which reads the input data in blocks of length \( L \) and outputs the result also in blocks of length \( L \):

```c
for each length-\( L \) input block \( x \) do:
    1. compute length-(\( L + M \)) output: \( y = h \ast x \)
    2. for \( i = 0, 1, \ldots, M-1 \):
        \( y(i) = y(i) + ytemp(i) \) (overlap)
        \( ytemp(i) = y(i + L) \) (save tail)
    3. for \( i = 0, 1, \ldots, L-1 \):
        output \( y(i) \)
```

It uses a temporary \( M \)-dimensional vector \( ytemp \) to store the last \( M \) samples of each previous block. Before processing the first block, \( ytemp \) must be initialized to zero.

After computing the length-(\( L + M \)) filter output \( y = h \ast x \), the first \( M \) samples of \( y \) are added to the last \( M \) samples of the previous block held in \( ytemp \). Then, the last \( M \) samples of the currently computed block \( y \) are saved in \( ytemp \) for use in the next iteration. Only the first \( L \) corrected output samples of \( y \) 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 \( x \) 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 \( N = 1024 \) 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 \( L \).

```c
/* blockcon.c - block convolution by overlap-add method */
void conv();
void blockcon(M, h, L, x, y, ytemp)
    double *h, *x, *y, *ytemp;
    int M, L;
{ int i;
    conv(M, h, L, x, y); /* compute output block \( y \) */
    for (i=0; i<M; i++)
```

The quantities \( h, x, y, ytemp \) are arrays and must be allocated in the main program, for example, using `calloc`:

```c
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 output 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.

The quantities \( h, x, y, ytemp \) are arrays and must be allocated in the main program for example, using `calloc`:

```c
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 output 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.

```c
    for (;;) {
        if (fscanf(fpx, "%lf", x+N) == EOF) goto last;
        for (i=0; i<M; i++)
            printf(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 \( N \leq L \). The entire last output block of length \( N+M \) is written into the output file. The last output samples represent the input-off transients. The file pointers, `fpx`, `fpy`, must be declared and defined in the main program by:

```c
    FILE *fpx, *fpy;
    fpx = fopen("x.dat", "r"); // open for read
    fpy = fopen("y.dat", "w"); // open for write
```

Separate 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,

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

(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:

\[
y(t_n) = \int h(t_n - \tau) x(\tau) \, d\tau = \lim_{T \to 0} \left[ T \sum_{m} h(t_n - t_m) x(t_m) \right] 
\]  

(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,

\[
y(t_n) \approx T \sum_{m} h(t_n - t_m) x(t_m)
\]  

(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 \( t_n \) by the sampled time instant, \( t_n = nT \), we finally obtain,

\[
y(t) = \sum_{m} h(t_n - t_m) x(t_m)
\]  

(4.2.4)

This can be implemented by the MATLAB code:

\[
y = T * \text{conv(h,x)};
\]  

(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:

\[
\frac{dy(t)}{dt} + ay(t) = ax(t) \implies h(t) = ae^{-at} u(t), \quad H(s) = \frac{a}{s + a}
\]  

(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,

\[
x(t) = F(t) = u(t) - u(t-t_d), \quad t_d > 0
\]  

(4.2.7)

The exact convolutional output due to the input \( F(t) \) was found in Example 1 of this set:

\[
y_{\text{exact}}(t) = G(t) = (e^{at} - e^{a(t-t_d)}) u(t) = \begin{cases} 1 - e^{-at}, & 0 \leq t \leq t_d \\ e^{-a(t-t_d)}[1 - e^{-at}], & t \geq t_d \\ 0, & t < 0 \end{cases}
\]  

(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 \( \tau \) and \( \alpha \) have already been defined previously),

\[
F = @(t) (t>=0) - (t>=td);
G = @(t) exp(-a*t) .* (exp(a*min(t,td)) - 1) .* (t>=0);
\]

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,

\[
x(t) = c_1 F(t-t_1) + c_2 F(t-t_2) + c_3 F(t-t_3)
\]  

(4.2.9)

\[
y(t) = c_1 G(t-t_1) + c_2 G(t-t_2) + c_3 G(t-t_3)
\]

(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 \( T_{\text{max}} = 25 \), 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 \([0, T_{\text{max}}]\) interval in steps of \( T \), that is, \( t = 0:T:T_{\text{max}} \).

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, \( N = \text{length}(f) \), convolutional outputs. This can be accomplished by redefining the computed output vector by:

\[
f = \text{f(1:N)};
\]

\[
x = \text{x(1:N)};
\]

\[
y = \text{y(1:N)};
\]

\[
T = \text{t(1:N)};
\]

With the above definitions, putting the input signal \( x(t) \), the output \( y(t) \), and the exact output \( y_{\text{exact}}(t) \) on the same graph, you should observe that the numerical approximation gets better as the sampling interval \( T \) gets smaller.
4.2. NUMERICAL EVALUATION OF CONTINUOUS-TIME CONVOLUTION

(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.

```matlab
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
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('t');
```

0.5
1
1.5
2
2.5
3

0
5
10
15
20
25

0
5
10
15
20
25

0
0.5
1
1.5
2
2.5
3

0
5
10
15
20
25

0
0.5
1
1.5
2
2.5
3

0
5
10
15
20
25

0
0.5
1
1.5
2
2.5
3

0
5
10
15
20
25

0
0.5
1
1.5
2
2.5
3

0
5
10
15
20
25

0
0.5
1
1.5
2
2.5
3

0
5
10
15
20
25
4.2. NUMERICAL EVALUATION OF CONTINUOUS-TIME CONVOLUTION

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,

\[
x(t) = \begin{cases} 
\sin(\omega_1 t), & 0 \leq t < 30 \\
\sin(\omega_0 t), & 30 \leq t < 70 \\
\sin(\omega_1 t), & 70 \leq t < 100 
\end{cases}
\]

where \( \omega_0 = 2 \) and \( \omega_1 = 3 \). It can be generated over the time interval, \( 0 \leq t \leq 100 \), by the following MATLAB code segment:

```matlab
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, \( t,t_d \),

\[
F = \Theta(t,t_d) \cdot (t<>0) - (t<>t_d);
\]

It is desired to eliminate the middle burst by means of a notch filter:

\[
H(s) = \frac{s^2 + \omega_0^2}{s^2 + \alpha s + \omega_0^2}
\]  

(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 \( Q = \omega_0/\Delta \omega = \omega_0/\alpha \). It can be verified that the impulse response of this filter is:

\[
h(t) = \delta(t) - g(t), \quad g(t) = \alpha e^{-\alpha t/2} \left[ \cos(\omega_r t) - \frac{\alpha}{2\omega_r} \sin(\omega_r t) \right] u(t)
\]

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),

\[
y(t) = \int_0^\infty h(t') x(t-t') \, dt' = \int_0^\infty [\delta(t') - g(t')] x(t-t') \, dt' = x(t) - \int_0^\infty g(t') x(t-t') \, dt'
\]

which can be implemented approximately by the MATLAB code:

\[
y = x - T * \text{conv}(g, x);
\]  

(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 \( H(\omega_0) \). See the blue line segments in the example graphs below.

(b) Calculate the 40-dB time constant of this filter given in general by,

\[
\tau = -\frac{\ln(10^{-40/20})}{|\text{Re}(p)|} = \frac{\ln(100)}{|\text{Re}(p)|}
\]  

(4.2.12)

where \( p \) is the stable pole closest to the \( j\omega \) axis. Is the value of \( \tau \) 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, \( \text{lsim} \), and make a plot of \( y(t) \) using the same scales as in part (a). Compare the outputs from the \( \text{lsim} \) and \( \text{conv} \) methods by computing the percentage error as the ratio,

\[
\text{Error} = 100 \cdot \frac{\|y_{\text{lsim}} - y_{\text{conv}}\|}{\|y_{\text{conv}}\|}
\]

where the norm \( \|y\| \) can be computed with the built-in function \( \text{norm} \).

\[\text{\footnotesize \text{\textsuperscript{1}the filter poles are at,} p = -\alpha/2 \pm j\omega_r, \text{in the left-hand s-plane}}\]
(d) The frequency and magnitude responses of the transfer function $H(s)$ of Eq. (4.2.10) are,

$$H(\omega) = \frac{\omega_0^2 - \omega^2}{\omega_0^2 - \omega^2 + \alpha j \omega} \Rightarrow |H(\omega)|^2 = \frac{(\omega^2 - \omega_0^2)^2}{(\omega^2 - \omega_0^2)^2 + \alpha^2 \omega^2}$$  \hspace{1cm} (4.2.13)

The (positive) left and right 3-dB frequencies $\omega_\pm$, that is, the solutions of the 3-dB half-power condition, $|H(\omega)|^2 = \frac{1}{2}$, are given by,

$$\omega_\pm = \pm \frac{\alpha}{2} \sqrt{\omega_0^2 + \frac{\alpha^2}{4}}$$  \hspace{1cm} (4.2.14)

Note that these frequencies satisfy the following useful relationships,†

$$\omega_+ - \omega_- = \frac{\alpha}{\omega_0^2} \omega_0 + \omega_- = \omega_0^2$$  \hspace{1cm} (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, $0 \leq \omega \leq 5$, 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 $\omega_1$ 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, $\alpha^2 \omega_+ \omega_- = \omega_0^3$, then, use the identity, $(\alpha_+ - \omega_-)^2 = \omega_0^2 + \omega_-^2 - 2 \omega_+ \omega_-$. 

4.2. NUMERICAL EVALUATION OF CONTINUOUS-TIME CONVOLUTION

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.
4.2. NUMERICAL EVALUATION OF CONTINUOUS-TIME CONVOLUTION

The typical MATLAB code for generating the above graphs is as follows:

```matlab
F = @(t,td) (t>=0) - (t>=td); % pulse function
Tmax = 100; T = Tmax/2000; % try also, T=0.01
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-y g ; % convolutional output
H1 = abs((w0^2 - w1^2)/(w0^2 - a^2/4 + 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)';
percent_error = 100 * norm(y-yf)/norm(y) % Error = 0.9725 for a = 0.3

percent_error = 100 * norm(y-yf)/norm(y) % Error = 0.9725 for a = 0.3
s = tf('s'); % transfer function class
H = (s^2 + w0^2)/(s^2 + a*s + w0^2); yf = lsim(H,x,t)';
figure; plot(t,y,'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('\omega'); % label x-axis
w = linspace(0,5,501); s = j*w;
H = abs((s^2+w0^2)/(s^2+a*s+w0^2))^2;
figure; plot(w,abs(H).^2), xlabel('\omega'), ylabel('|H(\omega)|^2');
figure; plot(w,abs(H).^2), xlabel('\omega'), ylabel('|H(\omega)|^2');
hold on; plot([w,abs(w)^2], [1,1]/2, 'r-'); % add 3-dB width
```

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.

![Block diagram of sample processing](image)

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
4.3. SAMPLE PROCESSING METHODS

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

\[ y(n) = w_1(n) \]

\[ w_1(n+1) = x(n) \]

Therefore, \( w_2(n) \) is the doubly delayed version of \( x(n) \):

\[ w_2(n) = w_1(n-1) = x((n-1)+1) = x(n-2) \]

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,

\[ w_2(n+1) = w_1(n) \]
\[ w_1(n+1) = x(n) \]

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:

```
for each input sample x do:
y := w_2
w_2 := w_1
w_1 := x
```

The repetitive sample processing algorithm describing these equations is:
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 $w_1(0) = w_2(0) = 0$):

<table>
<thead>
<tr>
<th>$n$</th>
<th>$x(n)$</th>
<th>$w_1(n)$</th>
<th>$w_2(n)$</th>
<th>$y(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$x_0$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>$x_1$</td>
<td>$x_0$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>$x_2$</td>
<td>$x_1$</td>
<td>$x_0$</td>
<td>$x_0$</td>
</tr>
<tr>
<td>3</td>
<td>$x_3$</td>
<td>$x_2$</td>
<td>$x_1$</td>
<td>$x_1$</td>
</tr>
<tr>
<td>4</td>
<td>$x_4$</td>
<td>$x_3$</td>
<td>$x_2$</td>
<td>$x_2$</td>
</tr>
<tr>
<td>5</td>
<td>$x_5$</td>
<td>$x_4$</td>
<td>$x_3$</td>
<td>$x_3$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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,

$$w_3(n) = w_2(n - 1)$$
$$w_2(n) = w_1(n - 1)$$
$$w_1(n) = x(n - 1)$$

Thus, $w_3(n) = w_2(n - 1) = w_1(n - 2) = x(n - 3)$. Their updates to time $n+1$ are:

$$w_3(n + 1) = w_2(n)$$
$$w_2(n + 1) = w_1(n)$$
$$w_1(n + 1) = x(n)$$

Therefore, the I/O equations for a triple delay will be:

$$y(n) = w_3(n)$$
$$w_3(n + 1) = w_2(n)$$
$$w_2(n + 1) = w_1(n)$$
$$w_1(n + 1) = x(n)$$

And the corresponding sample processing algorithm:

\[
\text{for each input sample } x \text{ do:} \\
\quad y := w_3 \\
\quad w_3 := w_2 \\
\quad w_2 := w_1 \\
\quad w_1 := x
\]

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 $w_i(n)$, $i = 1, 2, \ldots, D$. For convenience, the input is denoted by $w_0(n)$. The output of each register is the delayed version of its input:

$$w_i(n) = w_{i-1}(n-1), \quad \text{for } i = 1, 2, \ldots, D \quad (4.3.4)$$

Thus, $w_D(n) = x(n)$. The corresponding sample processing algorithm will be:

\[
\text{for each input sample } x \text{ do:} \\
\quad y := w_D \\
\quad w_D := x \\
\quad \text{for } i = D, D-1, \ldots, 1 \text{ do:} \\
\quad \quad w_i := w_{i-1}
\]

or, more simply:

\[
\text{for each input sample } w_0 \text{ do:} \\
\quad \text{for } i = D, D-1, \ldots, 1 \text{ do:} \\
\quad \quad w_i := w_{i-1}
\]

The following C routine *delay.c* is an implementation of this algorithm:
4.3. SAMPLE PROCESSING METHODS

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 \( x(n-1) \), \( x(n-2) \), and \( x(n-3) \).

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.](image)

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 \( w_1(n) \), \( w_2(n) \), and \( w_3(n) \) to describe the feedback arrangements, to implement several digital audio effects, such as digital reverb, and to implement periodic waveform generators.

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

\[
y(n) = h_0x(n) + h_1x(n-1) + \cdots + h_Mx(n-M)
\]  

(4.3.5)

with impulse response \( h = [h_0, h_1, \ldots, h_M] \). For example, a third-order filter

\[
h = [h_0, h_1, h_2, h_3]
\]

will have I/O equation:

\[
y(n) = h_0x(n) + h_1x(n-1) + h_2x(n-2) + h_3x(n-3)
\]  

(4.3.6)

so that each is a delayed version of the previous one. With these definitions, we can rewrite Eq. (4.3.6) in the form:

\[
y(n) = h_0w_0(n) + h_1w_1(n) + h_2w_2(n) + h_3w_3(n)
\]  

(4.3.7)

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...
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:

\[
\begin{align*}
\text{row of } & w_i \text{'s is the delayed (right-shifted) version of the previous row. The computed output samples } y(n) \text{ are subject to this algorithm:} \\
\text{for each input sample } x \text{ do:} \\
& w_0(n) = x(n) \\
& y(n) = h_0 w_0(n) + h_1 w_1(n) + h_2 w_2(n) + h_3 w_3(n) \\
& w_1(n+1) = w_1(n) \\
& w_2(n+1) = w_2(n) \\
& w_3(n+1) = w_3(n)
\end{align*}
\]

It can be mechanized by the following sample-by-sample processing algorithm:

\[
\text{for each input sample } x \text{ do:} \\
& w_0 = x(n) \\
& y = h_0 w_0 + h_1 w_1 + h_2 w_2 + h_3 w_3 \\
& w_1 = w_1(n) \\
& w_2 = w_2(n) \\
& w_3 = w_3(n) \\
\]

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 for the delays to empty out their contents and be filled with zeros again.

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):

<table>
<thead>
<tr>
<th>( n )</th>
<th>( x )</th>
<th>( w_0 )</th>
<th>( w_1 )</th>
<th>( w_2 )</th>
<th>( w_3 )</th>
<th>( y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_0 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( h_0 x_0 )</td>
</tr>
<tr>
<td>1</td>
<td>( x_1 )</td>
<td>( x_0 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( h_0 x_1 + h_1 x_0 )</td>
</tr>
<tr>
<td>2</td>
<td>( x_2 )</td>
<td>( x_1 )</td>
<td>( x_0 )</td>
<td>0</td>
<td>0</td>
<td>( h_0 x_2 + h_1 x_1 + h_2 x_0 )</td>
</tr>
<tr>
<td>3</td>
<td>( x_3 )</td>
<td>( x_2 )</td>
<td>( x_1 )</td>
<td>( x_0 )</td>
<td>0</td>
<td>( h_0 x_3 + h_1 x_2 + h_2 x_1 + h_3 x_0 )</td>
</tr>
<tr>
<td>4</td>
<td>( x_4 )</td>
<td>( x_3 )</td>
<td>( x_2 )</td>
<td>( x_1 )</td>
<td>0</td>
<td>( h_0 x_4 + h_1 x_3 + h_2 x_2 + h_3 x_1 )</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>( x_4 )</td>
<td>( x_3 )</td>
<td>0</td>
<td>( h_1 x_4 + h_2 x_3 + h_3 x_2 )</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( x_4 )</td>
<td>0</td>
<td>( h_2 x_4 + h_3 x_3 )</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( x_4 )</td>
<td>( h_3 x_4 )</td>
</tr>
</tbody>
</table>

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 \( M \)th order filter, we may define \( w_0(n) = x(n) \) and for \( i = 1, 2, \ldots, M \)

\[
w_i(n) = x(n - i)
\]

They satisfy

\[
w_i(n) = w_{i-1}(n-1), \quad i = 1, 2, \ldots, M
\]

Indeed, \( w_{i-1}(n-1) = x((n-1)-(i-1)) = x(n-i) = w_i(n) \). Therefore, at the next time instant:

\[
w_i(n+1) = w_{i-1}(n), \quad i = 1, 2, \ldots, M
\]

It follows that Eq. (4.3.5) can be written as

\[
y(n) = h_0 w_0(n) + h_1 w_1(n) + \cdots + h_M w_M(n)
\]
Thus, the FIR filter Eq. (4.3.5) is described by the following system:

\[
\begin{align*}
    w_0(n) &= x(n) \\
    y(n) &= h_0w_0(n) + h_1w_1(n) + \cdots + h_Mw_M(n) \\
    w_i(n+1) &= w_{i-1}(n), \quad \text{for } i = M, M-1, \ldots, 1
\end{align*}
\]

(4.3.14)

with corresponding sample processing algorithm:

\[
\begin{align*}
    \text{for each input sample } x \text{ do:} \\
    w_0 &= x \\
    y &= h_0w_0 + h_1w_1 + \cdots + h_Mw_M \\
    \text{for } i = M, M-1, \ldots, 1 \text{ do:} \\
    w_i &= w_{i-1}
\end{align*}
\]

(4.3.15)

Fig. 4.3.9 shows the corresponding direct form realization.

Example 4.3.1: Determine the sample processing algorithm of Example 4.1.1, which had filter and input

\[ h = [1, 2, -1, 1], \quad x = [1, 1, 2, 1, 2, 1, 1] \]

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:

\[
\begin{align*}
    w_0(n) &= x(n) \\
    y(n) &= w_0(n) + 2w_1(n) - w_2(n) + w_3(n) \\
    w_3(n+1) &= w_2(n) \\
    w_2(n+1) &= w_1(n) \\
    w_1(n+1) &= w_0(n)
\end{align*}
\]

(4.3.16)

The corresponding block diagram realization and sample processing algorithm are shown below:

\[
\begin{align*}
    x &= y \\
    w_0 &= w_0 \\
    w_1 &= w_1 \\
    w_2 &= w_2 \\
    w_3 &= w_3 \\
    \text{for each input sample } x \text{ do:} \\
    w_0 &= x \\
    y &= w_0 + 2w_1 - w_2 + w_3 \\
    w_3 &= w_2 \\
    w_2 &= w_1 \\
    w_1 &= w_0
\end{align*}
\]

The sample processing algorithm generates the following output samples:

\[
\begin{array}{c|cccc}
  n & x & w_0 & w_1 & w_2 & w_3 & y \\
  \hline
  0 & 1 & 1 & 0 & 0 & 0 & 1 \\
  1 & 1 & 1 & 0 & 0 & 3 \\
  2 & 2 & 1 & 1 & 0 & 3 \\
  3 & 1 & 1 & 2 & 1 & 5 \\
  4 & 2 & 1 & 2 & 1 & 3 \\
  5 & 2 & 2 & 1 & 2 & 7 \\
  6 & 1 & 1 & 2 & 2 & 4 \\
  7 & 1 & 1 & 2 & 2 & 3 \\
  8 & 0 & 0 & 1 & 1 & 3 \\
  9 & 0 & 0 & 0 & 1 & 0 \\
  10 & 0 & 0 & 0 & 0 & 1 \\
\end{array}
\]

The first three outputs correspond to the input-on transients (the internal delay registers are still filling up). The period \( 3 \leq n \leq 7 \) corresponds to steady state (all delays are filled). The last three outputs—in general, the last \( M \) outputs for an \( M \)th 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.

\[
\begin{align*}
    /* firexmpl.c - Example of FIR sample processing algorithm */ \\
    \#include <stdio.h> \\
    \#include <stdlib.h>
\end{align*}
\]

This declares calloc...
**4.3. SAMPLE PROCESSING METHODS**

```c
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);
        printf("%lf\n", y);
    }

    for (n=8; n<11; n++) { // input-off transients
        y = filter(0.0, w);
        printf("%lf\n", y);
    }
}

double filter(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 $x = [1,1,2,1,2,2,1,1]$, compute the output using the sample processing algorithm.

**Solution:** Because the filter has order $M = 4$, we define the following internal states:

- $w_0(n) = x(n)$
- $w_1(n) = x(n-1) = w_0(n-1)$
- $w_2(n) = x(n-2) = w_1(n-1)$
- $w_3(n) = x(n-3) = w_2(n-1)$
- $w_4(n) = x(n-4) = w_3(n-1)$

Then, the given I/O equation together with the state-updating equations will read:

- $y(n) = w_0(n) - w_4(n)$
- $w_4(n+1) = w_3(n)$
- $w_3(n+1) = w_2(n)$
- $w_2(n+1) = w_1(n)$
- $w_1(n+1) = w_0(n)$

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.

<table>
<thead>
<tr>
<th>n</th>
<th>x</th>
<th>$w_0$</th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
<th>$w_4$</th>
<th>$y = w_0 - w_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>-2</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>-2</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>
4.3. SAMPLE PROCESSING METHODS

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 firxmpl.c can be used to implement this example. The function \texttt{filter} will be in this case:

```c
double filter(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 \texttt{w} must be allocated as a 5-dimensional array in the main program.

4.3.3 Programming Considerations

The following C routine \texttt{fir.c} is an implementation of the sample processing algorithm Eq. (4.3.15):

```c
/* fir.c - FIR filter in direct form */

#include <stdio.h>
#include <stdlib.h>

doublefir(M, h, w, x)
{
    double y;
    int i;
    double h[M+1], w[M+1], x;
    w[M] = x; // read current input sample
    for (i=0; i<M; i++) // compute current output sample
        y += h[i] * w[i];
    return y;
}
```

It is the generalization of the example routine \texttt{filter} to the \(M\)th order case. It is patterned after the Fortran and C routines in [45]. The routine returns the computed output sample into a \texttt{double}, so that its typical usage will be of the form:

```c
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:

```c
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 \texttt{calloc} initializes \texttt{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 \(x\).\texttt{dat}, and the output samples are written one at a time into the output file \(y\).\texttt{dat}, as they are computed.

```c
FILE *fpf, *fpy;

fpf = fopen("x.dat", "r"); // input file
fpy = fopen("y.dat", "w"); // output file

while(fscanf(fpf, "%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
}
```

Filtering stops as soon as the end of file of \(x\).\texttt{dat} is detected and then the input-off transients are computed by making \(M\) additional calls to \texttt{fir} with zero input.

The \texttt{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

\[
y = \frac{1}{2} \sum_{i=0}^{M} h_i w_i
\]

and can be implemented by the following routine \texttt{dot.c}:

```c
/* dot.c - dot product of two length-(M+1) vectors */

#include <stdio.h>

double dot(M, h, w)
{
    double y;
    int i;
    double h[M+1], w[M+1];

    for (i=0; i<M; i++) // compute dot product
        y += h[i] * w[i];
    return y;
}
```

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:

```c
double *h, *w;

h = (double *) calloc(M+1, sizeof(double)); // (M+1)-dimensional
w = (double *) calloc(M+1, sizeof(double)); // (M+1)-dimensional
```
4.3. SAMPLE PROCESSING METHODS

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:

```c
/* 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;
{
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:

\[
\text{for each input sample } x \text{ do:}
\]
\[
\begin{align*}
  w_0 &= x \\
  y &= \text{dot}(M, h, w) \\
  \text{delay}(M, w)
\end{align*}
\]

(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 \( y = h^T w \), 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:

\[
T_{\text{instr}} = 30-80 \text{ nanoseconds}
\]

(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:

\[
\text{for each input sample } x \text{ do:}
\]
\[
\begin{align*}
  w_0 &= x \\
  y &= h_3 w_3 \\
  w_3 &= w_2 \\
  y &= y + h_2 w_2 \\
  w_2 &= w_1 \\
  y &= y + h_1 w_1 \\
  w_1 &= w_0 \\
  y &= y + h_0 w_0
\end{align*}
\]

This works because once the multiplication \( h_3 w_3 \) is performed, the current content of \( w_3 \) is no longer needed and can be updated to the next time instant. Similarly, once \( h_2 w_2 \) 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:

\[
\text{for each input sample } x \text{ do:}
\]
\[
\begin{align*}
  w_0 &= x \\
  y &= h_M w_M \\
  \text{for } i = M-1, \ldots, 1, 0 \text{ do:}
  \begin{align*}
    w_{i+1} &= w_i \\
    y &= y + h_i w_i
  \end{align*}
\end{align*}
\]

(4.3.18)

In earlier generations of DSP chips, the two operations:

\[
\begin{align*}
  w_{i+1} &= w_i \\
  y &= y + h_i w_i
\end{align*}
\]
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 $T_{\text{instr}}$ per filter tap, or, for an $M$th order filter:

$$T_{\text{proc}} = (M + 1)T_{\text{instr}} \quad (4.3.19)$$

As discussed in Chapter 1, this imposes a maximum limit on the allowed sampling rate for the application:

$$T \geq T_{\text{proc}} \Rightarrow f_s \leq \frac{1}{T_{\text{proc}}} \quad (4.3.20)$$

**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)

$$T = (M + 1)T_{\text{instr}} \Rightarrow M + 1 = \frac{T}{T_{\text{instr}}} = \frac{1}{f_s f_{\text{instr}}}$$

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

$$M + 1 = \frac{1}{f_s f_{\text{instr}}} = \frac{20 \cdot 10^6}{44.1 \cdot 10^3} = 453$$

This filter length is quite sufficient to implement several digital audio algorithms. \(\square\)

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:

```c
/* 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;
    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 $\{w_0, w_1, \ldots, w_M\}$. 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 $M = 3$.

![Wrapped linear delay-line buffer](image)

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 clock-
4. FIR FILTERING AND CONVOLUTION

4.3. SAMPLE PROCESSING METHODS

w 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,

\[
\mathbf{s}(n) = \begin{bmatrix} x_n \\ x_{n-1} \\ x_{n-2} \\ \vdots \\ x_{n-M} \end{bmatrix}, \quad \mathbf{s}(n+1) = \begin{bmatrix} x_{n+1} \\ x_n \\ x_{n-1} \\ \vdots \\ x_{n-M+1} \end{bmatrix} \tag{4.3.21}
\]

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, \( \ast p = x \), 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:

\[
w \leq p \leq w + M
\]

and therefore, it will always point at some w-register, say \( w[q] \),

\[
p = w + q \Rightarrow \ast p = p[0] = w[q]
\]

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 \( q \):

\[
0 \leq q \leq M
\]

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 \( q \):

\[
q = 0, 3, 2, 1, 0, 3, 2, 1, \ldots
\]

and in general, \( q \) cycles over the values: \( q = 0, M, M-1, \ldots \).

At each time instant, the pointer \( p \), or equivalently the offset \( q \), defines the vector \( \mathbf{s} \) of internal states. The sequence of pointers \( p, p+1, \ldots, p+M \), point at the components of the state vector, that is,

\[
s_i = p[i] = \ast (p+i) = \ast (w + q + i) = w[q+i], \quad i = 0, 1, \ldots, M
\]

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 \( q+i \leq M \). 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:

\[
s_i = w[(q+i)\%(M+1)] = w[(p-w+i)\%(M+1)]
\]

for \( i = 0, 1, \ldots, M \), where we solved Eq. (4.3.23) for \( q = p-w \).\(^1\) In particular, note that the first component of the state vector is:

\[
s_0 = w[q] = p[0] = \ast p = \ast (w + q)
\]

and the last component, corresponding to \( i = M \):

\[
s_M = w[(q+M)\%(M+1)] = w[(p-w+M)\%(M+1)]
\]

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 \( s_0 = w[q] = \ast p \), that is, \( \ast p = x \), the corresponding output sample \( y \) will be computed by the dot product:

\[
y = \sum_{i=0}^{M} h_i s_i = [h_0, h_1, \ldots, h_M] \begin{bmatrix} s_0 \\ s_1 \\ \vdots \\ s_M \end{bmatrix} = \mathbf{h}^T \mathbf{s}
\]

\(^1\)Some C compilers may require the cast: \( q = (\text{int}) (p-w) \).
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 \( i = 0, 1, 2, 3 \):

\[
q + i = 2 + i = 2, 3, 4, 5 \mod 4 = 2, 3, 0, 1
\]

![Fig. 4.3.14](image)

Internal states defined by circular pointers \( p + i \), \( i = 0, 1, \ldots, M \).

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 \( x_0 \) at the successive time instants \( n = 0, 1, \ldots, 7 \). An equivalent view is shown in Fig. 4.3.15. The \( w_i \) columns show the contents of the array \( w = [w_0, w_1, w_2, w_3] \) 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 \( w \) 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:

<table>
<thead>
<tr>
<th>( n )</th>
<th>( q )</th>
<th>( w_0 )</th>
<th>( w_1 )</th>
<th>( w_2 )</th>
<th>( w_3 )</th>
<th>( s_0 )</th>
<th>( s_1 )</th>
<th>( s_2 )</th>
<th>( s_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(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 \( x_0, x_{n-1}, x_{n-2}, x_{n-3} \). Therefore, the state vector \( s(n) \) at each time instant \( n \) is generated correctly according to Eq. (4.3.21).

![Fig. 4.3.15](image)

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 \( \text{wrap.c} \), 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 \( p = (w + M + 1) - (M + 1) = w \).

```c
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
    else
        \*p = (\*p) % M;  // 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:

```c
double \*w, \*p;
then \( p \) must be passed into \( \text{wrap.c} \) by its address, that is,
```

\( \text{wrap}(M, w, \&p) \);

With the help of the routine \( \text{wrap.c} \), an FIR filter can be implemented using a circular delay-line buffer by the following routine \( \text{cfir.c} \), which replaces \( \text{fir.c} \):
/* cfir.c - FIR filter implemented with circular delay-line buffer */

void wrap();

double cfir(M, h, w, p, x) double *h, *w, **p, x; int M;  
# {  
    int i;
    double y;
    **p = x;

    for (y=0, i=0; i<=M; i++) { 
        compute output sample y
        h += (*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 \( T_{proc} = (M+1)T_{instr} \).

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.

\[
\begin{align*}
\text{double } & h, \ w, \ *p; \\
\text{h} = (\text{double }*) \text{calloc(M+1, sizeof(double))}; \\
\text{w} = (\text{double }*) \text{calloc(M+1, sizeof(double))};  \\
& \text{also, initializes w to zero} \\
\text{p} = w; \\
& \text{initialize p} \\
\text{for } (n = 0; n < \text{Ntot}; n++) \\
&\quad y[n] = \text{cfir}(M, h, w, &p, x[n]);  \\
&\quad p \text{ passed by address}
\end{align*}
\]

The routine cfir mimics 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;

    for (y=0, h+=M, i=M; i>=0; i--) { 
        compute output sample y
        h += (*h--) * (*(*p)--);
        wrap(M, w, p);
    }

    (*p)++;  
    update circular delay line
    wrap(M, w, p);

    return y;
}  

\[
\begin{align*}
\text{double } & h, \ w, \ *p; \\
\text{h} = (\text{double }*) \text{calloc(M+1, sizeof(double))}; \\
\text{w} = (\text{double }*) \text{calloc(M+1, sizeof(double))};  \\
& \text{also, initializes w to zero} \\
\text{p} = w; \\
& \text{initialize p} \\
\text{for } (n = 0; n < \text{Ntot}; n++) \\
&\quad y[n] = \text{cfir1}(M, h, w, &p, x[n]);  \\
&\quad p \text{ passed by address}
\end{align*}
\]

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, DPS32C, DPS56K, 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 w0
ZAC; zero accumulator, y = 0
LT W3; load w3
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; $w_2 w_1$, load and shift
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 $w_1$ 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 $w_3$, $h_2$. 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 $w_1$ pointed to by $r3$, then post-increments the pointers.

In the fourth line, $r5$ points to the input data, that is, $w_1 = x$. 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.

```
c1r a
rep #M
mac x0,y0,a
macr x0,y0,a
```

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 c1r 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 $i$th 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 $w_{-1} M$. 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 IN Y2, R3; read new input sample x
STF Y3, *AR1++%; put x in w0 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=%, RO; $h_3 w_0$, and in parallel
|| ADDF3 RO, R2, R2; accumulate previous $R_0$ into $R_2$
ADDFF R0, 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 $h_0$, and the pointer $AR1$ points to the bottom of the $w$-buffer, that is, to $w_0$ which receives the current input sample $x_0$ 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 $i = M, \ldots, 1, 0$. 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 $(M+1)+3$ 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:
4.3. SAMPLE PROCESSING METHODS

4. FIR FILTERING AND CONVOLUTION

/* cdelay.c - circular buffer implementation of D-fold delay */

#include <stdio.h>

double wrap(int M, int q)

int M, *q;

{ /* p = w */
  for (n = 0; n < Ntot; n++) {
    write output
    *q = x[n];
    cdelay(D, w, &p);
  }
}

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++) {
  write output
  y[n] = w[(p-w+i)%((D+1));
  read input; equivalently, p[0] = x[n]
  cdelay(D, w, &p);
}

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, s = w, and its components are directly accessible as s_i = w[i], for i = 0, 1, ..., 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(int D, int w, int p, int i)

type: s_i = w[i], for i = 0, 1, ..., D

int D, *w, *p;

{ /* p = w */
  for (n = 0; n < Ntot; n++) {
    y[n] = tap(D, w, p, i);
    *p = x[n];
    cdelay(D, w, &p);
  }
}

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);
  *p = x[n];
  cdelay(D, w, &p);
}

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 * 44100 = 44100 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 p pointer. 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 ≤ q ≤ M, by wrapping it modulo-(M+1).

/* wrap2.c - circular wrap of pointer offset q, relative to array w */

#include <stdio.h>

int wrap2(int M, int q)

int M, *q;

{ /* p = w */
  for (n = 0; n < Ntot; n++) {
    write output
    *q = x[n];
    cdelay(D, w, &p);
  }
}

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);
  *p = x[n];
  cdelay(D, w, &p);
}

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 * 44100 = 44100 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 p pointer. 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 ≤ q ≤ M, by wrapping it modulo-(M+1).

/* wrap2.c - circular wrap of pointer offset q, relative to array w */

#include <stdio.h>

int wrap2(int M, int q)

int M, *q;

{ /* p = w */
  for (n = 0; n < Ntot; n++) {
    write output
    *q = x[n];
    cdelay(D, w, &p);
  }
}

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);
  *p = x[n];
  cdelay(D, w, &p);
}

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 * 44100 = 44100 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 p pointer. 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 ≤ q ≤ M, by wrapping it modulo-(M+1).

/* wrap2.c - circular wrap of pointer offset q, relative to array w */

#include <stdio.h>

int wrap2(int M, int q)

int M, *q;

{ /* p = w */
  for (n = 0; n < Ntot; n++) {
    write output
    *q = x[n];
    cdelay(D, w, &p);
  }
}
4.3. SAMPLE PROCESSING METHODS

183

4. FIR FILTERING AND CONVOLUTION

/* tap2.c - i-th tap of circular delay-line buffer */

double tap2(D, w, q, i) double *w;
int D, q, i;
{ i = 0,1,...,D
  return w[(q + i) % (D + 1)];
}

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):

Example 4.3.6: The circular buffer implementation of Example 4.3.1 is as follows:

where w is to be declared as a 4-dimensional array and initialized to zero. In terms of
the variable q, we have
4.4 PROBLEMS

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:

\[ s_i = \text{tap}(3,w,p,i) = w[(p-w+i) \mod 4] = w[i] \]

and the algorithm becomes the conventional one:

```
for each input sample x do:
    w_i = x
    y = w_0 + 2w_1 - w_2 + w_3
    delay(3,w)
```

where `delay` was replaced by `delay`.

4.4 Problems

4.1 Compute the convolution, \( y = h \ast x \), of the filter and input,

\[ h = [1, 1, 2, 1], \quad x = [1, 2, 1, 2, 1, 1, 1] \]

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:

\[ h = [2, -2, -1, 1], \quad x = [2, 2, 0, 1, -1, 0, 1, 2] \]

4.3 The impulse response \( h(n) \) of a filter is nonzero over the index range \( 3 \leq n \leq 6 \). The input signal \( x(n) \) to this filter is nonzero over the index range \( 10 \leq n \leq 20 \). 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 \( 0 < a < 1 \). 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: \( 0 \leq n \leq L - 1 \) and \( n \geq L \).

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: \( 0 \leq n \leq L - 1 \) and \( n \geq L \).

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,

\[ y_n = \sum_{i,j} h_{i,j} x_{n-i} \]

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:

\[ \text{blkfilt} \ h.h \text{.dat} \ L \ < \ x.dat \ > \ y.dat \]

It must have as command-line inputs a file of impulse response coefficients \( h.h \) (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 `blockconv.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:

\[ \text{firfilt} \ h.h < x.dat > y.dat \]

It must read and dynamically allocate the impulse response vector \( h \) from an input file, say `h.dat`, and must allocate the internal state vector \( w \). 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 \( h = h_1 * h_2 \) can be implemented by:
4.4. PROBLEMS

```
firfilt h1.dat | firfilt h2.dat < x.dat > y.dat
```

Alternatively or additionally, write a MATLAB version, say `firfilt.m`, with usage:

```matlab
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 4.1-4.1.10. It must produce identical results as the `firfilt` program. Rewrite versions of this program that use the alternative 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 `cdelayfilt.c`, that implements a plain delay by up to `D` samples, that is, `y(n) = x(n - i)`, `i = 0, 1, ..., D`, and has usage:

```
cdelayfilt 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 `delay.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, ..., 5`. Then, write another version of this program that uses the routines `delay2.c` 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 ≥ 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 `x = [1, 1, 2, 2, 4, ...]`. Using `convolution`, compute the first five output samples, `y(n), n = 0, 1, ..., 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 `x = [1, 1, 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 `w = [w0, w1, w2]` and a pointer `p` circulating over `w`.

Make a table of the numerical values of the contents of the array `w` for the successive time instants `0 ≤ n ≤ 10`. In each row, indicate that array element `wi`, 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 `h = [h0, h1, h2, h3]`. 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` for each time instant `0 ≤ n ≤ 10`.

4.21 Computer Experiment: Transposed FIR Realization. Generalize the block diagram of Fig. 4.4.1 to an arbitrary `M`th order filter with impulse response `h = [h0, h1, ..., hM]`. Write the corresponding sample processing algorithm and translate it into a C routine `firtr.c` that has usage:

```c
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 `cfirfilt.c` of Problem 4.10 that uses the routine `firtr.c` instead of `fir.c`. Test your program on Example 4.3.1.
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:

$$X(z) = \sum_{n=-\infty}^{\infty} x(n)z^{-n}$$

(z-transform) (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:

$$H(z) = \sum_{n=-\infty}^{\infty} h(n)z^{-n}$$

(transfer function) (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) $h = [h_0, h_1, h_2, h_3] = [2, 3, 5, 2]$

(b) $h = [h_0, h_1, h_2, h_3, h_4] = [1, 0, 0, 0, -1]$
5. Z-TRANSFORMS

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

\[
X(z) = a_1X_1(z) + a_2X_2(z)
\]

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,

\[
x(n) \xrightarrow{Z} X(z) \quad \Rightarrow \quad x(n-D) \xrightarrow{Z} z^{-D}X(z)
\]

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:

\[
y(n) = h(n) \ast x(n) \quad \Rightarrow \quad Y(z) = H(z)X(z)
\]

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 \( n \):

(a) \( h(n) = 2\delta(n) + 3\delta(n-1) + 5\delta(n-2) + 2\delta(n-3) \), (b) \( h(n) = \delta(n) - \delta(n-4) \)

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

\[
\delta(n-1) \xrightarrow{z^{-1}} z^{-1}, \quad \delta(n-2) \xrightarrow{z^{-2}} z^{-2}, \quad \delta(n-3) \xrightarrow{z^{-3}} z^{-3}, \quad \text{etc.}
\]

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:

(a) \( x(n) = u(n) \), (b) \( x(n) = -u(-n-1) \)

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

\[
x(n) - x(n-1) \xrightarrow{Z} X(z) - z^{-1}X(z) = 1 \quad \Rightarrow \quad X(z) = \frac{1}{1 - z^{-1}}
\]

Similarly, for case (b) we have the difference equation:

\[
x(n) - x(n-1) = -u(-n-1) + u(-(n-1)-1) = u(-n) - u(-n-1) = \delta(-n)
\]

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

\[
x(n) - x(n-1) \xrightarrow{Z} X(z) - z^{-1}X(z) = 1 \quad \Rightarrow \quad X(z) = \frac{1}{1 - z^{-1}}
\]

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

\[
h = [1, 2, -1, 1], \quad x = [1, 1, 2, 1, 2, 1, 1]
\]

have z-transforms:

\[
H(z) = 1 + 2z^{-1} - z^{-2} + z^{-3}
\]

\[
X(z) = 1 + z^{-1} + 2z^{-2} + z^{-3} + 2z^{-4} + 2z^{-5} + z^{-6} + z^{-7}
\]
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:
\[
y = h \ast x = [1, 3, 3, 5, 3, 7, 4, 3, 3, 0, 1]
\]
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 \( C \) for which the series (5.1.1) converges, that is,

\[
\text{Region of Convergence} = \{ z \in C \mid X(z) = \sum_{n=-\infty}^{\infty} x(n)z^{-n} \neq \infty \}
\]

The ROC is an important concept in many respects: It allows the unique inversion of the \( z \)-transform, as well as 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:
\[
x(n) = (0.5)^n u(n) = \{1, 0.5, 0.5^2, 0.5^3, \ldots \}
\]

Its \( z \)-transform will be:
\[
X(z) = \sum_{n=-\infty}^{\infty} (0.5)^n u(n)z^{-n} = \sum_{n=0}^{\infty} (0.5)^n z^{-n} = \sum_{n=0}^{\infty} \frac{(0.5z^{-1})^n}{1 - 0.5z^{-1}}
\]

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:
\[
1 + x + x^2 + x^3 + \cdots = \sum_{n=0}^{\infty} x^n = \frac{1}{1-x}
\]

which is valid only for \( |x| < 1 \) and diverges otherwise. Setting \( x = 0.5z^{-1} \) we find the sum:
\[
X(z) = \sum_{n=0}^{\infty} (0.5z^{-1})^n = \sum_{n=0}^{\infty} x^n = \frac{1}{1-x}, \quad \text{or,}
\]
\[
X(z) = \frac{1}{1-0.5z^{-1}} = \frac{z}{z - 0.5}
\]

where the convergence of the geometric series requires
\[
|x| = |0.5z^{-1}| < 1 \Rightarrow |z| > 0.5
\]

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:

\[
\text{ROC} = \{ z \in C \mid |z| > 0.5 \}
\]

Note, that the \( z \)-transform has a **pole** at \( z = 0.5 \). In summary, we have
\[
(0.5)^n u(n) \rightarrow \frac{z}{1-0.5z^{-1}}, \quad \text{with} \quad |z| > 0.5
\]

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:
\[
X(z) = -\sum_{n=-\infty}^{-1} (0.5)^n z^{-n} = -\sum_{n=0}^{\infty} (0.5)^{-1}z^{-n} = -\sum_{m=1}^{\infty} (0.5)^{-1}z^{-m}
\]

where we changed summation variables from \( n \) to \( m = -n \). To sum it, we use the following variant of the infinite geometric series:
\[
x + x^2 + x^3 + \cdots = \sum_{m=1}^{\infty} x^m = \frac{x}{1-x}
\]

which is valid only for \( |x| < 1 \) and diverges otherwise. Setting \( x = (0.5)^{-1}z \), we have
\[
X(z) = -\sum_{m=1}^{\infty} (0.5)^{-1}z^m = -\sum_{m=1}^{\infty} x^m = \frac{x}{1-x} = \frac{0.5^{-1}z}{1-0.5^{-1}z}, \quad \text{or,}
\]
\[
X(z) = \frac{z}{z-0.5} = \frac{1}{1-0.5z^{-1}}
\]
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

\[ |x| = |0.5^{-1}z| < 1 \Rightarrow |z| < 0.5 \]

which is the set of \( z \)'s that lie strictly inside the circle of radius 0.5, as shown below:

\[
\text{ROC} = \{ z \in \mathbb{C} \mid |z| < 0.5 \}
\]

To summarize, we have determined the \( z \)-transforms:

\[
(0.5)^nu(n) \xrightarrow{Z} \frac{1}{1 - 0.5z^{-1}}, \quad \text{with } |z| > 0.5
\]

\[
-(0.5)^nu(-n - 1) \xrightarrow{Z} \frac{1}{1 - 0.5z^{-1}}, \quad \text{with } |z| < 0.5
\]

The two signals have the same \( z \)-transform but completely disjoint ROCs. More generally, we have the result:

\[
d^nu(n) \xrightarrow{Z} \frac{1}{1 - az^{-1}}, \quad \text{with } |z| > |a|
\]

\[
-a^nu(-n - 1) \xrightarrow{Z} \frac{1}{1 - az^{-1}}, \quad \text{with } |z| < |a|
\]

(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:

\[
u(n) \xrightarrow{Z} \frac{1}{1 - z^{-1}}, \quad \text{with } |z| > 1
\]

\[
-u(-n - 1) \xrightarrow{Z} \frac{1}{1 - z^{-1}}, \quad \text{with } |z| < 1
\]

\[
(-1)^nu(n) \xrightarrow{Z} \frac{1}{1 + z^{-1}}, \quad \text{with } |z| > 1
\]

\[
-(1)^nu(-n - 1) \xrightarrow{Z} \frac{1}{1 + z^{-1}}, \quad \text{with } |z| < 1
\]

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. \( x(n) = (-0.8)^nu(n) \)
3. \( x(n) = (-0.8)^nu(n) - (0.8)^nu(n - 10) \)
4. \( x(n) = 0.5u(n) + (-1)^nu(n) = \{1, 0, 1, 0, 1, 0, 1, 0, 1, 0, \ldots \} \)
5. \( x(n) = 0.5((-0.8)^nu(n) + (0.8)^nu(n)) \)
6. \( x(n) = \cos\left(\frac{\pi n}{2}\right)u(n) = \{1, 0, -1, 0, 1, 0, -1, 0, 1, 0, \ldots \} \)
7. \( x(n) = 0.8\cos\left(\frac{\pi n}{2}\right)u(n) \)
8. \( x(n) = \frac{1}{2}\left[(0.8)^iu(n) + (-0.8)^iu(n)\right] \)
9. \( x(n) = \cos(\omega_0n)u(n) \) and \( x(n) = \sin(\omega_0n)u(n) \)
10. \( x(n) = \{1, 2, 3, 1, 2, 3, 1, 2, 3, \ldots \} \)

**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

\[
X(z) = \frac{1}{1 + 0.8z^{-1}}, \quad \text{with ROC: } |z| > 0 = 0.8
\]

For case (3), we write

\[
x(n) = (-0.8)^nu(n) - (-0.8)^{10}(-0.8)^{n-10}u(n - 10)
\]

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

\[
x(n) = a^n[u(n) - u(n - 10)] = \{1, a, a^2, a^3, a^4, a^5, a^6, a^7, a^8, 0, 0, 0, \ldots \}
\]
and therefore, its $z$-transform can be computed by the finite sum

$$X(z) = 1 + az^{-1} + a^2 z^{-2} + \cdots + a^n z^{-n}$$

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

$$X(z) = 1 + az^{-1} + a^2 z^{-2} + \cdots + a^n z^{-n} = \frac{1 - a^{N+1}}{1 - az^{-1}} = \frac{1 - (0.8) \cdot 10^{N+10}}{1 + 0.8 z^{-1}}$$

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,

$$X(z) = \frac{1}{1 - x} \bigg|_{x = z^{-2}} = \frac{1}{1 - z^{-2}}$$

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):

$$X(z) = \frac{1}{2} \left[ \frac{1}{1 - 0.8 z^{-1}} + \frac{1}{1 + 0.8 z^{-1}} \right] = \frac{1}{1 + 0.64 z^{-2}}$$

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):

$$x(n) = \cos \left( \frac{\pi n}{2} \right) u(n) = \frac{1}{2} \left[ e^{j \pi n/2} u(n) + e^{-j\pi n/2} u(n) \right] = \frac{1}{2} \left[ a^n u(n) + a^{-n} u(n) \right]$$

where $a = e^{j\pi/2} = j$ and $a^{-1} = -j$. 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

$$x(n) = (0.8)^n \cos \left( \frac{\pi n}{2} \right) u(n) = \frac{1}{2} \left[ (0.8)^n e^{j\pi n/2} u(n) + (0.8)^n e^{-j\pi n/2} u(n) \right]$$

which can be written as the signal of case (8):

$$x(n) = \frac{1}{2} \left[ (0.8)^n u(n) + (-0.8j)^n u(n) \right]$$

Thus, cases (7) and (8) are the same. Their $z$-transform is obtained using $a = \pm 0.8j$ in Eq. (5.2.3):

$$X(z) = \frac{1}{2} \left[ \frac{1}{1 - 0.8j z^{-1}} + \frac{1}{1 + 0.8j z^{-1}} \right] = \frac{1}{1 + 0.64 z^{-2}}$$

with ROC $|z| > 0.8$. The cosinewave in case (9) can be handled in a similar fashion. We write

$$\cos (\omega_0 n) u(n) = \frac{1}{2} \left[ e^{j\omega_0 n} + e^{-j\omega_0 n} \right] u(n) = \frac{1}{2} \left[ \frac{1}{1 - e^{j\omega_0 z^{-1}}} + \frac{1}{1 - e^{-j\omega_0 z^{-1}}} \right]$$

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

$$\sin (\omega_0 n) u(n) = \frac{1}{2j} \left[ e^{j\omega_0 n} - e^{-j\omega_0 n} \right] u(n) = \frac{1}{2j} \left[ \frac{1}{1 - e^{j\omega_0 z^{-1}}} - \frac{1}{1 - e^{-j\omega_0 z^{-1}}} \right]$$

which combines to give:

$$X(z) = \sin (\omega_0) \frac{z^{-1}}{1 - 2 \cos (\omega_0) z^{-1} + z^{-2}}$$

Finally, we consider case (10). Using the definition (5.1.1) and grouping the terms in groups of 3, we obtain:

$$X(z) = (1 + 2z^{-1} + 3z^{-2}) + (1 + 2z^{-1} + 3z^{-2}) z^{-3} +$$

$$+ (1 + 2z^{-1} + 3z^{-2}) z^{-6} + \left[ 1 + 2z^{-1} + 3z^{-2} \right] z^{-9} + \cdots$$

$$= (1 + 2z^{-1} + 3z^{-2}) \left[ 1 + z^{-3} + z^{-6} + z^{-9} + \cdots \right]$$

$$= 1 + 2z^{-1} + 3z^{-2}$$

$$\frac{1 - z^{-3}}{1 - z^{-3}}$$

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

$$x(n-3) = \{0, 0, 0, 1, 2, 3, 1, 2, 3, \ldots\}$$

and subtract it from $x(n)$ to get the difference equation

$$x(n) - x(n-3) = \{1, 2, 3, 0, 0, 0, 0, 0, 0, \ldots\} = \delta(n) + 2\delta(n-1) + 3\delta(n-2)$$

Then, taking $z$-transforms of both sides, we get

$$X(z) - z^{-3} X(z) = 1 + 2z^{-1} + 3z^{-2} \Rightarrow X(z) = \frac{1 + 2z^{-1} + 3z^{-2}}{1 - z^{-3}}$$

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,

\[
X(z) = \frac{1}{1 - 0.8z^{-1}} + \frac{1}{1 - 1.25z^{-1}} = 2 - 2.05z^{-1} + 0.25z^{-2}
\]

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.

\[
\text{Fig. 5.3.1 Causal and anticausal ROCs.}
\]

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—between 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

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

\[
x(n) = A_1 p_1^n u(n) + A_2 p_2^n u(n) + \cdots
\]

will have z-transform

\[
X(z) = \frac{A_1}{1 - p_1 z^{-1}} + \frac{A_2}{1 - p_2 z^{-1}} + \cdots
\]

with the restrictions \( |z| > |p_1|, |z| > |p_2| \), and so forth. Therefore, the common ROC of all the terms will be

\[
|z| > \max_i |p_i|
\]

that is, the outside of the circle defined by the pole of maximum magnitude. Similarly, if the signal is completely anticausal

\[
x(n) = -A_1 p_1^n u(-n - 1) - A_2 p_2^n u(-n - 1) - \cdots
\]

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:

\[
|z| < \min_i |p_i|
\]

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.
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,

$$X(z) = \frac{A_1}{1 - p_1 z^{-1}} + \frac{A_2}{1 - p_2 z^{-1}} + \frac{A_3}{1 - p_3 z^{-1}} + \frac{A_4}{1 - p_4 z^{-1}}$$

\[\text{stable/causal ROC}\] \hspace{1cm} \[\text{stable/anticausal ROC}\] \hspace{1cm} \[\text{stable/mixed ROC}\]

![Stable ROCs](image)

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

$$x(n) = [A_1 p_1^n + A_2 p_2^n + A_3 p_3^n + A_4 p_4^n] u(n)$$

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:

$$x(n) = [-A_1 p_1^n + A_2 p_2^n + A_3 p_3^n + A_4 p_4^n] u(-n - 1)$$

where $n$ 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

$$-A_1 p_1^n u(-n - 1) = -A_1 p_1^{-n} u(-n - 1) = -A_1 \left( \frac{1}{p_1} \right)^{|n|} u(-n - 1)$$

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 $|p_2| > |p_3| > 1$. Therefore, the stable signal will be

$$x(n) = [A_1 p_1^n + A_2 p_2^n] u(n) - [A_3 p_3^n + A_4 p_4^n] u(-n - 1)$$

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,

$$x(n) = (0.8)^n u(n) - (1.25)^{-n} u(-n - 1)$$

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 $\omega_0$

$$(\text{causal}) \quad x(n) = e^{j\omega_0 n} u(n)$$

$$(\text{anticausal}) \quad x(n) = -e^{j\omega_0 n} u(-n - 1)$$

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 $(-1)^n u(n)$ 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

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n) e^{-j\omega n} \quad \text{(DTFT)} \quad (5.4.1)$$

It is recognized as the evaluation of the $z$-transform on the unit circle, that is, at the $z$ points:

$$z = e^{j\omega} \quad (5.4.2)$$
Indeed, we have:

\[ X(z) \big|_{z=e^{j\omega}} = \sum_{n=-\infty}^{\infty} x(n) z^{-n} \big|_{z=e^{j\omega}} = \sum_{n=-\infty}^{\infty} x(n) e^{-jn\omega} = X(\omega) \]

The frequency response \( H(\omega) \) of a linear system \( h(n) \) with transfer function \( H(z) \) is defined in the same way, namely,

\[ H(\omega) = \sum_{n=-\infty}^{\infty} h(n) e^{-jn\omega} \]

(frequency response) \hfill (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 \( \omega \) is in units of \([\text{radians/sample}]\) and is related to the physical frequency \( f \) in Hz by

\[ \omega = \frac{2\pi f}{f_s} \] (digital frequency) \hfill (5.4.4)

The Nyquist interval \([-f_s/2, f_s/2]\) is the following interval in units of \( \omega \):

\[ -\pi \leq \omega \leq \pi \] (Nyquist interval) \hfill (5.4.5)

In Chapter 1, the quantity \( X(\omega) \) was denoted by

\[ \hat{X}(f) = \sum_{n=-\infty}^{\infty} x(nT) e^{-2jn\pi f / f_s} \]

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 \( \omega \), periodicity in \( f \) with period \( f_s \) becomes periodicity in \( \omega \) with period \( 2\pi \). 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:

\[ x(n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} X(\omega) e^{jn\omega} d\omega \] (inverse DTFT) \hfill (5.4.6)

It expresses \( x(n) \) as a linear combination of discrete-time sinusoids \( e^{jn\omega} \) 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

\[ x(n) = \frac{1}{f_s} \int_{-f_s/2}^{f_s/2} X(f) e^{2\pi jfn/f_s} df \]

As an example, consider a (double-sided) complex sinusoid of frequency \( \omega_0 \):

\[ x(n) = e^{j\omega_0 n}, \quad -\infty < n < \infty \]

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 \( 2\pi \). This is needed in order to make \( X(\omega) \) periodic with period \( 2\pi \). 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 \( \omega_0 \) 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:

\[ X(\omega) = 2\pi \delta(\omega - \omega_0), \quad -\pi \leq \omega \leq \pi \]

Therefore, Eq. \( (5.4.6) \) gives

\[ x(n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} X(\omega) e^{jn\omega} d\omega = \frac{1}{2\pi} \int_{-\pi}^{\pi} 2\pi \delta(\omega - \omega_0) e^{jn\omega} d\omega = e^{j\omega_0 n} \]

Similarly, for a linear combination of two sinusoids we have:

\[ x(n) = A_1 e^{j\omega_1 n} + A_2 e^{j\omega_2 n} \rightarrow X(\omega) = 2\pi A_1 \delta(\omega - \omega_1) + 2\pi A_2 \delta(\omega - \omega_2) \]

This can be verified in the same way, if we assume that both \( \omega_1 \) and \( \omega_2 \) lie in the Nyquist interval. In particular, for real-valued cosine and sine signals, we have:

\[ \cos(\omega_0 n) \rightarrow \pi \delta(\omega - \omega_0) + \pi \delta(\omega + \omega_0) \]
\[ \sin(\omega_0 n) \rightarrow -j\pi \delta(\omega - \omega_0) + j\pi \delta(\omega + \omega_0) \]

Another useful relationship is Parseval’s equation, which relates the total energy of a sequence to its spectrum:

\[ \sum_{n=-\infty}^{\infty} |x(n)|^2 = \frac{1}{2\pi} \int_{-\pi}^{\pi} |X(\omega)|^2 d\omega \] (Parseval) \hfill (5.4.7)
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 $\omega$ 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 $\omega$.

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

$$X(\omega) = \frac{1}{1 - e^{j\omega} e^{-j\omega}} = \frac{1}{1 - e^{j(\omega_0 - \omega)}}$$

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 $\omega_0$.

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 $z = z_1$.

$$X(z) = \frac{1 - z_1 z^{-1}}{1 - p_1 z^{-1}} = \frac{z - z_1}{z - p_1}$$

The corresponding spectrum and its magnitude are obtained by replacing $z$ by $e^{j\omega}$:

$$X(\omega) = \frac{e^{j\omega} - z_1}{e^{j\omega} - p_1} \Rightarrow |X(\omega)| = \frac{|e^{j\omega} - z_1|}{|e^{j\omega} - p_1|}$$

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, $|e^{j\omega} - p_1|$.

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 $\omega_1$ 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.](image)

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 \( \text{Re} \ X(\omega), \text{Im} \ X(\omega) \) or, in its polar form, by its magnitude and phase responses \( |X(\omega)|, \arg X(\omega) \). Thus,

\[
X(\omega) = \text{Re} \ X(\omega) + j \text{Im} X(\omega) = |X(\omega)| e^{j \arg X(\omega)}
\]

For real-valued signals \( x(n) \), the quantity \( X(\omega) \) satisfies the following so-called *hermitian* property:

\[
X(\omega)^* = X(-\omega)
\]

which translates to the following relationships for the magnitude and phase responses:

\[
|X(\omega)| = |X(-\omega)| \quad \text{arg} \ X(\omega) = - \text{arg} \ X(-\omega)
\]

(5.4.9)

that is, the magnitude response is even in \( \omega \) 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 \( Y(\omega) = H(\omega) X(\omega) \) evaluated on the unit circle takes the following frequency-domain form:

\[
Y(\omega) = H(\omega) X(\omega) \quad \text{(filtering in frequency domain)}
\]

(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:

\[
X(z) = \frac{1}{1 - 0.8 z^{-1}} + \frac{1}{1 - 1.25 z^{-1}}
\]

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 \( p_1, p_2, \ldots, p_M \), and \( D(z) \) may be assumed to be in the factored form:

\[
D(z) = (1 - p_1 z^{-1})(1 - p_2 z^{-1}) \cdots (1 - p_M z^{-1})
\]

The partial fraction expansion of \( X(z) \) is given by:

\[
X(z) = \frac{N(z)}{D(z)} = \frac{N(z)}{(1 - p_1 z^{-1})(1 - p_2 z^{-1}) \cdots (1 - p_M z^{-1})}
\]

The partial fraction expansion of \( X(z) \) is given by:

\[
X(z) = \frac{N(z)}{D(z)} = \frac{A_1}{1 - p_1 z^{-1}} + \frac{A_2}{1 - p_2 z^{-1}} + \cdots + \frac{A_M}{1 - p_M z^{-1}}
\]

(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:

\[
A_i = \left[ (1 - p_i z^{-1})X(z) \right]_{z = p_i}
\]

(5.5.2)

\[\text{We have assumed that all the poles are single poles.}\]
for \( i = 1, 2, \ldots, M \). 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

\[
X(z) = \frac{2 - 2.05 z^{-1}}{1 - 2.05 z^{-1} + z^{-2}} = \frac{2 - 2.05 z^{-1}}{(1 - 0.8 z^{-1})(1 - 1.25 z^{-1})}
\]

Because the numerator polynomial has degree one in the variable \( z^{-1} \), there is a PF expansion of the form:

\[
X(z) = \frac{2 - 2.05 z^{-1}}{(1 - 0.8 z^{-1})(1 - 1.25 z^{-1})} = \frac{A_1}{1 - 0.8 z^{-1}} + \frac{A_2}{1 - 1.25 z^{-1}}
\]

The two coefficients are obtained by Eq. (5.5.2) as follows:

\[
A_1 = [(1 - 0.8 z^{-1})X(z)]_{z=0.8} = \left[ \frac{2 - 2.05 z^{-1}}{1 - 1.25 z^{-1}} \right]_{z=0.8} = \frac{2 - 2.05}{1 - 1.25} = 1
\]

\[
A_2 = [(1 - 1.25 z^{-1})X(z)]_{z=1.25} = \left[ \frac{2 - 2.05 z^{-1}}{1 - 0.8 z^{-1}} \right]_{z=1.25} = 1
\]

which are as expected. \( \square \)

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:

\[
X(z) = \frac{N(z)}{D(z)} = \frac{N(z)}{(1 - p_1 z^{-1})(1 - p_2 z^{-1}) \cdots (1 - p_M z^{-1})} = A_0 + \frac{A_1}{1 - p_1 z^{-1}} + \frac{A_2}{1 - p_2 z^{-1}} + \cdots + \frac{A_M}{1 - p_M z^{-1}}
\]

(5.5.3)

The coefficients \( A_i, i = 1, 2, \ldots, M \) 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,

\[
A_0 = X(z) \bigg|_{z=0}
\]

(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 \textit{quotient} and \textit{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

\[
X(z) = \frac{2 - 2.05 z^{-1}}{(1 - 0.8 z^{-1})(1 - 1.25 z^{-1})} = \frac{z(2z - 2.05)}{(z - 0.8)(z - 1.25)}
\]

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

\[
X(z) = \frac{zA_1}{z - 0.8} + \frac{zA_2}{z - 1.25} = A_1 + \frac{A_2}{1 - 1.25 z^{-1}}
\]

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 \( z^{-1} \). \( \square \)

**Example 5.5.4:** Compute all possible inverse \( z \)-transforms of

\[
X(z) = \frac{6 + z^{-1}}{1 - 0.25 z^{-2}}
\]

**Solution:** Because the numerator has degree one in \( z^{-1} \), we have the PF expansion:

\[
X(z) = \frac{6 + z^{-1}}{1 - 0.25 z^{-2}} = \frac{6 + z^{-1}}{(1 - 0.5 z^{-1})(1 + 0.5 z^{-1})} = \frac{A_1}{1 - 0.5 z^{-1}} + \frac{A_2}{1 + 0.5 z^{-1}}
\]

where

\[
A_1 = \left[ \frac{6 + z^{-1}}{1 - 0.5 z^{-1}} \right]_{z=0.5} = 4, \quad A_2 = \left[ \frac{6 + z^{-1}}{1 + 0.5 z^{-1}} \right]_{z=-0.5} = 2
\]
The two poles at $\pm 0.5$ 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)

$$X(z) = \frac{10 + z^{-1} - z^{-2}}{1 - 0.25z^{-2}} = \frac{10 + z^{-1} - z^{-2}}{(1 - 0.5z^{-1})(1 + 0.5z^{-1})} = A_0 + \frac{A_1}{1 - 0.5z^{-1}} + \frac{A_2}{1 + 0.5z^{-1}}$$

where $A_1$ and $A_2$ are determined in the usual manner and $A_0$ is determined by evaluating $X(z)$ at $z = 0$:

$$A_0 = \left[\frac{10 + z^{-1} - z^{-2}}{1 - 0.25z^{-2}}\right]_{z=0} = \left[\frac{10z^2 + z - 1}{z^2 - 0.25}\right]_{z=0} = \frac{-1}{-0.25} = 4$$

$$A_1 = \left[\frac{10 + z^{-1} - z^{-2}}{1 + 0.5z^{-1}}\right]_{z=0.5} = 4, \quad A_2 = \left[\frac{10 + z^{-1} - z^{-2}}{1 - 0.5z^{-1}}\right]_{z=-0.5} = 2$$

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:

$$X(z) = (6 + z^{-5})W(z) = 6W(z) + z^{-3}W(z)$$

Taking inverse $z$-transforms of both sides and using the delay property, we find

$$x(n) = 6w(n) + w(n - 5) = 3(0.5)^n u(n) + 3(-0.5)^n u(n) + 0.5(0.5)^{n-5} u(n - 5) + 0.5(-0.5)^{n-5} u(n - 5)$$

The two expressions for $x(n)$ from the two techniques are equivalent.

Example 5.5.7: Determine all possible inverse $z$-transforms of

$$X(z) = \frac{7 - 9.5z^{-1} - 3.5z^{-2} + 5.5z^{-3}}{(1 - z^{-2})(1 - 0.5z^{-1})(1 - 1.5z^{-1})}$$
5.5. INVERSE Z-TRANSFORMS

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:

\[
x_1(n) = -[1 + (-1)^n + 3(0.5)^n + 2(1.5)^n]u(-n - 1)
\]

\[
x_2(n) = 3(0.5)^n u(n) - [1 + (-1)^n + 2(1.5)^n]u(-n - 1)
\]

\[
x_3(n) = [1 + (-1)^n + 3(0.5)^n]u(n) - 2(1.5)^n u(-n - 1)
\]

\[
x_4(n) = [1 + (-1)^n + 3(0.5)^n + 2(1.5)^n]u(n)
\]

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

\[(1.5)^n = (1.5)^{-|n|} = 0 \quad \text{as} \quad n \to -\infty\]

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

\[
X(z) = \frac{A_1}{1 - p_1 z^{-1}} + \frac{A_1^*}{1 - p_1^* z^{-1}} + \frac{A_2}{1 - p_2 z^{-1}} + \cdots
\]

where the PF expansion coefficients also come in complex-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

\[
x(n) = A_1 p_1^n u(n) + A_1^* p_1^{*n} u(n) + A_2 p_2^n u(n) + \cdots
\]

Because the first two terms are complex conjugates of each other, we may use the result that \(C + C^* = 2 \text{Re}(C)\), for any complex number \(C\), to write the first term as

\[
A_1 p_1^n + A_1^* p_1^{*n} = 2 \text{Re}[A_1 p_1^n]
\]

Writing \(A_1\) and \(p_1\) in their polar form, say, \(A_1 = B_1 e^{j\alpha_1}\) and \(p_1 = R_1 e^{j\omega_1}\), with \(B_1 > 0\) and \(R_1 > 0\), we have

\[
\text{Re}[A_1 p_1^n] = \text{Re}[B_1 e^{j\alpha_1} R_1^n e^{j\omega_1 n}] = B_1 R_1^n \text{Re}[e^{j\omega_1 n + j\alpha_1}]
\]

and taking the real part of the exponential, we find

\[
A_1 p_1^n + A_1^* p_1^{*n} = 2 \text{Re}[A_1 p_1^n] = 2 B_1 R_1^n \cos(\omega_1 n + \alpha_1)
\]

and for \(x(n)\)

\[
x(n) = 2 B_1 R_1^n \cos(\omega_1 n + \alpha_1) u(n) + A_2 p_2^n u(n) + \cdots
\]

Thus, complex-valued poles correspond to exponentially decaying sinusoids (if \(R_1 < 1\). The decay envelope \(R_1^n\) and the frequency \(\omega_1\) depend on the complex pole by \(p_1 = R_1 e^{j\omega_1}\).

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:

\[
\frac{A_1}{1 - p_1 z^{-1}} + \frac{A_1^*}{1 - p_1^* z^{-1}} = \frac{(A_1 + A_1^*) - (A_1 p_1^n + A_1^* p_1^{*n}) z^{-1}}{(1 - p_1 z^{-1})(1 - p_1^* z^{-1})}
\]

Using the identities

\[
(1 - p_1 z^{-1})(1 - p_1^* z^{-1}) = 1 - 2 \text{Re}(p_1) z^{-1} + |p_1|^2 z^{-2}
\]

and writing

\[
A_1 + A_1^* = 2 \text{Re}(A_1) = 2 B_1 \cos(\alpha_1)
\]

\[
A_1 p_1^n + A_1^* p_1^{*n} = 2 \text{Re}(A_1 p_1^n) = 2 B_1 R_1 \cos(\alpha_1 - \omega_1)
\]

we find

\[
\frac{A_1}{1 - p_1 z^{-1}} + \frac{A_1^*}{1 - p_1^* z^{-1}} = \frac{2 B_1 \cos(\alpha_1 - \omega_1) - 2 B_1 R_1 \cos(\alpha_1 - \omega_1) z^{-1}}{1 - 2 R_1 \cos(\omega_1) z^{-1} + R_1^2 z^{-2}}
\]

having real-valued coefficients.

Example 5.5.8: Determine all possible inverse z-transforms of

\[
X(z) = \frac{4 - 3z^{-1} + z^{-2}}{1 + 0.25z^{-2}}
\]
5.6 UNILATERAL Z-TRANSFORM

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:

\[ A_0 = \left[ \frac{4 - 3z^{-1} + z^{-2}}{1 + 0.25z^{-2}} \right]_{z=0} = 4, \quad A_1 = \left[ \frac{4 - 3z^{-1} + z^{-2}}{1 + 0.5jz^{-1}} \right]_{z=0} = 3j \]

Therefore,

\[ X(z) = 4 + \frac{3j}{1 - 0.5jz^{-1}} - \frac{3j}{1 + 0.5jz^{-1}} = 4 - \frac{3x^{-1}}{1 + 0.25x^{-2}} \]

The causal ROC is \(|z| > 0.5j\), resulting in

\[ x(n) = 4\delta(n) + 3j(0.5j)^nu(n) - 3j(-0.5j)^nu(n) \]

Because the last two terms are complex conjugates of each other, we may write them as

\[ x(n) = 4\delta(n) + 2Re\{3j(0.5j)^nu(n)\} = 4\delta(n) + 6(0.5)^nu(n)Re\{j^n\} \]

Writing \(j^n = e^{jn(n+1)/2}\) and taking real parts we find

\[ Re\{j^n\} = \cos\left(\frac{\pi(n+1)}{2}\right) = -\sin\left(\frac{n\pi}{2}\right) \]

and

\[ x(n) = 4\delta(n) - 6(0.5)^n\sin\left(\frac{n\pi}{2}\right)u(n) \]

Similarly, we find

\[ x(n) = 4\delta(n) + 6(0.5)^n\sin\left(\frac{n\pi}{2}\right)u(-n-1) \]

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.

\[ \square \]

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 \(n \geq 0\), that is,

\[ n = [\cdots -2 -1 0 1 2 3 \cdots] \]

\[ x_n = [\cdots x_{-2} x_{-1} x_0 x_1 x_2 x_3 \cdots] \]

\[ x_n^+ = [\cdots 0 0 x_0 x_1 x_2 x_3 \cdots] \] (5.6.1)

Similarly, the causal parts of the left-shifted time-advanced versions, \(x_{n+1}^+\) and \(x_{n+2}^+\), will be,

\[ n = [\cdots -2 -1 0 1 2 3 \cdots] \]

\[ x_{n+1} = [\cdots x_{-1} x_0 x_1 x_2 x_3 x_4 \cdots] \]

\[ x_{n+1}^+ = [\cdots 0 0 x_0 x_1 x_2 x_3 x_4 \cdots] \]

\[ x_{n+2} = [\cdots x_0 x_1 x_2 x_3 x_4 \cdots] \]

\[ x_{n+2}^+ = [\cdots 0 0 x_0 x_2 x_4 x_5 \cdots] \] (5.6.2)

And, the causal parts of the right-shifted time-delayed versions, \(x_n^-\) and \(x_{n-2}^-\), will be,

\[ n = [\cdots -2 -1 0 1 2 3 \cdots] \]

\[ x_{n-3} = [\cdots x_{-3} x_{-2} x_{-1} x_0 x_1 x_2 \cdots] \]

\[ x_{n-3}^- = [\cdots x_{-3} x_{-2} x_{-1} x_0 x_1 x_2 \cdots] \]

\[ x_{n-2}^- = [\cdots 0 0 x_0 x_2 x_4 x_5 \cdots] \]

\[ x_{n-2}^- = [\cdots 0 0 x_0 x_2 x_4 x_5 \cdots] \] (5.6.3)

Denoting by \(X^+(z)\), the z-transform of \(x_n^+\),

\[ X^+(z) = \sum_{n=0}^{\infty} x_n z^{-n} = x_0 + x_1 z^{-1} + z_2 z^{-2} + \cdots \] (unilateral z-transform) (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, \(x_{n+1}^+, x_{n+2}^-, \) will be related to \(X^+(z)\) via,

\[ x_n^+ \xrightarrow{z} X^+(z) \]

\[ x_{n+1}^+ \xrightarrow{z} z[X^+(z) - x_0] \]

\[ x_{n+2}^- \xrightarrow{z} z^2[X^+(z) - x_0 - z^{-1}x_1] \]

\[ x_{n-3}^- \xrightarrow{z} z^{-3}[X^+(z) + zx_{-1}] \]

\[ x_{n-2}^- \xrightarrow{z} z^{-2}[X^+(z) + z^2x_{-2} + zx_{-1}] \] (5.6.5)

for example, we see from Eq. (5.6.2) that,

\[ x_{n+1}^+ \xrightarrow{z} x_1 + z^{-1}x_2 + z^{-2}x_3 + \cdots = z[z^{-1}x_1 + z^{-2}x_2 + z^{-3}x_3 + \cdots] \]

\[ z\{x_0 + z^{-1}x_1 + z^{-2}x_2 + z^{-3}x_3 + \cdots\} - zx_0 = z X^+(z) - zx_0 \]

Such relationships are useful in solving difference equations with given initial condi-
tions. They are akin to the Laplace transform differentiation properties,
\[
x(t) \rightarrow L X(s)
\]
\[
\dot{x}(t) \rightarrow sX(s) - x(0^-)
\]
(5.6.6)

with the usual definition of the unilateral Laplace transform [597,598],

\[
x(t) \rightarrow L X(s) = \int_0^\infty x(t) e^{-st} dt
\]
(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:

\[
P = \frac{ry_0 (1 + r)^N}{(1 + r)^N - 1}
\]
(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, \(i_n = ry_n\), therefore, only the amount, \(b_n = P - ry_n\), is used to reduce the balance, so that the next balance will be:

\[
y_{n+1} = y_n - b_n = y_n - (P - ry_n) = (1 + r)y_n - P
\]

or, defining the quantity, \(a = 1 + r\), for convenience,

\[
y_{n+1} = ay_n - P, \quad n = 0, 1, 2, \ldots
\]
(5.6.9)

This recursion can be solved analytically, with solution,

\[
y_n = \left[y_0 - \frac{P}{r}\right] a^n + \frac{P}{r}, \quad n = 0, 1, 2, \ldots
\]
(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,

\[
y_n = \frac{a^n - a_0}{a^n - 1} y_0, \quad n = 0, 1, 2, \ldots, N
\]
(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 \(n = 0, 1, \ldots, N\), 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.)

<table>
<thead>
<tr>
<th>(n)</th>
<th>(y(n))</th>
<th>(i(n))</th>
<th>(b(n))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10000.00</td>
<td>50.00</td>
<td>143.33</td>
</tr>
<tr>
<td>1</td>
<td>9856.47</td>
<td>49.28</td>
<td>144.04</td>
</tr>
<tr>
<td>2</td>
<td>9712.63</td>
<td>48.56</td>
<td>144.76</td>
</tr>
<tr>
<td>3</td>
<td>9567.86</td>
<td>47.84</td>
<td>145.49</td>
</tr>
<tr>
<td>4</td>
<td>9422.37</td>
<td>47.11</td>
<td>146.22</td>
</tr>
<tr>
<td>57</td>
<td>574.23</td>
<td>2.87</td>
<td>190.46</td>
</tr>
<tr>
<td>58</td>
<td>383.78</td>
<td>1.92</td>
<td>191.41</td>
</tr>
<tr>
<td>59</td>
<td>192.37</td>
<td>0.96</td>
<td>192.37</td>
</tr>
<tr>
<td>60</td>
<td>0.00</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{balance_and_interest.png}
\caption{Balance and interest versus time.}
\end{figure}

**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)

\[
y'_n \rightarrow z Y'(z)
\]
\[
y_{n+1} \rightarrow z [Y'(z) - y_0]
\]

Then, Eq. (5.6.8) reads, in the time and \(z\) domains, where \(u(n)\) is the unit-step,

\[
y'^n = ay^n - Pu(n)
\]
\[
z [Y'(z) - y_0] = a Y'(z) - \frac{P}{1 - z^{-1}}
\]
5.6. UNILATERAL Z-TRANSFORM

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):

\[
y_n^* = y_0 a^n u(n) - \frac{p}{r} \left[ a^n u(n) - u(n) \right] = \left[ y_0 - \frac{p}{r} \right] a^n + \frac{p}{r}, \quad n = 0, 1, 2, \ldots
\]

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,

\[
\begin{align*}
y_1 &= ay_0 - p \\
y_2 &= ay_1 - P = a[ay_0 - P] - P = a^2 y_0 - P(1 + a) \\
y_3 &= ay_2 - P = a[a^2 y_0 - (1 + a)P] - P = a^3 y_0 - P(1 + a + a^2) \\
y_4 &= ay_3 - P = a[a^3 y_0 - (1 + a + a^2)P] - P = a^4 y_0 - P(1 + a + a^2 + a^3)
\end{align*}
\]

thus, by induction, we conclude that,

\[
y_n = a^n y_0 - P(1 + a + a^2 + \cdots + a^{n-1}) = a^n y_0 - P\frac{a^n - 1}{a - 1} = \left[ y_0 - \frac{p}{a - 1} \right] a^n + \frac{p}{a - 1}
\]

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,

\[
y_{n+1}^* = ay_n^* - Pu(n)
\]

can be expressed as the sum a particular solution and a homogeneous solution, the latter being the solution of the homogeneous equation,

\[
y_{n+1}^* = ay_n^* \Rightarrow y_{n+1}^* = y_{\text{homog}}(n) = c a^n
\]

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,

\[
y_c = ay_c - P \Rightarrow y_c = -\frac{P}{1 - a} = \frac{P}{r}
\]

It follows that the complete solution will be, for \( n \geq 0 \),

\[
y_n = y_{\text{homog}}(n) + y_{\text{part}}(n) = c a^n + \frac{P}{r}
\]

The constant \( c \) is fixed by requiring the initial condition, \( y_n = y_0 \) at \( n = 0 \), which gives,

\[
y_0 = c + \frac{P}{r} \Rightarrow c = y_0 - \frac{P}{r} \Rightarrow y_n = \left[ y_0 - \frac{P}{r} \right] a^n + \frac{P}{r}, \quad n \geq 0
\]

Note also that the total interest paid can be calculated by summing up the individual interests,

\[
I_{\text{tot}} = \sum_{n=0}^{N-1} r y_n
\]

Using Eq. (5.6.11), we find

\[
I_{\text{tot}} = r \sum_{n=0}^{N-1} y_0 a^n - r \sum_{n=0}^{N-1} \frac{P}{r} [a^n - 1] = NP - \left[ \frac{P}{r} - y_0 \right] r \sum_{n=0}^{N-1} a^n
\]

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,

\[
I_{\text{tot}} = \sum_{n=0}^{N-1} r y_n = NP - y_0
\]

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, \( y_n = 0 \).

\[\Box\]

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. \( x(n) = u(n - 5) \)

da. \( x(n) = u(n + 5) \)

5.3 Compute the z-transform of the following sequences and determine the corresponding region of convergence:

a. \( x(n) = (-0.5)^n u(n) \)

b. \( x(n) = (-0.5)^n [u(n) - u(n - 10)] \)

c. \( x(n) = (0.5)^n u(n) + (-0.5)^n u(n) \)
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. \( x(n) = (0.8j)^nu(n) + (-0.8j)^nu(n) \)

5.5 Compute the \( z \)-transform of the following sequences and determine the corresponding region of convergence:

a. \( x(n) = (0.25)^nu(n) + 4^nu(n) \)

b. \( x(n) = (0.25)^nu(n) - 4^n u(-n - 1) \)

c. \( x(n) = -(0.25)^nu(-n - 1) - 4^nu(-n - 1) \)

d. Explain why \( x(n) = -(0.25)^nu(-n - 1) + 4^n u(n) \) 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 sequences and determine the corresponding region of convergence:

\[
5.11 \text{Show the } z^{-}\text{transform of a triangular signal:}
\]

\[
X(z) = \left( \frac{1}{1 - z^{-1}} \right) \quad \text{ROC } |z| > 1
\]

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 \( X(z) = 1 - z^{-1}X(z) \).

5.12 Using Euler’s formula and the \( z \)-transforms of \( \cos(\omega_n)u(n) \) and \( \sin(\omega_n)u(n) \), derive the \( z \)-transforms of the signals \( x(n) = R^n \cos(\omega(n))u(n) \) and \( x(n) = R^n \sin(\omega(n))u(n) \).

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 \( X(z) = 1 - z^{-1}X(z) \).

5.16 Without using partial fractions, determine the causal inverse \( z \)-transforms of:

a. \( X(z) = \frac{1}{1 + z^{-2}} \)

b. \( X(z) = \frac{1}{1 - z^{-2}} \)

c. \( X(z) = \frac{1}{1 + z^{-4}} \)

d. \( X(z) = \frac{1}{1 - z^{-8}} \)

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:

\[
H(z) = Q(z) + \sum_{k=1}^{\infty} b_k z^{-k} + z^{-1} \frac{b_0}{1 + a_1 z^{-1} + a_2 z^{-2}}
\]

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:

\[
1 - z^{-D} = \prod_{k=0}^{D-1} (1 - z_k z^{-1})
\]

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:

\[
x(n) \xrightarrow{z} X(z) \Rightarrow a^n x(n) \xrightarrow{z} X(z/a)
\]

For \( a = e^{j\omega_0} \), show that in the frequency domain this property becomes:

\[
x(n) \rightarrow X(\omega) \Rightarrow e^{j\omega_0 n} x(n) \rightarrow X(\omega - \omega_0)
\]

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.

![Equivalent descriptions of digital filters](image)

*Fig. 6.1.1 Equivalent descriptions of digital filters.*
6.2. TRANSFER FUNCTIONS

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:

\[
H(z) = \frac{5 + 2z^{-1}}{1 - 0.8z^{-1}}
\]  
(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:

\[
A_0 = H(z) \bigg|_{z=0} = \frac{5 + 2z^{-1}}{1 - 0.8z^{-1}} \bigg|_{z=0} = \frac{5z + 2}{z - 0.8} \bigg|_{z=0} = \frac{2}{-0.8} = -2.5
\]

\[
A_1 = (1 - 0.8z^{-1})H(z) \bigg|_{z=0.8} = (5 + 2z^{-1}) \bigg|_{z=0.8} = 5 + 2/0.8 = 7.5
\]

Assuming the filter is causal, we find:

\[
h(n) = -2.5\delta(n) + 7.5(0.8)^nu(n)
\]  
(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 \( h(n) \):

\[
h(n) = 0.8h(n-1) + 5\delta(n) + 2\delta(n-1)
\]  
(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,

\[
y_n = h_0x_n + h_1x_{n-1} + h_2x_{n-2} + h_3x_{n-3} + \cdots
\]

\[
= 5x_n + 7.5\left[ (0.8)x_{n-1} + (0.8)^2x_{n-2} + (0.8)^3x_{n-3} + \cdots \right]
\]

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:

\[
y(n) = 0.8y(n-1) + 5x(n) + 2x(n-1)
\]  
(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

\[
Y(z) = 0.8z^{-1}Y(z) + 5X(z) + 2z^{-1}X(z) \quad \Rightarrow \quad (1 - 0.8z^{-1})Y(z) = (5 + 2z^{-1})X(z)
\]

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
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

\[ v_1(n) = x(n-1) \Rightarrow v_1(n+1) = x(n) \]

where \( v_1(n) \) is the content of the x-delay at time \( n \). Similarly, we define:

\[ w_1(n) = y(n-1) \Rightarrow w_1(n+1) = y(n) \]

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:

\[
\text{(compute output)} \quad y(n) = 0.8w_1(n) + 5x(n) + 2v_1(n) \\
\text{(update states)} \quad v_1(n+1) = x(n) \\
\text{} \quad w_1(n+1) = y(n)
\]

It may be written as the repetitive sample processing algorithm:

\[
\text{for each input sample } x \text{ do:} \\
\quad y = 0.8w_1 + 5x + 2v_1 \\
\quad v_1 = x \\
\quad w_1 = y
\]

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:

\[
H(z) = \frac{5(1 + 0.4z^{-1})}{1 - 0.8z^{-1}} \Rightarrow H(\omega) = \frac{5(1 + 0.4e^{-j\omega})}{1 - 0.8e^{-j\omega}}
\]

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:

\[
|H(\omega)| = \sqrt{\frac{25}{1 - 0.8^2}} \text{ when passing near zeros.}
\]

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 \( z = -0.4 \) when \( \omega = 0 \) and therefore there must be a dip because \( e^{j\omega} \) is closest to the zero \( z = 0.8 \). 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:

\[
H(\omega) \big|_{\omega=0} = H(z) \big|_{z=-1} = \frac{5 + 2}{1 - 0.8} = 35 \\
H(\omega) \big|_{\omega=\pi} = H(z) \big|_{z=-1} = \frac{5 - 2}{1 + 0.8} = \frac{5}{3} = \frac{35}{21}
\]

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)
6.2. TRANSFER FUNCTIONS

\[ H(z) = \frac{5 + 2z^{-1}}{1 - 0.8z^{-1}} = -2.5 + \frac{7.5}{1 - 0.8z^{-1}} \]

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 \( H_2(z) = \frac{7.5}{1 - 0.8z^{-1}} \). 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 \( H(z) \).](image)

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

\[ y(n) = w(n) - 2.5x(n) \quad (6.2.6) \]

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

\[ w(n) = 0.8w(n-1) + 7.5x(n) \quad (6.2.7) \]

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

\[ Y(z) = W(z) - 2.5X(z) \]
\[ W(z) = 0.8z^{-1}W(z) + 7.5X(z) \quad \Rightarrow \quad W(z) = \frac{7.5X(z)}{1 - 0.8z^{-1}} \]

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

\[ w_0(n) = w(n) \]
\[ w_1(n) = w(n-1) \quad \Rightarrow \quad w_1(n+1) = w_0(n) \]

Then, Eqs. (6.2.6) and (6.2.7) can be replaced by the system:

\[ w_0(n) = 0.8w_1(n) + 7.5x(n) \]
\[ y(n) = w_0(n) - 2.5x(n) \]
\[ w_1(n+1) = w_0(n) \]

which can be written in the algorithmic form:

\[ \text{for each input sample } x \text{ do:} \]
\[ w_0 = 0.8w_1 + 7.5x \]
\[ y = w_0 - 2.5x \]
\[ w_1 = w_0 \]

(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

\[ Y(z) = H(z)X(z) = \frac{5 + 2z^{-1}}{1 - 0.8z^{-1}}X(z) \]

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

![Fig. 6.2.4 Canonical form realization of \( H(z) \).](image)
6.2. TRANSFER FUNCTIONS

\[ Y(z) = (5 + 2z^{-1})W(z) \]

Writing these equations in the time domain, we obtain

\[ W(z) = \frac{1}{1 - 0.8z^{-1}}X(z) \Rightarrow W(z) = 0.8z^{-1}W(z) + X(z) \]

or,

\[ w(n) = 0.8w(n - 1) + x(n) \]

Similarly,

\[ Y(z) = 5W(z) + 2z^{-1}W(z) \Rightarrow y(n) = 5w(n) + 2w(n - 1) \]

Thus, we obtain the system of I/O equations

\[ w(n) = 0.8w(n - 1) + x(n) \]

\[ y(n) = 5w(n) + 2w(n - 1) \]

which are mechanized in the block diagram of Fig. 6.2.4. Introducing internal states

\[ w_0(n) = w(n) \]

\[ w_1(n) = w(n - 1) \]

we rewrite the above system as:

\[ w_0(n) = 0.8w_1(n) + x(n) \]

\[ y(n) = 5w_0(n) + 2w_1(n) \]

\[ w_1(n + 1) = w_0(n) \]

which can be written in the algorithmic form:

\[
\text{for each input sample } x \text{ do:} \\
w_0 = 0.8w_1 + x \\
y = 5w_0 + 2w_1 \\
w_1 = w_0
\]

(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,

\[ w_1(n) = 2x(n - 1) + 0.8y(n - 1) \]

The complete I/O description of this realization is then given by the system:

\[ y(n) = w_1(n) + 5x(n) \]

\[ w_1(n + 1) = 2x(n) + 0.8y(n) \]

which translates to the following sample processing algorithm:

\[
\text{for each input sample } x \text{ do:} \\
y = w_1 + 5x \\
w_1 = 2x + 0.8y
\]

(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:

\[ Y(z) = W_1(z) + 5X(z) \]

\[ zW_1(z) = 2X(z) + 0.8Y(z) \]

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)
{
    double y;
    w[0] = 0.8 * w[1] + x;  // update internal state
    y = 5 * w[0] + 2 * w[1];  // compute output
    w[1] = w[0];
    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:

```c
w = (double *) calloc(2, sizeof(double));
for (n=0; n<N; n++)
    y[n] = filter(x[n], w);
```

It can also be written as:

\[
y_n = -a_1 y_{n-1} - \cdots - a_M y_{n-M} + b_0 x_n + b_1 x_{n-1} + \cdots + b_L x_{n-L}
\]  (6.2.12)

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

\[
Y(z) = H(z) X(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_L z^{-L}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_M z^{-M}} X(z)
\]

then, multiplying by the denominator:

\[(1 + a_1 z^{-1} + \cdots + a_M z^{-M}) Y(z) = (b_0 + b_1 z^{-1} + \cdots + b_L z^{-L}) X(z)\]

and finally, transforming back to the time domain:

\[
y_n + a_1 y_{n-1} + \cdots + a_M y_{n-M} = b_0 x_n + b_1 x_{n-1} + \cdots + b_L x_{n-L}
\]  (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.

Example 6.2.1: Determine the transfer function of the following third-order FIR filter with impulse response:

\[ h = [1, 6, 11, 6] \]

**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:

\[
H(z) = h_0 + h_1 z^{-1} + h_2 z^{-2} + h_3 z^{-3} = 1 + 6z^{-1} + 11z^{-2} + 6z^{-3}
\]
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 block diagram realization and the sample-by-sample processing algorithm are:

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 \( h = [1, 0, 0, 0, -1] \). 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,
\[
z^4 - 1 \quad \Rightarrow \quad z = e^{j\pi k/4}, \quad k = 0, 1, 2, 3 \quad \Rightarrow \quad z = 1, j, -1, -j
\]
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, \( 0 \leq \omega \leq \pi \), 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:

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:
\[
(\text{a}) \quad y(n) = 0.25y(n - 2) + x(n) \\
(\text{b}) \quad y(n) = -0.25y(n - 2) + x(n)
\]
**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 pass filter—attenuating the intermediate frequencies between low and high. Indeed, the value of $H(z)$ at $\omega = 0, \pi$ or $z = \pm 1$ is

$$ H(0) = H(\pi) = H(z) \bigg|_{z=1} = \frac{1}{1 - 0.25} = \frac{4}{3} $$

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

$$ H(\pi/2) = H(z) \bigg|_{z=j} = \frac{1}{1 - 0.25(-1)} = \frac{4}{5} $$

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 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:

$$ h(n) = 2 \text{Re}[A_1 (0.5)^n e^{jn\pi/2}] u(n) = 2 \text{Re}(0.5 (0.5)^n e^{jn\pi/2}) u(n) $$

$$ = (0.5)^n \cos(\pi n/2) u(n) $$

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 band pass 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:

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 \( \omega_0 \) 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

\[
y(n) = \sum_m h(m)x(n-m) = \sum_m h(m)e^{j(n-m)\omega_0} = e^{j\omega_0 n} \sum_m h(m)e^{-j\omega_0 m}, \quad \text{or},
\]

\[
y(n) = H(\omega_0) e^{j\omega_0 n}
\]

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 \( \omega \) was replaced by \( \omega_0 \) 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 \( \omega_0 \) resulting in Eq. (6.3.1). To summarize, an infinite double-sided input sinusoid of frequency \( \omega_0 \) reappears at the output \textit{unchanged} in frequency but modified by the frequency response factor \( H(\omega_0) \):

\[
e^{j\omega_0 n} \xrightarrow{H(\omega_0)} H(\omega_0) e^{j\omega_0 n}
\]

Therefore, Eq. (6.3.2) can be written in the form:

\[
e^{j\omega_0 n} \xrightarrow{H(\omega_0)} |H(\omega_0)| e^{j\arg H(\omega_0)}
\]

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:

\[
\begin{align*}
\cos(\omega_0 n) & \xrightarrow{H(\omega_0)} |H(\omega_0)| \cos(\omega_0 n + \arg H(\omega_0)) \\
\sin(\omega_0 n) & \xrightarrow{H(\omega_0)} |H(\omega_0)| \sin(\omega_0 n + \arg H(\omega_0))
\end{align*}
\]

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.

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 \( \omega_1 \) and \( \omega_2 \), resulting in the same linear combination of the corresponding outputs, that is,

\[
A_1 e^{j\omega_1 n} + A_2 e^{j\omega_2 n} \xrightarrow{H(\omega)} A_1 H(\omega_1) e^{j\omega_1 n} + A_2 H(\omega_2) e^{j\omega_2 n}
\]

which shows that the effect of filtering is to change the \textit{relative amplitudes} and phases of the two sinusoids from the values \( [A_1, A_2] \) to the values \( [A_1 H(\omega_1), A_2 H(\omega_2)] \). In the frequency domain, we have for the spectra of the input and output signals:
6.3. SINUSOIDAL RESPONSE

\[ A_1 \delta(\omega - \omega_1) + A_2 \delta(\omega - \omega_2) \rightarrow A_1 H(\omega_1) \delta(\omega - \omega_1) + A_2 H(\omega_2) \delta(\omega - \omega_2) \]

where for simplicity, we dropped a common factor of \(2\pi\). 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.](image)

If one of the sinusoids, say \(\omega_1\), 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:

\[ y(n) = A_1 H(\omega_1) e^{j\omega_1 n} + A_2 H(\omega_2) e^{j\omega_2 n} = A_1 e^{j\omega_1 n} \]

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:

\[ y(n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} Y(\omega) e^{j\omega n} d\omega = \frac{1}{2\pi} \int_{-\pi}^{\pi} H(\omega)X(\omega) e^{j\omega n} d\omega \quad (6.3.5) \]

Another useful filtering concept is that of the phase delay defined in terms of the phase response \(\arg H(\omega)\) as follows:

\[ d(\omega) = -\frac{\arg H(\omega)}{\omega} \Rightarrow \arg H(\omega) = -\omega d(\omega) \quad (6.3.6) \]

Similarly, the group delay of a filter is defined as:

\[ d_g(\omega) = -\frac{d}{d\omega} \arg H(\omega) \quad (6.3.7) \]

The sinusoidal response of Eqs. (6.3.2) or (6.3.3) can be expressed in terms of the phase delay as follows:

\[ e^{j\omega n} \rightarrow \frac{H(\omega)}{|H(\omega)|} e^{j\omega(n-d(\omega))} \quad (6.3.8) \]

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 \(\omega\), \(\arg H(\omega) = -\omega D\). Such filters cause every frequency component to be delayed by the same amount \(D\), thus corresponding to an overall delay in the output:

\[ e^{j\omega n} \rightarrow \frac{H(\omega)}{|H(\omega)|} e^{j\omega(n-D)} \quad (6.3.9) \]

This overall delay can also be seen by the inverse DTFT formulas:

\[ x(n) = \int_{-\pi}^{\pi} X(\omega) e^{j\omega n} d\omega \rightarrow \frac{H(\omega)}{|H(\omega)|} \int_{-\pi}^{\pi} |H(\omega)| X(\omega) e^{j\omega(n-D)} d\omega \]

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:

\[ X(z) = e^{j\omega_0 n} u(n) \rightarrow X(z) = \frac{1}{1 - e^{j\omega_0 z^{-1}}} \]

having \(|z| > |e^{j\omega_0}| = 1\). Assume a filter of the form:

\[ H(z) = \frac{N(z)}{D(z)} = \frac{N(z)}{(1 - p_1 z^{-1})(1 - p_2 z^{-1}) \cdots (1 - p_M z^{-1})} \]
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,\(^1\) 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:

$$C = \frac{1}{1 - e^{j\omega_0}z^{-1}} Y(z) \big|_{z=e^{j\omega_0}} = \frac{H(z)}{1 - e^{j\omega_0}z^{-1}} \big|_{z=e^{j\omega_0}}$$

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,

$$C = H(z) \big|_{z=e^{j\omega_0}} = H(\omega_0)$$

(6.3.10)

Therefore, the PF expansion will read:

$$Y(z) = \frac{H(\omega_0)}{1 - e^{j\omega_0}z^{-1}} + \frac{B_1}{1 - p_1z^{-1}} + \frac{B_2}{1 - p_2z^{-1}} \cdots + \frac{B_M}{1 - p_Mz^{-1}}$$

Taking the causal inverse $z$-transform (with ROC $|z| > 1$), we find for $n \geq 0$:

$$y(n) = H(\omega_0)e^{j\omega_0n} + B_1p_1^n + B_2p_2^n + \cdots + B_Mp_M^n$$

(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:

\[^1\text{This assumption is not critical. The same conclusions can be drawn otherwise.}\]

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 $\omega_0$.

**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:

$$y(n) = H(\omega_0)e^{j\omega_0n} + B_1(0.8)^n, \quad n \geq 0$$

For large $n$, the term $(0.8)^n$ drops to zero and the output settles into its steady-state sinusoidal response

$$y(n) = H(\omega_0)e^{j\omega_0n}$$

where $H(\omega_0) = \frac{5 + 2e^{-j\omega_0}}{1 - 0.8e^{-j\omega_0}}$.

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

$$\rho = \max_i |p_i|$$

\[\square\]
we may define the effective time constant to be the time $n_{\text{eff}}$ at which the quantity $\rho^n$ 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 $n_{\text{eff}}$ such that:

$$
\rho^{n_{\text{eff}}} = \varepsilon
$$

where $\varepsilon$ is the desired level of smallness, for example, $\varepsilon = 1\% = 0.01$. It follows that:

$$
n_{\text{eff}} = \frac{\ln \varepsilon}{\ln \rho} = \frac{\ln(1/\varepsilon)}{\ln(1/\rho)} \quad \text{(time constant)} \quad (6.3.12)
$$

Because both $\varepsilon$ and $\rho$ 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 $n_{\text{eff}}$ becomes larger if the slowest pole is pushed closer to the unit circle, that is, increasing $\rho$ toward one, and also if we require a smaller threshold $\varepsilon$.

The value $\varepsilon = 1\%$ 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 $\varepsilon = 0.1\% = 10^{-3}$.

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 if all of 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, $x(n) = \sin(\omega_0 n), 0 \leq n < 300$, 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 $\omega_0$ is unity. Determine and plot the output of the filter $y(n)$ over the interval $0 \leq n < 450$, by iterating the difference equation of the filter.

**Solution:** The 1% time constant of this filter is computed from Eq. (6.3.12),

$$
n_{\text{eff}} = \frac{\ln \varepsilon}{\ln \rho} = \frac{\ln(0.01)}{\ln(0.97)} = 151.2 \text{ samples}
$$

The frequency and magnitude responses are

$$
H(\omega) = \frac{b}{1 - ae^{-j\omega}} \Rightarrow |H(\omega)| = \frac{b}{\sqrt{1 - 2a \cos \omega + a^2}}
$$

The requirement that $|H(\omega_0)| = 1$ leads to the condition on $b$:

$$
|H(\omega_0)| = \frac{b}{\sqrt{1 - 2a \cos \omega_0 + a^2}} = 1 \Rightarrow b = \sqrt{1 - 2a \cos \omega_0 + a^2} = 0.3096
$$

The value of the frequency response at $\omega_0$ becomes then,

$$
H(\omega_0) = \frac{b}{1 - ae^{-j\omega_0}} = 0.2502 - 0.9682j = 1 \cdot e^{j\mu \omega_0}
$$

so that its phase response will be $\arg H(\omega_0) = -\pi \cdot 0.1\text{ radians}$. 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:

```c
for (y1=0, n=0; n<450; n++) {
  if (n < 300)
    x = sin(\omega_0 * n);
  else
    x = 0;
  y[n] = a * y1 + b * x;
  y1 = y[n];/* y1 = y[n-1] */
}
```

**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 $150 \leq n < 450$, by iterating the difference equation of the filter.

**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, $x(n) = e^{j\omega_0 n}, 0 \leq n < 299$, and take imaginary parts of the answer at the end. Using convolution, we obtain

$$
y(n) = \sum_{m=\max(0,n-299)}^{n} h(m)x(n-m) = \sum_{m=\max(0,n-299)}^{n} h^m u(\omega_0(n-m))
$$

where we used the impulse response $h(m) = h^m u(m)$. 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,
6.3. **SINUSOIDAL RESPONSE**

$$0 \leq m < \infty, \quad 0 \leq n - m \leq 299$$

which are equivalent to max(0, n - 299) ≤ m ≤ n. It follows that if 0 ≤ n ≤ 299, then

$$y(n) = e^{j\phi n} \sum_{m=0}^{n} b a^m e^{-j\omega_0 (n-m)} = e^{j\phi n} b \frac{1 - a^{n+1} e^{-j\omega_0 (n+1)}}{1 - a e^{-j\omega_0}}$$

Setting $H_0 = H(\omega_0) = b/(1 - a e^{-j\omega_0})$ and noting that $(1 - a e^{-j\omega_0})H_0 = b$ which gives $b - H_0 = -H_0 a e^{-j\omega_0}$, we obtain

$$y(n) = H_0 e^{j\omega_0 n} + (b - H_0) a^n,$$  for $0 \leq n \leq 299$

On the other hand, if 300 ≤ n ≤ 449, then the limits of summation become:

$$y(n) = e^{j\omega_0 n} \sum_{m=0}^{m_2} b a^m e^{-j\omega_0 m}$$

and we must use the finite geometric series formula

$$\sum_{m=m_1}^{m_2} x^m = \frac{x^{m_1} - x^{m_2+1}}{1 - x}$$

with $x = a e^{-j\omega_0}$, to obtain:

$$y(n) = H_0 e^{j\omega_0 n}[a^{n-299} e^{-j\omega_0 (n-299)} - a^{n+1} e^{-j\omega_0 (n+1)}]$$

Noting that the factor $e^{j\omega_0 n}$ cancels, we obtain

$$y(n) = H_0[a^{n-299} e^{j\omega_0 n} - a^{n+1} e^{-j\omega_0 (n+1)}] = H_0 a e^{-j\omega_0} (a^{300} - a^{300}) a^{n-300}$$

At n = 300, we have $y(300) = H_0 a e^{-j\omega_0} (a^{300} - a^{300})$, Therefore, we can write

$$y(n) = y(300) a^{n-300}, \quad 300 \leq n < \infty$$

Thus, the input-output transients are exponentially decaying as seen in Fig. 6.3.4. Note that the two expressions of $y(n)$ for $n \leq 299$ and $n \geq 300$ 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 $b - H_0 = -aH_0 e^{-j\omega_0}$,

$$ay(299) = a[H_0 e^{j\omega_0 n} + (b - H_0) a^{299}]$$

$$= aH_0 e^{j\omega_0 n} e^{-j\omega_0} - a^2 H_0 e^{-j\omega_0} a^{299}$$

$$= aH_0 e^{-j\omega_0} (e^{j\omega_0} - a^{300}) = y(300)$$

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 $n ≤ 299$

$$y(n) = H_0 e^{j\omega_0 n} + (b - H_0) a^n = e^{j(\omega_0 n + \phi_0)} + (b - e^{j\phi_0}) a^n$$

Taking imaginary parts, we find

$$y(n) = \sin(\omega_0 n + \phi_0) - a^n \sin \phi_0, \quad \text{for } 0 \leq n \leq 299$$

and

$$y(n) = y(300) a^{n-300}, \quad \text{for } n ≥ 300$$

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 $n$:

$$x(n) = e^{j\omega_0 n} (u(n) - u(n - 300)) = e^{j\omega_0 n} u(n) - e^{j300\omega_0} e^{j\omega_0 n - 300} u(n - 300)$$

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

$$Y(z) = H(z) X(z) = \frac{b(1 - e^{j\omega_0 z^{-300}})}{(1 - az^{-1})(1 - e^{j\omega_0 z^{-1}})}$$

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, $Y(z) = \left(1 - e^{j300\omega_0 z^{-300}}\right) W(z)$, we find

$$y(n) = w(n) - e^{j300\omega_0} w(n - 300)$$

which gives rise to the following expression for $y(n)$:

$$y(n) = H_0 e^{j\omega_0 n} u(n) + (b - H_0) a^n u(n) - e^{j300\omega_0} [H_0 e^{j\omega_0 (n-300)} u(n - 300) + (b - H_0) a^{n-300} u(n - 300)]$$

We leave it up to the reader to verify that this expression agrees with the separate expressions given above for $n ≤ 299$ and $n ≥ 300$. \(\square\)
6.3. SINUSOIDAL RESPONSE

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:

\[
y(n) = H(0) + B_1 p_1^n + B_2 p_2^n + \cdots + B_M p_M^n, \quad n \geq 0
\]

Thus, in the limit of large \( n \), the output will settle into a constant value given by

\[
y(n) \to H(0) = H(z) \bigg|_{z=1} \quad \text{as} \quad n \to \infty
\]

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 \( z = 1 \) in the definition of \( H(z) \), Eq. (5.1.2), we can express \( H(0) \) in the alternative form:

\[
H(0) = H(z) \bigg|_{z=1} = \sum_{n=0}^{\infty} h(n) \quad \text{(DC gain)} \tag{6.3.14}
\]

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:

\[
y(n) = H(\pi) e^{j\pi n} + B_1 p_1^n + B_2 p_2^n + \cdots + B_M p_M^n, \quad n \geq 0
\]

And, in the limit of large \( n \), the output tends to

\[
y(n) \to H(\pi) (-1)^n \quad \text{as} \quad n \to \infty
\]

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 \( H(z) \):

\[
H(\pi) = H(z) \bigg|_{z=-1} = \sum_{n=0}^{\infty} (-1)^n h(n) \quad \text{(AC gain)}
\]

**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 \( n_{\text{eff}} \) 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, \( H(\omega_0) + B_1 = 3 \), which can be solved for \( B_1 = 5 - H(\omega_0) \). Thus, for a general \( \omega_0 \) we may write:

\[
y(n) = H(\omega_0) e^{j\omega_0 n} + (5 - H(\omega_0)) (0.8)^n, \quad n \geq 0
\]

For the DC unit-step response, we have setting \( \omega_0 = 0 \):

\[
H(0) = \left[ \frac{5 + 2 e^{j\omega_0}}{1 - 0.8 e^{-j\omega_0}} \right]_{\omega_0=0} = \frac{5 + 2}{1 - 0.8} = 35
\]

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:

- **DC response**
- **Alternating response**

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. \( \Box \)

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 $\theta_1$, 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:

$$
H(z) = \frac{1}{1-z^{-1}}, \quad H(z) = \frac{1}{1+z^{-1}}, \quad H(z) = \frac{1}{1+z^{-2}}
$$

have poles at $p_1 = 1 = e^{j\pi}$, $p_1 = -1 = e^{-j\pi}$, and $\pm j = e^{j\pi/2}$, respectively.

Suppose also that all other poles lie inside the unit circle. Then the transient response Eq. (6.3.11) will be:

$$
y(n) = y(n) = H(\omega_0) e^{j\omega_0 n} + B_1 p_1^n + B_1^* p_1^{-n} + B_2 p_2^n + \cdots, \quad \text{or,}
$$

$$
y(n) = H(\omega_0) e^{j\omega_0 n} + B_1 e^{j\theta_1 n} + B_1^* e^{-j\theta_1 n} + B_2 p_2^n + \cdots
$$

In the limit of large $n$, the terms $p_1^n$, $p_1^{-n}$, etc., will drop to zero exponentially, but the $p_1^n$ being sinusoidal will not. Thus, the filter output tends to

$$
y(n) \rightarrow H(\omega_0) e^{j\omega_0 n} + B_1 e^{j\theta_1 n} + B_1^* e^{-j\theta_1 n}
$$

for large $n$. There is no true sinusoidal response even though the output is a sum of sinusoids. Once the sinusoidal pole terms $e^{j\omega_0 n}$ 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:

$$
Y(z) = H(z) X(z) = \frac{N(z)}{1 - e^{j\omega_0} z^{-1}} (1 - p_1 z^{-1}) \cdots (1 - p_M z^{-1})
$$

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 $|z| > |a|$):

$$
\frac{1}{(1 - az^{-1})^2} \rightarrow (n + 1) a^n u(n)
$$

we find for the output signal:

$$
y(n) = B_1 p_1^n + B_1' (n + 1) p_1^n + B_2 p_2^n + \cdots + B_M p_M^n, \quad \text{or,}
$$

$$
y(n) = B_1 e^{j\omega_0 n} + B_1' (n + 1) e^{j\omega_0 n} + B_2 p_2^n + \cdots + B_M p_M^n
$$

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 $h = [h_0, h_1, \cdots, h_M]$. For a causal sinusoidal input $x(n) = e^{j\omega_0} u(n)$, 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 $n \geq 0$:

$$
y(n) = e^{j\omega_0 n} \sum_{m=0}^{\min(n,M)} h(m) e^{-j\omega_0 m}
$$

When $n \geq M$, the upper summation limit becomes $M$, giving

$$
y(n) = e^{j\omega_0 n} \sum_{m=0}^{M} h(m) e^{-j\omega_0 m} = H(\omega_0) e^{j\omega_0 n}, \quad n \geq M
$$

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

$$
H(z) = \frac{G(1 + bz^{-1})}{1 - az^{-1}}
$$

(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,
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):

$$H(0) = \frac{G(1+b)}{1-a}, \quad H(\pi) = \frac{G(1-b)}{1+a}$$

Therefore, the attenuation of the highest frequency relative to the lowest one is:

$$\frac{H(\pi)}{H(0)} = \frac{(1-b)(1-a)}{(1+b)(1+a)} \quad (6.4.2)$$

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 $n_{\text{eff}}$, 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 $a$:

$$a = e^{1/n_{\text{eff}}} = (0.01)^{1/20} \approx 0.8$$

With this value of $a$, requiring that $H(\pi)/H(0) = 1/21$, Eq. (6.4.2) would give:

$$\frac{(1-b)(1-0.8)}{(1+b)(1+0.8)} = \frac{1}{21} \Rightarrow b = 0.4$$

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 $0 \leq b \leq 1$, 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:

$$H(z) = \frac{G}{1-0.8z^{-1}}, \quad \frac{H(\pi)}{H(0)} = \frac{1}{9}$$

and setting $b = 1$,

$$H(z) = \frac{G(1+z^{-1})}{1-0.8z^{-1}}, \quad \frac{H(\pi)}{H(0)} = \frac{1}{9}$$

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 $\omega_0$. Such frequency response is shown in Fig. 6.4.2.

To make a peak at $\omega = \omega_0$, we place a pole inside the unit circle along the ray with phase angle $\omega_0$, 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:

$$H(z) = \frac{G}{(1-Re^{j\omega_0}z^{-1})(1-Re^{-j\omega_0}z^{-1})} = \frac{G}{1 + a_1z^{-1} + a_2z^{-2}}$$

(6.4.3)

where $a_1$ and $a_2$ are related to $R$ and $\omega_0$ by

$$a_1 = -2R\cos\omega_0, \quad a_2 = R^2$$

The gain $G$ may be fixed so as to normalize the filter to unity at $\omega_0$, 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

\[ 20 \log_{10} \frac{|H(\omega)|}{|H(\omega_0)|} = 10 \log_{10} \left( \frac{1}{2} \right) = -3 \text{ dB} \]

This equation has two solutions, say \( \omega_1 \) and \( \omega_2 \), the first to the left of \( \omega_0 \) 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, \( R \leq 1 \), the full width is given approximately by

\[ \Delta \omega \approx 2(1 - R) \tag{6.4.4} \]

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 \( |OP| = R \). Therefore, the distance \( |PQ| \) to the unit circle will be \( |PQ| = 1 - R \).

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:

\[ |H(z_A)| = \frac{G}{|z_A - p||z_A - p^*|}, \quad |H(z_Q)| = \frac{G}{|z_Q - p||z_Q - p^*|} \]

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, \( |z_A - p^*| \approx |z_Q - p^*| \). Thus, we have for the ratio:

\[ \frac{|H(z_A)|}{|H(z_Q)|} = \frac{|z_A - p|}{|z_Q - p|} \sim \frac{|PQ|}{|PA|} \]

Then, the 3-dB condition that \( |H(z_A)|/|H(z_Q)| = 1/\sqrt{2}, \) implies \( |PQ|/|PA| = 1/\sqrt{2}, \) or, \( |PA| = \sqrt{2}|PQ| \), 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 \( |AB| = 2|QA| = 2|PQ| = 2(1 - R) \). 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 \( |AB| \) and therefore, \( \Delta \omega = |AB| = 2(1 - R) \).

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 \( n \geq 0 \):

\[ h(n) = \frac{G}{\sin \omega_0} R^n \sin(\omega_0 n + \omega_0) \]

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

\[ (1 + a_1 z^{-1} + a_2 z^{-2}) Y(z) = G X(z) \]

and in the time domain:

\[ y(n) + a_1 y(n - 1) + a_2 y(n - 2) = G x(n) \]

or,

\[ y(n) = -a_1 y(n - 1) - a_2 y(n - 2) + G x(n) \tag{6.4.5} \]

A block diagram realization is shown in Fig. 6.4.4. The corresponding sample processing algorithm is obtained by introducing the internal states

\[ w_1(n) = y(n - 1) \]
\[ w_2(n) = y(n - 2) = w_1(n - 1) \]
The difference equation may be replaced by the system:

\[ y(n) = -a_1 w_1(n) - a_2 w_2(n) + G x(n) \]
\[ w_2(n + 1) = w_1(n) \]
\[ w_1(n + 1) = y(n) \]

which gives rise to the following sample processing algorithm:

```
for each input sample x do:
    y = -a_1 w_1 - a_2 w_2 + G x
    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

\[ \omega_0 = \frac{2\pi f_0}{f_s} = 0.1\pi \] [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:

![Magnitude Response](image)

![Impulse Response](image)

The effective time constant of this filter is about \( n_{eff} = \ln \epsilon / \ln R = 458 \) time samples. The graph only plots until \( n = 300 \). \[ \square \]

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:

\[ z_1 = r e^{j\omega_0}, \quad z_1^* = r e^{-j\omega_0} \]

where \( r \) is restricted to the range \( 0 \leq r \leq 1 \). The transfer function becomes:

\[ H(z) = \frac{(1 - r e^{j\omega_0} z^{-1})(1 - r e^{-j\omega_0} z^{-1})}{(1 - R e^{j\omega_0} z^{-1})(1 - R e^{-j\omega_0} z^{-1})} = \frac{1 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \]  

(6.4.6)

where the filter coefficients are given in terms of the parameters \( r, R \), and \( \omega_0 \):

\[ b_1 = -2R \cos \omega_0 \quad b_2 = r^2 \]
\[ a_1 = -2R \cos \omega_0 \quad a_2 = R^2 \]  

(6.4.7)

The corresponding magnitude squared response is:

\[ |H(\omega)|^2 = \frac{(1 - 2R \cos(\omega - \omega_0) + r^2)(1 - 2R \cos(\omega + \omega_0) + r^2)}{(1 - 2R \cos(\omega - \omega_0) + R^2)(1 - 2R \cos(\omega + \omega_0) + R^2)} \]

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, \( r \approx R \) or \( r \gg R \), 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
6.4. POLE/ZERO DESIGNS

A parametric equalizer filter is shown in Fig. 6.4.5. The transfer function takes the form:

\[ H(z) = \frac{1 + b_1 z^{-1} + b_2 z^{-2}}{1 + R b_1 z^{-1} + R^2 b_2 z^{-2}} = \frac{N(z)}{N(\rho^{-1} z)} \]

where \( N(z) \) is the numerator polynomial having zeros at the two notch locations \( z = e^{j\omega_0} \):

\[ N(z) = 1 + b_1 z^{-1} + b_2 z^{-2} = 1 - 2z^{-1} \cos \omega_0 + z^{-2} = (1 - e^{j\omega_0} z^{-1}) (1 - e^{-j\omega_0} z^{-1}) \]

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 \( \omega_i, i = 1, 2, \ldots, M \), then \( N(z) \) is defined as the \( M \)th degree polynomial with zeros at \( z_i = e^{j\omega_i}, i = 1, 2, \ldots, M \):

\[ N(z) = \prod_{i=1}^{M} (1 - e^{j\omega_i} z^{-1}) \]

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

\[ a_1 = R b_1 = -2R \cos \omega_0, \quad a_2 = R^2 b_2 = R^2 \]
6.4. POLE/ZERO DESIGNS

\[ H(z) = \frac{N(z)}{N(\rho^{-1}z)} = \frac{1 + b_1z^{-1} + b_2z^{-2} + \cdots + b_Mz^{-M}}{1 + \rho b_1z^{-1} + \rho^2b_2z^{-2} + \cdots + \rho^M b_Mz^{-M}} \]  

(6.4.9)

that is, the denominator coefficients are chosen as the scaled versions of the numerator coefficients,

\[ a_i = \rho^i b_i, \quad i = 1, 2, \ldots, M \]

If \( \rho \) is near one, \( \rho \leq 1 \), 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

\[ \omega_1 = 2\pi f_1 f_s = \frac{2\pi \cdot 60}{600} = 0.2\pi \quad [\text{radians/sample}] \]

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 \( i = 0, 1, \ldots, 9 \). Because \( f_1 = 10f_1 \), 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 \( f_{11} = 11f_1 \) gets aliased with \( f_{11} - f_1 = 11f_1 - 10f_1 = f_1 \), 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:

\[ N(z) = 1 - z^{-10} = \prod_{i=0}^{9} (1 - e^{j\omega_1}z^{-1}) \]

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 \( 0 \leq \omega \leq \pi \)).

---

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 \( i = 0, 1, \ldots, 9 \). Because \( f_1 = 10f_1 \), 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 \( f_{11} = 11f_1 \) gets aliased with \( f_{11} - f_1 = 11f_1 - 10f_1 = f_1 \), 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:

\[ N(z) = 1 - z^{-10} = \prod_{i=0}^{9} (1 - e^{j\omega_1}z^{-1}) \]

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 \( 0 \leq \omega \leq \pi \)).

---

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 \( i = 0, 1, \ldots, 9 \). Because \( f_1 = 10f_1 \), 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 \( f_{11} = 11f_1 \) gets aliased with \( f_{11} - f_1 = 11f_1 - 10f_1 = f_1 \), 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:

\[ N(z) = 1 - z^{-10} = \prod_{i=0}^{9} (1 - e^{j\omega_1}z^{-1}) \]

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 \( 0 \leq \omega \leq \pi \)).

---

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 \( i = 0, 1, \ldots, 9 \). Because \( f_1 = 10f_1 \), 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 \( f_{11} = 11f_1 \) gets aliased with \( f_{11} - f_1 = 11f_1 - 10f_1 = f_1 \), 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:

\[ N(z) = 1 - z^{-10} = \prod_{i=0}^{9} (1 - e^{j\omega_1}z^{-1}) \]

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 \( 0 \leq \omega \leq \pi \)).
6.4. POLE/ZERO DESIGNS

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:

\[
N(z) = \frac{1 - z^{-20}}{1 - z^{-1}} = \frac{1 - z^{10}}{1 - z^{10}} = (1 + z^2 + z^4 + z^6 + z^8) (1 + z^{-10})
\]

Therefore, we find for the notch transfer function:

\[
H(z) = \frac{N(z)}{N(r^{-1}z)} = \frac{(1 + z^2 + z^4 + z^6 + z^8) (1 + z^{-10})}{(1 + r^2 z^{-2} + r^4 z^{-4} + r^6 z^{-6} + r^8 z^{-8}) (1 + r^{10} z^{-10})}
\]

The resulting 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 reverber 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:

\[
y(n) = h(n) \ast x(n)
\]

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

\[
H_{\text{inv}}(z) = \frac{1}{H(z)}
\]

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:

\[
y(n) = x(n) = H_{\text{inv}}(n)Y(n)
\]
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

\[
y_{\text{hall}}(n) = h_{\text{hall}}(n) \ast x(n)
\]  

(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 \( H_{\text{inv}}(z) \). 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:

\[
y(n) = h(n) \ast x(n) + v(n)
\]

(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:
6. TRANSFER FUNCTIONS

\[ \tilde{x}(n) = h_{\text{inv}}(n) * y(n) = x(n) + \tilde{v}(n) \]  \hspace{1cm} (6.5.7)

where \( \tilde{v}(n) \) is the filtered noise:

\[ \tilde{v}(n) = h_{\text{inv}}(n) * v(n) \]

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 \( \tilde{v}(n) \) to be a much amplified version of \( v(n) \), rendering \( x(n) \) a very poor and almost useless estimate of the desired signal \( y(n) \). 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:

\[ h_{\text{inv}}(n) = 0.4\delta(n) - 0.6(1.25)^n u(-n - 1) = \begin{cases} 0 & \text{if } n \geq 1 \\ 0.4 & \text{if } n = 0 \\ -0.6(1.25)^n & \text{if } n \leq -1 \end{cases} \]

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:

\[ \tilde{h}_{\text{inv}}(n) = \begin{cases} h_{\text{inv}}(n) & \text{if } n \geq -D \\ 0 & \text{if } n < -D \end{cases} \]  \hspace{1cm} (6.5.8)

The approximate impulse response \( \tilde{h}_{\text{inv}}(n) \) 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 \( \tilde{x}(n) \) be the deconvolved output using \( \tilde{h}_{\text{inv}}(n) \), that is,

\[ \tilde{x}(n) = \tilde{h}_{\text{inv}}(n) * y(n) \]

Subtracting it from the exact output \( x(n) \) of Eq. (6.5.3), we have

\[ x(n) - \tilde{x}(n) = h_{\text{inv}}(n) * y(n) - \tilde{h}_{\text{inv}}(n) * y(n) = (h_{\text{inv}}(n) - \tilde{h}_{\text{inv}}(n)) * y(n) \]

\[ = \sum_{m=-\infty}^{\infty} (h_{\text{inv}}(m) - \tilde{h}_{\text{inv}}(m)) y(n - m) \]

\[ = \sum_{m=-\infty}^{-D-1} h_{\text{inv}}(m) y(n - m) \]

where all the terms for \( m \geq -D \) were dropped because \( \tilde{h}_{\text{inv}}(m) \) agrees with \( h_{\text{inv}}(m) \) there, and \( \tilde{h}_{\text{inv}}(m) = 0 \) for \( m < -D \). Assuming the signal \( y(n) \) is bounded by some maximum value \( |y(n)| \leq A \), we find

\[ |x(n) - \tilde{x}(n)| \leq A \sum_{m=-\infty}^{-D-1} |h_{\text{inv}}(m)| \]  \hspace{1cm} (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:

\[ |x(n) - \tilde{x}(n)| \leq A \sum_{m=-\infty}^{-D-1} |0.6(1.25)^m| = 2.4A(1.25)^{-D} \]

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 \( B \):
6.6 PROBLEMS

6.6.1 Transfer Functions

In summary, when the inverse filter transfer function \( H_{inv}(z) \) has some poles outside the unit circle, one must choose the anticausal but stable impulse response \( h_{inv}(n) \) 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

6.6.1 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:

\( y(n) = -0.8y(n-1) + x(n) \)

\( y(n) = 0.8y(n-1) + x(n) + x(n-1) \)

\( y(n) = 0.8y(n-1) + x(n) - 0.5x(n-1) \)

\( y(n) = 0.8y(n-1) + x(n) + 0.25x(n-2) \)

\( y(n) = 0.9y(n-1) - 0.2y(n-2) + x(n) + x(n-1) - 6x(n-2) \)

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 \( 0 < \omega < \pi \), and finally, draw the direct and canonical block diagram realizations of the difference equation and state the corresponding sample-by-sample filtering algorithms.

6.6.2 A unit-step signal \( x(n) = u(n) \) is applied to the input of the linear systems:

\( y(n) = x(n) + 6x(n-1) + 11x(n-2) + 6x(n-3) \)

\( y(n) = x(n) - x(n-4) \)

Using z-transforms, determine the corresponding output signals \( y(n) \), for all \( n \geq 0 \). Repeat for the alternating-step input \( x(n) = (-1)^nu(n) \).

6.6.3 Repeat Problem 6.2 for the following systems:

\( y(n) = 0.25y(n-2) + x(n) \)

\( y(n) = -0.25y(n-2) + x(n) \)

6.6.4 A unit-step signal \( x(n) = u(n) \) is applied to the inputs of the systems of Problem 6.1.

\( y(n) = x(n) + 6x(n-1) + 11x(n-2) + 6x(n-3) \)

\( y(n) = x(n) - x(n-4) \)

Using z-transforms, derive expressions for the corresponding output signals \( y(n) \) for all \( n \geq 0 \), and determine which part of \( y(n) \) is the steady-state part and which the transient part.

6.6.4.1 Repeat for the input \( x(n) = (-1)^nu(n) \).

6.6.4.2 Repeat for the input \( x(n) = (0.5)^nu(n) \) applied only to Problem 6.1(d).

6.6.4.3 Repeat for the input \( x(n) = (0.5)^n\cos(\pi n/2)u(n) \) applied to Problem 6.1(e) only.

6.6.8 The first few Fibonacci numbers are:

\( h = [0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots] \)

6.6.5 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:

\( h(n) = \delta(n-5) \)

\( h(n) = u(n-5) \)

\( h(n) = 0.8u(n) \)

\( h(n) = 2(0.8)^n\cos(\pi n/2)u(n) \)

\( h(n) = (0.8)^n\sin(n) \)

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 \( 0 < \omega < \pi \), and finally, draw the direct and canonical realizations implementing the I/O difference equation and state the corresponding sample-by-sample processing algorithms.

6.6.6 Find the transfer function \( H(z) \) and express it as the ratio of two polynomials of the system having impulse response:

\( h(n) = \sum_{m=0}^{\infty} (0.5)^m\delta(n-8m) = \delta(n) + (0.5)^2\delta(n-8) + (0.5)^3\delta(n-16) + \cdots \)

Then, draw a block diagram realization and write its sample processing algorithm.

6.6.7 A digital reverberation processor has frequency response:

\( H(\omega) = \frac{-0.5 + e^{-j\omega}}{1 - 0.5e^{-j\omega}} \)

where \( \omega \) is the digital frequency in [radians/sample]. Determine the causal impulse response \( h(n) \), for all \( n \geq 0 \), and sketch it versus \( n \). [Hint: Do not use partial fractions.]

6.6.8 The first few Fibonacci numbers are:

\( h = [0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots] \)

where each is obtained by summing the previous two.

6.6.9 Determine the linear system \( H(z) \) whose causal impulse response is \( h \), and express it as a rational function in \( z^{-1} \).

6.6.10 Using partial fractions, derive an expression for the \( r \)-th Fibonacci number in terms of the poles of the above filter.

6.6.11 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, \( \phi = (1 + \sqrt{5})/2 \).

6.6.12 Show that the filter’s poles are the two numbers \( \{\phi, -\phi^{-1}\} \). Show that the geometric sequence:

\( y = [0, 1, \phi, \phi^2, \phi^3, \ldots] \)

satisfies the same recursion as the Fibonacci sequence (for \( n \geq 3 \)). Show that \( y \) may be considered to be the output of the filter \( h \) for a particular input. What is that input?
6.6. PROBLEMS

271

6. TRANSFER FUNCTIONS

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., \( h_n = 2h_{n-1} + h_{n-2} \)):

\[
h = [0, 1, 2, 5, 12, 29, \ldots ]
\]

Determine the linear system \( H(z) \) whose causal impulse response is \( h \), and express it as a rational function in \( z^{-1} \). Using partial fractions, derive an expression for the \( n \)th 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:

\[
y = [0, 1, \theta, \theta^2, \theta^3, \ldots ]
\]

satisfies the same recursion as the Pell sequence (for \( n \geq 3 \)). Show that \( y \) may be considered to be the output of the filter \( h \) for a particular input. What is that input?

6.10 For a particular causal filter, it is observed that the input signal \( (0.5)^nu(n) \) produces the output signal \( (0.5)^nu(n) + (0.4)^nu(n) \). What input signal produces the output signal \( (0.4)^nu(n) \)?

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 \( (0.7)^nu(n) \) is applied to the input of an unknown causal LTI filter, and the signal \( (0.7)^nu(n) + (0.5)^nu(n) \) is observed at the output. What is the causal input signal that will cause the output \( (0.5)^nu(n) \)? 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 \( H(z) = \frac{1}{1 + a_1z^{-1} + a_2z^{-2}} \), which has a peak at \( f_0 = 250 \text{ Hz} \) and a 3-dB width of \( \Delta f = 20 \text{ Hz} \) and is operating at a rate of \( f_s = 5 \text{ kHz} \). 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 \( n_{\text{eff}} \), that is, corresponding to \( \epsilon = 10^{-2} \). Compare the approximate and exact values of \( n_{\text{eff}} \).

6.14 For any stable and causal filter, let \( \tau_{40} \) and \( \tau_{60} \) denote its 40-dB and 60-dB time constants, expressed in seconds. Show that they are related by: \( \tau_{60} = 1.3 \tau_{40} \).

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 \( \tau_{60} \) is in seconds and \( \Delta f \) is the 3-dB width in Hz. When is the approximation valid?

6.16 It is desired to generate the following periodic waveform:

\[
h(n) = [1, 2, 3, 4, 0, 0, 0, 1, 2, 3, 4, 0, 0, 0, \ldots ]
\]

where the dots indicate the periodic repetition of the 8 samples \( [1, 2, 3, 4, 0, 0, 0] \).

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:

\[
h = [0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, \ldots ]
\]

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

\[
h(n) = A + B(-1)^n + 2C \cos \left( \frac{\pi n}{2} \right) + 2D \sin \left( \frac{\pi n}{2} \right), \quad n \geq 0
\]

What are the values of \( A, B, C, D \)?

6.18 Consider the system: \( H(z) = \frac{1 + z^{-1} + z^{-2} + z^{-3}}{1 - z^{-2}} \).

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 \( x = [3, 2, 1] \) 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: \( H(z) = \frac{1 + z^{-1} + z^{-2} + z^{-3}}{1 - z^{-2}} \).

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:

\[
\begin{align*}
w_0 &= x + w_1 \\
y &= w_0 + w_2 \\
w_2 &= w_1 \\
w_1 &= w_0
\end{align*}
\]

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.

\[ H(z) = \frac{1 - z^{-16}}{1 - az^{-1}} \]

\[ a = \frac{1}{2} \]

\[ 0 < a < 1 \]

- 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 \( 0 \leq \omega \leq 2\pi \).
- c. Determine the causal/stable impulse response \( h(n) \) for all \( n \geq 0 \). 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.)

6.22 A digital filter has transfer function, where \( 0 < a < 1 \):

\[ H(z) = \frac{1}{1 - az^{-1}} \]

- 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 \( 0 \leq \omega \leq 2\pi \).
- c. Determine the causal/stable impulse response \( h(n) \) for all \( n \geq 0 \). 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 \( n \geq 0 \), of \( H(z) = \frac{0.3 + 0.15z^{-1}}{1 - 0.5z^{-1}} \).

6.24 Let \( H(z) = \frac{1 - az^{-1}}{1 - \frac{1}{2}z^{-1}} \) 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 \( H(z) = \frac{3 - 3z^{-1}}{1 - 1.5z^{-1}} \). 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 \( 0 \leq |\omega| \leq \omega_c \), 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 \( 0 \leq |\omega| \leq \omega_c \) 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 \( H(z) = \frac{N(z)}{(1 - p_1z^{-1})(1 - p_2z^{-1}) \cdots (1 - p_Mz^{-1})} \), 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:

\[ h(n) = \sum_{i=1}^{M} a_i p_i^n u(n), \quad \text{where} \quad a_i = \frac{N(p_i)}{\prod_{j \neq i} (1 - p_j p_i^{-1})} \]
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

\[
H(z) = \frac{N(z)}{D(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}
\]

having I/O difference equation:

\[
y(n) = -a_1 y_{n-1} - a_2 y_{n-2} + b_0 x_n + b_1 x_{n-1} + b_2 x_{n-2}
\]

The direct form realization is the block diagram representation of this difference equation. It is depicted in Fig. 7.1.1.

![Block diagram of a second-order IIR filter](image)

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 IIR 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:

\[
\begin{align*}
v_0(n) &= x(n) \\
v_1(n) &= x(n-1) = y_0(n-1) & \text{and} & \quad w_1(n) &= y(n-1) = w_0(n-1) \\
v_2(n) &= x(n-2) = v_1(n-1) & \text{and} & \quad w_2(n) &= y(n-2) = w_1(n-1)
\end{align*}
\]

The quantities \(v_1(n), v_2(n), w_1(n), \text{and } w_2(n)\) 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:

\[
\begin{align*}
v_1(n+1) &= v_0(n) & \quad w_1(n+1) &= w_0(n) \\
v_2(n+1) &= v_1(n) & \quad w_2(n+1) &= w_1(n)
\end{align*}
\]

Therefore, we may replace Eq. (7.1.2) by the system:

\[
\begin{align*}
v_0(n) &= x(n) \\
w_0(n) &= -a_1 v_1(n) - a_2 v_2(n) + b_0 y_0(n) + b_1 v_1(n) + b_2 v_2(n) \\
y(n) &= w_0(n) \\
v_2(n+1) &= v_1(n) & \quad w_2(n+1) &= w_1(n) \\
v_1(n+1) &= v_0(n) & \quad w_1(n+1) &= w_0(n)
\end{align*}
\]

It can be replaced by the following repetitive sample processing algorithm:

\[
\text{for each input sample } x \text{ do:} \\
v_0 = x \\
w_0 = -a_1 v_1 - a_2 v_2 + b_0 y_0 + b_1 v_1 + b_2 v_2 \\
y = w_0 \\
v_2 = v_1, \quad w_2 = w_1 \\
v_1 = v_0, \quad w_1 = w_0
\]

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:

\[ H(z) = \frac{N(z)}{D(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_L z^{-L}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_M z^{-M}} \] (7.1.4)

having an \( L \)th degree numerator and \( M \)th degree denominator. The corresponding I/O difference equation is:

\[ y_n = -a_1 y_{n-1} - a_2 y_{n-2} - \cdots - a_M y_{n-M} + b_0 x_n + b_1 v_1(n) + \cdots + b_L v_L(n) \] (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:

\[ v_i(n) = x(n-i), \quad i = 0, 1, \ldots, L \]
\[ w_i(n) = y(n-i), \quad i = 0, 1, \ldots, M \] (7.1.6)

They are updated in time by

\[ v_i(n+1) = v_i(n), \quad i = 1, 2, \ldots, L \]
\[ w_i(n+1) = w_i(n), \quad i = 1, 2, \ldots, M \] (7.1.7)

These can be shown easily, for example:

\[ w_i(n+1) = y(n+1-i) = y(n-(i-1)) = w_{i-1}(n) \]

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:

\[
\begin{align*}
\text{for each input sample } x & \text{ do:} \\
v_0 &= x \\
w_0 &= -a_1 w_1 - \cdots - a_M w_M + b_0 v_0 + b_1 v_1 + \cdots + b_L v_L \\
y &= w_0 \\
v_i &= v_{i-1}, \quad i = L, L-1, \ldots, 1 \\
w_i &= w_{i-1}, \quad i = M, M-1, \ldots, 1
\end{align*}
\] (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:

\[ [v_1, v_2, \ldots, v_L] = [0, 0, \ldots, 0], \quad [w_1, w_2, \ldots, w_M] = [0, 0, \ldots, 0] \]

The following C routine dir.c is an implementation of this algorithm:

\[
\text{/* dir.c - IIR filtering in direct form */}
\]

\[
\begin{align*}
\text{double dir(M, a, L, b, w, v, x)} \quad \text{usage: } y & \text{ = dir}(M, a, L, b, v, w, x); \\
double \ast a, \ast b, \ast w, \ast v, \ast x; \\
\text{int } M, L; \\
\{ \\
\text{int } i; \\
v[0] = x; \quad \text{current input sample} \\
w[0] = 0; \quad \text{current output to be computed} \\
\text{for } (i=0; \ i< L; \ i++) \\
w[0] = b[i] * v[i]; \quad \text{numerator part} \\
\text{for } (i=1; \ i< M; \ i++) \\
w[0] = a[i] * w[i]; \quad \text{denominator part} \\
\text{for } (i=L; \ i>= 1; \ i--) \\
v[i] = v[i-1]; \quad \text{reverse-order updating of } v \\
\text{for } (i=M; \ i>= 1; \ i--) \\
w[i] = w[i-1]; \quad \text{reverse-order updating of } w \\
\text{return } w[0]; \quad \text{current output sample}
\}
\]

Note that \( b, a \) are the numerator and denominator coefficient vectors:

\[ b = [b_0, b_1, b_2, \ldots, b_L], \quad a = [1, a_1, a_2, \ldots, a_M] \] (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:

The filter coefficient and state vectors are in this example:

\[
\begin{align*}
a &= [a_0, a_1, a_2, a_3, a_4] = [1, 0.2, -0.3, 0, 0.5] \\
b &= [b_0, b_1, b_2, b_3] = [2, -3, 0, 4] \\
w &= [w_0, w_1, w_2, w_3, w_4], \quad v = [v_0, v_1, v_2, v_3]
\end{align*}
\]

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:

\[
dot(L, b, v) = b^T v = b_0 v_0 + b_1 v_1 + \cdots + b_L v_L
\]
\[
dot(M, a, w) = a^T w = a_0 w_0 + a_1 w_1 + \cdots + a_M w_M
\]

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,

\[
w_0 = 0 \Rightarrow \dot{w}(M, a, w) = a_1 w_1 + \cdots + a_M w_M
\]

Therefore, we may replace Eq. (7.1.8) by the more compact version:

\[
\begin{align*}
\text{for each input sample } x &\text{ do:} \\
v_0 &= x \\
w_0 &= -0.2w_1 + 0.3w_2 - 0.5w_4 + 2v_0 - 3v_1 + 4v_3 \\
y &= w_0 \\
w_1 &= w_3 \\
w_2 &= w_2, \quad v_3 = v_2 \\
w_3 &= w_1, \quad v_2 = v_1 \\
w_4 &= w_0, \quad v_1 = v_0
\end{align*}
\]

The following routine dir2.c is an implementation of this algorithm. It is the IIR version of the routine fir2 of Chapter 4.

\[
\begin{align*}
\text{ /* dir2.c - IIR filtering in direct form */}
double dot(); \\
void delay(); \\
double dir2(M, a, L, b, w, v, x) usage: y = dir2(M, a, L, b, w, v, x);
\end{align*}
\]
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,

\[ y_n = (b_0 x_n + b_1 x_{n-1} + b_2 x_{n-2}) + (-a_1 y_{n-1} - a_2 y_{n-2}) \]

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.](image1)

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

\[ H(z) = \frac{1}{D(z)} \cdot N(z) \]

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.](image2)
7.2. CANONICAL FORM

\[ w(n) = x(n) - a_1 w(n-1) - a_2 w(n-2) \]
\[ y(n) = b_0 w(n) + b_1 w(n-1) + b_2 w(n-2) \]

(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

\[ W(z) = X(z) - a_1 z^{-1}W(z) - a_2 z^{-2}W(z) \]
\[ Y(z) = b_0 W(z) + b_1 z^{-1}W(z) + b_2 z^{-2}W(z) \]

which can be solved for \( W(z) \) and \( Y(z) \):

\[ W(z) = \frac{1}{1 + a_1 z^{-1} + a_2 z^{-2}} X(z) = \frac{1}{D(z)} X(z) \]
\[ Y(z) = \frac{b_0}{D(z)} X(z) \]

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:

\[ w_0(n) = w(n) \]
\[ w_1(n) = w(n-1) = w_0(n-1) \Rightarrow w_1(n+1) = w_0(n) \]
\[ w_2(n) = w(n-2) = w_1(n-1) \Rightarrow w_2(n+1) = w_1(n) \]

Therefore, the system (7.2.1) can be rewritten as:

\[ w_0(n) = x(n) - a_1 w_1(n) - a_2 w_2(n) \]
\[ y(n) = b_0 w_0(n) + b_1 w_1(n) + b_2 w_2(n) \]
\[ w_2(n+1) = w_1(n) \]
\[ w_1(n+2) = w_0(n) \]

which translates to the following sample processing algorithm:

for each input sample \( x \) do:
\[ w_0 = x - a_1 w_1 - a_2 w_2 \]
\[ y = b_0 w_0 + b_1 w_1 + b_2 w_2 \]
\[ w_2 = w_1 \]
\[ w_1 = w_0 \]

(7.2.2)

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:

\[ (1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_M z^{-M})W(z) = X(z) \]
\[ Y(z) = (b_0 + b_1 z^{-1} + \cdots + b_L z^{-L})W(z) \]

or, equivalently

\[ W(z) = X(z) - (a_1 z^{-1} + a_2 z^{-2} + \cdots + a_M z^{-M})W(z) \]
\[ Y(z) = (b_0 + b_1 z^{-1} + \cdots + b_L z^{-L})W(z) \]

which become in the time domain:

\[ w(n) = x(n) - a_1 w(n-1) - \cdots - a_M w(n-M) \]
\[ y(n) = b_0 w(n) + b_1 w(n-1) + \cdots + b_L w(n-L) \]

(7.2.3)

The block diagram realization of this system is shown in Fig. 7.2.4 for the case \( M = L \). If \( M > L \) one must draw the maximum number of common delays, that is, \( K = \max(M, L) \). Defining the internal states by

\[ w_i(n) = w(n-i) = w_{i-1}(n-1), \quad i = 0, 1, \ldots, K \]

we may rewrite the system (7.2.3) in the form:

Fig. 7.2.4  Canonical realization form of \( M \)th order IIR filter.
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, this leads to the following sample-by-sample filtering algorithm:

\[
\begin{align*}
    w_0(n) &= x(n) - a_1 w_1(n) - \cdots - a_M w_M(n) \\
    y(n) &= b_0 w_0(n) + b_1 w_1(n) + \cdots + b_L w_L(n) \\
    w_i(n+1) &= w_{i-1}(n), \quad i = K, K-1, \ldots, 1
\end{align*}
\] (7.2.4)

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, \([w_1, w_2, \ldots, w_K] = [0, 0, \ldots, 0] \). The following C routine is an implementation of this algorithm:

```c
/* can.c - IIR filtering in canonical form */

double can(M, a, L, b, w, x)
double *a, *b, *w, x;
int M, L;
{
    int K, i;
    double y = 0;
    w[0] = x; /* current input sample */
    for (i=1; i<=M; i++)
        w[0] -= a[i] * w[i];
    for (i=0; i<=L; i++)
        y += b[i] * w[i];
    for (i=K; i>=1; i--)
        w[i] = w[i-1];
    return y; 
}
```

The vectors \( a, b \) must be allocated just as for the direct form routine \( \text{dir} \). The state vector \( w \) must be allocated to dimension \( K+1 \), that is,

\[
    w = (\text{double} *) \text{malloc}(K+1, \text{sizeof(double)});
    w = [w_0, w_1, \ldots, w_K]
\]

The same program segment illustrating the usage of \( \text{dir} \) also illustrates the usage of \( \text{can} \). The only difference is that now there is only one state vector \( w \) instead of \( w, v \):

```c
for (n = 0; n < Ntot; n++)
    y[n] = can(M, a, L, b, w, x[n]);
```

The canonical realization form of Example 7.1.1 is illustrated in Fig. 7.2.5. The difference equations describing this block diagram in the time domain are obtained from the two adder equations:

\[
\begin{align*}
    w(n) &= x(n) - 0.2 w(n-1) + 0.3 w(n-2) - 0.5 \text{w}(n-4) \\
    y(n) &= 2 w(n) - 3 w(n-1) + 4 w(n-3)
\end{align*}
\]

The corresponding sample processing algorithm is:

```c
for each input sample x do:
    w[0] = x; /* current input sample */
    for (i=1; i<=M; i++)
        w[0] -= a[i] * w[i];
    for (i=0; i<=L; i++)
        y += b[i] * w[i];
    for (i=K; i>=1; i--)
        w[i] = w[i-1];
    return y;
```

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.
7.3. TRANSPOSED FORM

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:

\[
\text{for each input sample } x \text{ do:}
\]
\[
w_0 = 0
\]
\[
w_0 = x - \text{dot}(M,a,w)
\]
\[
y = \text{dot}(L,b,w)
\]
\[
delay(K,w)
\]

The C routine can2.c is an implementation.

```c
/* 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;
    w[0] = 0;
    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,

\[
\text{initialize } v_1, v_2
\]
\[
\text{for each input sample } x \text{ do,}
\]
\[
y = b_0 x + v_1
\]
\[
v_1 = b_1 x - a_1 y + v_2
\]
\[
v_2 = b_2 x - a_2 y
\]

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, \(v_1(n+1)\), \(v_2(n+1)\), it follows that this realization is described by the following system of first-order difference equations,

\[
y(n) = b_0 x(n) + v_1(n)
\]
\[
v_1(n+1) = b_1 x(n) - a_1 y(n) + v_2(n)
\]
\[
v_2(n+1) = b_2 x(n) - a_2 y(n)
\]

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...
7. DIGITAL FILTER REALIZATIONS

7.3 Transposed Form

clearer, consider the case of a 2nd order transfer function

\[ H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} = \frac{N(z)}{D(z)} \]

and demonstrate the algebraic equivalence of the direct, canonical, and transposed realizations working in the z-domain. For the direct form, we have,

\[ Y(z) = H(z)X(z) = \left[ \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \right] X(z) \]

\[ Y(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \cdot X(z) \]

\[ H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \]

so that the transposed block diagram realizes the last equation, or, in the time domain,

\[ z V_2(z) = b_2 X(z) - a_2 Y(z) \]

\[ v_2(n + 1) = b_2 x(n) - a_2 y(n) \]

\[ z V_1(z) = b_1 X(z) - a_1 Y(z) + V_2(z) \]

\[ v_1(n + 1) = b_1 x(n) - a_1 y(n) + v_2(n) \]

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,

\[ s(n + 1) = A s(n) + B x(n) \]

\[ y(n) = C s(n) + D x(n) \]

(ABCD state-space realization) \hspace{1cm} (7.4.1)

where the state vector \( s(n) \) 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,

\[
\text{initialize } s, \text{ then, for each input sample } x, \text{ do, }
\]

\[ y = Cs + Dx, \quad \text{output } s = As + Bx, \quad \text{next state} \]

(ABCD sample processing algorithm) \hspace{1cm} (7.4.2)

![Fig. 7.4.1 ABCD state-space realization.](image)

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,

\[ s(n) = \begin{bmatrix} w_1(n) \\ w_2(n) \end{bmatrix} = \text{canonical,} \quad s(n) = \begin{bmatrix} v_1(n) \\ v_2(n) \end{bmatrix} = \text{transposed} \] \hspace{1cm} (7.4.3)
7.4. STATE-SPACE REALIZATIONS

The corresponding $A$, $B$, $C$, $D$ matrices have dimensions, $2 \times 2$, $2 \times 1$, $1 \times 2$, and $1 \times 1$, respectively, and are given as follows in the two cases,

\begin{align*}
\text{(canonical): } & 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): } & 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{align*}

where we defined the parameters,

\begin{equation}
\begin{aligned}
c_1 &= b_1 - b_0 a_1 \\
c_2 &= b_2 - b_0 a_2
\end{aligned}
\end{equation}

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{align*}
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)] \\
y(n) &= v_1(n) + b_0 x(n)
\end{align*}

or, reassembled in ABCD form,

\[
\begin{bmatrix} v_1(n+1) \\ v_2(n+1) \end{bmatrix} = \begin{bmatrix} -a_1 & 1 \\ -a_2 & 0 \end{bmatrix} \begin{bmatrix} v_1(n) \\ v_2(n) \end{bmatrix} + \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} x(n) \\
y(n) &= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} v_1(n) \\ v_2(n) \end{bmatrix} + b_0 x(n)
\]

Similarly, we have for the canonical form,

\begin{align*}
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)
\end{align*}

or, reassembled in ABCD form,

\[
\begin{bmatrix} w_1(n+1) \\ w_2(n+1) \end{bmatrix} = \begin{bmatrix} -a_1 & -a_2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} w_1(n) \\ w_2(n) \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} x(n) \\
y(n) &= \begin{bmatrix} c_1, c_2 \end{bmatrix} \begin{bmatrix} w_1(n) \\ w_2(n) \end{bmatrix} + b_0 x(n)
\]

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,

\[
\begin{array}{c}
A \rightarrow A^T \\
B \rightarrow C^T \\
C \rightarrow B^T \\
D \rightarrow D
\end{array}
\]

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. The corresponding impulse response is obtained by inverting Eq. (7.4.6) causally,

\[
\begin{bmatrix} w_1(n+1) \\ w_2(n+1) \end{bmatrix} = \begin{bmatrix} -a_1 & -a_2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} w_1(n) \\ w_2(n) \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} x(n) \\
y(n) &= \begin{bmatrix} c_1, c_2 \end{bmatrix} \begin{bmatrix} w_1(n) \\ w_2(n) \end{bmatrix} + b_0 x(n)
\]

\[
H(z) = \frac{Y(z)}{X(z)} = C(z I - A)^{-1} B + D
\]

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,

\[
h(n) = C A^n B u(n - 1) + D \delta(n)
\]

We demonstrate Eq. (7.4.6) explicitly for the canonical realization with parameters given by Eq. (7.4.4),

\[
\begin{bmatrix} \det(z I - A) \\ \det(z I - A) \end{bmatrix} = \begin{bmatrix} z & 0 \\ 0 & z \end{bmatrix} - \begin{bmatrix} -a_1 & -a_2 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} z + a_1 & -a_2 \\ -1 & z \end{bmatrix}
\]

\[
(z I - A)^{-1} = \frac{1}{\det(z I - A)} \begin{bmatrix} z & a_2 \\ 1 & z + a_1 \end{bmatrix} = \frac{1}{z^2 + a_1 z + a_2} \begin{bmatrix} z & a_2 \\ 1 & z + a_1 \end{bmatrix}
\]

\[
C (z I - A)^{-1} B = \frac{1}{z^2 + a_1 z + a_2} c_1 c_2 \begin{bmatrix} z & a_2 \\ 1 & z + a_1 \end{bmatrix} = \frac{c_1 z + c_2}{z^2 + a_1 z + a_2}
\]

so that the complete transfer function is,
7. DIGITAL FILTER REALIZATIONS

7.4. STATE-SPACE REALIZATIONS

State-space parameters: $ABCD$ which define the
\[ z^{-1} C(zI - A)^{-1} B + D = \frac{C_1 z + C_2}{z^2 + a_1 z + a_2} + b_0 \]
\[ \frac{(b_1 - b_0 a_1) z + (b_2 - b_0 a_2)}{z^2 + a_1 z + a_2} + b_0 = \frac{b_0 z^2 + b_1 z + b_2}{z^2 + a_1 z + a_2} \]
\[ = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \]

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,
\[ [A,B,C,D] = tf2ss(num,den); \]
which is 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),
\[
\text{num} = [b_0, b_1, b_2]; \\
\text{den} = [1, a_1, a_2]; \\
[A,B,C,D] = tf2ss(num,den); \quad \% \text{canonical state-space form}
\]

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,
\[
p = \sigma + j\omega \\
\sigma = \frac{a_1}{2}, \quad \omega = \sqrt{a_2 - \frac{a_1^2}{4}}, \quad \left( \text{must have of course, } a_2 > \frac{a_1^2}{4} \right) \\
a_1 = b_1 - b_0 a_1, \quad a_2 = b_2 - b_0 a_2 \\
\alpha_r = a_1, \quad \alpha_i = \frac{a_1 \sigma + a_2}{2\omega} \\
\alpha = \alpha_r + j\alpha_i \\
P = |\alpha| \quad Q = \text{Im} \left[ \frac{\alpha}{1 - \text{Im}(\alpha)} \right], \quad k = \sqrt{\frac{P + Q}{P - Q}} \\
B_1 = \sqrt{|\alpha| - \alpha_i}, \quad B_2 = -\text{sign}(\alpha_r) \sqrt{|\alpha| + \alpha_i} \\
C_1 = \frac{\alpha_r}{B_1}, \quad C_2 = \frac{\alpha_r}{B_2}
\]

which define the $ABCD$ state-space parameters:
\[
A = \begin{bmatrix} \sigma & \omega k \\ -\omega/k & \sigma \end{bmatrix}, \quad B = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}, \quad C = [C_1, C_2], \quad D = b_0
\]

In the special case when $\alpha_r = 0$ and $\alpha_i > 0$, we have:
\[
B_1 = 0, \quad B_2 = -\sqrt{\frac{2|\alpha|}{P + Q}}, \quad C_1 = \sqrt{2|\alpha|(P - Q)}, \quad C_2 = 0
\]

and in the case, $\alpha_r = 0$ and $\alpha_i < 0$:
\[
B_1 = \sqrt{\frac{2|\alpha|}{P - Q}}, \quad B_2 = 0, \quad C_1 = 0, \quad C_2 = -\sqrt{2|\alpha|(P + Q)}
\]

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:

```c
/* sos.c - IIR filtering by single second order section */

double sos(a, b, w, x) 
double *a, *b, *w, x;
{ 
    double y;
    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
\[
a = \text{double *}(\text{calloc}(3, \text{sizeof(double)})); \quad a = [1, a_1, a_2] \\
b = \text{double *}(\text{calloc}(3, \text{sizeof(double)})); \quad b = [b_0, b_1, b_2] \\
w = \text{double *}(\text{calloc}(3, \text{sizeof(double)})); \quad w = [w_0, w_1, w_2]
\]

The cascade realization form of a general transfer function assumes that the transfer function is the product of such second-order sections:
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 $z^{-1}$ coefficients $b_{i2}$ or $a_{i2}$ are zero, the actual numerator and denominator orders will be $L \leq 2K$ and $M \leq 2K$.

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.

![Cascade of second-order sections](image)

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 $i$th 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 $i$th section becomes the input to the $(i+1)$th section $H_{i+1}(z)$, that is,

$$x_{i+1}(n) = y_i(n), \quad i = 0, 1, \ldots, K - 1$$

Each section has its own internal state vector $w_i(n) = [w_{0i}(n), w_{1i}(n), w_{2i}(n)]$, $i = 0, 1, \ldots, K - 1$, where the numbers $w_{0i}(n)$, $w_{1i}(n)$ are the contents of the section's delay registers at the $n$th 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:

$$x_0(n) = x(n)$$

for $i = 0, 1, \ldots, K - 1$ do:

$$w_i(n) = x_i(n) - a_{i1} w_{i1}(n - 1) - a_{i2} w_{i2}(n - 2)$$

$$y_i(n) = b_{i0} w_{i0}(n) + b_{i1} w_{i1}(n - 1) + b_{i2} w_{i2}(n - 2)$$

$$x_{i+1}(n) = y_i(n)$$

$$y(n) = y_{K-1}(n)$$

(7.5.2)

It can be translated to the following sample processing algorithm:

for each input sample $x$ do:

$x_0 = x$

for $i = 0, 1, \ldots, K - 1$ do:

$$w_{0i}(n) = x_i(n) - a_{i1} w_{i1}(n - 1) - a_{i2} w_{i2}(n - 2)$$

$$y_i(n) = b_{i0} w_{i0}(n) + b_{i1} w_{i1}(n - 1) + b_{i2} w_{i2}(n - 2)$$

$$w_{1i}(n) = w_{i1}(n)$$

$$w_{2i}(n) = w_{i2}(n)$$

$$x_{i+1}(n) = y_i(n)$$

$$y(n) = y_{K-1}(n)$$

(7.5.3)

where, the internal state vector $w_i$ of the $i$th section is defined at time $n$ by:

$$w_{0i}(n) = w_i(n)$$

$$w_{1i}(n) = w_{i1}(n - 1), \quad \text{for} \quad i = 0, 1, \ldots, K - 1$$

$$w_{2i}(n) = w_{i2}(n - 2)$$

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 $i$th section. For example, if $K = 4$ as in Fig. 7.5.1, we define

$$A = \begin{bmatrix} 1 & a_{01} & a_{02} \\ 1 & a_{11} & a_{12} \\ 1 & a_{21} & a_{22} \\ 1 & a_{31} & a_{32} \end{bmatrix}, \quad B = \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \\ b_{30} & b_{31} & b_{32} \end{bmatrix}, \quad W = \begin{bmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \\ w_{20} & w_{21} & w_{22} \\ w_{30} & w_{31} & w_{32} \end{bmatrix}$$

$$H(z) = \prod_{i=0}^{K-1} H_i(z) = \prod_{i=0}^{K-1} \frac{b_{i0} + b_{i1} z^{-1} + b_{i2} z^{-2}}{1 + a_{i1} z^{-1} + a_{i2} z^{-2}}$$

(7.5.1)
7.5. CASCADE FORM

The \( i \)th rows of these matrices are the three-dimensional coefficient vectors and states of the \( i \)th section, that is,

\[
\begin{align*}
\mathbf{a}_i &= [1, a_{i1}, a_{i2}] \\
\mathbf{b}_i &= [b_{i0}, b_{i1}, b_{i2}], \quad \text{for } i = 0, 1, \ldots, K - 1 \\
\mathbf{w}_i &= [w_{i0}, w_{i1}, w_{i2}]
\end{align*}
\]  

(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 \texttt{sos}:

\[
\text{for each input sample } x \text{ do:} \\
\begin{align*}
y &= x \\
\text{for } i = 0, 1, \ldots, K - 1 \text{ do:} \\
y &= \text{sos} (\mathbf{a}_i, \mathbf{b}_i, \mathbf{w}_i, y)
\end{align*}
\]

(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 \texttt{cas.c}:

```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 \( K \times 3 \) 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 \( i \)th 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

```c
double **A, **B, **W;
A = (double **) calloc(K, sizeof(double *));
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:

```c
double A[K][3], B[K][3], W[K][3];
```

In that case, the declarations inside \texttt{cas} must also be modified to read:

```c
double A[][3], B[][3], W[][3];
```

The quantities \( A[i] \), \( B[i] \), \( W[i] \) are the \( i \)th 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 \texttt{cas}; this is accomplished indirectly by \texttt{callloc}. The usage of \texttt{cas} is the same as \texttt{can}; that is,

```c
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:

\[
H(z) = \frac{3 - 4z^{-1} + 2z^{-2}}{1 - 0.4z^{-1} + 0.5z^{-2}} = H_5(z)H_1(z)
\]

\[
= \frac{9 - 4z^{-2} + 4z^{-4}}{1 + 0.84z^{-2} + 0.25z^{-4}}
\]

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:

\[
\begin{align*}
w_0(n) &= x(n) + 0.4w_0(n-1) - 0.5w_0(n-2) \\
x_1(n) &= 3w_0(n) - 4w_0(n-1) + 2w_0(n-2) \\
w_1(n) &= x_1(n) - 0.4w_1(n-1) - 0.5w_1(n-2) \\
y(n) &= 3w_1(n) + 4w_1(n-1) + 2w_1(n-2)
\end{align*}
\]

where \( x_1(n) \) is the output of \( H_1(z) \) and the input to \( H_5(z) \). The corresponding sample processing algorithm is:

```c
for each input sample x do:
    w_0 = x + 0.4w_0 - 0.5w_0[n-2]
    w_1 = 3w_0 - 4w_1 + 2w_0[n-2]
    w_1 = x_1 - 0.4w_1 - 0.5w_1[n-2]
    y = 3w_1 + 4w_1[n-1] + 2w_1[n-2]
```

The coefficient and state matrices in the routine \texttt{cas} are in this case:

```c
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_0 & w_0 & w_0 \\ w_0 & w_1 & w_0 \\ w_1 & w_1 & w_1 \end{bmatrix}
```
For the canonical case, we have the coefficient vectors for the numerator and denominator polynomials:

\[ b = [9, 0, -4, 0, 4], \quad a = [1.00, 0.00, 0.84, 0.00, 0.25] \]

The difference equation at the input and output adders of Fig. 7.5.3 are:

\[
\begin{aligned}
    w(n) &= x(n) - 0.84w(n-2) - 0.25w(n-4) \\
    y(n) &= 9w(n) - 4w(n-2) + 4w(n-4)
\end{aligned}
\]

Defining the internal states as \( w_i(n) = w(n-i) \), \( i = 0, 1, 2, 3, 4 \), we find the sample processing algorithm:

```c
for each input sample x do:
    w0 = x - 0.84w2 - 0.25w4
    y = 9w0 - 4w2 + 4w4
    w4 = w3
    w3 = w2
    w2 = w1
    w1 = w0
```

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.](image)

**Example 7.5.2:** Consider the filter

\[
H(z) = \frac{1 + z^{-1} + z^{-2}}{1 - 0.7z^{-2}} = \frac{1 - z^{-2}}{1 - 0.6z^{-1} + 0.4z^{-2}} \frac{1 - z^{-1} + z^{-2}}{1 + 0.5z^{-1} + 0.3z^{-2}} = \frac{1 - z^{-6}}{1 - 0.1z^{-1} - 0.3z^{-2} + 0.09z^{-3} - 0.16z^{-4} - 0.014z^{-5} - 0.084z^{-6}}
\]

To illustrate the usage of the routines cas and can, we generated and filtered the following "step" input signal of length 100:

```c
for (n=0; n<100; n++) {
    ycasc[n] = cas(3, A, B, W, x[n]);
    ycan[n] = can(6, a, 6, b, w, x[n]);
}
```

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:

\[
b = [1, 0, 0, 0, 0, 0, -1] \\
a = [1, -0.1, -0.3, 0.09, -0.16, -0.014, -0.084]
\]

These quantities, as well as the cascade state matrix \( W \) and canonical internal state vector \( w \), 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:

```c
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 can 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, \( p = \pm 0.7 = \pm 0.8367 \). Therefore, the \( c = 1\% = 0.01 \) time constant...
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:

\[
(1 - p_1 z^{-1})(1 - p_1^* z^{-1}) = 1 - 2 \text{Re}(p_1) z^{-1} + |p_1|^2 z^{-2}
\]

\[
= 1 - 2R_1 \cos(\theta_1) z^{-1} + R_1^2 z^{-2}
\]

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 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:

\[
(1 - 0.9z^{-1})
\]

\[
(1 - (-0.5 + 0.7j) z^{-1})(1 - (-0.5 - 0.7j) z^{-1}) = (1 + z^{-1} + 0.74z^{-2})
\]

\[
(1 - (0.8 + 0.4j) z^{-1})(1 - (0.8 - 0.4j) z^{-1}) = (1 - 1.6z^{-1} + 0.8z^{-2})
\]

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}) 
(1 - (-0.7 + 0.4j)z^{-1})(1 - (-0.7 - 0.4j)z^{-1}) = (1 + 1.4z^{-1} + 0.65z^{-2})

Therefore, a possible pairing/ordering of SOS factors for $H(z)$ will be:

$$H(z) = \frac{1 - 0.9z^{-1}}{1 + 0.8z^{-1}}, \frac{1 + z^{-1} + 0.74z^{-2}}{1 + 1.4z^{-1} + 0.85z^{-2}} \cdot (1 - 1.6z^{-1} + 0.8z^{-2})$$

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, $h_2 = 0$. 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

$$1 + 0.96z^{-2} + 0.64z^{-4} = 0 \Rightarrow (z^{-2})^2 + 0.96(z^{-2}) + 0.64 = 0$$

which has two solutions:

$$z^2 = -0.96 \pm \sqrt{0.96^2 - 4 \cdot 0.64} = -0.48 \pm 0.64j$$

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:

$$(1 - (0.4 + 0.8j)z^{-1})(1 - (0.4 - 0.8j)z^{-1}) = 1 - 0.8z^{-1} + 0.8z^{-2}$$

Similarly, we find for the numerator polynomial:

$$1 - 0.48z^{-2} + 0.5476z^{-4} = 0 \Rightarrow z^2 = 0.24 \pm 0.7j$$

and taking square roots:

$$z = \pm \sqrt{0.24 \pm 0.7j} = \pm (0.7 \pm 0.5j)$$

The quadratic factors are:

$$(1 - (0.7 + 0.5j)z^{-1})(1 - (0.7 - 0.5j)z^{-1}) = 1 - 1.4z^{-1} + 0.74z^{-2}$$

$$(1 + (0.7 + 0.5j)z^{-1})(1 + (0.7 - 0.5j)z^{-1}) = 1 + 1.4z^{-1} + 0.74z^{-2}$$

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:

$$1 + z^{-8} = 0 \Rightarrow z^8 = -1 = e^{j\pi} = e^{j2\pi k / 8}$$

where we multiplied by $e^{j2\pi k / 8}$ for integer $k$. Taking eighth roots of both sides we find:

$$z_k = e^{j(2k + 1)\pi / 8}, \quad k = 0, 1, \ldots, 7$$

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:

$$(1 - z_0z^{-1})(1 - z_7z^{-1}) = 1 - 2 \cos\left(\frac{\pi}{8}\right)z^{-1} + z^{-2} = 1 - 1.8478z^{-1} + z^{-2}$$

$$(1 - z_1z^{-1})(1 - z_6z^{-1}) = 1 - 2 \cos\left(\frac{3\pi}{8}\right)z^{-1} + z^{-2} = 1 - 0.7654z^{-1} + z^{-2}$$

$$(1 - z_2z^{-1})(1 - z_5z^{-1}) = 1 - 2 \cos\left(\frac{5\pi}{8}\right)z^{-1} + z^{-2} = 1 + 0.7654z^{-1} + z^{-2}$$

$$(1 - z_3z^{-1})(1 - z_4z^{-1}) = 1 - 2 \cos\left(\frac{7\pi}{8}\right)z^{-1} + z^{-2} = 1 + 1.8478z^{-1} + z^{-2}$$

Similarly, the filter poles are the roots of the denominator:

$$1 - 0.0625z^{-8} = 0 \Rightarrow z^8 = 0.0625 = 0.0625e^{j2\pi k / 8} = (0.5)^4 e^{j2\pi k / 8}$$

which has the eight solutions:

$$p_k = \sqrt[8]{0.5}e^{j2\pi k / 8}, \quad k = 0, 1, \ldots, 7$$
Of these, \( p_0 = \sqrt{0.5} \) and \( p_4 = \sqrt{0.5}e^{i\pi/4} = -\sqrt{0.5} \) are real and may be paired together into one SOS. The rest are complex and can be paired in complex conjugate pairs: \( \{p_1, p_7\}, \{p_2, p_6\}, \{p_3, p_5\} \), resulting into the quadratic factors:

\[
(1 - p_0z^{-1})(1 - p_zz^{-1}) = (1 - \sqrt{0.5}z^{-1})(1 + \sqrt{0.5}z^{-1}) = 1 - 0.5z^{-2}
\]

\[
(1 - p_1z^{-1})(1 - p_7z^{-1}) = 1 - \sqrt{2}(\cos(\frac{2\pi}{8})z^{-1} + 0.5z^{-2}) = 1 - z^{-1} + 0.5z^{-2}
\]

\[
(1 - p_2z^{-1})(1 - p_6z^{-1}) = 1 - \sqrt{2}(\cos(\frac{4\pi}{8})z^{-1} + 0.5z^{-2}) = 1 + 0.5z^{-2}
\]

\[
(1 - p_3z^{-1})(1 - p_5z^{-1}) = 1 - \sqrt{2}(\cos(\frac{6\pi}{8})z^{-1} + 0.5z^{-2}) = 1 + z^{-1} + 0.5z^{-2}
\]

Finally, we obtain the factorization of \( H(z) \):

\[
H(z) = \frac{1 - 1.8478z^{-1} + z^{-2}}{1 - 0.5z^{-2}} \cdot \frac{1 - 0.7654z^{-1} + z^{-2}}{1 - z^{-1} + 0.5z^{-2}} \cdot \frac{1 + 0.7654z^{-1} + z^{-2}}{1 + 0.5z^{-2}} \cdot \frac{1 + 1.8478z^{-1} + z^{-2}}{1 + z^{-1} + 0.5z^{-2}}
\]

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 is more efficient than the cascade one, since it involves only one multiplier and an 8-fold delay. The canonical realization and the corresponding sample programming algorithm are shown in Fig. 7.6.2. Here, \( w = [w_0, w_1, \ldots, w_8] \) is the 9-dimensional internal state vector.

Example 7.6.4: Sharpens 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

\[
H_1(z) = \frac{N(z)}{N(p^{-1}z)} = \frac{1 + z^{-8}}{1 + p^{-1}z^{-8}}
\]

For example, we may choose \( p^8 = 0.94 \), or \( p = 0.9923 \). The factorization of \( N(p^{-1}z) \) into SOSs is obtained from that of \( N(z) \) by replacing \( z \) by \( z/p \) or \( z^{-1} \) by \(pz^{-1} \) in each factor. This gives:

\[
H_2(z) = \frac{1}{1 - 0.0625z^{-8}} \rightarrow H_2(z) = \frac{1 - r^8z^{-8}}{1 - R^8 z^{-8}}
\]

where \( r < R \). For example, we may choose \( r^8 = 0.96 \) or \( r = 0.9940 \), 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:

\[
H_2(z) = \frac{1 - r^8z^{-8}}{1 - R^8 z^{-8}} = \frac{1 - r^8z^{-2}}{1 - R^8 z^{-2}} \cdot \frac{1 - \sqrt{2}Rz^{-1} + R^2z^{-2}}{1 - \sqrt{2}Rz^{-1} + R^2z^{-2}}
\]

\[
\cdot \frac{1 + r^8z^{-2}}{1 + R^8 z^{-2}} \cdot \frac{1 + \sqrt{2}Rz^{-1} + R^2z^{-2}}{1 + \sqrt{2}Rz^{-1} + R^2z^{-2}}
\]

Thus, the new transfer function will be
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.

\[ H(z) = H_1(z)H_2(z) = \left[ \frac{1 + z^{-8}}{1 + 0.94z^{-8}} \right] \cdot \left[ \frac{1 - 0.96z^{-8}}{1 - 0.98z^{-8}} \right] \]

These convolutions may be implemented recursively by defining:

\[ d_i = a_0 * a_1 * \cdots * a_{i-1} \]

and noting that $d_i$ satisfies the recursion:

\[ d_{i+1} = d_i * a_i \quad (7.6.2) \]

where it must be initialized to $d_0 = \delta$, that is, a delta function, $d_0(n) = \delta(n)$. We may take $d_0$ to be the one-dimensional vector: $d_0 = [1]$. A few of the recursion steps will be:

\[
\begin{align*}
  d_1 &= d_0 * a_0 = \delta * a_0 = a_0 \\
  d_2 &= d_1 * a_1 = a_0 * a_1 \\
  d_3 &= d_2 * a_2 = (a_0 * a_1) * a_2 \\
  d_4 &= d_3 * a_3 = (a_0 * a_1 * a_2) * a_3, \text{ etc.}
\end{align*}
\]

The recursion ends at $i = K-1$ with the desired answer $a = d_K$. Note that the intermediate vector $d_i$ has order $2i$ and length $2i + 1$; similarly, the resulting vector $d_{i+1}$ has length $2i + 3$.

During the recursion (7.6.2), there is no need to save the intermediate vectors $d_i$. Therefore, the recursion can be stated in the following algorithmic form:

A variation, which helps the implementation in terms of the routine conv of Chapter 4, is to keep updating $a$ during each step:

\[
\begin{align*}
  a &= \delta \\
  \text{for } i = 0, 1, \ldots, K-1 \text{ do:} & \\
  d &= a_i \ast a \\
  a &= d
\end{align*}
\]

and similarly for the numerator vector $b$. The following C routine cas2can.c is an implementation of this algorithm:

```c
/* cas2can.c - cascade to canonical */
#include <stdlib.h> declares calloc

void conv();
```
7.6. CASCADE TO CANONICAL

void cas2can(K, A, a)
double **A, *a;
int K;
{
  int i,j;
double *d;
d = (double *) calloc(2*K+1, sizeof(double));
a[0] = 1;

  for(i=0; i<K; i++) {
    for(j=0; j<2*i+3; j++)
      a[j] = d[j];
    d = conv(2, A[i], 2*i+1, a, d);
  }
}

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 \( a \). 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:

\[
H(z) = \frac{1 - 0.9z^{-1}}{1 + 0.8z^{-1}} \cdot \frac{1 + z^{-1} + 0.74z^{-2}}{1 + 1.4z^{-1} + 0.65z^{-2}} \cdot \frac{1 - 1.6z^{-1} + 0.8z^{-2}}{1 + 0.9z^{-1}}
\]

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:

\[
cas2can(K, A, a); \quad \text{denominator coefficients}
\]
\[
cas2can(K, B, b); \quad \text{numerator coefficients}
\]

will return the vectors:

\[
a = [1, 2.2, 1.77, 0.52, 0, 0]
b = [1, -1.5, 0.48, -0.33, 0.9376, -0.5328]
\]

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.

![Typical DSP chip realization of a second-order section.](image)

Fig. 7.7.1

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:

\[
\text{for each input sample } x \text{ do:}
\]
\[
w_0 := x - a_1 w_1
\]
\[
w_0 := w_0 - a_2 w_2
\]
\[
y := b_1 w_1 + b_2 w_2
\]
\[
y := y + b_1 w_1
\]
\[
w_1 := w_0, \quad y := y + b_0 w_0
\]

(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 \( a_1 \) 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 \( b_1 \) coefficient to the lowest. In the general case, we can rewrite Eq. (7.2.5), where for simplicity we assumed \( L = M \):
7.7. HARDWARE REALIZATIONS AND CIRCULAR BUFFERS

where $T_{\text{instr}}$ is the time for a basic instruction, such as MAC or MACD. Recall from Section 4.3.4 that $T_{\text{instr}}$ is of the order of 30–80 nanoseconds, which corresponds to an instruction rate of $f_{\text{inst}} = 1/T_{\text{instr}} = 12.5$–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,

$$f_s = \frac{1}{T_{\text{proc}}} = \frac{1}{N_{\text{inst}}T_{\text{instr}}} = \frac{f_{\text{inst}}}{N_{\text{inst}}}$$ (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 $T_{\text{proc}}$ 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):

$$T_{\text{SOS}} \approx 5T_{\text{instr}}$$ (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:

$$N_{\text{instr}} = 5K + C$$ (K-section IIR filter) (7.7.8)

where $C$ is any additional overhead for the $K$-section filter. Therefore, the total processing time will be:

$$T_{\text{proc}} = (5K + C)T_{\text{instr}} = KT_{\text{SOS}} + CT_{\text{instr}}$$ (7.7.9)

Ignoring the possible small overhead term, we find the maximum sampling rate $f_s$ for implementing $K$ second-order sections:

$$f_s = \frac{1}{T_{\text{proc}}} = \frac{1}{KT_{\text{SOS}}} = \frac{f_{\text{inst}}}{5K}$$ (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 $T_{\text{SOS}}$ seconds. In this case, the total processing
7.7 HARDWARE REALIZATIONS AND CIRCULAR BUFFERS

7. DIGITAL FILTER REALIZATIONS

313

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 $T_{SOS}$ 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 Pipeline the operation of multiple DSP processors.](image)

At the $n$th 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 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 $z^{-K}$.

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, $T_{instr} = 4T_{clock}$. Therefore, its instruction rate is $f_{instr} = T_{clock}/4$. A typical MAC instruction represents two floating point operations: one addition and one multiplication. Therefore, the chip achieves a computational rate of $f_{FLOPS} = 2f_{instr} = f_{clock}/2$ FLOPS.

At a clock rate of $f_{clock} = 50$ MHz, it achieves an instruction rate of $f_{instr} = 50/4 = 12.5$ MIPS, and a computational rate of $f_{FLOPS} = 50/2 = 25$ MFLOPS (megaflops). The time per instruction is $T_{instr} = 1/f_{instr} = 1/12.5 = 80$ nanoseconds.

An order-$M$ FIR filter can be implemented (with in-line code) with

$$N_{instr} = (M + 1) + 11 = M + 12 \quad \text{(instructions per sample)}$$

Therefore, the processing time per sample will be

$$T_{proc} = (M + 12)T_{instr}$$
7. DIGITAL FILTER REALIZATIONS

7.7. HARDWARE REALIZATIONS AND CIRCULAR BUFFERS

for each input sample x do:
   for i = 1, 2, ..., M determine states:
      si = tap(M, w, p, i)
   s0 = x - a1s1 - ... - aMsm
   y = b0s0 + b1s1 + ... + bMsm
   *p = s0
   cdelay(M, 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:
   s1 = tap(4, w, p, 1)
   s2 = tap(4, w, p, 2)
   s3 = tap(4, w, p, 3)
   s4 = tap(4, w, p, 4)
   s0 = x - 0.2s1 + 0.3s2 - 0.5s4
   y = 2s0 - 3s1 + 4s3
   *p = s0
   cdelay(4, w, &p)

The statement *p = s0 puts the computed value of the 0th component s0 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 w is nine-dimensional. The algorithm is stated below, where only the output s8 of the 8-fold delay line is needed:

for each input sample x do:
   s8 = tap(8, w, p, 8)
   s0 = x + 0.0625s8
   y = s0 + s8
   *p = s0
   cdelay(8, w, &p)

Note that the output s8 was available before the input s0 could be computed.

Example 7.7.4: The input signal x = [1, 3, 2, 5, 4, 6] is applied to the filter:

\[ H(z) = \frac{1 + z^{-1} + 2z^{-2}}{1 - z^{-3}} \]
7.7. HARDWARE REALIZATIONS AND CIRCULAR BUFFERS

Draw the canonical realization and write its circular buffer sample processing algorithm. Iterate the algorithm nine times for \( n = 0, 1, \ldots, 8 \) and compute the corresponding output \( y(n) \). Make a table of the circular buffer entries \( w \) and the filter’s internal states \( s \).

**Solution:** The block diagram realization and its circular sample processing algorithm are:

```
for each input sample x do:
    s1 = tap(3, w, p, 1)
    s2 = tap(3, w, p, 2)
    s3 = tap(3, w, p, 3)
    s0 = x + s3
    y = s0 + s1 + 2s2
    *p = s0
delay(3, w, &p)
```

The entries of the linear buffer \( w = \{w_0, w_1, w_2, w_3\} \) over which the circular pointer circulates are shown in the following table.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( x )</th>
<th>( w_0 )</th>
<th>( w_1 )</th>
<th>( w_2 )</th>
<th>( w_3 )</th>
<th>( s_0 )</th>
<th>( s_1 )</th>
<th>( s_2 )</th>
<th>( s_3 )</th>
<th>( y = s_0 + s_1 + 2s_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>6</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>2</td>
<td>3</td>
<td>7</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>2</td>
<td>2</td>
<td>27</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>6</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>28</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>29</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>27</td>
</tr>
</tbody>
</table>
7. HARDWARE REALIZATIONS AND CIRCULAR BUFFERS

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 \( w_0 \) and \( w_1 \) of the two sections, that is, \( p_0 = w_0 \) and \( p_1 = w_1 \). The sample processing algorithm is:

\[
\text{for each input sample } x \text{ do:}
\]
\[
s_1 = \text{tap}(2, w, p, 1)
\]
\[
s_2 = \text{tap}(2, w, p, 2)
\]
\[
s_0 = x - a_1 s_1 - a_2 s_2
\]
\[
y = b_0 s_0 + b_1 s_1 + b_2 s_2
\]
\[
*p = s_0
\]
\[
\text{cdelay}(2, w, &p)
\]

where \( a_k \) read input sample \( x \)

\[
\text{for } i = 0, 1, \ldots, K - 1 \text{ do:}
\]
\[
y = x
\]
\[
y = \text{cos}(a_i, b_i, w_i, &p_i, y)
\]

As in Eq. (7.5.5), the cascade of \( K \) second-order sections can be realized by \( K \) successive calls to the routine \( \text{cos} \):

\[
\text{for each input sample } x \text{ do:}
\]
\[
s_1 = \text{tap}(2, w_0, p_0, 1)
\]
\[
s_2 = \text{tap}(2, w_0, p_0, 2)
\]
\[
s_0 = x + 0.4 s_1 - 0.5 s_2
\]
\[
y = 3 s_0 + 4 s_1 + 2 s_2
\]
\[
*p_0 = s_0
\]
\[
\text{cdelay}(2, w_0, &p_0)
\]
\[
s_1 = \text{tap}(2, w_1, p_1, 1)
\]
\[
s_2 = \text{tap}(2, w_1, p_1, 2)
\]
\[
s_0 = x_1 - 0.4 s_1 - 0.5 s_2
\]
\[
y = 3 s_0 + 4 s_1 + 2 s_2
\]
\[
*p_1 = s_0
\]
\[
\text{cdelay}(2, w_1, &p_1)
\]

where the output \( x_i \) of the first section becomes the input to the second. \( \square \)

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 \( \text{FIR} \) routine \( \text{cfir2} \) of Section 4.3.4. The following routine \( \text{ccan2} \) replaces \( \text{ccan} \):

\[
\text{for each input sample } x \text{ do:}
\]
\[
s_1 = \text{tap}(2, w_0, p_0, 1)
\]
\[
s_2 = \text{tap}(2, w_0, p_0, 2)
\]
\[
s_0 = x - a_1 s_1 - a_2 s_2
\]
\[
y = b_0 s_0 + b_1 s_1 + b_2 s_2
\]
\[
*p = s_0
\]
\[
\text{cdelay}(2, w_0, &p)
\]

As in the \( \text{cas} \), we save the individual buffers \( w_i \) as the rows of the matrix \( W \). Similarly, we save the individual pointers \( p_i \) in an \( \text{array} \) 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 \( \text{cas} \) is illustrated below:

\[
\text{array of } K \text{ pointers}
\]
\[
\text{for } (n = 0; n < Ntot; n++)
\]
\[
y[n] = \text{ccas}(K, A, B, W, x[n]);
\]

Its usage is illustrated by the following program segment. Note that \( q \) must be passed by address:

\[
\text{for each input sample } x \text{ do:}
\]
\[
s_1 = \text{tap}(2, w, p, 1)
\]
\[
s_2 = \text{tap}(2, w, p, 2)
\]
\[
s_0 = x - a_1 s_1 - a_2 s_2
\]
\[
y = b_0 s_0 + b_1 s_1 + b_2 s_2
\]
\[
*p = s_0
\]
\[
\text{cdelay}(2, w, &p)
\]

where each of the \( K \) sections has its own three-dimensional buffer \( w_i \) and corresponding circular pointer \( p_i \). The following routine \( \text{ccas} \) is an implementation, replacing \( \text{cas} \):

\[
\text{for each input sample } x \text{ do:}
\]
\[
\text{Solution:}
\]
\[
\text{Let } p_0 \text{ and } p_1 \text{ denote the circular pointers of the two sections. Initially they point at the first elements of the three-dimensional buffers } w_0 \text{ and } w_1 \text{ of the two sections, that is, } p_0 = w_0 \text{ and } p_1 = w_1. \text{ The sample processing algorithm is:}
\]
\[
\text{for each input sample } x \text{ do:}
\]
\[
y = x
\]
\[
\text{for } i = 0, 1, \ldots, K - 1 \text{ do:}
\]
\[
y = \text{cos}(a_i, b_i, w_i, &p_i, y)
\]
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:

```c
/* 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;
}
```

```
/* ccas2.c - circular buffer implementation of cascade realization */
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);  
    return y;
}
```

The i-th 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:

```c
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 ≥ 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:

![Diagram of delta-sigma A/D converter]

\[ e(n) = \text{quantization noise} \]

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,

\[ H_s(z) = z^{-1} \quad \text{and} \quad H_e(z) = (1 - z^{-1})^2 \]

7.3 A digital filter has transfer function:

\[ H(z) = \frac{z^{-1} (1 + 2z^{-2}) (1 + 3z^{-2})}{1 - z^{-6}} \]
7.8. PROBLEMS

7. DIGITAL FILTER REALIZATIONS

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:

\[
\begin{align*}
v(n) &= x(n) + v(n - 1) \\
y(n) &= v(n) + v(n - 2) + v(n - 4)
\end{align*}
\]

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:

\[
w(n) = 0.75w(n - 10) + x(n) \\
y(n) = -0.75w(n) + w(n - 10)
\]

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 \( \omega \).

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:
  w0 = x + 0.64w4
  y = w0 + w3
  w4 = w2
  w3 = w2
  w2 = w1
  w1 = w0
```

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,

\[
H(z) = (1 + z^{-2})^3, \quad H(z) = \frac{1}{1 + 0.81z^{-2}}, \quad H(z) = \frac{1 - z^{-4}}{1 - 0.9z^{-1}}
\]

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 response \( |H(\omega)| \) versus \( \omega \).

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.
7.8. PROBLEMS

7.12 Consider a stable system with transfer function \( H(z) = \frac{1}{1 + \frac{1}{16} z^{-4}} \).

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 \( \omega \), 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

\[ H_0(z) = \frac{0.313(1 + z^{-1})}{1 - 0.373z^{-1}}, \quad H_1(z) = \frac{0.147(1 + 2z^{-1} + z^{-2})}{1 - 1.122z^{-1} + 0.712z^{-2}} \]

\[ H_2(z) = \frac{0.117(1 + 2z^{-1} + z^{-2})}{1 - 0.891z^{-1} + 0.360z^{-2}}, \quad H_3(z) = \frac{0.103(1 + 2z^{-1} + z^{-2})}{1 - 0.780z^{-1} + 0.190z^{-2}} \]

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:

\[ x(n) = \begin{cases} 1 & \text{if } 0 \leq n < 50 \\ 0 & \text{if } 50 \leq n < 100 \end{cases} \]

Using the cascade routine cas compute the filter output \( y_a \) for \( 0 \leq n \leq 99 \). Repeat using the routines dir and can. In three parallel columns, print the signal samples \( y_a \) computed by the three routines cas, dir, and can.

d. On the same graph, plot the two signals \( x_a \) and \( y_a \) 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 \( 0 \leq n < 50 \), 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:

\[ \text{canfilt a.dat b.dat < x.dat > y.dat} \]

It must read and dynamically allocate the denominator and numerator coefficient vectors \( a \) and \( b \) from two input files, say a.dat and b.dat, and must allocate the internal state vector \( w \). Using the routine can, 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:

\[ \text{casfilt A.dat B.dat < x.dat > y.dat} \]

It must read and dynamically allocate the denominator and numerator coefficient matrices \( A \) and \( B \) from two input files, say A.dat and B.dat (stored in row-wise fashion). Using the routine cas, 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:

\[ y = \text{casfilt}(B, A, x); \]

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:

\[ x(n) = s(n) + v(n), \quad \text{where} \quad s(n) = A_0 \cos(w_0 n) + A_1 \cos(w_1 n) \cos(w_2 n) \]

where \( n = 0, 1, \ldots, 499 \), 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 \( n = 0, 1, \ldots, 499 \) 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 \( 0 \leq n < 200 \).
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: $H(z) = \frac{6 - 2z^{-3}}{1 - 0.5z^{-1}}$.

a. Draw the canonical realization form of $H(z)$ and write the corresponding sample processing algorithm in both 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 $n = 0, 1, \ldots, 6$ filling the entries of the following table:

<table>
<thead>
<tr>
<th>$n$</th>
<th>$x$</th>
<th>$w_0$</th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
<th>$s_0$</th>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$s_3$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

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 $x = \{1, 2, 3\}$ is applied. Compute the corresponding output for $n = 0, 1, \ldots, 6$ 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:

$$1 + a_1 z^{-1} + a_2 z^{-2} = (1 - p z^{-1})(1 - p^* z^{-1})$$

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:

$$dp = \frac{-pd_{a_1} + da_2}{p - p^*}$$

7.21 Consider a fourth-order denominator polynomial for an IIR filter with conjugate poles:

$$1 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} + a_4 z^{-4} = (1 - p z^{-1})(1 - p^* z^{-1})(1 - p_z z^{-1})(1 - p_{z^*} z^{-1})$$

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 $p_z$ is close to $p$:

$$dp_{p_0} = \frac{-p_0 d_{a_1} + p_0^* d_{a_2} + p_0 d_{a_3} + da_4}{(p_0 - p_z)(p_0 - p_{z^*})}$$

$$dp_{p_1} = \frac{-p_1 d_{a_1} + p_1^* d_{a_2} + p_1 d_{a_3} + da_4}{(p_1 - p_z)(p_1 - p_{z^*})}$$

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 $y$.

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:

$$y = \text{transp}(w, a, b, v, x);$$

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 transposed 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 $y_D(n)$. 
7.8. PROBLEMS

329

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?

7.28 The three block diagrams (a,b,c) shown below are equivalent descriptions of the same low-pass reverberator filter of the form:

\[ H(z) = \frac{1}{1 - G(z)z^{-4}} \]

where \( G(z) = \frac{0.2}{1 - 0.8z^{-1}} \)

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:

\[ z^{-4} \rightarrow \frac{z^{-4}}{1 - 0.8z^{-4}} \]

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 \( H(z) = \frac{B(z)}{A(z)} \) 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.](image)

In its simpler, standard, form, each section contains one delay element \( z^{-1} \) and one multiplier \( y \) referred to as a “reflection coefficient”, whereas the normalized form contains two multipliers, the reflection coefficient \( y \) and its associated “transmission” coefficient, \( \tau = \sqrt{1 - y^2} \).

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 to 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 \( y \)s and \( \tau \)s) 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 \( y \)-coefficients as reflection coefficients arises from this wave propagation context, where the \( y \)s 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 \( y \)-coefficients are also known in this context as “partial-correlation” coefficients (PARCOR).

The layer recursions—are 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 \( y \)-coefficients extracted from \( H(z) \) by applying the layer recursions have magnitude less than one, \(|y| < 1\).

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.
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,

\[ H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}}{1 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}} \]  

(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, \(y_1, y_2, y_3\), and the ladder coefficients, \(d_0, d_1, d_2, d_3\).

![Fig. 8.3.1 Standard lattice realization.](image)

To derive the lattice realization for the filter, \(H(z) = \frac{B(z)}{A(z)}\), 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 inputs, \(f(n)\), and \(r(n)\), as depicted in the upper part of Fig. 8.3.2, so that the overall input to this structure is \(f(n)\)

and \(r(n)\), for, \(p = 1, 2, \ldots, M\).

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 \(2 \times 2\) transition matrices, for example, we have for \(M = 3\),

\[ \begin{bmatrix} F_3(z) \\ R_3(z) \end{bmatrix} = \begin{bmatrix} 1 & -y_3 z^{-1} \\ -y_3 & z^{-1} \end{bmatrix} \begin{bmatrix} 1 & -y_2 z^{-1} \\ -y_2 & z^{-1} \end{bmatrix} \begin{bmatrix} 1 & -y_1 z^{-1} \\ -y_1 & z^{-1} \end{bmatrix} \begin{bmatrix} F_0(z) \\ R_0(z) \end{bmatrix} \]

or, since we assumed a common input, \(F_0(z) = R_0(z)\),

\[ \begin{bmatrix} F_3(z) \\ R_3(z) \end{bmatrix} = \begin{bmatrix} 1 & -y_3 z^{-1} \\ -y_3 & z^{-1} \end{bmatrix} \begin{bmatrix} 1 & -y_2 z^{-1} \\ -y_2 & z^{-1} \end{bmatrix} \begin{bmatrix} 1 & -y_1 z^{-1} \\ -y_1 & z^{-1} \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} F_0(z) \]
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,

$$A_3(z) = \frac{F_3(z)}{F_0(z)}, \quad A_3^R(z) = \frac{R_3(z)}{R_0(z)} = \frac{R_1(z)}{F_0(z)},$$

where, $A_3(z), A_3^R(z)$, will be the two order-3 polynomials resulting from the matrix operation,

$$A_3(z) = \begin{bmatrix} 1 & -y_3 z^{-1} & 1 & -y_1 z^{-1} & 1 \\ -y_3 & -y_2 z^{-1} & -y_1 z^{-1} & z^{-1} \\ \end{bmatrix}$$

(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,

$$A(z) = a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}, \quad a = [a_0, a_1, a_2, a_3]$$

$$A^R(z) = a_3 + a_2 z^{-1} + a_1 z^{-2} + a_0 z^{-3}, \quad a^R = [a_3, a_2, a_1, a_0]$$

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 $F(z)$. Although Eq. (8.3.6) establishes a relationship between the polynomial coefficients, $\{a_1, a_2, a_3\}$, and the reflection coefficients $\{y_1, y_2, y_3\}$, 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,

$$A_2(z) = \begin{bmatrix} 1 & -y_2 z^{-1} & 1 & -y_1 z^{-1} & 1 \\ -y_2 & -y_1 z^{-1} & -y_1 z^{-1} & z^{-1} \\ \end{bmatrix}$$

$$A_1(z) = \begin{bmatrix} 1 & -y_1 z^{-1} & 1 \\ -y_1 & -y_1 z^{-1} & 1 \\ \end{bmatrix}$$

(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,

$$\begin{bmatrix} A_3(z) \\ A_3^R(z) \end{bmatrix} = \begin{bmatrix} 1 & -y_3 z^{-1} & 1 & -y_1 z^{-1} & 1 \\ \end{bmatrix}$$

$$\begin{bmatrix} A_2(z) \\ A_2^R(z) \end{bmatrix} = \begin{bmatrix} 1 & -y_2 z^{-1} & 1 & -y_1 z^{-1} & 1 \\ \end{bmatrix}$$

$$\begin{bmatrix} A_1(z) \\ A_1^R(z) \end{bmatrix} = \begin{bmatrix} 1 & -y_1 z^{-1} & 1 \\ \end{bmatrix}$$

(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 $p = 1, 2, \ldots, M$, starting with, $A_0(z) = A_0^R(z) = 1$,

$$\begin{bmatrix} A_p(z) \\ A_p^R(z) \end{bmatrix} = \begin{bmatrix} 1 & -y_p z^{-1} & 1 & -y_{p-1} z^{-1} & \ldots & -y_1 z^{-1} & \ldots & 1 \\ \end{bmatrix}$$

(8.3.9)

(8.3.10)

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 $p$th lattice section, while $A_p^R(z)$ is the transfer function to the $R_p(z)$ output, for $p = 1, 2, \ldots, M$,

$$F_p(z) = A_p(z) F_0(z)$$

$$R_p(z) = A_p^R(z) F_0(z)$$

In the time-domain, the forward Levinson recursion (8.3.9) reads as follows, building an order-$p$ filter from an order-$(p-1)$ one,

$$a_0 = \begin{bmatrix} a_0 \\ 0 \\ \end{bmatrix} = -y_p \begin{bmatrix} 0 \\ 0 \\ \vdots \\ a_{p-1} \end{bmatrix}, \quad p = 1, 2, \ldots, M \quad (\text{forward Levinson})$$

(8.3.11)

where $a_p = [a_0, a_1, a_2, \ldots, a_p]^T$ is the column vector of the coefficients of $A_p(z)$, starting with $a_0 = a_0^R = [1]$. Explicitly, we have,

$$\begin{bmatrix} 1 \\ a_{p-1} \\ a_{p-2} \\ \vdots \\ a_{p-M+1} \end{bmatrix} = \begin{bmatrix} 1 \\ a_{p-1,1} \\ a_{p-1,2} \\ \vdots \\ a_{p,1} \end{bmatrix} \Rightarrow \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ \end{bmatrix}$$

(8.3.12)

The inverse of this recursion is the backward Levinson recursion, which starts with the final order-$M$ filter, $a_M = [1, a_M, a_{M-1}, \ldots, a_{M-1}]^T$, and proceeds downwards to order-0, extracting the reflection coefficients $y_p$ in the process, for, $p = M, M-1, \ldots, 1$,

$$y_p = -a_{p,p}$$

(8.3.12)

or, explicitly,

$$\begin{bmatrix} 1 \\ a_{p-1,1} \\ a_{p-1,2} \\ \vdots \\ a_{p-M+1,1} \end{bmatrix} = \begin{bmatrix} 1 \\ a_{p-1,1} \\ a_{p-1,2} \\ \vdots \\ a_{p-M+1,1} \end{bmatrix} + y_p \begin{bmatrix} 1 \\ a_{p-1,1} \\ a_{p-1,2} \\ \vdots \\ a_{p-M+1,1} \end{bmatrix}$$

(8.3.13)

where by construction, the bottom entry of $a_{p-1}$ 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,
8.3. STANDARD, REARRANGED, AND NORMALIZED LATTICE

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),

$$f_p(n) = f_{p-1}(n) - y_p r_{p-1}(n-1) \Rightarrow f_{p-1}(n) = f_p(n) + y_p r_{p-1}(n-1)$$

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 $p = 0, 1, \ldots, M$.

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 $p$th lattice section is, for $p = 0, 1, \ldots, M$,

$$R_p(z) = \frac{A_p^R(z)}{A_M(z)} F_0(z) = \frac{A_p^R(z)}{A_M(z)} \frac{A_M(z)F_0(z)}{A_M(z)} = \frac{A_p^R(z)}{A_M(z)} X(z) \quad (8.3.15)$$

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,

$$Y(z) = \sum_{p=0}^{M} d_p R_p(z) = \frac{\sum_{p=0}^{M} d_p A_p^R(z)}{A_M(z)} X(z) \quad (8.3.16)$$

thus, resulting in the overall transfer function from $X(z)$ to $Y(z)$,

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{p=0}^{M} d_p A_p^R(z)}{A_M(z)} = \frac{B_M(z)}{A_M(z)} \quad (8.3.17)$$

where the numerator polynomial $B_M(z)$ is expressible as a linear combination of the reversed polynomials, $A_p^R(z)$,

$$B_M(z) = \sum_{p=0}^{M} d_p A_p^R(z) \quad (8.3.18)$$

This can be written in the time domain in the matrix form,

$$b_M = \sum_{p=0}^{M} d_p a_p^R = [a_0^R, a_1^R, \cdots, a_M^R] \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_M \end{bmatrix} = A d \quad (8.3.19)$$

where $A$ is a unit upper-tridiagonal matrix consisting of the reversed prediction polynomials. For example, for $M = 3$, we have,

$$A = \begin{bmatrix} a_0^R & a_1^R & a_2^R \\ 0 & a_1^R & a_2^R \\ 0 & 0 & a_3^R \end{bmatrix} = \begin{bmatrix} 1 & a_{11} & a_{22} & a_{33} \\ 0 & 1 & a_{21} & a_{32} \\ 0 & 0 & 1 \end{bmatrix}$$

and Eq. (8.3.19) becomes,

$$b = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} 1 & a_{11} & a_{22} & a_{33} \\ 0 & 1 & a_{21} & a_{32} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix} = Ad \quad (8.3.20)$$

Solving for the ladder weights in terms of the numerator coefficients $b$ can be done efficiently with MATLAB’s backslash operator,

$$d = A^{-1}b = A \backslash b \quad (8.3.21)$$

To summarize, given the numerator and denominator coefficients, $b$, $a$, of a transfer function, one subjects the numerator coefficients $a$ to the backward Levinson recursions generating the reflection coefficients $y_p$ 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 $y_p$ and $d_p$ coefficients are given, one subjects the $y_p$’s to the forward Levinson recursion generating all the prediction polynomials, gathered in the matrix $A$, and then constructs the numerator coefficients $b$ 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 $p$th 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,

$$f_{p-1}(n) = f_p(n) + y_p w_p(n) \quad \text{for } p = M-1, M-2, \ldots, 1, do,$$

$$r_p(n) = w_p(n) - y_p f_{p-1}(n)$$

$$w_{p+1}(n+1) = r_p(n) \quad \text{end}$$

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,
\[
\begin{align*}
  f_M(n) &= x(n) \\
  f_{M-1}(n) &= f_M(n) + Mw_M(n) \\
  r_M(n) &= w_M(n) - Mf_{M-1}(n) \\
  y(n) &= d_Mr_M(n)
\end{align*}
\]
for each input sample \( x \), do,
\[
\begin{align*}
  f &= x \\
  f &= f + yMw_M \\
  r &= w_M - yMf \\
  y &= dMr_M(n)
\end{align*}
\]

so that upon exit from the \( n \)th time pass, we have accumulated all of the output terms,
\[
y(n) = d_0r_0(n) + d_1r_1(n) + \cdots + d_Mr_M(n)
\]

**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, \( \tau_p^2 = 1 - \gamma_p^2 \).

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,
\[
\begin{align*}
  f_p(n) &= f_p(n) + \gamma_p wp(n) \\
  w_p(n) &= r_p(n) - \gamma_p f_{p-1}(n)
\end{align*}
\]

that is,
\[
\begin{align*}
  f_{p-1}(n) &= f_p(n) + y_p w_p(n) \\
  r_p(n) &= w_p(n) - y_p f_{p-1}(n) \\
  &= w_p(n) - y_p \left[ f_p(n) + y_p w_p(n) \right] \\
  &= (1 - \gamma_p^2) w_p(n) - y_p f_{p-1}(n) \\
  &= \tau_p^2 w_p(n) - y_p f_{p-1}(n), \quad \text{where } \tau_p = \sqrt{1 - \gamma_p^2}, \quad \tau_p^2 = 1 - \gamma_p^2
\end{align*}
\]
Thus, the \( p \)th section is now described as follows, for, \( p = M, M - 1, \ldots, 1 \),

\[
\begin{align*}
\bar{y}_p &= y_p w_p(n) \\
\bar{f}_{p-1}(n) &= f_{p-1}(n) - y_p \bar{w}_p(n) \\
r_p(n) &= \tau_p^2 \bar{f}_p(n) - y_p \bar{f}_p(n) \\
d_p &= \frac{\sigma_M}{\sigma_p} c_p
\end{align*}
\]  

(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, \( \tau_p^2 = 1 - y_p^2 \), multiplying the \( w_p(n) \) signal is shared equally between the right-going signal \( \bar{w}_p(n) \) and the upper left-going signal \( \bar{f}_p(n) \). The resulting \( p \)th section with the rescaled signals, \( \bar{f}_p(n), \bar{r}_p(n), \bar{w}_p(n) \), becomes then,

\[
\begin{align*}
\bar{w}_p(n) &= \bar{f}_{p-1}(n) - y_p \bar{f}_p(n) \\
\bar{f}_{p-1}(n) &= \tau_p \bar{f}_p(n) + y_p \bar{w}_p(n) \\
\bar{r}_p(n) &= \tau_p \bar{w}_p(n) - y_p \bar{f}_p(n)
\end{align*}
\]  

(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 \( c_p \).

\[
\begin{align*}
\bar{w}_p(n) &= \bar{f}_{p-1}(n) - y_p \bar{f}_p(n) \\
\bar{f}_{p-1}(n) &= \tau_p \bar{f}_p(n) + y_p \bar{w}_p(n) \\
\bar{r}_p(n) &= \tau_p \bar{w}_p(n) - y_p \bar{f}_p(n)
\end{align*}
\]  

(8.3.26)

where,

\[
\begin{align*}
\tau_p &= (1 - y_p^2)^{1/2}, \quad p = 1, 2, \ldots, M \\
\sigma_0 &= 1 \\
\sigma_p &= \tau_1 \tau_2 \cdots \tau_p = \tau_p \sigma_{p-1}, \quad p = 1, 2, \ldots, M
\end{align*}
\]

In the normalized form, the overall input and output signals, \( x(n), y(n) \), will have z-transforms, noting that, \( R_p(z) = A_p^R(z) F_0(z) \),

\[
\begin{align*}
X(z) &= \sum_{p=0}^{M} c_p R_p(z) = \sum_{p=0}^{M} c_p \sigma_p^{-1} R_p(z) = \sum_{p=0}^{M} c_p \sigma_p^{-1} A_p^R(z) F_0(z) \\
Y(z) &= \sigma_0^{-1} M \sum_{p=0}^{M} c_p R_p(z) = \sigma_0^{-1} M \sum_{p=0}^{M} c_p \sigma_p^{-1} A_p^R(z) F_0(z)
\end{align*}
\]

so that the resulting transfer function will be,

\[
H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{p=0}^{M} c_p \sigma_p^{-1} A_p^R(z) F_0(z)}{\sigma_0^{-1} M A_M(z) F_0(z)}
\]

or,

\[
H(z) = \frac{\sum_{p=0}^{M} c_p \sigma_p^{-1} A_p^R(z) A_M(z)}{A_M(z)} = B_M(z)
\]  

(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),

\[
d_p = c_p \sigma_p^{-1} \sigma_M, \quad p = 0, 1, \ldots, M
\]  

(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,
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, \texttt{dir2nl} and \texttt{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,

\[
\begin{align*}
\texttt{nl2dir.m} & \quad \texttt{gamma,d} -> \texttt{b,a} \\
M & = \text{length}(g); \\
d & = d(:); g = g(:); \\
a & = 1; A = \text{eye}(M+1); \\
& \text{for } p = 1:M, \\
& \quad a = [a; 0] - g(p)^{*}[0; \text{flip}(a)]; \\
& \quad A(1:p+1,p+1) = \text{flip}(a); \\
& \quad A(1:M+1,M+1) = \text{flip}(a); \\
& \quad \text{end} \\
& \quad b = A^d; \\
\end{align*}
\]

\[
\begin{align*}
\texttt{dir2nl.m} & \quad \texttt{b,a} -> \texttt{gamma,d} \\
& \quad g = \text{zeros}(M,1); \\
& \quad A(1:M+1,M+1) = \text{flip}(a); \\
& \quad \text{for } p = 1:M+1, \\
& \quad \quad g(p) = -a(end); \\
& \quad \quad a = (a + g(p)^{*}\text{flip}(a))/(1-g(p)^2); \\
& \quad \quad A(1:p,p) = \text{flip}(a); \\
& \quad \quad A(1:M+1,M+1) = \text{flip}(a); \\
& \quad \quad \text{end} \\
& \quad d = A^b; \\
\end{align*}
\]

The essential code in these functions is given below. We note also that the built-in MATLAB functions, \texttt{tf2latc} and \texttt{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, $k_p = -y_p$. The ladder coefficients $d_p$ are the same.

8.5 Filtering in Lattice Realizations

The filtering operations in the lattice realization are implemented by the following MATLAB function, \texttt{nlfilt}, either in the normalized or the standard forms. Its inputs and outputs are described below.
8.5 Filtering in Lattice Realizations

% 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,
        r = tr(p)*w(p) - g(p)*f;
        f = tf(p)*f + g(p)*w(p);
        if p>M, w(p+1) = r; end
        y(n) = y(n) + d(p+1)*r;
    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,

$$[b,a] = \text{lat2dir}(g,d,\text{type});$$

$$\omega = \ldots \% \text{define desired vector of digital frequencies [rads/sample]}$$

$$H = \text{freqz}(b,a,\omega);$$

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; fliplr(a)];
B = B + d(p+1) * freqz(fliplr(a),1,w);
end
A = freqz(a,1,w); % denominator DTFT
H = B./A; % frequency response H(\omega)

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 $p = 1, 2, \ldots, M$. 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,

\[ a_3 = \begin{bmatrix} 1 \\ 0.1 \\ -0.26 \end{bmatrix}, \quad b_3 = \begin{bmatrix} 7 \\ -1.14 \\ -3.9 \\ 1 \end{bmatrix} \]

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 \( y_1, y_2, y_3 \) in the process, and also express \( B_3(z) \) as a linear combination of the reversed polynomials in the form,

\[ B_3(z) = d_0 A_0^R(z) + d_1 A_1^R(z) + d_2 A_2^R(z) + d_3 A_3^R(z) \]  

(8.8.1)

The steps are as follows. First we extract \( y_3 \) and calculate \( a_2 \),

\[ y_3 = -a_{33} = -0.6 \Rightarrow a_2 = \frac{a_3 + y_3 a_3^R}{1 - y_3^2} = \begin{bmatrix} 1 \\ 0.1 \\ -0.26 \end{bmatrix} + \begin{bmatrix} 0.6 \\ -0.26 \\ 0.1 \end{bmatrix} \begin{bmatrix} 1 \\ -0.6 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.4 \\ -0.5 \\ 0 \end{bmatrix} \]

and after deleting the bottom zero in \( a_2 \), we repeat the recursion to get \( y_2 \) and \( a_1 \),

\[ y_2 = -a_{22} = 0.5 \Rightarrow a_1 = \frac{a_2 + y_2 a_2^R}{1 - y_2^2} = \begin{bmatrix} 1 \\ 0.4 \\ -0.5 \end{bmatrix} + \begin{bmatrix} 0.5 \\ 0.4 \end{bmatrix} \begin{bmatrix} 1 \\ -0.5 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.8 \\ 0 \end{bmatrix} \]

and after deleting the bottom zero of \( a_1 \), we have,

\[ a_1 = \begin{bmatrix} 1 \\ 0.8 \end{bmatrix} \Rightarrow y_1 = -a_{11} = -0.8 \]

thus, \([y_1, y_2, y_3] = [-0.8, 0.5, -0.6] \). The feed-forward numerator coefficients \( d_p \) are calculated by solving the equation (8.3.20), which reads here,

\[ \begin{bmatrix} 1 & 0.8 & -0.5 & 0.6 \\ 0 & 1 & 0.4 & -0.26 & d_0 \\ 0 & 0 & 1 & 0.1 & d_1 \\ 0 & 0 & 0 & 1 & d_2 \end{bmatrix} = \begin{bmatrix} 7 \\ 1.14 \\ -3.9 \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ -4 \end{bmatrix} \]

with the solution obtained easily with the backward substitution. Finally, we verify Eq. (8.8.1)

\[ B_3(z) = d_0 A_0^R(z) + d_1 A_1^R(z) + d_2 A_2^R(z) + d_3 A_3^R(z) \]

\[ = 2 + 3(0.8 + z^{-1}) + (-4)(-0.5 + 0.4z^{-1} + z^{-2}) + (0.6 - 0.26z^{-1} + 0.1z^{-2} + z^{-3}) \]

\[ = 7 + 1.14z^{-1} - 3.9z^{-2} + z^{-3} \]

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,

\[ \sigma_0 = 1 \quad d_0 = \tau_1 \tau_2 \tau_3 c_0 \]

\[ \sigma_1 = \tau_1 \quad d_1 = \tau_1 \tau_2 c_1 \]

\[ \sigma_2 = \tau_1 \tau_2 \quad d_2 = \tau_1 \tau_3 c_2 \]

\[ \sigma_3 = \tau_1 \tau_2 \tau_3 \quad d_3 = c_3 \]

where

\[ \tau_1 = \sqrt{1 - y_1^2} = \sqrt{1 - (-0.8)^2} = 0.6 \]

\[ \tau_2 = \sqrt{1 - y_2^2} = \sqrt{1 - (0.5)^2} = \frac{\sqrt{3}}{2} = 0.8660 \]

\[ \tau_3 = \sqrt{1 - y_3^2} = \sqrt{1 - (-0.6)^2} = 0.8 \]

Example 2

A third-order normalized lattice filter has reflection coefficients and ladder coefficients:

\[ [y_1, y_2, y_3] = [0.5, -0.6, -0.5] \]

\[ [c_0, c_1, c_2, c_3] = \frac{1}{6} [40, 15\sqrt{3}, 8\sqrt{3}, 6] = [6.6667, 4.3301, 2.3094, 1] \]

Determine the transfer function, \( H(z) = B_3(z)/A_3(z) \), of this filter, as well as the lower order linear prediction polynomials, \( A_2(z) \), 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,

\[ x = [1, 2, 3, 4, 5, 6, 7, 8] \]
We start by building the polynomials $A_1(z), A_2(z), A_3(z)$ using the forward Leis-
son recursion expressed in vectorial form. Starting with the 0th order polynomial,
$A_0 = a_0^R = [1]$, we construct the order-1 polynomial $A_1(z)$,

$$
A_1(z) = 1 - 0.5z^{-1}
$$

Then, we construct the order-2 polynomial,

$$
A_2(z) = 1 - 0.8z^{-1} + 0.6z^{-2}
$$

and finally the order-3 polynomial, which is the denominator of $H(z)$,

$$
A_3(z) = 1 - 0.5z^{-1} + 0.2z^{-2} + 0.5z^{-3}
$$

Next, we transform the ladder coefficients $c_p$ into standard un-normalized ones $d_p$,

$$
\tau_1 = \sqrt{1 - \gamma_1^2} = \frac{\sqrt{3}}{2}, \quad \tau_2 = \sqrt{1 - \gamma_2^2} = 0.8, \quad \tau_3 = \sqrt{1 - \gamma_3^2} = \frac{\sqrt{3}}{2}
$$

$$
d_0 = \tau_1 \tau_2 \tau_3 c_0 = \frac{\sqrt{3}}{2} \cdot 0.8 \cdot \frac{\sqrt{3}}{2} \cdot \frac{40}{6} = 4
$$

$$
d_1 = \tau_2 \tau_3 c_1 = 0.8 \cdot \frac{\sqrt{3}}{2} \cdot 15 \frac{\sqrt{3}}{6} = 3
$$

$$
d_2 = \tau_3 c_2 = \frac{\sqrt{3}}{2} \cdot \frac{8 \sqrt{3}}{6} = 2
$$

$$
d_3 = c_3 = 1
$$

and calculate the numerator polynomial coefficients from Eq. (8.3.20),

$$
\begin{bmatrix}
 b_0 \\
 b_1 \\
 b_2 \\
 b_3
\end{bmatrix} =
\begin{bmatrix}
 1 & a_{11} & a_{21} & a_{31} \\
 0 & 1 & a_{12} & a_{22} \\
 0 & 0 & 1 & a_{13} \\
 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
 d_0 \\
 d_1 \\
 d_2 \\
 d_3
\end{bmatrix}
= \begin{bmatrix}
 1 & -0.5 & 0.6 & -0.5 & 4 \\
 0 & 1 & -0.8 & 0.2 & 3 \\
 0 & 0 & 1 & -0.5 & 2 \\
 0 & 0 & 0 & 1 & 1
\end{bmatrix}
\begin{bmatrix}
 4.2 \\
 1.5 \\
 1.6 \\
 1
\end{bmatrix}
$$

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,

$$
\begin{align*}
 f_1 &= x \\
 f_2 &= f_3 + y_3 w_3 \\
 r_3 &= w_3 - y_3 f_2 \\
 y &= d_3 r_3 \\
 f_1 &= f_2 + y_2 w_2 \\
 r_2 &= w_2 - y_2 f_1 \\
 w_3 &= r_2 \\
 y &= y + d_2 r_2 = d_2 r_2 + d_3 r_3 \\
 f_0 &= f_1 + y_1 w_1 \\
 r_1 &= w_1 - y_1 f_0 \\
 w_2 &= r_1 \\
 y &= y + d_1 r_1 = d_1 r_1 + d_2 r_2 + d_3 r_3 \\
 r_0 &= f_0 \\
 w_1 &= r_0 \\
 y &= y + d_0 r_0 = d_0 r_0 + d_1 r_1 + d_2 r_2 + d_3 r_3
\end{align*}
$$

The MATLAB implementation for this example is as follows,

$$
g1 = 0.5; g2 = -0.6; g3 = -0.5; \quad \% \text{reflection coefficients} \\
d0 = 4; d1 = 3; d2 = 2; d3 = 1; \quad \% \text{ladder coefficients} \\
b = [4.2, 1.6, 1.5, 1]; \quad \% \text{numerator coefficients} \\
a = [1, -0.5, 0.2, 0.3]; \quad \% \text{denominator coefficients}
$$

$$
x = [1 2 3 4 5 6 7 8]'; \quad \% \text{input signal}
$$

$$
w1 = 0; w2 = 0; w3 = 0; \quad \% \text{initial states}
$$

$$
y = \text{zeros(size(x));}
$$

for $n = 1:length(x)$,

$$
f = x(n); \quad \% \text{section 3} \\
 f = f + g3 \cdot w3; \\
r = w3 - g3 \cdot f; \\
y(n) = y(n) + d3 \cdot r; \\
f = f + g2 \cdot w2; \quad \% \text{section 2} \\
r = w2 - g2 \cdot f; \\
w3 = r;
$$
y(n) = y(n) + d2*r;
f = f + g1*w1; % section 1
r = w1 - g1*f;

y(n) = y(n) + d1*r;
r = f;
w1 = r; % final output sample

yf = filter(b,a,x); % compare with output from FILTER

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,

The following MATLAB program is a concrete implementation.

```matlab
% ladder coefficients
c = [1 2 -4 5];
% reflection coefficients
ga = [0.6, -0.8, -0.6];
% tau = \sqrt{1 - ga.^2};
tau = sqrt(1-ga.^2);
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

% input signal
x = [1 2 3 4 5]';
y = zeros(size(x)); % output has same size as x
```
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 $p_j$, $j = 1, 2, \ldots, M$, 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 $i$th pole, we may write,

$$A(z) = \sum_{m=0}^{M} a_m z^{-m} = (1 - p_i z^{-1}) F(z), \quad F(z) = \prod_{j=1, j \neq i}^{M} (1 - p_j z^{-1})$$

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,

$$\sum_{m=0}^{M} (da_m) p_i^{-m} = -(dp_i) p_i^{-1} F(p_i) = -(dp_i) p_i^{-1} \prod_{j=1, j \neq i}^{M} (1 - p_j p_i^{-1}) = -(dp_i) p_i^{-1} \prod_{j=1, j \neq i}^{M} (p_i - p_j)$$

from which we obtain the partial derivatives,

$$\frac{\partial p_i}{\partial a_m} = - \frac{p_i^{M-m}}{M \prod_{j=1, j \neq i}^{M} (p_i - p_j)}, \quad \text{for } i, m = 1, 2, \ldots, M \quad (8.9.1)$$

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:

$$H(z) = \frac{1}{1 - 1.8955z^{-1} + 0.9939z^{-2}} \cdot \frac{1}{1 - 1.6065z^{-1} + 0.9859z^{-2}}$$

(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 $0 \leq \omega \leq \pi/2$. Then, using the function filter calculate the filter’s impulse response $h(n)$ for $n = 0, 1, \ldots, 599$, 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:

$$H(z) = G \cdot \frac{(1 + z^{-1})^2}{1 + a_{11}z^{-1} + a_{12}z^{-2}} \cdot \frac{(1 + z^{-1})^2}{1 + a_{21}z^{-1} + a_{22}z^{-2}} \cdot \frac{(1 + z^{-1})^2}{1 + a_{31}z^{-1} + a_{32}z^{-2}}$$

$$= G \cdot \frac{(1 + z^{-1})^6}{1 + a_1z^{-1} + a_2z^{-2} + a_3z^{-3} + a_4z^{-4} + a_5z^{-5} + a_6z^{-6}}$$

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:

$$a_1 = [1, a_{11}, a_{12}] = [1, -1.86711351, 0.96228613]$$
$$a_2 = [1, a_{21}, a_{22}] = [1, -1.84679822, 0.89820764]$$
$$a_3 = [1, a_{31}, a_{32}] = [1, -1.85182222, 0.86344488]$$

(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:

$$a = a_1 \ast a_2 \ast a_3 = [1, a_1, a_2, a_3, a_4, a_5, a_6]$$

as well as the normalized lattice coefficients, $y_p$, $c_p$, 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,
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 $0 \leq f \leq 1.2$ kHz, and verify that it meets the prescribed 1 dB passband specification.

Using the normalized lattice form implemented by your function, \textit{nlfilt}, calculate and plot the impulse response of this filter, $h(n)$, for $n = 0, 1, \ldots, 200$. Moreover, calculate it also using the function \textit{filter}, and compare the two methods.

c. Determine the quantized version of $a$, say $\hat{a}$, 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 $\hat{a}$ as the filter denominator vector, calculate in dB the magnitude response.

Then, round the normalized lattice coefficients $y_p, c_p$ to 5-digit fractional precision and calculate the corresponding magnitude response of the normalized lattice using your function \textit{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 $\hat{a}$, 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 $y_p, c_p$ to 4-digit precision, and verify that the resulting $y_p$’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 \textit{lat2freq}.

In addition, round to four-digit fractional precision the \textit{individual} second-order coefficient vectors $a_i, i = 1, 2, 3$, 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 $n = 0, 1, \ldots, 200$, of the 3-digit normalized lattice and cascade forms, and the exact response.
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):

\[
T \hat{X}(f) = X(f), \quad -\frac{f_s}{2} \leq f \leq \frac{f_s}{2} \tag{9.1.1}
\]

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:

\[
T \hat{X}(f) = X(f) + X(f - f_s) + X(f + f_s) + \cdots, \quad -\frac{f_s}{2} \leq f \leq \frac{f_s}{2} \tag{9.1.2}
\]
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, $x(nT)$, $0 \leq n \leq L - 1$. This time-windowing process is illustrated in Fig. 9.1.1.

In terms of the time samples $x(nT)$, the original sampled spectrum $X(f)$ and its time-windowed version $\hat{X}(f)$ are given by:

$$\hat{X}(f) = \sum_{n=-\infty}^{\infty} x(nT) e^{-j2\pi fnT}$$

$$\hat{X}_L(f) = \sum_{n=0}^{L-1} x(nT) e^{-j2\pi fnT}$$ (9.1.3)

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:

$$T_L = LT$$ (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 $L$:

$$w(n) = \begin{cases} 
1, & \text{if } 0 \leq n \leq L - 1 \\
0, & \text{otherwise}
\end{cases}$$ (9.1.5)

Then, define the windowed signal as follows:

$$x_L(n) = x(n)w(n) = \begin{cases} 
x(n), & \text{if } 0 \leq n \leq L - 1 \\
0, & \text{otherwise}
\end{cases}$$ (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:

$$X_L(\omega) = \sum_{n=0}^{L-1} x(n)e^{-j\omega n}$$

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n}$$

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 $\omega$.

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 $x_L(n) = x(n)w(n)$:

$$X_L(\omega) = \int_{-\pi}^{\pi} X(\omega')W(\omega - \omega') \frac{d\omega'}{2\pi}$$ (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:

$$W(z) = \sum_{n=0}^{L-1} w(n)z^{-n} = \sum_{n=0}^{L-1} 1 - z^{-1} = \frac{1 - z^{-L}}{1 - z^{-1}}$$

Setting $z = e^{j\omega}$, we find for $W(\omega)$:

$$W(\omega) = \frac{1 - e^{-jL\omega}}{1 - e^{-j\omega}} = \frac{\sin(\omega L/2)}{\sin(\omega/2)} e^{-j\omega(L-1)/2}$$ (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.

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 \(k = \pm 1, \pm 2, \ldots\) (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:

\[
\Delta \omega_w = \frac{2\pi}{L} \quad \text{(rectangular window width)} \tag{9.1.10}
\]

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:

\[
\Delta f_w = \frac{f_s}{L} = \frac{1}{L} = \frac{1}{T} \tag{9.1.11}
\]

We will see shortly that the mainlobe width \(\Delta f_w\) 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 \(L\):

\[
\frac{|W(\omega)|}{W(0)} \bigg|_{\omega=3\pi/L} = \frac{\sin(\omega L/2)}{L \sin(\omega/2)} = \frac{\sin(3\pi/2)}{L \sin(3\pi/2L)} = \frac{1}{L} \cdot \frac{1}{(3\pi/2)} = \frac{2}{3\pi}
\]

We assumed that \(L\) was fairly large (typically, \(L \geq 10\)), and used the small-x approximation \(\sin x = x\) with \(x = 3\pi/2L\). In decibels, the relative sidelobe level is

\[
R = 20 \log_{10} \left| \frac{W(\omega)}{W(0)} \right|_{\omega=3\pi/L} \approx 20 \log_{10} \left( \frac{2}{3\pi} \right) = \frac{-13.46}{dB}
\]

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:

\[
x(t) = e^{2\pi i f_1 t}, \quad -\infty < t < \infty \implies x(n) = e^{2\pi i f_1 nT} = e^{i\omega_1 n}, \quad -\infty < n < \infty
\]

where \(\omega_1 = 2\pi f_1 = 2\pi f_1 / f_s\). The spectrum of the analog signal \(x(t)\) is the Fourier transform:

\[
X(f) = \int_{-\infty}^{\infty} x(t) e^{-2\pi i ft} dt = \int_{-\infty}^{\infty} e^{-2\pi i f_1 t} e^{-2\pi i f} dt = \delta(f - f_1)
\]

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:

\[
\cos(2\pi f_1 t) = \frac{1}{2} e^{2\pi if_1 t} + \frac{1}{2} e^{-2\pi if_1 t} \implies \frac{1}{2} \delta(f - f_1) + \frac{1}{2} \delta(f + f_1)
\]

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 \(-f_s/2 \leq f \leq f_s/2\):

\[
X(\omega) = X(f) = \frac{1}{T} X(f) = \frac{1}{T} \delta(f - f_1)
\]

Using the delta function property, \(|a| \delta(ax) = \delta(x)\), we can express the spectrum in terms of the digital frequency \(\omega = 2\pi f / f_s = 2\pi f T\), as follows:

\[
2\pi \delta(\omega - \omega_1) = \frac{1}{T} 2\pi T \delta(2\pi f_1 T - 2\pi f) = \frac{1}{T} \delta(f - f_1)
\]

Therefore, the spectrum of the sampled signal will be, over the Nyquist interval:

\[
X(\omega) = 2\pi \delta(\omega - \omega_1), \quad -\pi \leq \omega \leq \pi \tag{9.1.12}
\]

Outside the Nyquist interval, the spectral line is replicated at multiples of \(2\pi\), that is, \(2\pi \delta(\omega - \omega_1 - 2\pi n)\). 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):

\[
x(n) = \int_{-\pi}^{\pi} X(\omega) e^{i\omega n} \frac{d\omega}{2\pi} = \int_{-\pi}^{\pi} 2\pi \delta(\omega - \omega_1) e^{i\omega n} \frac{d\omega}{2\pi} = e^{i\omega_1 n}
\]

The windowed sinusoid consists of the \(L\) samples:

\[
x_L(n) = e^{i\omega_1 n}, \quad n = 0, 1, \ldots, L - 1
\]
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:

$$X_L(\omega) = W(\omega - \omega_1)$$

This is the translation of $W(\omega)$ centered about $\omega_1$, 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 $\omega_1$ and replacing it by $W(\omega - \omega_1)$.

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 width is inversely proportional to the window length. The smaller the desired separation, the longer the data record.

The mainlobe width of $W(\omega)$ determines the amount of achievable frequency resolution. The mainlobes, 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:

$$w(n) = \left\{ \begin{array}{ll} 0.54 - 0.46 \cos \left( \frac{2\pi n}{L - 1} \right), & \text{if } 0 \leq n \leq L - 1 \\ 0, & \text{otherwise} \end{array} \right.$$

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.
9.1. FREQUENCY RESOLUTION AND WINDOWING

9. DTFT AND SPECTRAL ANALYSIS

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 spectral 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 Welch method consists of dividing the total length-\(L\) data record into \(K\) contiguous segments of length \(N\), such that \(L = KN\). 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 Welch 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 spectral 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 Welch method consists of dividing the total length-\(L\) data record into \(K\) contiguous segments of length \(N\), such that \(L = KN\). 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 Welch method consists of dividing the total length-\(L\) data record into \(K\) contiguous segments of length \(N\), such that \(L = KN\). 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 Welch method consists of dividing the total length-\(L\) data record into \(K\) contiguous segments of length \(N\), such that \(L = KN\). 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 Welch method consists of dividing the total length-\(L\) data record into \(K\) contiguous segments of length \(N\), such that \(L = KN\). 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.

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 ... 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 $\Delta f = f_s / L = 10 / 100 = 0.1 \text{ kHz}$. Thus, the closest $f_3$ to $f_1$ and $f_2$ will be:

$$f_3 = f_1 + \Delta f = 1.1 \text{ kHz}, \quad \text{and} \quad f_3 = f_2 - \Delta f = 1.9 \text{ kHz}$$

In the Hamming case, the minimum resolvable frequency separation doubles, that is, $\Delta f = c f_s / L = 2 \cdot 0.2 = 0.4 \text{ kHz}$, which gives $f_3 = 1.2 \text{ kHz}$ or $f_1 = 1.8 \text{ kHz}$.

Example 9.1.3: The sinusoid $x(t) = \cos(2\pi f_0 t)$, where $f_0 = 50 \text{ Hz}$ is sampled at a rate of $f_s = 1 \text{ kHz}$. The sampled signal is $x(n) = \cos(\omega_0 n)$, where $\omega_0 = 2\pi f_0 / f_s = 2\pi \cdot 50 / 1000 = 0.1 \pi$ radians/sample. A length-$L$ portion of $x(n)$ is windowed by a rectangular and a Hamming window, that is, for $n = 0, 1, \ldots, L - 1$:

$$x_L(n) = w_{\text{rec}}(n) x(n) = \cos(\omega_0 n)$$

$$x_L(n) = w_{\text{ham}}(n) x(n) = \left[ 0.54 - 0.46 \cos\left( \frac{2\pi n}{L - 1} \right) \right] \cos(\omega_0 n)$$

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, $0 \leq \omega \leq 0.2\pi$. The spectra were computed by successive calls to the routine `dtft` of the next section, for 200 equally spaced values of $\omega$ in the interval $0 \leq \omega \leq 0.2\pi$.

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 $f_1 = 2 \text{ kHz}$, $f_2 = 2.5 \text{ kHz}$, and $f_3 = 3 \text{ kHz}$:

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

where $t$ is in milliseconds, is sampled at a rate of 10 kHz. We consider four data records of lengths $L = 10, 20, 40, \text{ and } 100$ 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, $0 \leq f \leq f_c$. 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 $x$ 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 $f_s$. 

\[\top\]
9.1. FREQUENCY RESOLUTION AND WINDOWING

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, $f_s/2 = 0.25$.

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.

9.2 DTFT Computation

In this section, we turn our attention to the computational aspects of the DTFT. We
consider a length-$L$ signal $x(n)$, $n = 0, 1, \ldots, L - 1$, 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:

$$X(\omega) = \sum_{n=0}^{L-1} x(n)e^{-j\omega n} \quad \text{(DTFT of length-$L$ signal)}$$  \hspace{1cm} (9.2.1)

This expression may be computed at any desired value of $\omega$ in the Nyquist interval
$-\pi \leq \omega \leq \pi$. 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 $-\pi \leq \omega \leq \pi$ onto the right-sided one $0 \leq \omega \leq 2\pi$. We will refer to the latter
as the DTFT Nyquist interval. This mapping is shown in Fig. 9.2.1.

The positive-frequency subinterval $0 \leq \omega \leq \pi$ remains unchanged, but the negative-frequency one, $-\pi \leq \omega \leq 0$, gets mapped onto the second half of the DFT Nyquist interval, $\pi \leq \omega \leq 2\pi$. 

Fig. 9.1.8 Rectangular and Hamming spectra for $L = 10$ and 20.

Fig. 9.1.9 Rectangular and Hamming spectra for $L = 40$ and 100.
This algorithm can then be applied to any point on the unit circle \( z = e^{\jmath \omega} \) to evaluate \( X(\omega) \). The C-functions, \texttt{dtfc} and \texttt{dtfr},\footnote{included in the C-function collection of this book} implement the computation of the DTFT at a single frequency and at a range of frequencies, respectively.

Alternatively, one may use the built-in MATLAB function \texttt{freqz} to evaluate \( X(\omega) \) at any vector of frequencies \( \omega \), with usage:

\begin{verbatim}
  % 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
\end{verbatim}

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

\[ \{ f_1, -f_1 \} \leftrightarrow \{ f_1, f_L - f_1 \} \]

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:

\[
X(\omega) = \sum_{n=0}^{L-1} x(n)e^{-j\omega n} = \sum_{n=0}^{L-1} x(n)z^{-n} \bigg|_{z=e^{j\omega}} \tag{9.2.2}
\]

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 \( X(z) \):

\[
\begin{align*}
\text{for each complex } z & \text{ do:} \\
X &= 0 \\
\text{for } n = L-1 \text{ down to } n = 0 & \text{ do:} \\
X &= x_n + z^{-1}X
\end{align*}
\tag{9.2.3} \text{(Hörner’s rule)}
\]

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:

\[
\begin{align*}
X &= x_0 + z^{-1}X = x_0 \\
x_0 + z^{-1}X &= x_1 + z^{-1}x_2 + z^{-2}x_3 \\
x_1 + z^{-1}X &= x_2 + z^{-1}x_3 \\
x_2 + z^{-1}X &= x_3 \\
x_3 & = X(z)
\end{align*}
\]

Example 9.2.1: In Example 16.1.1, we discussed the generation of dual sinuosoidal tones for DTMF touch-tone phones. Each keypress generates two frequencies \( \omega_H \) and \( \omega_L \), one from the high and one from the low group of frequencies. A total of \( 4 \times 4 = 16 \) 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 \texttt{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 \( \delta f \) 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 \( f_s = 8 \text{ kHz} \) and rectangular windowing gives the minimum length

\[
L = f_s/\Delta f = 8000/268 = 30 \text{ samples.}
\]

9.3 Window Parameters

We saw above that the effect of windowing a signal consisting of a sum of sinuosoidal components is to broaden the (ideally infinitely narrow) spectral peaks at the sinuosoid 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 sinuosoids are too weak, they will
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 mainlobe (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 $\omega_1, \omega_2$ 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 $\omega_1, \omega_2$ get smeared and replaced by the shifted versions of $W(\omega)$,

$$x(n) = A_1 e^{j\omega_1 n} w(n) + A_2 e^{j\omega_2 n} w(n), \quad n = 0, 1, \ldots, L - 1$$

$$X(\omega) = A_1 W(\omega - \omega_1) + A_2 W(\omega - \omega_2)$$

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 $\omega_0$ of the first null, that is, if $\omega_2 - \omega_1 \geq \omega_0$. Since the value of $\omega_0$ is roughly equal to the 3-dB width, an alternative resolvability criterion is to require that,

$$\omega_2 - \omega_1 \geq \Delta \omega_{3dB}$$  \hspace{1cm} (frequency resolution condition)  \hspace{1cm} (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 $|X(\omega)| = |A_1 W(\omega - \omega_1) + A_2 W(\omega - \omega_2)|$ near $\omega_1$ will not necessarily be at precisely $\omega_1$, 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 Doerr’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 $\Delta \omega_{3dB}$ 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,

$$\Delta \omega_{3dB} = 0.886 \frac{2\pi b}{L} \quad \Rightarrow \quad \Delta f_{3dB} = 0.886 \frac{f_b}{L}$$

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

$$b = 0.0124 R + 1.0221$$

which works well over the range, $20 \leq R \leq 120$ dB, with a 2-percent error. An even better approximation with a 0.4-percent error is,

$$b = -0.000051 R^2 + 0.019549 R + 0.813943, \quad 20 \leq R \leq 120$$

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,

$$w(n) = 0.54 - 0.46 \cos \left( \frac{\pi n}{M} \right), \quad n = 0, 1, \ldots, 2M, \quad M = \frac{L - 1}{2}$$

$$w(n) = \frac{I_0 \left( \alpha \sqrt{1 - (n - M)^2/M^2} \right)}{I_0(\alpha)}, \quad n = 0, 1, \ldots, 2M$$

The Kaiser window shape parameter $\alpha$ (denoted by $\beta$ in MATLAB), can be calculated in terms of $R$ by the Kaiser-Schafer approximation [224], where $R$ is in dB.$^1$

$^1$A different approximation is used in the FIR filter design problem, discussed in Chap. 11.
3-dB broadening factor

\[ b = \begin{cases} 0, & R \leq 13.26 \\ 0.76609 (R - 13.26)^{0.4} + 0.09834 (R - 13.26), & 13.26 < R \leq 60 \\ 0.12438 (R + 6.3), & 60 < R \leq 120 \end{cases} \]  

(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 \( \alpha \) required for the calculation of the window samples \( w(n) \). The window samples \( w(n) \) can be calculated with the built-in MATLAB function `kaiser`

\[
w = \text{kaiser}(L, \alpha); \quad \text{Lx1 column vector}
\]

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,

\[
\Delta \omega_{\text{null}} = \frac{2\pi c}{L - 1} \quad \Rightarrow \quad \Delta f_{\text{null}} = \frac{f_s c}{L - 1}, \quad c = \frac{12(R + 12)}{155}
\]  

(9.3.8)

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.

This window maximizes the power contained in the mainlobe. More specifically, for a length-\( L \) window, \( w(n) \), \( n = 0, 1, \ldots, L - 1 \), the time-bandwidth parameter \( B \) defines the mainlobe band, \([ -\omega_c, \omega_c ] \), via the relationship,

\[
B = \frac{\omega_c}{2\pi} = \frac{2\pi B}{L}
\]  

(9.3.9)

The window \( w(n) \) is chosen to maximize the power within the mainlobe band, that is,

\[
J = \frac{1}{2\pi} \int_{-\omega_c}^{\omega_c} |W(\omega)|^2 \, d\omega
\]

(9.3.10)

where \( W(\omega) \) is the DTFT of the window,

\[
W(\omega) = \sum_{n=0}^{L-1} w(n) e^{-jn\omega}
\]

9. DTFT and Spectral Analysis

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:

\[
w_c = \text{chebwin}(L, R); \quad \text{% built-in function, Lx1 vector}
w_c = \frac{w_c}{\text{max}(w_c)}; \quad \text{% normalize to unity at its middle}
\]

Its broadening factor can be computed by the following approximation, which is accurate over the range, \( 13 \leq R \leq 180 \) dB:

\[
b = 0.65 + 0.0195 R - 0.00005 R^2
\]  

(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:

<table>
<thead>
<tr>
<th>window</th>
<th>3-dB broadening factor</th>
<th>sidelobe attenuation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rectangular</td>
<td>( b = 1 )</td>
<td>13.26</td>
</tr>
<tr>
<td>Hamming</td>
<td>( b = 1.47 + 0.97/L )</td>
<td>42.67</td>
</tr>
<tr>
<td>Kaiser</td>
<td>( b = 0.0124 R + 1.0221 )</td>
<td>variable ( R )</td>
</tr>
<tr>
<td>DPSS</td>
<td>( b = 0.0124 R + 1.0221 )</td>
<td>variable ( R )</td>
</tr>
<tr>
<td>Chebyshev</td>
<td>( b = 0.65 + 0.0195 R - 0.00005 R^2 )</td>
<td>variable ( R )</td>
</tr>
</tbody>
</table>

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.

\[
x(n) = e^{j\omega_0 n}, \quad \omega_0 = \frac{2\pi f_1}{f_s}, \quad n = 0, 1, \ldots, L - 1
\]
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,

\[
X(f) = \sum_{n=0}^{L-1} w(n) x(n) e^{-2\pi fn/fs} = \text{DTFT of windowed signal}
\]

\[
X_{\text{norm}}(f) = \frac{X(f)}{|X(f_1)|} = \text{normalized spectrum}
\]

\[
20 \log_{10} |X_{\text{norm}}(f)| = \text{spectrum in dB}
\]

In addition, the window time functions, \( w(n), n = 0, 1, \ldots, L-1 \), 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 \( \omega_i \), the interaction between the shifted windowed spectra \( W(\omega + \omega_i) \) 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, \( x(n) = \cos(\omega_1 n) \), \( n = 0, 1, \ldots, L-1 \). 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.

\[
x(n) = w(n) \cos(\omega_1 n) \quad \Rightarrow \quad X(\omega) = \frac{1}{2} [W(\omega - \omega_1) + W(\omega + \omega_1)]
\]

**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 \( \Delta f \) is determined to be \( L = 169 \), from the MATLAB code segment,
9.3 WINDOW PARAMETERS

9.4 ADDITIONAL DETAILS ON WINDOWS

The required DTFT calculations can be done with the help of the `freqz` function, with MATLAB code similar to the following,

```matlab
R = 60; % sidelobe attenuation and 3-dB width
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 down to the desired 70-dB level.

**Example 9.3.3:** Consider next the following analog signal consisting of three sinusoids:

$$x(t) = 10 \cos(2\pi f_1 t) + 3 \cos(2\pi f_2 t) + \cos(2\pi f_3 t)$$

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 $R = 70$; $Df = 0.1$; % sidelobe attenuation and 3-dB width
$b = 0.0244 * 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, $L = 169$

We saw earlier that the 3-dB width of a length-$L$ rectangular window is given by

$$\Delta f_{3dB} = \frac{f_s}{L} \cdot \frac{12(R + 12)}{155}$$

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,

$$\Delta \omega_{3dB} = 0.886 \frac{2\pi}{L}$$
9.4. ADDITIONAL DETAILS ON WINDOWS

This can be derived by considering the DTFT of the length-\(L\) rectangular window,

\[
W(\omega) = \frac{1 - e^{-jL\omega}}{1 - e^{-j\omega}} = e^{-j\omega(L-1)/2} \frac{\sin(\omega L/2)}{\sin(\omega/2)}
\]

\[
= e^{-j\pi x(L-1)/L} \frac{\sin(\pi x)}{\sin(\pi x/L)} , \quad \text{where } \omega = \frac{2\pi x}{L}
\]

\((9.4.1)\)

**Fig. 9.3.3** Rectangular, Hamming, Kaiser, DPSS, and Chebyshev windows for \(L = 169\).

and its magnitude, normalized to unity gain at DC,

\[
\frac{W(\omega)}{W(0)} = \frac{\sin(\omega L/2)}{L \sin(\omega/2)} = \frac{\sin(\pi x)}{L \sin(\pi x/L)} , \quad \omega = \frac{2\pi x}{L}
\]

**Fig. 9.3.4** Windows designed with \(R = 70\) dB and \(L = 169\).
9.4. ADDITIONAL DETAILS ON WINDOWS

For $L > 10$, this can be approximated well by a sinc-function,

$$F(x) = \frac{\sin(\pi x)}{L \sin(\pi x/L)} = \frac{\sin(\pi x)}{L \cdot \pi x/L} = \frac{\sin(\pi x)}{\pi x} = \text{sinc}(x)$$

The 3-dB frequency is obtained by solving the half-power condition,

$$\left| \frac{W(\omega)}{W(0)} \right|^2 = \frac{1}{2} \Rightarrow F^2(x) = \frac{1}{2} \Rightarrow |F(x)| = \frac{1}{\sqrt{2}}$$

which can be implemented easily using MATLAB’s `fzero` function,

```matlab
% 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,

$$\omega_3 = \frac{2\pi x_3}{L} = 0.443 \frac{2\pi}{L} \Rightarrow \Delta \omega_{3\text{dB}} = 2\omega_3 = 0.886 \frac{2\pi}{L}$$

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, $1 \leq x \leq 2$, because as we noted earlier, the sidelobe frequency is approximately near $\omega = \frac{2\pi x}{L}$, with $x \approx 1.5$,

```matlab
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)));
```

so that,

$$\omega_{\text{side}} = \frac{2\pi x_{\text{side}}}{L}, \quad -20 \log_{10} \left| \frac{W(\omega_{\text{side}})}{W(0)} \right| = 13.2615 \text{ dB}$$

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,

$$w(n) = u(n) - u(n - L) = \text{length}-L \text{ rectangular window}$$

$$w_H(n) = 0.54 w(n) - 0.46 \cos \left( \frac{2\pi n}{L - 1} \right) w(n)$$

$$= 0.54 w(n) - 0.23 e^{2\pi j n/(L-1)} w(n) - 0.23 e^{-2\pi j n/(L-1)} w(n)$$

$$W_H(\omega) = 0.54 W(\omega) - 0.23 W \left( \omega - \frac{2\pi}{L - 1} \right) - 0.23 W \left( \omega + \frac{2\pi}{L - 1} \right)$$

or, using the $x$-variable and the sinc approximation,

$$W(x) = e^{-\pi x/(L-1)/L} \frac{\sin(\pi x)}{\sin(\pi x/L)} \approx e^{-\pi x/(L-1)/L} \frac{L \sin(\pi x)}{\pi x}, \quad \omega = \frac{2\pi x}{L}$$

$$W_H(x) = 0.54 W(x) - 0.23 W \left( x - \frac{L}{L - 1} \right) - 0.23 W \left( x + \frac{L}{L - 1} \right)$$

Fig. 9.3.5 Windows designed with $R = 70$ dB and $L = 839$. 
The exact DC value, $W_H(0)$, can be expressed in the simple closed form,

$$W_H(0) = 0.54L - 0.46 \quad (9.4.2)$$

By noting that the maximum sidelobe always occurs in the range, $4 \leq x \leq 5$, 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,

$$\omega_{\text{side}} = \frac{2\pi x_{\text{side}}}{L}, \quad x_{\text{side}} = 4.5 - \frac{2.1}{L} + \frac{2.2}{L^2}$$

$$R_{\text{side}} = -20 \log_{10} \left| \frac{W_H(\omega_{\text{side}})}{W_H(0)} \right| = -42.67 + \frac{1.5}{L} + \frac{513}{L^2} \quad (9.4.3)$$

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 $L$,

$$\left| \frac{W_H(\omega_{3dB})}{W_H(0)} \right|^2 = \frac{1}{2} \quad (9.4.4)$$

from which we obtain the approximate broadening factor for the full width $\Delta\omega_{3dB}$, valid again for $L > 10$,

$$\Delta\omega_{3dB} = 2\omega_{3dB} = 0.886 \frac{2\pi b}{L}, \quad b = 1.47 + 0.97 \frac{1}{L} \quad (9.4.5)$$

### 9.4.3 Kaiser Window

The symmetric Kaiser window is based on the following Fourier transform pair in continuous time,

$$2\tau_0 \sinh \left[ \sqrt{\tau_0^2 \Omega^2 - \tau_0^2 \Omega^2} \right] = \int_{-\tau_0}^{\tau_0} I_0(\omega_0 \sqrt{1 - t^2/\tau_0^2}) e^{i\omega t} dt$$

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 $\tau_0$,

$$\tau_0 = MT_s$$

$$t = kT_s, \quad -M \leq k \leq M$$

$$\omega = \Omega T_s = \text{digital frequency}$$

$$\tau_0 \Omega = MT_s \Omega = M\omega$$

then, the sampled and normalized (to unity at its middle) symmetric window becomes,

$$w(k) = I_0 \left( \frac{\alpha \sqrt{1 - k^2/M^2}}{I_0(\alpha)} \right), \quad -M \leq k \leq M$$

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,

$$W(\omega) = 2M \frac{\sinh \left[ \sqrt{\alpha^2 - M^2\omega^2} \right]}{I_0(\sqrt{\alpha^2 - M^2\omega^2})}, \quad \omega = \frac{2\pi f}{T_s}$$

$$\frac{W(\omega)}{W(0)} = \frac{\alpha}{\sinh(\alpha)} \cdot \frac{\sinh \left[ \sqrt{\alpha^2 - M^2\omega^2} \right]}{\sqrt{\alpha^2 - M^2\omega^2}}$$

The causal version of the window is obtained by delaying $w(k)$ by $M$ samples,

$$w(n) = I_0 \left( \frac{\alpha \sqrt{1 - (n-M)^2/M^2}}{I_0(\alpha)} \right), \quad n = 0, 1, \ldots, 2M$$

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 $\alpha$ and $R$ are (see EWA/Ch.20 [46] for computational details),

$$b = 0.0124 R + 1.0221$$

$$R = 13.26 + 20 \log_{10} \left[ \frac{\sinh(\alpha)}{\alpha} \right]$$

with $R$ in dB, and the exact relationship between $R$ and $\alpha$ can be replaced by the Kaiser-Schäfer approximation:

$$\alpha = \begin{cases} 0, & R \leq 13.26 \\ 0.76609 (R - 13.26)^{0.4} + 0.09834 (R - 13.26), & 13.26 < R \leq 60 \\ 0.12438 (R + 6.3), & 60 < R \leq 120 \end{cases}$$

### 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 $\omega_c$ to be determined from $\alpha$ and $L$, the minimization criterion is as mentioned in Eq. (9.3.10),

$$J = \frac{1}{2\pi} \int_{-\omega_c}^{\omega_c} |W(\omega)|^2 d\omega$$

$$J = \frac{1}{2\pi} \int_{-\pi}^{\pi} |W(\omega)|^2 d\omega = \max$$
where $W(\omega)$ is the DTFT of the window $w_n$,

$$W(\omega) = \sum_{n=0}^{L-1} w_n e^{-j\omega n}, \quad w = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{L-1} \end{bmatrix}$$

The performance index $J$ can be expressed directly in terms of the window time samples $w$ as a Rayleigh quotient,

$$J = \frac{w^TAw}{w^Tw} = \max$$

where $A$ is the $L \times L$ "prolate matrix" defined in terms of its matrix elements,

$$A_{nm} = \frac{\sin(2\pi W_c(n-m))}{\pi(n-m)}, \quad n,m = 0,1,\ldots,L-1, \quad W_c = \frac{\omega_c}{2\pi}$$

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, $\lambda_0$:

$$Aw = \lambda_0 w$$

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, cond$(A) = 5.1063 \times 10^{16}$,

<table>
<thead>
<tr>
<th>$i$</th>
<th>$\lambda_i$</th>
<th>$\lambda_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.99999999998517786000</td>
<td>0.00000000000000920186</td>
</tr>
<tr>
<td>1</td>
<td>0.99999999997951462700</td>
<td>0.00000000000000004034</td>
</tr>
<tr>
<td>2</td>
<td>0.99999998710540139000</td>
<td>0.0000000000000000920186</td>
</tr>
<tr>
<td>3</td>
<td>0.99999517388508363000</td>
<td>0.00000000000000001958</td>
</tr>
<tr>
<td>4</td>
<td>0.99987947149714950000</td>
<td>0.00000000000000000000</td>
</tr>
<tr>
<td>5</td>
<td>0.99724570999562000000</td>
<td>0.00000000000000000000</td>
</tr>
<tr>
<td>6</td>
<td>0.97588122145542644000</td>
<td>0.00000000000000000000</td>
</tr>
<tr>
<td>7</td>
<td>0.83446090480119717000</td>
<td>0.00000000000000000000</td>
</tr>
<tr>
<td>8</td>
<td>0.45591142240931063000</td>
<td>0.00000000000000000000</td>
</tr>
<tr>
<td>9</td>
<td>0.11887181815895912000</td>
<td>0.00000000000000000000</td>
</tr>
<tr>
<td>10</td>
<td>0.01567636516215985600</td>
<td>0.00000000000000000000</td>
</tr>
</tbody>
</table>

These were generated by the following MATLAB code:

```matlab
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 $\alpha$ parameter works well over the range $14 \leq L \leq 120$ dB:

$$w_c = \frac{0.95\alpha/\pi + 0.14}{L}$$

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 attenu-
ation 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 \( m \)th Chebyshev polynomial \( T_m(x) \) is:

\[
T_m(x) = \cos(m \acos(x))
\]

If \(|x| > 1\), the inverse cosine \( \acos(x) \) becomes imaginary, and the expression can be rewritten in terms of hyperbolic cosines:

\[
T_m(x) = \cosh(m \acos(x))
\]

Setting \( x = \cos(\theta) \), or \( \theta = \acos(x) \), 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.,

\[
\begin{align*}
\cos(0\theta) &= 1 & T_0(x) &= 1 \\
\cos(1\theta) &= \cos \theta & T_1(x) &= x \\
\cos(2\theta) &= 2 \cos^2 \theta - 1 & T_2(x) &= 2x^2 - 1 \\
\cos(3\theta) &= 4 \cos^3 \theta - 3 \cos \theta & T_3(x) &= 4x^3 - 3x \\
\cos(4\theta) &= 8 \cos^4 \theta - 8 \cos^2 \theta + 1 & T_4(x) &= 8x^4 - 8x^2 + 1
\end{align*}
\]

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 \( T_6(x) \) and \( T_{10}(x) \).

The Dolph-Chebyshev window is defined such that its sidelobes will correspond to a portion of the equiripple range \(|x| \leq 1\) 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 \( m \)th terms:

\[
\begin{align*}
\cos(m\omega) &= \cos\left(2m \frac{\omega}{2}\right) = T_{2m}(u) \\
\cos((m-1/2)\omega) &= \cos\left((2m-1) \frac{\omega}{2}\right) = T_{2m-1}(u)
\end{align*}
\]

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,

\[
W(\omega) = T_{L-1}(x), \quad x = x_0 \cos\left(\frac{\omega}{2}\right)
\]

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 \(|x| \leq 1\), we will have \( W_{\text{side}} = 1 \). Thus, we find:

\[
R_a = T_{L-1}(x_0) = \cosh((L - 1) \acos(x_0))
\]

which can be solved for \( x_0 \) in terms of \( R_a \):

\[
x_0 = \cosh\left(\frac{\acos(R_a)}{L - 1}\right)
\]

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:

\[
T_{L-1}(x) = \cos((L - 1) \acos(x)) = 0 \quad \Rightarrow \quad x_i = \cos\left(\frac{(i - 1/2) \pi}{L - 1}\right)
\]

for \( i = 1, 2, \ldots, L - 1 \). Solving for the corresponding frequencies through \( x_i = x_0 \cos(\omega_i/2) \), we find the DTFT zeros:

\[
\omega_i = 2 \acos\left(\frac{x_i}{x_0}\right), \quad z_i = e^{j \omega_i}, \quad i = 1, 2, \ldots, L - 1
\]
Fourier Optics, Apertures, Spatial Arrays

The symmetric z-transform of the window is then constructed in terms of its zeros:

\[ W(z) = \prod_{i=1}^{L-1} (1 - z_i z^{-1}) \]

The inverse z-transform of \( W(z) \) are the window coefficients \( w(n) \). The typical MATLAB code is,

\[ L_1 = L-1; \quad \text{% number of zeros} \]
\[ R_a = 10^{(R/20)}; \quad \text{% sidelobe level in absolute units} \]
\[ x_0 = \cosh(\text{acosh}(R_a)/L_1); \quad \text{% scaling factor} \]
\[ i = 1:L_1; \]
\[ x_i = \cos(i\pi(i-0.5)/L_1); \quad \text{% L1 zeros of Chebyshev polynomial} \]
\[ \omega_m = 2 \sqrt{x_i/x_0}; \quad \text{% L1 zeros in omega-space} \]
\[ z_i = \exp(j\omega_m i); \quad \text{% L1 zeros of W(z) polynomial} \]
\[ w = \text{real}(\text{poly}(z_i)); \quad \text{% zeros-to-polynomial-coefficients} \]
\[ w = \text{chebwin}(L,R); \quad \text{% see also the more accurate function poly2 in the EWA toolbox} \]

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

\[ w = \text{chebwin}(L,R); \quad \text{% in dB} \]

The 3-dB frequency \( \omega_3 \) is defined by the half-power condition:

\[ W(\omega_3) = T_{L-1}(x_3) = \frac{T_{L-1}(x_0)}{\sqrt{2}} = \frac{R_a}{\sqrt{2}} \]

\[ \cosh((L-1)\text{acosh}(x_3)) = \frac{R_a}{\sqrt{2}} \]

Solving for \( x_3 \) and the corresponding 3-dB angle \( \omega_3 \), \( x_3 = x_0 \cos(\omega_3/2) \),

\[ x_3 = \cosh \left( \frac{\text{acosh}(R_a/\sqrt{2})}{L-1} \right), \quad \omega_3 = 2 \cos \left( \frac{x_3}{x_0} \right) \]

From \( \omega_3 \), one calculates the 3-dB width, \( \Delta\omega_{3\text{dB}} = 2\omega_3 \). However, the following approximate relationship works well for the broadening factor relative to the rectangular window, over the range \( 13 \leq R \leq 180 \text{ dB} \),

\[ \Delta\omega_{3\text{dB}} = 0.886 \frac{2\pi b}{L}, \quad b = 0.65 + 0.0195 R - 0.00005 R^2 \]

### 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, \( t, \Omega \), are replaced one-dimensionally by the space and wavenumber variables, \( x, k_x, k_y, k_z \).

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 along the \( x \)-axis, \( x_n = nd, n = 0, 1, 2, \ldots, N - 1 \) 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, \( w_n \),

\[ W(\psi) = \sum_{m=0}^{N-1} w_m e^{j\phi m}, \quad \psi = k_x d = k d \cos \phi = \frac{2\pi d}{\lambda} \cos \phi \]

where \( k_x = k \cos \phi \) is the wavenumber along the \( x \)-direction and \( \phi \), the azimuthal angle, and \( k \) is the vacuum wavenumber related to the wavelength \( \lambda \), by \( k = 2\pi/\lambda \). and \( \psi \) plays the role of digital spatial frequency in units of radians per space sample. An array that is scanned towards a particular direction \( \phi_0 \), with \( \psi_0 = k d \cos \phi_0 \), will have an array factor, \( A(\psi) = W(\psi - \psi_0) \).

![Fig. 9.5.1 One-dimensional antenna array.](image-url)

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.

\[ \text{† from the Greek word for “no feet”}\]
### Table 9.5.1 Duality between time-domain and space-domain signal processing.

<table>
<thead>
<tr>
<th><strong>Discrete-time signal processing</strong></th>
<th><strong>Discrete-space array processing</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>time-domain sampling, ( t_n = nT )</td>
<td>space-domain sampling, ( x_n = n\hat{d} )</td>
</tr>
<tr>
<td>sampling time interval, ( T )</td>
<td>sampling space interval, ( \hat{d} )</td>
</tr>
<tr>
<td>sampling rate, ( 1/T ) [samples/sec]</td>
<td>sampling rate, ( 1/\hat{d} ) [samples/meter]</td>
</tr>
<tr>
<td>frequency, ( \Omega ) [radians/sec]</td>
<td>wavenumber, ( k_x ) [radians/meter]</td>
</tr>
<tr>
<td>digital frequency, ( \omega = \Omega T )</td>
<td>digital wavenumber, ( \psi = k_x \hat{d} )</td>
</tr>
<tr>
<td>Nyquist interval, ( -\pi \leq \omega \leq \pi )</td>
<td>Nyquist interval, ( -\pi \leq \psi \leq \pi )</td>
</tr>
<tr>
<td>sampling theorem, ( \Omega \leq \pi/T )</td>
<td>sampling theorem, ( k_x \leq \pi/\hat{d} )</td>
</tr>
<tr>
<td>spectral images</td>
<td>grating lobes or fringes</td>
</tr>
<tr>
<td>frequency response, ( A(\omega) )</td>
<td>array factor, ( A(\psi) )</td>
</tr>
<tr>
<td>( z )-domain, ( z = e^{j\omega} )</td>
<td>( z )-domain, ( z = e^{j\psi} )</td>
</tr>
<tr>
<td>transfer function, ( A(z) )</td>
<td>transfer function, ( A(z) )</td>
</tr>
<tr>
<td>DTFT and inverse DTFT</td>
<td>DSFT and inverse DSFT</td>
</tr>
<tr>
<td>pure sinusoid, ( e^{j\omega_0} )</td>
<td>narrow beam, ( e^{-j\phi_0} )</td>
</tr>
<tr>
<td>windowed sinusoid, ( w(n)e^{j\omega_0} )</td>
<td>windowed narrow beam, ( w(n)e^{-j\phi_0} )</td>
</tr>
<tr>
<td>resolution of multiple sinusoids</td>
<td>resolution of multiple beams</td>
</tr>
<tr>
<td>frequency shifting by AM modulation</td>
<td>phased array scanning</td>
</tr>
<tr>
<td>filter design by window method</td>
<td>array design by window method</td>
</tr>
<tr>
<td>bandpass FIR filter design</td>
<td>angular sector array design</td>
</tr>
<tr>
<td>frequency-sampling design</td>
<td>Woodward-Lawson design</td>
</tr>
<tr>
<td>DFT</td>
<td>Blass matrix</td>
</tr>
<tr>
<td>FFT</td>
<td>Butler matrix</td>
</tr>
</tbody>
</table>

#### 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:

\[
R_{xx}(k) = E[x(n+k)x(n)] \quad \text{(autocorrelation function)}
\]

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:

\[
S_{xx}(\omega) = \sum_{k=-\infty}^{\infty} R_{xx}(k)e^{-jk\omega} \quad \text{(power spectrum)}
\]

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 \( S_{xx}(\omega) \),

\[
R_{xx}(k) = E[x(n+k)x(n)] = \int_{-\pi}^{\pi} S_{xx}(\omega)e^{j\omega k} \frac{d\omega}{2\pi}
\]

In particular, setting \( k = 0 \), we obtain the average power, or variance, of the signal \( x(n) \):

\[
\sigma_x^2 = R_{xx}(0) = E[x(n)^2] = \int_{-\pi}^{\pi} S_{xx}(\omega) \frac{d\omega}{2\pi} = \int_{-f_s/2}^{f_s/2} S_{xx}(f) \frac{df}{f_s}
\]

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 \( S_{xx}(z) \) or \( S_{xx}(f) \):

\[
S_{xx}(z) = \sum_{k=-\infty}^{\infty} R_{xx}(k) z^{-k}
\]

White noise has a delta-function autocorrelation and a flat spectrum, as shown below.

\[ R_{xx}(k) = \sigma_x^2 \delta(k) \]

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:

\[ R_{xx}(k) = E[x(n+k)x(n)] = E[x(n+k)]E[x(n)] = 0 \]

whereas for \( k = 0 \), we get the variance

\[ R_{xx}(0) = E[x(n)^2] = \sigma_x^2 \]

Combining them into a single equation, we have:

\[ R_{xx}(k) = \sigma_x^2 \delta(k) \quad \text{(white noise autocorrelation)} \]
9.6. PERIODOGRAM AND ITS IMPROVEMENTS

Only the \( k = 0 \) term survives the sum giving the flat spectral density (over the Nyquist interval):

\[
S_{xx}(f) = \sigma_x^2, \quad \text{for } -\frac{f_s}{2} \leq f \leq \frac{f_s}{2} \quad \text{(white noise spectrum)}
\]

Given a length-\( N \) block of signal samples \( x(n), n = 0, 1, \ldots, N-1 \), 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:

\[
\hat{R}_{xx}(k) = \frac{1}{N} \sum_{n=0}^{N-1-k} x(n+k)x(n) \quad \text{(sample autocorrelation)}
\]

for \( k = 0, 1, \ldots, N-1 \). The negative tail can be defined using the symmetry property \( \hat{R}_{xx}(-k) = \hat{R}_{xx}(k) \), so that we can write, for \( |k| \leq N-1 \),

\[
\hat{R}_{xx}(k) = \frac{1}{N} \sum_{n=0}^{N-1-|k|} x(n+|k|)x(n) \quad \text{(sample autocorrelation)}
\]

The rule of thumb is that only about the first 5-10% of the lags are statistically reliable, that is, \( 0 \leq k \leq N/10 \). The built-in MATLAB function, \texttt{xcorr}, computes \( \hat{R}_{xx}(k) \), for \(-M \leq k \leq M\), with any \( M \leq N-1 \), with usage,

\[
Rxx = \texttt{xcorr(x,M)}; \quad \% \text{with } M = \text{length(x)-1}
\]

\% examples
\[
x = 0:5;
x0 = \texttt{xcorr(x,3)};
Rxx = [14 26 40 55 40 26 14]
\]

\% examples
\[
x = xcorr(x,5);
Rxx = [0 5 14 26 40 55 40 26 14 5 0]
\]

with the outputs listed in the order:

\[
[\hat{R}_{xx}(-M), \ldots, \hat{R}_{xx}(-1), \hat{R}_{xx}(0), \hat{R}_{xx}(1), \ldots, \hat{R}_{xx}(M)]
\]

Alternatively, one can use the function, \texttt{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,

\[
Rxy = \texttt{acf(x,y,M)} = \text{autocorrelation of length-N vector } x
\]

\% examples
\[
x = \texttt{length-N vector}
y = \texttt{length-N vector}
M = \texttt{maximum lag (typically, } M < N-1, \text{ pads zeros if } M > N-1)
\]

\[
Rxy = [Rxy(0), Rxy(1), \ldots, Rxy(M)], \text{ row vector of non-negative lags}
\]

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):

\[
\hat{R}_{xx}(k) \overset{\text{asymptotically unbiased}}{\longrightarrow} R_{xx}(k) \quad \text{as } N \rightarrow \infty
\]

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 \( S_{xx}(\omega) \):

\[
\hat{S}_{xx}(\omega) = \sum_{k=-(N-1)}^{N-1} \hat{R}_{xx}(k)e^{-j\omega k} \quad \text{(periodogram spectrum)} \quad (9.6.1)
\]

Using the definition of \( \hat{R}_{xx}(k) \), and rearranging summations, we can express the periodogram in the alternative way:

\[
\hat{S}_{xx}(\omega) = \frac{1}{N} |X_N(\omega)|^2 \quad \text{(periodogram spectrum)} \quad (9.6.2)
\]

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 \texttt{freqz},

\[
X_N(\omega) = \sum_{n=0}^{N-1} x(n)e^{-j\omega n}
\]

\[
\hat{S}_{\text{mod}}(\omega) = \frac{1}{N} |X_w(\omega)|^2 \quad \text{modified periodogram} \quad (9.6.3)
\]

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,

\[
X_w(\omega) = \sum_{n=0}^{N-1} w(n)x(n)e^{-j\omega n}
\]

\[
\hat{S}_{\text{mod}}(\omega) = \frac{1}{N} |X_w(\omega)|^2 \quad \text{modified periodogram} \quad (9.6.3)
\]

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 \( N \),

\[
S_{xx}(\omega) = \lim_{N \to \infty} E\left[\hat{S}_{xx}(\omega)\right] = \lim_{N \to \infty} E\left[\frac{1}{N} |X_N(\omega)|^2\right]
\]

Since the sample autocorrelation is an asymptotically unbiased and consistent estimate of the true autocorrelation function, \( R_{xx}(k) = E\left[x(n+k)x(n)\right] \), 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 \( N \to \infty \), 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.

### Periodogram Averaging

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 \( \omega \). Denoting by \( S_i(\omega) \), \( i = 1, 2, \ldots, K \), the periodogram of the \( i \)-th segment, the averaged periodogram is calculated by,

\[
\hat{S}_w(\omega) = \frac{1}{K} \sum_{i=1}^{K} S_i(\omega) = \text{averaged periodogram} \quad (9.6.4)
\]

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,

\[
L = KN \quad \text{contiguous} \\
L = \frac{1}{2}(K+1)N \quad 50\% \text{overlapping} \quad (9.6.5)
\]

These capture the tradeoff between statistical reliability, which requires a large \( K \), and frequency resolution for each segment, which requires a large \( N \), since, \( \Delta \omega \approx 2\pi b/N \).

### Periodogram Smoothing

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 \( \hat{S}_{\text{per}}(\omega) \),

\[
\hat{S}_w(\omega) = W(\omega) * \hat{S}_{\text{per}}(\omega) = \int_{-\pi}^{\pi} W(\omega') \hat{S}_{\text{per}}(\omega - \omega') \frac{d\omega'}{2\pi} \quad (9.6.7)
\]

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 \( \omega \). An example of such window is the symmetric Bartlett window of length \( 2M + 1 \), or, length \( 2M - 1 \), since it vanishes at \( k = \pm M \),

\[
w(k) = 1 - \frac{|k|}{M}, \quad -M \leq k \leq M \quad \text{(Bartlett window)} \quad (9.6.8)
\]

having a non-negative DTFT, known as Fejér’s kernel,

\[
W(\omega) = \sum_{k=-M}^{M} \left( 1 - \frac{|k|}{M} \right) e^{-j\omega k} = \sum_{k=-M}^{M} w(k) e^{-j\omega k} = \frac{1}{M} \frac{\sin^2 \left( \frac{M\omega}{2} \right)}{\sin^2 \left( \frac{\omega}{2} \right)} \quad (9.6.9)
\]

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 \( M < L - 1 \), 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, \( x(n), n = 0, 1, \ldots, L - 1 \), first remove its sample mean, and then,

\[
\hat{R}_{xx}(k) = \frac{1}{L} \sum_{n=0}^{L-1} x(n + |k|) x(n), \quad -M \leq k \leq M
\]

\[
\hat{S}_w(\omega) = \sum_{k=-M}^{M} w(k) \hat{R}_{xx}(k) e^{-j\omega k} = \text{smoothed periodogram} \quad (9.6.6)
\]
Another lag window is the Parzen window, defined over \(-M \leq k \leq M\),
\[
w(k) = \begin{cases} 
1 - 6 \left( \frac{|k|}{M} \right)^2 + 6 \left( \frac{|k|}{M} \right)^3, & |k| \leq \frac{M}{2} \\
2 \left( 1 - \frac{|k|}{M} \right)^3, & \frac{M}{2} < |k| \leq M 
\end{cases}
\] (9.6.10)

with non-negative DTFT,
\[
W(\omega) = \frac{4 \left[ 3 - 2 \sin^2 \left( \frac{\omega}{2} \right) \right] \sin^4 \left( \frac{M \omega}{2} \right)}{M^4 \sin^4 \left( \frac{\omega}{2} \right)} 
\] (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 random 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:
\[
S_{yy}(\omega) = |H(\omega)|^2 S_{xx}(\omega) 
\] (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
\[
S_{yy}(\omega) = |H(\omega)|^2 \sigma_x^2 
\] (9.7.2)

Similarly, in the \(z\)-transform notation:
\[
S_{yy}(z) = H(z)H(z^{-1}) \sigma_x^2 
\] (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 \(z^{-1} = e^{-j\omega}\), we have:
\[
H(\omega)^* = \left( \sum_{n} h(n)e^{-j\omega n} \right)^* = \sum_{n} h(n)e^{j\omega n} = H(z^{-1}) 
\]

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 \(\sigma_y^2\). Using Eq. (9.7.1) applied to \(y(n)\), we have:
\[
\sigma_y^2 = \int_{-\pi}^{\pi} |S_{yy}(\omega)|^2 \frac{d\omega}{2\pi} = \sigma_x^2 \int_{-\pi}^{\pi} |H(\omega)|^2 \frac{d\omega}{2\pi} 
\]

which can be written in the form:
\[
\text{NRR} = \frac{\sigma_y^2}{\sigma_x^2} = \int_{-\pi}^{\pi} |H(\omega)|^2 \frac{d\omega}{2\pi} = \sum_{n} |h(n)|^2 
\] (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

\[
H(z) = \frac{1}{1 - z^{-1}}, \quad h(n) = u(n) 
\]

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:
\[
y(n) = x(n) + x(n - 1) + \cdots + x(1) + x(0) \quad (9.7.5)
\]

If \(x(n)\) is a zero-mean, white noise signal with variance \(\sigma_x^2\), 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 \( \sigma_x^2 \), by assumption. It follows that:

\[ \sigma_y^2(n) = E[y(n)^2] = (n+1)\sigma_x^2 \quad \text{(9.7.6)} \]

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 \( 0 < \rho \leq 1 \)

\[ H(z) = \frac{1}{1-z^{-1}} \quad \text{becomes} \quad H(\rho z^{-1}) = \frac{1}{1-\rho^2 z^{-2}} \]

More generally, a marginally stable filter can be stabilized by the substitution \( z \rightarrow \rho^{-1}z \), 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

\[ H(z) \rightarrow H(\rho^{-1}z), \quad h(n) \rightarrow \rho^m h(n) \quad \text{(9.7.7)} \]

### 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,

```matlab
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 \( 0 \leq f \leq f_s/2 \).

(b) Determine the frequency \( f_{\text{max}} \) of the dominant cycle, i.e., the frequency of the maximum of the periodogram spectrum, and calculate the estimated period \( 1/f_{\text{max}} \) 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:

```
% \( f = \) \ldots \quad \% \text{frequency range}
% \( S = \) \ldots \quad \% \text{in dB, as above}
% \( S_{\text{max}} = \max(S) \); \quad \% \text{max } S \text{ and its index}
% \( f_{\text{max}} = f(max) \); \quad \% \text{frequency at } S_{\text{max}}
```

(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,
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 x 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 pwerch 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,
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 \( k = 0, 1, \ldots, 63 \) 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, \( k = 0, 1, \ldots, N - 1 \), 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 \( 0 \leq k \leq 15 \) 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 dftfr.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:

- group A = 697, 770, 852, 941 Hz
- group B = 1209, 1336, 1477 Hz

A digital DTMF receiver computes the spectrum of the received dual-tone signal and deter-
mines 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, \( x(n), n = 0, 1, \ldots, 255 \). 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 \( n \).
9.9. PROBLEMS

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 \( 0 \leq f \leq 4 \text{ kHz} \).

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 \( 0 \leq f \leq 4 \text{ kHz} \).

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,

\[
X(\omega) = \sum_{n=0}^{L-1} x(n) e^{-j\omega n}
\]  

(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 \( \omega \) with period \( 2\pi \), so that we only need to evaluate \( X(\omega) \) within the symmetric Nyquist interval, \( -\pi \leq \omega \leq \pi \), or, within the right-sided Nyquist interval, \( 0 \leq \omega \leq 2\pi \), and mentioned that \( X(\omega) \) can be evaluated at any range of \( \omega \)s using, for example, MATLAB’s built-in function \texttt{freqz},

\[
\begin{align*}
\% \ x = \ldots & \quad \% \ \text{define length-}L \ \text{signal} \ x(n) \\
\% \ \text{omega} = \ldots & \quad \% \ \text{choose a range of omega’s} \\
X = \text{freqz}(x,1,\text{omega}); & \quad \% \ \text{DFT} \ X(\omega) \ \text{of same length as omega}
\end{align*}
\]

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, \( 0 \leq \omega \leq 2\pi \). These so-called “DFT frequencies” are defined in radians per sample as follows:

\[
\omega_k = \frac{2\pi k}{N}, \quad k = 0, 1, \ldots, N - 1
\]  

(10.1.2)

or, in Hz

\[
f_k = \frac{k f_s}{N}, \quad k = 0, 1, \ldots, N - 1
\]  

(10.1.3)

Thus, the \( N \)-point DFT will be, for \( k = 0, 1, \ldots, N - 1 \):

\[
X(\omega_k) = \sum_{n=0}^{L-1} x(n) e^{-j\omega_k n}
\]  

(N-point DFT of length-\( L \) signal)  

(10.1.4)
10.1. DISCRETE FOURIER TRANSFORM

Usually, \( X(\omega_k) \) is denoted by \( X_k \) in terms of the DFT index \( k = 0, 1, \ldots, N - 1 \). The C-function\(^1\) `dft` can be used to evaluate the array \( X_k \). Alternatively, the MATLAB computation in terms of `freqz` is,

\[
\begin{align*}
\text{freqz}(x, \omega, \text{N}) & \quad \text{defined length-}\text{N} \text{ signal } x(n) \\
\text{N} & \quad \text{DFT index} \\
\omega & \quad \text{N}-\text{dimensional vector of DFT frequencies} \\
X_k & \quad \text{N}-\text{dimensional DFT vector}
\end{align*}
\]

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 \( 2\pi \) is reflected in the periodicity of the DFT, \( X_k = X(\omega_k) \), in the index \( k \) with period \( N \), which follows from,

\[ \omega_{k+N} = \frac{2\pi(k + N)}{N} = 2\pi \frac{k}{N} + \omega_k + 2\pi \]

so that,

\[ X_{k+N} = X(\omega_{k+N}) = X(\omega_k + 2\pi) = X(\omega_k) = X_k \quad (10.1.5) \]

A consequence of this property is also the mapping of the negative-half, \( -\pi < \omega \leq 0 \), of the symmetric Nyquist interval, \( -\pi \leq \omega \leq 0 \), into the second-half, \( \pi \leq \omega \leq 2\pi \), of the right-sided Nyquist interval, \( 0 \leq \omega \leq 2\pi \), that is,

\[ X_{-k} = X_{N-k}, \quad k = 1, 2, \ldots, N - 1 \quad (10.1.6) \]

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, \( \omega_k = \Delta \omega_{\text{bin}}, k = 0, 1, \ldots, N - 1 \),

\[ \Delta \omega_{\text{bin}} = \frac{2\pi}{N} \quad \text{or} \quad \Delta f_{\text{bin}} = \frac{f_s}{N} \quad (10.1.7) \]

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,

\[ X^*_k = X_{-k} = X_{N-k}, \quad k = 0, 1, 2, \ldots, N - 1 \quad (10.1.8) \]

so that for example, \( X_0 = X_N \) (i.e., real-valued), \( X_1 = X_{N-1}, X_2 = X_{N-2} \), etc.

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, \( k = 0, 1, \ldots, 15 \),

\[ X_k = [16, 15, 14, 13, 12, 11, 10, 9, 8, 9, 10, 11, 12, 13, 14, 15] \]

will get remapped to the symmetric one, with indices, \( -8 \leq k \leq 7 \),

\[ X_k^{\text{shifted}} = [8, 9, 10, 11, 12, 13, 14, 15, 16, 15, 14, 13, 12, 11, 10, 9] \]

and shown in Fig. 10.1.2, implemented with MATLAB code,

\[ X = [16, 15, 14, 13, 12, 11, 10, 9, 8, 9, 10, 11, 12, 13, 14, 15]; \]

\[ X_{\text{shifted}} = \text{fftshift}(X); \]

\[ X_{\text{shifted}} = [8, 9, 10, 11, 12, 13, 14, 15, 16, 15, 14, 13, 12, 11, 10, 9] \]

\[ \text{Fig. 10.1.2 FFTSHIFT example.} \]

\[ X(\omega_k) = \sum_{n=0}^{L-1} x(n) z_k^n \quad (10.1.9) \]
where
\[ z_k = e^{j\omega k} = e^{j2\pi nk/N}, \quad k = 0, 1, \ldots, N - 1 \] (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:

\[ x = [x_0, x_1, \ldots, x_{L-1}] \]
\[ x_D = [x_0, x_1, \ldots, x_{L-1}, 0, 0, \ldots, 0] \]

Because \( x_D(n) = x(n) \) for \( 0 \leq n \leq L - 1 \) and \( x_D(n) = 0 \) for \( L \leq n \leq L + D - 1 \), the corresponding DTFTs will remain the same:

\[
X_D(\omega) = \sum_{n=0}^{L+D-1} x_D(n) e^{-j\omega n} = \sum_{n=0}^{L-1} x_D(n) e^{-j\omega n} + \sum_{n=L}^{L+D-1} x_D(n) e^{-j\omega n}
\]
\[
= \sum_{n=0}^{L-1} x(n) e^{-j\omega n} = X(\omega)
\]

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:

\[ x = [x_0, x_1, \ldots, x_{L-1}] \]
\[ x_D = [0, 0, \ldots, 0, x_0, x_1, \ldots, x_{L-1}] \]

will have DTFTs and DFTs:

\[ X_D(\omega) = e^{-j\omega D} X(\omega) \]
\[ X_D(\omega_k) = e^{-j\omega_D\omega} X(\omega_k), \quad k = 0, 1, \ldots, N - 1 \] (10.2.2)

### 10.3 Physical versus Computational Resolution

The bin width \( \Delta f_{\text{bin}} \) 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 DFT 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 \( 0 \leq k_0 \leq N - 1 \), such that

\[ f_0 = f_{k_0} = k_0 f_s/N \]

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 \( N - k_0 \):

\[ -f_0 = -k_0 f_s/N \]
\[ f_s - f_0 = f_s - k_0 f_s/N = (N - k_0) f_s/N \]

In summary, for each sinusoid with peaks at \( \pm f_0 \), we would like our DFT to show these peaks at the integers:
10.3. PHYSICAL VERSUS COMPUTATIONAL RESOLUTION

\[ L = 10, \; N = 32 \]

\[ L = 10, \; N = 64 \]

\[ L = 20, \; N = 32 \]

\[ L = 20, \; N = 64 \]

**Fig. 10.3.1** Physical versus computational resolution in DTFT computation.

\[ \{ f_0, -f_0 \} \Rightarrow \{ f_0, f_x - f_0 \} \Rightarrow \{ k_0, N - k_0 \} \quad (10.3.2) \]

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:

\[ \frac{f_1}{f_s} = 0.20, \; \frac{f_2}{f_s} = 0.25, \; \frac{f_3}{f_s} = 0.30 \quad (10.3.3) \]

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:

\[ \begin{align*}
  k_1 & = N \frac{f_1}{f_s} = 32 \cdot 0.20 = 6.4, \quad N - k_1 = 25.6 \\
  k_2 & = N \frac{f_2}{f_s} = 32 \cdot 0.25 = 8, \quad N - k_2 = 24 \\
  k_3 & = N \frac{f_3}{f_s} = 32 \cdot 0.30 = 9.6, \quad N - k_3 = 22.4
\end{align*} \]

Similarly, for \( N = 64 \), we find:

\[ \begin{align*}
  k_1 & = N \frac{f_1}{f_s} = 64 \cdot 0.20 = 12.8, \quad N - k_1 = 51.2 \\
  k_2 & = N \frac{f_2}{f_s} = 64 \cdot 0.25 = 16, \quad N - k_2 = 48 \\
  k_3 & = N \frac{f_3}{f_s} = 64 \cdot 0.30 = 19.2, \quad N - k_3 = 44.8
\end{align*} \]

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 \( N = 32 \):

\[ \begin{align*}
  k_1 & = 6.4 \Rightarrow k_1 = 6 \Rightarrow \frac{f_1}{f_s} = k_1 \frac{1}{N} = 0.1875 \\
  k_3 & = 9.6 \Rightarrow k_3 = 10 \Rightarrow \frac{f_3}{f_s} = k_3 \frac{1}{N} = 0.3125
\end{align*} \]

and for \( N = 64 \):

\[ \begin{align*}
  k_1 & = 12.8 \Rightarrow k_1 = 13 \Rightarrow \frac{f_1}{f_s} = k_1 \frac{1}{N} = 0.203125 \\
  k_3 & = 19.2 \Rightarrow k_3 = 19 \Rightarrow \frac{f_3}{f_s} = k_3 \frac{1}{N} = 0.296875
\end{align*} \]

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

\[ k = N \frac{f}{f_s} \quad (frequency \ in \ units \ of \ the \ DFT \ index) \quad (10.3.4) \]
The Nyquist interval $0 \leq f \leq f_s$ corresponds to the index interval $0 \leq k \leq N$. The $N$-point DFT is at the integer values $k = 0, 1, \ldots, N - 1$. For plotting purposes, the graph of the spectrum over the full interval $0 \leq k \leq N$ has been split into two side-by-side graphs covering the half-intervals: $0 \leq k \leq N/2$ and $N/2 \leq k \leq N$.

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 $2.5 \leq f \leq 5$ 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 $x$ of time samples. Its output is an $N$-dimensional vector $X$ of frequency samples. It is desired to compute the spectrum over the full interval $0 \leq f < 5$ 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 $x$ of time samples. Its output is an $N$-dimensional vector $X$ of frequency samples.

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 full interval:

$$\Delta f_{\text{bin}} = \frac{5 - 2.5}{128} = \frac{10}{N} \Rightarrow N = 512$$

The number of collected samples is $L = T_s f_s = (10 \text{msec}) \times (10 \text{kHz}) = 100$. Thus, the subroutine’s 512-dimensional input vector $x$ 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, $128 \leq k < 256$. □

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:

$$X = \begin{bmatrix} X_0 \\ X_1 \\ \vdots \\ X_{N-1} \end{bmatrix} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_L \\ \end{bmatrix}$$

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:

$$X = AX$$

where we denoted the DFT components by $X_k = X(\omega_k)$, $k = 0, 1, \ldots, N - 1$.

The matrix elements $A_{kn}$ are defined from Eq. (10.1.4):

$$A_{kn} = e^{-j\omega_k n} = e^{-2\pi jkn/N} = W_N^{kn}$$

$N \times L$ DFT matrix

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 $\Delta f_{\text{bin}}$.

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 $k = Nf_s/f_s = 128 \cdot 5/40 = 16$. The negative frequency $-5$ kHz is represented by the DFT index $N - k = 128 - 16 = 112$. □
for $0 \leq k \leq N - 1$ and $0 \leq n \leq L - 1$. For convenience, we defined the so-called twiddle factor $W_N$ as the complex number:

$$W_N = e^{-2\pi j/N}$$

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:

$$A_{0n} = 1, \quad 0 \leq n \leq L - 1 \quad \text{and} \quad A_{nk} = 1, \quad 0 \leq k \leq N - 1$$

The matrix $A$ can be built from its second row ($k = 1$), consisting of the successive powers of $W_N$:

$$A_{1n} = W_N^n, \quad n = 0, 1, \ldots, L - 1$$

It follows from the definition that the $k$th row is obtained by raising the second row to the $k$th power—element by element:

$$A_{kn} = W_N^k = (W_N^n)^k = A_{1n}$$

Some examples of twiddle factors, DFT matrices, and DFTs are as follows: For $L = N$ and $N = 2, 4, 8$, we have:

$$W_2 = e^{-2\pi j/2} = e^{-\pi j} = -1$$
$$W_4 = e^{-2\pi j/4} = e^{-\pi j/2} = \cos(\pi/2) - j\sin(\pi/2) = -j$$
$$W_8 = e^{-2\pi j/8} = e^{-\pi j/4} = \cos(\pi/4) - j\sin(\pi/4) = \frac{1 - j}{\sqrt{2}}$$

The corresponding 2-point and 4-point DFT matrices are:

$$A = \begin{bmatrix} 1 & 1 \\ 1 & W_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$
$$A = \begin{bmatrix} 1 & 1 \\ 1 & W_4 \\ 1 & W_4^2 \\ 1 & W_4^3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -j & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & j \end{bmatrix}$$

And, the 2-point and 4-point DFTs of a length-2 and a length-4 signal will be:

$$\begin{bmatrix} X_0 \\ X_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} x_0 + x_1 \\ x_0 - x_1 \end{bmatrix}$$
$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -j & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & j \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

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 $N$th roots of unity; indeed, in the notation of Eq. (10.1.10), it is the root $z_N$ and is shown in Fig. 10.1.3. Actually, all the successive powers $W_N^k$, $k = 0, 1, \ldots, N - 1$ are $N$th roots of unity, but in reverse order (i.e., clockwise) than the $z_k$ of Eq. (10.1.10):

$$W_N^k = e^{-2\pi jk/N} = z_{-k} = z_k^{-1}, \quad k = 0, 1, \ldots, N - 1$$

Figure 10.4.1 shows $W_N$ and its successive powers for the values $N = 2, 4, 8$. Because $W_2^8 = 1$, the exponents in $W_N^k$ can be reduced modulo-$N$, that is, we may replace them by $W_N^{kn \mod(N)}$.

![Twiddle lookup table](image)

For example, using the property $W_4^2 = 1$, 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

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W_4 & W_4^2 & W_4^3 \\ 1 & W_4^2 & W_4^4 & W_4^6 \\ 1 & W_4^3 & W_4^6 & W_4^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W_4 & W_4^2 & W_4^3 \\ 1 & W_4^2 & W_4^4 & W_4^6 \\ 1 & W_4^3 & W_4^5 & W_4 \end{bmatrix}$$

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,

$$\text{DFTmat} = @(N,L) \exp(-2\pi i * \text{zeros}(N-1:N:N-1)' / N * (0:L-1)); \quad \text{Usage: } A = \text{DFTmat}(N,L)$$
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 \( x \) 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 \( \tilde{x} \) has length \( N \).

The length \( L \) of the signal \( x \) 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 \( N \).

\[
\tilde{x} = x_0 + x_1 + \ldots + x_{N-1}
\]

\( \tilde{x} \) is periodic in \( N \) with period \( N \), that is, \( \tilde{x}(n) = \tilde{x}(n+N) = \tilde{x}(n+N) \). The definition (10.5.4) evaluates only one basic period \( \tilde{x} \) of \( \tilde{x} \), 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 \( 0 \leq n \leq N-1 \), thus, their sum is the wrapped signal \( \tilde{x} \).

\[
\tilde{x}(n) = \sum_{m=0}^{\infty} x(mN+n), \quad -\infty < n < \infty
\]

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 \( 0 \leq n \leq N-1 \) of \( \tilde{x}(n) \), which is all that is needed in the DFT.

\[\begin{bmatrix}
1 & 2 & 3 & 4 \\
-2 & -1 & 0 & 1 \\
3 & 4 & 5 & 0
\end{bmatrix}
\]

\[\begin{bmatrix}
1 & 2 & 3 & 4 \\
-2 & -1 & 0 & 1 \\
3 & 4 & 5 & 0
\end{bmatrix}
\]

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 \( \tilde{x} \) has the same \( N \)-point DFT as the original unwrapped signal \( x \), that is,
where \( \tilde{X}_k = \tilde{X}(\omega_k) \) is the \( N \)-point DFT of the length-\( N \) signal \( \tilde{x}(n) \):

\[
\tilde{X}_k = \tilde{X}(\omega_k) = \sum_{n=0}^{N-1} \tilde{x}(n) e^{-j\omega_k n}, \quad k = 0, 1, \ldots, N - 1
\]

(10.5.7)

In the notation of Eq. (10.4.2), we may write:

\[
\tilde{X}_k = \sum_{n=0}^{N-1} W_N^k \tilde{x}(n) = \sum_{n=0}^{N-1} A_{kn} \tilde{x}(n)
\]

(10.5.8)

where \( \tilde{A} \) is the DFT matrix defined as in Eq. (10.4.3):

\[
\tilde{A}_{kn} = W_N^k, \quad 0 \leq k \leq N - 1, \quad 0 \leq n \leq N - 1
\]

(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 \( \tilde{x} \) in the compact matrix form:

\[
\tilde{X} = \text{DFT}(\tilde{x}) = \tilde{A} \tilde{x}
\]

(10.5.10)

Thus, the above theorem can be stated in vector form:

\[
\tilde{x} = X = Ax = \tilde{A} \tilde{x}
\]

(10.5.11)

Symbolically, we will write \( \text{DFT}(\tilde{x}) = \text{DFT}(x) \) 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 \( m \)-th submatrix will be:

\[
A_{k,mN+n} = W_N^{kmN} = W_N^{mN+n}
\]

Using the property \( W_N^{N} = 1 \), it follows that \( W_N^{mN+1} = 1 \), and therefore:

\[
A_{k,mN+n} = W_N^{kN} = A_{kn} = \tilde{A}_{kn}, \quad 0 \leq k, n \leq N - 1
\]

Thus, in general, \( A \) is partitioned in the form:

\[
A = [\tilde{A}, \tilde{A}, \tilde{A}, \ldots]
\]

(10.5.12)

As an example, consider the case \( L = 8, N = 4 \). The \( 4 \times 8 \) DFT matrix \( A \) can be partitioned into two \( 4 \times 4 \) identical submatrices, which are equal to \( \tilde{A} \). Using \( W_4 = e^{-2\pi j/4} = -j \), we have:

\[
A = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & W_4 & W_4^2 & W_4^3 & W_4 & W_4^2 & W_4^3 & W_4^2 \\
1 & W_4^2 & W_4^3 & W_4^2 & W_4 & W_4^2 & W_4^3 & W_4^2 \\
1 & W_4^3 & W_4^2 & W_4^3 & W_4 & W_4^2 & W_4^3 & W_4^2 \\
\end{bmatrix}
\]

\[
= \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & -j & -1 & j & 1 & -j & -1 & j \\
1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
1 & j & -1 & -j & 1 & j & -1 & -j \\
\end{bmatrix}
\]

\[
= [\tilde{A}, \tilde{A}, \tilde{A}, \tilde{A}]
\]

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):

\[
X = Ax = [\tilde{A}, \tilde{A}, \tilde{A}, \tilde{A}] \begin{bmatrix}
x_0 \\
x_1 \\
x_2 \\
x_3 \\
\end{bmatrix} = \tilde{A}x_0 + \tilde{A}x_1 + \tilde{A}x_2 + \tilde{A}x_3 = \tilde{X}
\]

Figure 10.5.3 illustrates the relative dimensions of these operations. The DFT (10.5.10) of \( \tilde{x} \) requires \( N^2 \) complex multiplications, whereas that of \( x \) requires \( NL \). Thus, if \( L > N \), it is more efficient to first wrap the signal mod-\( N \) and then take its DFT.

\[
\begin{array}{c}
N \\
L
\end{array}
\]

\[
\begin{array}{c}
\tilde{X} \\
\tilde{A}
\end{array}
\]

\[
\begin{array}{c}
\tilde{x} \\
\tilde{A}
\end{array}
\]

\[
\begin{array}{c}
N \\
\tilde{A}
\end{array}
\]

\[
\begin{array}{c}
\tilde{x} \\
\tilde{A}
\end{array}
\]

\[
\begin{array}{c}
\tilde{x} \\
\tilde{A}
\end{array}
\]

\[
\begin{array}{c}
\tilde{x} \\
\tilde{A}
\end{array}
\]

\[
\begin{array}{c}
\tilde{x} \\
\tilde{A}
\end{array}
\]

\[
\begin{array}{c}
\tilde{x} \\
\tilde{A}
\end{array}
\]

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 \( x \) and (b) computing the DFT of its mod-4 reduction.
The 4×8 DFT matrix was worked out above. The corresponding DFT is:

\[
X = Ax = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & -j & -1 & j & 1 & -j & -1 & j \\
1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
1 & j & -1 & j & -1 & j & -1 & j
\end{bmatrix}
= \begin{bmatrix}
6 & 8+4j \\
3 & -2 \\
2 & 8-4j \\
1 & 1
\end{bmatrix}
\]

The same DFT can be computed by the DFT matrix \(\tilde{A}\) acting on the wrapped signal \(\tilde{x}\), determined in Example 10.5.1:

\[
\tilde{X} = \tilde{A}\tilde{x} = \begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & -j & -1 & j \\
1 & -1 & 1 & -1 \\
1 & j & -1 & j
\end{bmatrix}
\begin{bmatrix}
5 \\
0 \\
-3 \\
4
\end{bmatrix}
= \begin{bmatrix}
6 \\
8+4j \\
-2 \\
8-4j
\end{bmatrix}
\]

The two methods give identical results.

**Example 10.5.3:** The length \(L\) of the signal \(x\) 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

\[
x(n) = \begin{cases}
1 & n = 0, 1, \ldots, N-1 \\
0 & n < 0, n \geq N
\end{cases}
\]

To compute its \(N\)-point DFT, we determine its \(z\)-transform and evaluate it at the \(N\)th root of unity points \(z_k = e^{j2\pi k/N}\). This gives:

\[
X(z) = \sum_{n=0}^{N-1} x(n)z^{-n} = \sum_{n=0}^{N-1} a^n z^{-n} = \frac{a^N - 1}{1 - az^{-1}}
\]

Next, we compute its mod-\(N\) reduction by the sum (10.5.4):

\[
\tilde{x}(n) = \sum_{m=0}^{N-1} x(mN+n) = \sum_{m=0}^{N-1} a^m a^n = \frac{a^n - 1}{1 - a}, \quad n = 0, 1, \ldots, N-1
\]

where we used the geometric series sum. Computing its \(z\)-transform, we find:

\[
\tilde{X}(z) = \sum_{n=0}^{N-1} \tilde{x}(n)z^{-n} = \frac{1 - a^N z^{-N}}{1 - az^{-1}}
\]

Evaluating it at \(z = z_k\) and using the property that \(z_k^N = 1\), we find

\[
\tilde{X}_k = \tilde{X}(z_k) = \frac{1 - a^N z_k^{-N}}{1 - az_k^{-1}} = \frac{1 - a^N}{1 - az_k^{-1}} = X_k
\]

Thus, even though \(x(n)\) and \(\tilde{x}(n)\) are different and have different \(z\)-transforms and DFTTs, their \(N\)-point DFTs are the same.

The following C routine \texttt{modwrap.c} implements the modulo-\(N\) reduction operation. If \(L < N\), it pads \(N - L\) zeros at the end of \(x\) so that \(\tilde{x}\) 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.

```c
/* modwrap.c - modulo-N wrapping of length-L signal */

void modwrap(L, x, N, xtilde);

int L, N;

double *x, *xtilde;

{ double *x, *xtilde;

int n, r, m, M;

r = L % N;
M = (L - r) / N;

for (n = 0; n < N; n++)
{xtilde[n] = x[M * N + n];
if (n < r) xtilde[n] += x[m * N + n];
if (n < L, pad N - L zeros at end remaining blocks
for (m = 0; m < M; m++)
xtildel[n] += x[m * N + n];
if (L < N, this loop is skipped}
}

Usage: modwrap(L, x, N, xtilde);

int L, N;

double *x, *xtilde;

{ int n, r, m, M;

r = L % N;
M = (L - r) / N;

for (n = 0; n < N; n++)
{xtilde[n] = x[M * N + n];
if (n < r) xtilde[n] += x[m * N + n];
if (n < L, pad N - L zeros at end remaining blocks
for (m = 0; m < M; m++)
xtildel[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:

```c
modwrap(L, x, N, xtilde);

dft(X, N, xtilde, X);

DFTs() = DFTs()
```

Assuming \(L\) is a multiple of \(N\), \(L = MN\), the computational cost of the routine \texttt{modwrap} is \(N(M - 1) = MN\) MACs, whereas that of the above \texttt{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 \texttt{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 \texttt{datawrap} also implements the wrapping process. The FFT function in MATLAB, \texttt{X = fft(x, N)}, has the peculiar syntax that it truncates a length-\(L\) input signal vector \(x\) to length \(N\), if \(N \leq L\), discarding those parts of \(x\) 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 \(N(N-1) = 896\). The cost of the three methods will be:
10.5. MODULO-N REDUCTION

\( \text{(direct DFT)} \quad LN = 1024 \cdot 128 = 131,072 \)

\( \text{(wrapped DFT)} \quad N^2 + N(M - 1) = 128^2 + 128 \cdot (8 - 1) = 17,280 \)

\( \text{(wrapped FFT)} \quad \frac{1}{2} N \log_2 (N) + N(M - 1) = \frac{1}{2} \cdot 128 \cdot 7 + 128 \cdot (8 - 1) = 1,344 \)

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:

\( \text{(1024-point FFT)} \quad \frac{1}{2} L \log_2 (L) = \frac{1}{2} \cdot 1024 \cdot 10 = 5,120 \)

The DFT frequencies of the desired 128-point DFT are a subset of the DFT frequencies of the 1024-point DFT; indeed, we have:

\[ \omega_k = \frac{2 \pi k}{128} = \frac{2 \pi (8k)}{1024}, \quad k = 0, 1, \ldots, 127 \]

Thus, the 128-point DFT can be extracted from the 1024-point FFT by taking every eighth entry, that is, \( X_{128}(k) = X_{1024}(8k) \).

\[ \square \]

The two signals \( x \) and \( \bar{x} \) are not the only ones that have a common DFT. Any other signal that has the same \( \text{mod}-N \) reduction as \( x \) will have the same DFT as \( x \). To see this, consider a length-\( L \) signal \( y \) such that \( \hat{y} = \hat{x} \); then its \( N \)-point DFT can be obtained by applying Eq. (10.5.11):

\[ Y = A y = \hat{A} \hat{y} = \hat{A} \bar{x} = A x = X \]

For example, the following length-8 signals all have the same 4-point DFT,

\[
\begin{bmatrix}
    x_0 \\
    x_1 \\
    x_2 \\
    x_3 \\
    x_4 \\
    x_5 \\
    x_6 \\
    x_7 \\
\end{bmatrix} =
\begin{bmatrix}
    x_0 + x_4 \\
    x_1 + x_5 \\
    x_2 + x_6 \\
    x_3 + x_7 \\
    0 \\
    0 \\
    x_6 \\
    x_7 \\
\end{bmatrix} =
\begin{bmatrix}
    x_0 + x_4 \\
    x_1 + x_5 \\
    x_2 + x_6 \\
    x_3 + x_7 \\
    0 \\
    0 \\
    0 \\
    0 \\
\end{bmatrix} =
\begin{bmatrix}
    x_0 + x_4 \\
    x_1 + x_5 \\
    x_2 + x_6 \\
    x_3 + x_7 \\
\end{bmatrix}
\]

because all have the same \( \text{mod}-4 \) reduction:

\[ \bar{x} = \begin{bmatrix}
    x_0 + x_4 \\
    x_1 + x_5 \\
    x_2 + x_6 \\
    x_3 + x_7 \\
\end{bmatrix} \]

The above signals have a bottom half that becomes progressively zero, until the last vector which is recognized as the \( \bar{x} \), viewed as a length-8 vector. In fact, the \( \text{mod}-N \)

wrapped signal \( \bar{x} \) is unique in the above class of signals in the sense that it is \textit{shortest} signal, that is, of length \( N \), that has the same DFT as the signal \( x \).

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 \( y \) and \( x \) have equal \( \text{mod}-N \) reductions, \( \hat{y} = \hat{x} \) 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 \( N \)th roots of unity and using the equality of their \( z \)-transforms:

\[ F(z_k) = X(z_k) - Y(z_k) = X_k - Y_k = 0, \quad k = 0, 1, \ldots, N - 1 \]

Thus, the \( N \) complex numbers \( z_k \) are roots of the difference polynomial \( F(z) \). Therefore, \( F(z) \) will be divisible by the \( N \)th 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 \( N \)th root-of-unity zeros. Therefore, we can write:

\[ X(z) - Y(z) = F(z) = (1 - z^{-N}) Q(z) \quad \text{or,} \]

\[ X(z) = Y(z) + (1 - z^{-N}) Q(z) \quad \text{(10.5.13)} \]

Because \( X(z) \) and \( Y(z) \) have degree \( L - 1 \), it follows that \( Q(z) \) is an \textit{arbitrary} polynomial of degree \( L - 1 - N \). Denoting the coefficients of \( Q(z) \) by \( q(n) \), \( 0 \leq n \leq L - 1 - N \), we may write Eq. (10.5.13) in the time domain:

\[ x(n) = y(n) + q(n) - q(n - N) \]

\[ n = 0, 1, \ldots, L - 1 \quad \text{(10.5.14)} \]

Thus, any two \( x(n) \) and \( y(n) \) related by Eq. (10.5.14) will have the same \( N \)-point DFT. The \( \text{mod}-N \) reduction \( \bar{x} \) and its \( z \)-transform \( \hat{X}(z) \) are also related by Eq. (10.5.13):

\[ X(z) = (1 - z^{-N}) Q(z) + \hat{X}(z) \quad \text{(10.5.15)} \]

Because \( \hat{X}(z) \) has degree \( N - 1 \), Eq. (10.5.15) represents the division of the polynomial \( X(z) \) by the \( \text{DFT} \) polynomial \( 1 - z^{-N} \), with \( \hat{X}(z) \) being the \textit{remainder} polynomial and \( Q(z) \) the \textit{quotient} polynomial. The remainder \( \hat{X}(z) \) is the unique polynomial satisfying Eq. (10.5.15) that has \textit{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 $x$ from its $N$-point DFT $X$, that is, inverting the relationship:

$$X = Ax = \tilde{A}x$$  \hspace{1cm} (10.6.1)

When $L > N$, the matrix $A$ is not invertible. As we saw, there are in this case several possible solutions $x$, all satisfying Eq. (10.6.1) and having the same mod-$N$ reduction $\tilde{x}$.

Among these solutions, the only one that is uniquely obtainable from the knowledge of the DFT vector $X$ is $\tilde{x}$. The corresponding DFT matrix $\tilde{A}$ is an $N \times N$ square invertible matrix. Thus, we define the inverse DFT by

$$\tilde{x} = \text{IDFT}(X) = \tilde{A}^{-1}X$$  \hspace{1cm} (inverse DFT) \hspace{1cm} (10.6.2)

or, component-wise,

$$\tilde{x}_n = \sum_{k=0}^{N-1} (\tilde{A}^{-1})_{nk}X_k , \hspace{1cm} n = 0, 1, \ldots, N - 1$$  \hspace{1cm} (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 $\tilde{A}$:

$$\frac{1}{N} \tilde{A}^\ast \tilde{A} = I_N$$  \hspace{1cm} (10.6.4)

where $I_N$ is the $N$-dimensional identity matrix and $\tilde{A}^\ast$ 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:

$$\frac{1}{4} \tilde{A}^\ast = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ -j & 1 & j & -1 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Multiplying both sides of Eq. (10.6.4) by $\tilde{A}^{-1}$, we obtain for the matrix inverse:

$$\tilde{A}^{-1} = \frac{1}{N} \tilde{A}^\ast$$  \hspace{1cm} (10.6.5)

Thus, the IDFT (10.6.2) can be written in the form:

$$\tilde{x} = \text{IDFT}(X) = \frac{1}{N} \tilde{A}^\ast X$$  \hspace{1cm} (inverse DFT) \hspace{1cm} (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:

$$\tilde{A}^\ast X = (\tilde{A}X^\ast)^\ast = [\text{DFT}(X^\ast)]^\ast$$

where the matrix $\tilde{A}$ acting on the conjugated vector $X^\ast$ is the DFT of that vector. Dividing by $N$, we have:

$$\tilde{X} = \frac{1}{N} \text{IDFT}(X^\ast) = \frac{1}{N} [\text{DFT}(X^\ast)]^\ast$$  \hspace{1cm} (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.

$$\text{IFFT}(X) = \frac{1}{N} [\text{FFT}(X^\ast)]^\ast$$  \hspace{1cm} (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:

$$\tilde{x} = \text{IDFT}(X) = \frac{1}{N} \tilde{A}^\ast X = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -j & 1 & -1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 6 \\ 8 + 4j \\ -2 \\ 8 - 4j \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \\ -3 \\ 4 \end{bmatrix}$$

and using Eq. (10.6.7), we conjugate $X$ and transform it:

$$\frac{1}{N} (\tilde{A}X^\ast)^\ast = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ -1 & 1 & 1 & -1 \\ -j & -1 & j & 1 \end{bmatrix} \begin{bmatrix} 6 \\ 8 - 4j \\ -2 \\ 8 + 4j \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \\ -3 \\ 4 \end{bmatrix}$$

where the final overall conjugation was omitted because $\tilde{x}$ is real. \hspace{1cm} $\square$

Using Eq. (10.4.3) the matrix elements of $\tilde{A}^{-1}$ are:

$$(\tilde{A}^{-1})_{nk} = \frac{1}{N} \tilde{A}^\ast_{nk} = \frac{1}{N} (W_N^{nk})^\ast = \frac{1}{N} W_N^{-nk}$$

where we used the property $W_N^k = e^{2\pi i k/N} = W_N^{k+N}$. Then, Eq. (10.6.3) can be written in the form:

$$(\text{IDFT}) \hspace{1cm} \tilde{x}_n = \frac{1}{N} \sum_{k=0}^{N-1} W_N^{-nk}X_k , \hspace{1cm} n = 0, 1, \ldots, N - 1$$  \hspace{1cm} (10.6.9)

In terms of the DFT frequencies $\omega_k$, we have $X_k = X(\omega_k)$ and

$$W_N^{-nk} = e^{2\pi i nk/N} = e^{2\pi i n/4}$$

Therefore, the inverse DFT can be written in the alternative form:

$$(\text{IDFT}) \hspace{1cm} \tilde{x}(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(\omega_k)e^{i\omega_k n} , \hspace{1cm} n = 0, 1, \ldots, N - 1$$  \hspace{1cm} (10.6.10)
It expresses the signal $\tilde{x}(n)$ as a sum of $N$ complex sinusoids of frequencies $\omega_k$, 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 DFTs that use all frequencies, not just the $N$ DFT frequencies:

$$X(\omega) = \sum_{n=0}^{L-1} x(n)e^{-j\omega n}, \quad x(n) = \int_{0}^{2\pi} X(\omega)e^{j\omega n} \frac{d\omega}{2\pi} \quad (10.6.11)$$

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:

$$\int_{0}^{2\pi} X(\omega)e^{j\omega n} \frac{d\omega}{2\pi} \approx \sum_{k=0}^{N-1} X(\omega_k)e^{j\omega_k n} \Delta \omega_{\text{bin}}$$

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 $x$, it is necessary to have $\tilde{x} = x$. This happens only if the DFT length $N$ is at least $L$, so that there will be only one length-$N$ sub-block in $x$ and there will be nothing to wrap around. Thus, we have the condition:

$$\tilde{x} = x \quad \text{only if} \quad N \geq L \quad (10.6.12)$$

If $N = L$, then Eq. (10.6.12) is exact. If $N > L$, then we must pad $N - L$ zeros at the end of $x$ 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 $x$ that get wrapped around. Thus, we also have the condition:

$$\tilde{x} \neq x \quad \text{if} \quad N < L \quad (10.6.13)$$

### 10.7 Sampling of Periodic Signals and the DFT

The inverse DFT (10.6.10) defines the signal $\tilde{x}(n)$ for $n = 0, 1, \ldots, N - 1$. 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:

$$e^{j\omega n} = e^{j\omega (n+N)} = e^{j\omega nk/n}e^{j\omega nk} = e^{j\omega k}e^{j\omegakn/N} = e^{j\omega kn}$$

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 $\tilde{x} = x$, or, $x(n) = \tilde{x}(n)$. It follows that the periodic signal $x(n)$ may be represented by the discrete Fourier series (DFS):

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(\omega_k)e^{j\omega kn} \quad \text{(DFS)} \quad (10.7.1)$$

with the DFT playing the role of Fourier series coefficients:

$$X(\omega_k) = \sum_{n=0}^{N-1} x(n)e^{-j\omega_k n} \quad \text{(DFS coefficients)} \quad (10.7.2)$$

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 $f_m = mf_1$:

$$x(t) = \sum_{m=-\infty}^{\infty} c_m e^{j2\pi mf_1 t}$$

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:

$$f_k = kf_1 = k \frac{f_1}{N}, \quad k = 0, 1, \ldots, N - 1$$
10.7. SAMPLING OF PERIODIC SIGNALS AND THE DFT

Given an integer \( m \), we determine its quotient and remainder of the division by \( N \):

\[
m = qN + k, \quad 0 \leq k \leq N - 1
\]

and therefore, the corresponding harmonic will be:

\[
f_m = mf_1 = qNf_1 + kf_1 = qf_k + f_k
\]

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:

\[
x(nT) = \sum_{m=-\infty}^{\infty} c_m e^{2\pi j f_m n / f_s} = \sum_{k=0}^{N-1} \sum_{q=-\infty}^{\infty} c_{qN+k} e^{2\pi j (qf_k + f_k) n / f_s}
\]

where we wrote the summation over \( m \) as an equivalent double summation over \( q \) and \( k \). Noting that,

\[
e^{2\pi j f_k n / f_s} = e^{2\pi j k n / N} = 1
\]

and defining the aliased Fourier series amplitudes,

\[
b_k = \sum_{q=-\infty}^{\infty} c_{qN+k}, \quad k = 0, 1, \ldots, N - 1
\]

we obtain:

\[
x(nT) = \sum_{k=0}^{N-1} b_k e^{2\pi j f_k n / f_s} = \sum_{k=0}^{N-1} b_k e^{j \omega_k n}
\]

Comparing it with Eq. (10.7.1), we may identify the aliased amplitudes:

\[
b_k = \frac{1}{N} X(\omega_k), \quad k = 0, 1, \ldots, N - 1
\]

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:

\[
x_{al}(t) = \sum_{k=0}^{N-1} b_k e^{2\pi j f_k t}
\]

with the proviso that those harmonics \( f_k \) that lie in the right half of the Nyquist interval, \( f_s / 2 < f_k \leq f_s \), 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 \( x_{al}(t) \) 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:

\[
x = \begin{bmatrix} 0 \\ 1 \\ 0 \\ -1 \end{bmatrix} \Rightarrow X = AX = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & j & -1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ -2j \\ 0 \\ 0 \end{bmatrix}
\]

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:

\[
[f_0, f_1, f_2, f_3] = [0, 1, 2, 3] = [0, 1, 2, -1]
\]

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:

\[
x_{al}(t) = b_1 e^{2\pi j \mu t} + b_2 e^{-2\pi j \mu t} = \frac{1}{2j} e^{2\pi j \mu t} - \frac{1}{2j} e^{-2\pi j \mu t} = \sin(2\pi t)
\]

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:

\[
x = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ -1 \end{bmatrix} \Rightarrow X = AX = \begin{bmatrix} 0 \\ -2j(\sqrt{2} + 1) \\ 0 \\ -2j(\sqrt{2} - 1) \\ 0 \\ 0 \\ 2j(\sqrt{2} - 1) \\ -2j(\sqrt{2} + 1) \end{bmatrix} \begin{bmatrix} 0 \\ -j(\sqrt{2} + 1)/4j \\ 0 \\ -j(\sqrt{2} - 1)/4j \\ 0 \\ 0 \\ -j(\sqrt{2} - 1)/4j \\ -j(\sqrt{2} + 1)/4j \end{bmatrix}
\]

These amplitudes correspond to the frequencies \( f_k = kf_1 \):

\[
[0, 1, 2, 3, 4, 5, 6, 7] = [0, 1, 2, 3, -3, -2, -1]
\]

It follows that the aliased signal will be:

\[
x_{al}(t) = \frac{\sqrt{2} + 1}{4j} e^{2\pi j \mu t} + \frac{\sqrt{2} - 1}{4j} e^{2\pi j \mu t} - \frac{\sqrt{2} - 1}{4j} e^{-2\pi j \mu t} - \frac{\sqrt{2} + 1}{4j} e^{-2\pi j \mu t} = \frac{\sqrt{2} + 1}{2} \sin(2\pi t) + \frac{\sqrt{2} - 1}{2} \sin(6\pi t)
\]

which agrees with Example 1.4.6. The above 8-point DFT can be done using the \( 8 \times 8 \) DFT matrix, or, more quickly using an FFT by hand, as done in Example 10.8.3. \( \square \)
Example 10.7.2: Without performing any DFT or FFT computations, determine the 16-point DFT of the signal:

\[ x(n) = 1 + 2 \sin\left(\frac{\pi n}{2}\right) + 2 \cos\left(\frac{3\pi n}{4}\right) + \cos(\pi n), \quad n = 0, 1, \ldots, 15 \]

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:

\[ x(n) = 1 - je^{j\pi n/2} + je^{-j\pi n/2} + e^{j3\pi n/4} + e^{-j3\pi n/4} + e^{j\pi n} \]

Shifting the negative frequencies by \( 2\pi \), 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:

\[ x(n) = \frac{1}{16} \left[ 16e^{j\omega_0} - 16je^{j\omega_2} + 16e^{j\omega_4} + 16e^{j\omega_8} + 16e^{j\omega_{10}} + 16je^{j\omega_{12}} \right] \]

where the frequencies, their negatives, and their relative locations with respect to the 16 DFT roots of unity are as follows:

\[
\begin{align*}
\omega_4 &= \frac{2\pi \cdot 4}{16} = \frac{\pi}{2} \\
\omega_{12} &= \frac{2\pi \cdot 12}{16} = 2\pi - \frac{2\pi \cdot 4}{16} = 2\pi - \omega_4 \\
\omega_6 &= \frac{2\pi \cdot 6}{16} = \frac{3\pi}{4} \\
\omega_{10} &= \frac{2\pi \cdot 10}{16} = 2\pi - \frac{2\pi \cdot 6}{16} = 2\pi - \omega_6 \\
\omega_8 &= \frac{2\pi \cdot 8}{16} = \pi
\end{align*}
\]

Comparing with the IDFT, we identify the coefficients of the exponentials:

\[ X_0 = 16, \quad X_4 = -16j, \quad X_6 = X_8 = X_{10} = 16, \quad X_{12} = 16j \]

Thus, the 16-point DFT vector will be:

\[ X = [16, 0, 0, 0, -16j, 0, 16, 0, 16, 0, 16, 0, 16j, 0, 0, 0]^T \]

The 8-point DFT is obtained by picking every other entry of \( X \), that is,

\[ X = [16, 0, -16j, 16, 16, 16, 16j, 0]^T \quad (10.7.6) \]

This follows because the 8-point DFT frequencies are a subset of the 16-point ones, that is, \( \omega_k = 2\pi k/8 = 2\pi (2k)/16, k = 0, 1, \ldots, 7 \).

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 \( 0 \leq n \leq 7 \), 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:

\[ \omega_2 = \frac{2\pi - 2}{8} = \frac{\pi}{2}, \quad \omega_3 = \frac{2\pi - 3}{8} = \frac{3\pi}{4}, \quad \omega_4 = \frac{2\pi - 4}{8} = \pi \]

and \( x(n) \) can be written as:

\[ x(n) = \frac{1}{8} \left[ 8e^{j\omega_0} - 8je^{j\omega_2} + 8e^{j\omega_4} + 8e^{j\omega_8} + 8je^{j\omega_{10}} + 8je^{j\omega_{12}} \right] \]

comparing with the 8-point IDFT,

\[ x(n) = \frac{1}{8} \sum_{k=0}^{7} X_k e^{j\omega_k n} \]

we obtain:

\[ X = [8, 0, -8j, 8, 8, 8j, 0]^T \]

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.

\[ \Box \]

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:

\[ N = 2^B \quad \Rightarrow \quad B = \log_2(N) \quad (10.8.1) \]

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.
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:

\[
\frac{1}{2}NB = \frac{1}{2}N\log_2(N) \quad \text{(FFT computational cost)} \quad (10.8.3)
\]

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 \( 2N \cdot 4 = 12 \) 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 2 as a power of 2, in Eq. (10.8.1).

Given a length-\( N \) sequence \( x(n) \), for \( n = 0, 1, \ldots, N-1 \), its \( N \)-point DFT \( X(k) \) can be written in the component-form of Eq. (10.4.2):

\[
X(k) = \sum_{n=0}^{N-1} W_N^{kn} x(n), \quad k = 0, 1, \ldots, N-1 \quad (10.8.4)
\]

The summation index \( n \) ranges over both even and odd values in the range \( 0 \leq n \leq N-1 \). By grouping the even-indexed and odd-indexed terms, we may rewrite Eq. (10.8.4) as

\[
X(k) = \sum_{n=0}^{N/2-1} W_N^{k(2n)} x(2n) + \sum_{n=0}^{N/2-1} W_N^{k(2n+1)} x(2n+1)
\]

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 \( 0 \leq 2n \leq N-1 \). 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:

\[
0 \leq 2n \leq N-2 \Rightarrow 0 \leq n \leq \frac{N}{2} - 1
\]

Similarly, for the odd-indexed terms, we must have \( 0 \leq 2n+1 \leq N-1 \). Now the upper limit can be realized, but the lower one cannot; the smallest odd index is unity. Thus, we have:

\[
1 \leq 2n+1 \leq N-1 \Rightarrow 0 \leq 2n \leq N-2 \Rightarrow 0 \leq n \leq \frac{N}{2} - 1
\]

Therefore, the summation limits are the same for both terms:

\[
X(k) = \sum_{n=0}^{N/2-1} W_N^{k(2n)} x(2n) + \sum_{n=0}^{N/2-1} W_N^{k(2n+1)} x(2n+1) \quad (10.8.5)
\]
This expression leads us to define the two length- \( (N/2) \) subsequences:

\[
g(n) = x(2n), \quad n = 0, 1, \ldots, N/2 - 1
\]

\[
h(n) = x(2n + 1), \quad n = 0, 1, \ldots, N/2 - 1
\]

and their \( (N/2) \)-point DFTs:

\[
G(k) = \sum_{n=0}^{N/2-1} W_N^{kn} g(n), \quad k = 0, 1, \ldots, N/2 - 1
\]

\[
H(k) = \sum_{n=0}^{N/2-1} W_N^{kn} h(n)
\]

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 \( 2 \) and \( N/2 \) are related as follows:

\[
W_N = e^{-2\pi j/(N/2)} = e^{-4\pi j/N} = W_N^2
\]

Therefore, we may write:

\[
W_N^{2n} = (W_N)^{2n} = W_N^{kn}, \quad W_N^{2(n+1)} = W_N^{kn}W_N^{2n} = W_N W_N^{kn}
\]

Using the definitions (10.8.6), Eq. (10.8.5) can be written as:

\[
X(k) = \sum_{n=0}^{N/2-1} W_N^{kn} g(n) + W_N^{kN/2} \sum_{n=0}^{N/2-1} W_N^{kn} h(n)
\]

and using Eq. (10.8.7),

\[
X(k) = G(k) + W_N^k H(k), \quad k = 0, 1, \ldots, N - 1
\]

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 property that any DFT is periodic in \( N \)-dimensional DFT vector \( X \), the additional multiplications may be reduced by half to \( N/2 \). To see this, we split the full index range \( 0 \leq k \leq N - 1 \) into two half-ranges parametrized by the two indices \( k \) and \( k + N/2 \):

\[
0 \leq k \leq N/2 - 1 \Rightarrow \quad N/2 \leq k + N/2 \leq N - 1
\]

Therefore, we may write the \( N \) equations (10.8.8) as two groups of \( N/2 \) equations:

\[
X(k) = G(k) + W_N^k H(k)
\]

\[
X(k + N/2) = G(k + N/2) + W_N^{k+N2} H(k + N/2)
\]

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:

\[
X(k) = G(k) + W_N^k H(k)
\]

\[
X(k + N/2) = G(k) - W_N^k H(k)
\]

They are known as the butterfly merging equations. The upper group generates the upper half of the \( N \)-dimensional DFT vector \( X \); 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:

\[
\begin{bmatrix}
X_0 \\
X_1 \\
\vdots \\
X_{N/2-1}
\end{bmatrix} = \begin{bmatrix}
G_0 \\
G_1 \\
\vdots \\
G_{N/2-1}
\end{bmatrix} + \begin{bmatrix}
H_0 \\
H_1 \\
\vdots \\
H_{N/2-1}
\end{bmatrix} \times \begin{bmatrix}
W_0^N \\
W_1^N \\
\vdots \\
W_{N/2-1}^N
\end{bmatrix}
\]

(10.8.10)

\[
\begin{bmatrix}
X_{N/2} \\
X_{N/2+1} \\
\vdots \\
X_{N-1}
\end{bmatrix} = \begin{bmatrix}
G_0 \\
G_1 \\
\vdots \\
G_{N/2-1}
\end{bmatrix} - \begin{bmatrix}
H_0 \\
H_1 \\
\vdots \\
H_{N/2-1}
\end{bmatrix} \times \begin{bmatrix}
W_0^N \\
W_1^N \\
\vdots \\
W_{N/2-1}^N
\end{bmatrix}
\]

where the indicated multiplication is meant to be component-wise. Together, the two equations generate the full DFT vector \( X \). The operations are shown in Fig. 10.8.2.
For $N = 4$, we have $W_4^0 = -j$, and only the powers $W_4^0, W_4^1$ appear:

\[
\begin{align*}
X_0 &= [G_0] + H_0 W_4^0 \\
X_1 &= [G_1] + H_1 W_4^1 \\
X_2 &= [G_0] - H_0 W_4^0 \\
X_3 &= [G_1] - H_1 W_4^1
\end{align*}
\]

And, for $N = 8$, we have:

\[
\begin{align*}
X_0 &= [G_0] + H_0 W_8^0 \\
X_1 &= [G_1] + H_1 W_8^1 \\
X_2 &= [G_2] + H_2 W_8^2 \\
X_3 &= [G_3] + H_3 W_8^3 \\
X_4 &= [G_0] - H_0 W_8^0 \\
X_5 &= [G_1] - H_1 W_8^1 \\
X_6 &= [G_2] - H_2 W_8^2 \\
X_7 &= [G_3] - H_3 W_8^3
\end{align*}
\]

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 $X = [X_0]$ of a 1-point signal $x = [x_0]$ 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 $x$ is divided into two length-$(N/2)$ blocks $g$ and $h$ by putting every other sample into $g$ and the remaining samples into $h$.

During the second stage, the same subdivision is applied to $g$, resulting into the blocks $\{a, b\}$ and to $h$ resulting into the blocks $\{c, d\}$, and so on. Eventually, the signal $x$ 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:

\[x \rightarrow \{g, h\} \rightarrow \{a, b, c, d\} \rightarrow \cdots \rightarrow \text{1-point signals}\]

and the merging process rebuilds the corresponding DFTs:

\[\begin{align*}
X_0 &= \{A_0, A_1, G_0, G_1, X_0\} \\
X_1 &= \{A_1, A_0, G_1, G_0, X_1\} \\
X_2 &= \{A_1, A_0, G_1, G_0, X_2\} \\
X_3 &= \{A_0, A_1, G_0, G_1, X_3\} \\
X_4 &= \{A_0, A_1, G_0, G_1, X_4\} \\
X_5 &= \{A_1, A_0, G_1, G_0, X_5\} \\
X_6 &= \{A_1, A_0, G_1, G_0, X_6\} \\
X_7 &= \{A_0, A_1, G_0, G_1, X_7\}
\end{align*}\]

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 $0 \leq n \leq N - 1$, 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 $\{b_2, b_1, b_0\}$, which are zero or one:

\[n = (b_2 b_1 b_0) = b_2 2^2 + b_1 2^1 + b_0 2^0\]
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:

\[
  r = \text{bitrev}(n) = (b_0 b_1 b_2) = b_0 2^2 + b_1 2^1 + b_2 2^0
\]

We observe in Fig. 10.8.5 that the overall effect of the successive shuffling stages is to put the \( n \)th sample of the input array into the \( r \)th 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.

The following C routine \texttt{fft.c} implements the FFT algorithm, as described above. It consists of two parts: bit-reversal and merging.

```c
#include <cmplx.h>
void shuffle(), dftmerge();

void 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 \texttt{shuffle.c}, which calls the routines \texttt{swap.c} and \texttt{bitrev.c} that implement the swapping of the bit-reversed locations:

```c
#include <cmplx.h>

void swap();
int bitrev();

void shuffle(N, X)
    complex *X;
    int N;
{    
    int n, r, B=1;
    while ( (N >> B) > 0 )    
        B++;
        B--;
    N = 2^B
    for (n = 0; n < N; n++) {    
        r = bitrev(n, B);
        if (r < n) continue;
        swap(X+n, X+r);    
    }
}
```

```c
/* shuffle.c - in-place shuffling (bit-reversal) of a complex array */

#include <cmplx.h>

void shuffle(N, X)
    complex *X;
    int N;
{    
    int n, r, B=1;
    while ( (N >> B) > 0 )    
        B++;
        B--;
    N = 2^B
    for (n = 0; n < N; n++) {    
        r = bitrev(n, B);
        if (r < n) continue;
        swap(X+n, X+r);    
    }
}
```

```c
/* bitrev.c - bit reverse of a B-bit integer n */

#include <cmplx.h>

int bitrev(n, B)
    int n, B;
{    
    int m, r;
    for (r=0, m=1; m<0; m--)    
        if ((n >> m) == 1) {    
            if (r == 1) r = two(B-1-m);
            n -= two(m);
        }
    return(r);
}
```

A \( B \)-bit number \( n \) and its reverse can be expressed in terms of their bits as:
The routine `bitrev` builds \( r \) by determining if the \( m \)th 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 \( M = 2, 4, \ldots, N \).

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 ifft(N, X)
complex *X; int N;
{
    int k;
    for (k=0; k<N; k++)
        X[k] = conjg(X[k]);
    fft(N, X);
    for (k=0; k<N; k++)
        X[k] = rdiv(conjg(X[k]), (double)N);
}
```

Next, we present some FFT examples. In the merging operations from 2-point to 4-point DFTs and from 4-point DFTs to 8-DFTs, the following twiddle factors are used:

\[
\begin{bmatrix}
W_0^4 \\
W_1^4
\end{bmatrix} = \begin{bmatrix}
1 \\
-j
\end{bmatrix}, \quad \begin{bmatrix}
W_0^8 \\
W_1^8
\end{bmatrix} = \begin{bmatrix}
1 \\
(1-j)/\sqrt{2}
\end{bmatrix}, \quad \begin{bmatrix}
W_2^8 \\
W_3^8
\end{bmatrix} = \begin{bmatrix}
-j \\
-(1+j)/\sqrt{2}
\end{bmatrix}
\]

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.](image)

**Example 10.8.2:** Using the FFT algorithm, compute the 8-point DFT of the following 8-point signal:

\[ x = [4, -3, 2, 0, -1, -2, 3, 1]^T \]

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.](image)

**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.](image)
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 DFTs:

\[
y = h \ast x \quad \iff \quad Y(\omega) = H(\omega)X(\omega) \]

(10.10.1)

Therefore, \( y(n) \) can be recovered by the inverse DFT of the product of the two DFTs:

\[
y(n) = \frac{1}{N} \sum_{\omega=0}^{\pi} Y(\omega) e^{j\omega n} = \frac{1}{N} \sum_{\omega=0}^{\pi} H(\omega)X(\omega) e^{j\omega n} \]

(10.10.2)

Symbolically, we write Eq. (10.10.2) as:

\[
y = \text{IDFT}(\text{DFT}(h) \cdot \text{DFT}(x)) \]

(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 \( \omega \)-integration requires knowledge of \( Y(\omega) \) at a continuous range of \( \omega \)'s.

A practical approach is to replace all the DFTs 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:

\[
\tilde{y}(n) = \frac{1}{N} \sum_{k=0}^{N-1} Y(\omega_k) e^{j\omega_k n} = \frac{1}{N} \sum_{k=0}^{N-1} H(\omega_k)X(\omega_k) e^{j\omega_k n} \]

(10.10.4)

for \( n = 0, 1, \ldots, N - 1 \), or, written symbolically:

\[
\tilde{y} = \text{IDFT}(\text{DFT}(h) \cdot \text{DFT}(x)) \]

(10.10.5)

Because the unwrapped \( y \) is the ordinary convolution \( y = h \ast x \), we can write the above as the wrapped convolution:

\[
\tilde{y} = \tilde{h} \ast \tilde{x} = \text{IDFT}(\text{DFT}(h) \cdot \text{DFT}(x)) \quad \text{(mod-N circular convolution)} \]

(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:

\[
\tilde{y} = h \ast x = \text{IFFT}(\text{FFT}(h) \cdot \text{FFT}(x)) \]

(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 \( h \), \( h \), and the inverse FFT) plus the cost of the \( N \) complex multiplications \( Y(\omega_k)H(\omega_k)X(\omega_k) \), \( k = 0, 1, \ldots, N - 1 \). Thus, the total number of multiplications to implement Eq. (10.10.7) is:

\[
3 \frac{1}{2} N \log_2(N) + N \]

(10.10.8)

Some alternative ways of expressing \( \tilde{y} \) 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, \( \text{DFT}(h) = \text{DFT}(\tilde{h}) \) and \( \text{DFT}(x) = \text{DFT}(\tilde{x}) \). Thus, we can write:

\[
\tilde{y} = \tilde{h} \ast \tilde{x} = \text{IDFT}(\text{DFT}(h) \cdot \text{DFT}(x)) \]

(10.10.9)

\[
= \text{IDFT}(\text{DFT}(\tilde{h}) \cdot \text{DFT}(\tilde{x})) \]

(10.10.10)

\[
= \text{DFT}(\text{DFT}(h)) \cdot \text{DFT}(\text{DFT}(x)) \]

(10.10.11)

\[
= \text{DFT}(\text{DFT}(\tilde{h})) \cdot \text{DFT}(\text{DFT}(\tilde{x})) \]

According to Eq. (10.6.12), in order for the circular convolution \( \tilde{y} \) 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 \( N \):

\[
\tilde{y} = y \quad \text{only if} \quad N \geq L_y = L + M \]

(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:

\[
h = [1, 2, -1, 1], \quad x = [1, 1, 2, 1, 2, 1, 1] \]
**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:

\[ y = x * h = [1, 3, 3, 3, 7, 4, 3, 3, 0, 1] \]

The mod-3 circular convolution is obtained by dividing $y$ into length-3 contiguous blocks, wrapping them around, and summing them to get:

\[ y = [1, 3, 3][5, 3, 7][4, 3, 3][0, 1, 0] \Rightarrow \tilde{y} = [10, 10, 13] \]

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:

- (mod-5): \( y = [1, 3, 3, 5, 3][7, 4, 3, 3, 0][1] \Rightarrow \tilde{y} = [9, 7, 6, 8, 3] \)
- (mod-7): \( y = [1, 3, 3, 5, 5, 3, 7, 4][3, 3, 0, 1] \Rightarrow \tilde{y} = [4, 6, 3, 6, 3, 7, 4] \)
- (mod-9): \( y = [1, 3, 3, 5, 7, 4, 3, 3, 0, 1] \Rightarrow \tilde{y} = [1, 4, 3, 5, 7, 4, 3, 3] \)
- (mod-11): \( y = [1, 3, 3, 5, 5, 3, 7, 4, 3, 0, 1] \Rightarrow \tilde{y} = [1, 3, 3, 5, 7, 4, 3, 3, 0, 1] \)

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 $\tilde{y}$ resemble $y$ more and more. \( \square \)

**Example 10.10.2:** Recompute the length-3 circular convolution of the previous example by first wrapping mod-3 the signals $h$ and $x$, performing their linear convolution, and wrapping it mod-3.

**Solution:** We find for the mod-3 reductions:

- \( h = [1, 2, -1][1] \Rightarrow \tilde{h} = [2, 2, -1] \)
- \( x = [1, 1, 2][1, 2, 2][1, 1] \Rightarrow \tilde{x} = [3, 4, 4] \)

The convolution of the wrapped signals is:

\[ \tilde{h} * \tilde{x} = [2, 2, -1]*[3, 4, 4] = [6, 14, 13, 4, -4] \]

and, its mod-3 reduction:

\[ \tilde{h} * \tilde{x} = [6, 14, 13][4, -4] \Rightarrow \overline{\tilde{h} * \tilde{x}} = [10, 10, 13] \]

which agrees with $\tilde{y}$, in accordance with Eq. (10.10.9). \( \square \)

**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.

- \( h = [1, 2, 2, 1], \quad x = [1, 3, 3, 1] \)

**Solution:** The linear convolution is:

\[ y = h * x = [1, 2, 2, 1]*[1, 3, 3, 1] = [1, 5, 11, 14, 11, 5, 1] \]

wrapping it mod-4, we get:

\[ y = [1, 5, 11, 14][11, 5, 1] \Rightarrow \tilde{y} = [12, 10, 12, 14] \]

Alternatively, we compute the 4-point DFTs of $h$ and $x$:

\[ H = \begin{bmatrix} 1 & 1 & -j & 1 \\ 1 & -j & -1 & j \\ 1 & j & -1 & -1 \\ -1 + j & -1 - j \end{bmatrix} \]

\[ X = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & 1 & j \\ 1 & j & -1 & -j \\ 1 & -j & -1 & -j \end{bmatrix} \]

Multiplying them pointwise, we get:

\[ Y = \begin{bmatrix} H_0X_0 \\ H_1X_1 \\ H_2X_2 \\ H_3X_3 \end{bmatrix} = \begin{bmatrix} 48 \\ 4j \\ -4j \end{bmatrix} \]

To take the inverse DFT, we conjugate, take the 4-point DFT, divide by 4, and conjugate the answer:

\[ \tilde{y} = \text{IDFT}(Y) = \frac{1}{N}[\text{DFT}(Y^*)]^T \]

\[ \tilde{y} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & 1 & -1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 48 \\ 4j \\ -4j \end{bmatrix} = \begin{bmatrix} 12 \\ 10 \\ 12 \\ 14 \end{bmatrix} \]

The final conjugation is not necessary because $\tilde{y}$ is real. \( \square \)

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:

\[ Y(\omega_k) = H(\omega_k)X(\omega_k) \Rightarrow H(\omega_k) = \frac{Y(\omega_k)}{X(\omega_k)} \quad k = 0, 1, \ldots, N - 1 \]
Then, taking the inverse DFT, we have:

\[ \tilde{h}(n) = \frac{1}{N} \sum_{k=0}^{N-1} H(\omega_k) e^{i\omega_k n} = \frac{1}{N} \sum_{k=0}^{N-1} Y(\omega_k) e^{i\omega_k n} \]  
\[ (10.10.11) \]

or, symbolically,

\[ \tilde{h}(n) = H(\omega_k) e^{i\omega_k n} \]
\[ (10.10.12) \]

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

\[ h(n) = Aa^n + Bb^n, \quad n \geq 0 \]

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:

\[ \tilde{h}(n) = \sum_{m=0}^{\infty} h(mN + n) = \frac{A}{1 - a^N} a^n + \frac{B}{1 - b^N} b^n, \quad 0 \leq n \leq N - 1 \]

If \( N \) is large enough such that \( a^N \) and \( b^N \) are small enough to be ignored, then \( \tilde{h}(n) = h(n) \) for \( n = 0, 1, \ldots, N - 1 \).

### 10.11. **Overlap-Add and Overlap-Save Methods**

When the length \( L \) of the input signal \( x \) 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 \( x_0, x_1, x_2, \ldots \), is convolved with the order-\( M \) filter \( h \) producing the outputs blocks:

\[ y_0 = h \ast x_0 \]
\[ y_1 = h \ast x_1 \]
\[ y_2 = h \ast x_2 \]  
\[ (10.11.1) \]

and so on. The resulting blocks are pieced together according to their absolute timing. Block \( y_0 \) starts at absolute time \( n = 0 \); block \( y_1 \) starts at \( n = L \) because the corresponding input block \( x_0 \) starts then; block \( y_2 \) 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 \( x \) of length \( L \) and outputs the result also in blocks of length \( L \):

for each length-\( L \) input block \( x \), do:

1. compute the length-\( (L+M) \) output \( y \):
   \[ y = h \ast x, \quad \text{or, alternatively,} \]
   \[ y = \text{IFFT}(\text{FFT}(h) \cdot \text{FFT}(x)) \]
2. for \( i = 0, 1, \ldots, M-1 \):
   \[ y(i) = y(i) + y_{\text{temp}}(i) \quad \text{(overlap)} \]
   \[ y_{\text{temp}}(i) = y(i + L) \quad \text{(save tail)} \]
3. for \( i = 0, 1, \ldots, L-1 \):
   \[ \text{output } y(i) \]

The algorithm uses a temporary \( M \)-dimensional vector \( y_{\text{temp}} \) to store the last \( M \) samples of each previous block. Before processing the first block, \( y_{\text{temp}} \) must be initialized to zero.

After computing the length-\( (L+M) \) filter output, \( y = h \ast x \), the first \( M \) samples of \( y \) are added to the last \( M \) samples of the previous block held in \( y_{\text{temp}} \). Then, the last \( M \) samples of the currently computed block \( y \) are saved in \( y_{\text{temp}} \) for use in the next iteration. Only the first \( L \) corrected output samples of \( y \) are sent to the output.

A fast version of the method can be obtained by performing the convolutions of the
10.11. OVERLAP-ADD AND OVERLAP-SAVE METHODS

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:

\[
N = L + M \quad \Rightarrow \quad L = N - M \quad (10.11.3)
\]

With this choice of \( N \), there would be no wrap-around errors, and the outputs of the successive input blocks \( \{x_0, x_1, \ldots \} \), can be computed by:

\[
y_0 = \tilde{y}_0 = \text{IFFT} \left( \text{FFT}(h) \cdot \text{FFT}(x_0) \right)
y_1 = \tilde{y}_1 = \text{IFFT} \left( \text{FFT}(h) \cdot \text{FFT}(x_1) \right)
y_2 = \tilde{y}_2 = \text{IFFT} \left( \text{FFT}(h) \cdot \text{FFT}(x_2) \right)
\]

and so on. In counting the computational cost of this method, the FFT of \( h \) need not be counted. It can be computed once, \( H = \text{FFT}(h) \), 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:

\[
\frac{1}{2}N \log_2 N + N = N \left( \log_2 N + 1 \right)
\]

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:

\[
\frac{\text{fast}}{\text{slow}} = \frac{N \left( \log_2 N + 1 \right)}{(M+1)N} \geq \frac{\log_2 N}{M} \quad (10.11.5)
\]

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).

**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:

\[
x = [1, 1, 1, 1, 3, 3, 3, 3, 1, 1, 1, 2, 2, 2, 1, 1, 1, 1, 1]
\]

with the “short” filter:

\[
h = [1, -1, -1, 1]
\]

**Solution:** For comparison, we compute the linear convolution using the convolution table:

\[
y = [1, 0, -1, 0, 2, 0, -2, 0, 2, 1, 0, -1, 0, -1, 0, 1, 0, -1, 0, 1]
\]

For the overlap-save method, we divide the input into length-8 blocks which overlap by \( M = 3 \) points. These blocks are:

\[
x = [1, 1, 1, 1, 3, (3, 3, 3), 1, 1, 1, 2, 2, 2, (1, 1, 1), 1, 0, 0, 0, 0]
\]
10.12. COMPUTER EXPERIMENT – FAST CONVOLUTION

Convolving these blocks with \( h \) gives:

\[
\begin{align*}
y_0 &= h \ast [1,1,1,3,3,3] = [1,0,-1,0,2,0,-2,0,-3,0,3] \\
y_1 &= h \ast [3,3,3,1,1,2,2] = [3,0,-3,-2,0,2,1,0,-3,0,2] \\
y_2 &= h \ast [1,2,2,2,1,1,1] = [1,1,-1,0,-1,0,1,-1,0,1] \\
y_3 &= h \ast [1,1,1,0,0,0,0] = [1,0,-1,0,-1,0,1,0,0,0]
\end{align*}
\]

Reducing them modulo-8 and ignoring the first \( M \) points (indicated by *),

\[
\begin{align*}
\tilde{y}_0 &= [*,*,*+,0,2,0,-2,0] \\
\tilde{y}_1 &= [*,*,*-,2,0,2,1,0] \\
\tilde{y}_2 &= [*,*,*-,0,-1,0,1] \\
\tilde{y}_3 &= [*,*,*+,0,-1,0,1,0]
\end{align*}
\]

These would be the outputs computed via the FFT method. Putting them together, we obtain the overall output signal:

\[
y = [*,*,*,0,2,0,-2,0][-2,0,2,1,0][-1,0,-1,0,1][0,-1,0,1,0]
\]

With the exception of the first 3 points, the answer is correct.

\[\square\]

10.12 Computer Experiment – Fast Convolution

Write two MATLAB functions, \texttt{ovadd} and \texttt{ovsave}, that implement the overlap-add and overlap-save methods using both time-domain convolution or the FFT. They should have syntax,

\[
y = \texttt{ovadd}(h,x,N,'t'); \quad \text{using time-domain convolution}
y = \texttt{ovadd}(h,x,N,'f'); \quad \text{using the FFT}
\]

\[
y = \texttt{ovsave}(h,x,N,'t'); \quad \text{using time-domain convolution}
y = \texttt{ovsave}(h,x,N,'f'); \quad \text{using the FFT}
\]

\% 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,

\[
y = \texttt{conv}(h,x);
\]

In implementing the FFT versions, you need to perform the FFT of the filter \( h \) 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 \texttt{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 \texttt{ovadd} and \texttt{ovsave} functions.

\[
\begin{align*}
h &= [1 2 -1 1]; \quad \% \text{filter} \\
x &= [1 1 2 1 2 1 3 3 1 1]; \quad \% \text{input} \\
y &= \texttt{conv}(h,x); \quad \% \text{expected output}
\end{align*}
\]

\%--------------------------------------------

\% \texttt{ovadd(h,x,N,'t')} \% using time-domain convolution
\% \texttt{ovadd(h,x,N,'f')} \% using the FFT

\% \texttt{ovsave(h,x,N,'t')} \% using time-domain convolution
\% \texttt{ovsave(h,x,N,'f')} \% using the FFT

\% \texttt{ovadd(h,x,N,'t')} \% using time-domain convolution
\% \texttt{ovadd(h,x,N,'f')} \% using the FFT

\% \texttt{ovsave(h,x,N,'t')} \% using time-domain convolution
\% \texttt{ovsave(h,x,N,'f')} \% using the FFT

\% x = input signal vector
\% y = output signal vector
\% N = FFT length

\% additional zeros are padded at the end
\% to make complete columns

\% \texttt{ovadd(h,x,N,'t')} \% using time-domain convolution
\% \texttt{ovadd(h,x,N,'f')} \% using the FFT

\% \texttt{ovsave(h,x,N,'t')} \% using time-domain convolution
\% \texttt{ovsave(h,x,N,'f')} \% using the FFT

\% \texttt{conv(h,x)}; \% time-domain method

\% \texttt{real(ifft(H.*fft(s,N), N))}; \% FFT method

\% \texttt{ys(m) = y(m) + ytemp}; \% correct last M samples
\% \texttt{ys(m) = y(m) + ytemp}; \% save last M samples to be used in next block
\% \texttt{Y = [Y,ys]}; \% corrected output blocks

\% end \texttt{s-loop}
\% \texttt{ys} = \texttt{conv}(h,s); \% \texttt{time-domain method}
\% \texttt{ys} = \texttt{real(ifft(H.*fft(s,N), N))}; \% \texttt{FFT method}

\% additional zeros are padded at the end
\% to make complete columns

\% \texttt{ovadd(h,x,N,'t')} \% using time-domain convolution
\% \texttt{ovadd(h,x,N,'f')} \% using the FFT

\% \texttt{ovsave(h,x,N,'t')} \% using time-domain convolution
\% \texttt{ovsave(h,x,N,'f')} \% using the FFT

\% x = input signal vector
\% y = output signal vector
\% N = FFT length

\% additional zeros are padded at the end
\% to make complete columns

\% \texttt{ovadd(h,x,N,'t')} \% using time-domain convolution
\% \texttt{ovadd(h,x,N,'f')} \% using the FFT

\% \texttt{ovsave(h,x,N,'t')} \% using time-domain convolution
\% \texttt{ovsave(h,x,N,'f')} \% using the FFT

\% x = input signal vector
\% y = output signal vector
\% N = FFT length

\% additional zeros are padded at the end
\% to make complete columns

\% \texttt{ovadd(h,x,N,'t')} \% using time-domain convolution
\% \texttt{ovadd(h,x,N,'f')} \% using the FFT

\% \texttt{ovsave(h,x,N,'t')} \% using time-domain convolution
\% \texttt{ovsave(h,x,N,'f')} \% using the FFT

\% x = input signal vector
\% y = output signal vector
\% N = FFT length
10.12. COMPUTER EXPERIMENT - FAST CONVOLUTION

% 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 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 FFT at once
% end
% 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 =
% 0 2 1 % pads M zeros in front
% 0 1 3 % input blocks overlap by M samples
% 0 2 3 % pad extra M zeros at end in order
% 1 2 1
% 1 1 1
% 2 1 0
% 1 3 0

% H = fft(h(:),N); % N-point FFT of filter
Y = []; % collect output blocks
for s = Xbuff % loop over columns 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 points
end
y = Y(:); % concatenate output blocks

% y' =
% 1 3 3 5 3 7 4 3 6 9 5 3 4 0 1
% matches expected result

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 \( td \) corresponding to the time it takes for light to travel from radar to target and back to radar, that is, such that, \( 2d = ctd \), or, \( d = ctd/2 \). By measuring the time delay \( td \), 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, \( s(t) \), \( 0 \leq t \leq T \), and a reflected, delayed and attenuated, version of \( s(t) \), received in the presence of noise \( v(t) \)

\[
x_{rec}(t) = as(t - td) + v(t)
\]

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 \( td \).
10.13. COMPUTER EXPERIMENT – MATCHED FILTERING

(i) A correlator approach, whereby one computes the cross-correlation between the sampled received signal and the known pulse, that is,

\[ R_{xx}(k) = E[x_{\text{rec}}(n+k)s(n)] \]

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 \( x_{\text{rec}}(t) \) is filtered by an FIR filter whose impulse response is the reverse of the transmitted pulse, that is,

\[ h(t) = s(T - t), \quad 0 \leq t \leq T \]

and one determines the time at which the filter output is maximum,

\[ y'(t) = h(t) * x_{\text{rec}}(t) = \text{max} \]

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,

\[ s(t) = \sin(20\pi t + 10\pi t^2), \quad 0 \leq t \leq 1 \]

where \( t \) is in units of msec. The sinusoid is sampled at a rate of \( f_s = 1 \) MHz, and the sampled matched filter \( h(n) = h(nT_s) \) 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 \( \alpha = 0.5 \), that is, \( s_{\text{rec}}(t) = 0.5s(t-3) \), and measured over a time period, \( 0 \leq t \leq 5 \) msec.

The upper-right graph shows the same signal \( s_{\text{rec}}(t) \), deeply buried in noise, as in Eq. (10.13.1), such that the received pulse \( s_{\text{rec}}(t) \) 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,

```matlab
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 \( x_{\text{rec}}(t) \) 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 \( x_{\text{rec}}(t) \). 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.

---

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

\[ \mathbf{X} = [0, 4, -4j, 4, 0, 4, -4j, 4] \]
10.14. PROBLEMS

Using the FFT algorithm, compute the inverse DFT: \( x = \text{IFFT}(X) \). Using the given FFT \( X \), express \( x \) as a sum of real-valued (cosinusoidal signals.

10.4 When a very large FFT of a very large data set is required (e.g., of size \( 2^{36} \) 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 \( h = [1, 2, 1, 1], x = [1, 1, 1, 1] \) in two ways: (a) by computing their linear convolution and then reducing the result mod-4, (b) by first reducing \( h \) and \( x \) mod-4, computing the linear convolution of the reduced signals, and reducing the result mod-4.

10.6 Compute the 8-point FFT of \( x = [4, 2, 4, -6, 2, 4, -6] \). 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 \( x(n) \), \( n = 0, 1, \ldots, 7 \) as a linear combination of real-valued sinusoidal signals. Does your \( x(n) \) agree with the given sequence?

10.7 Let \( x = [1, 2, 3, 4, 5] \). (a) Determine a length-6 signal that has the same 5-point DFT as \( x \). (b) Determine a length-7 signal that has the same 5-point DFT as \( x \). Your answers should be nontrivial, that is, do not increase the length of \( x \) by padding zeros at its end.

10.8 Show the property:
\[
\frac{1}{N} \left[ 1 + W_N^k + W_N^{2k} + \cdots + W_N^{(N-1)k} \right] = \delta(k), \quad k = 0, 1, \ldots, N - 1
\]

10.9 Show the following properties:

a. \( W_N = W_{2N}^0 = W_{4N}^0 = \cdots = W_{pN}^0 \)

b. \( X_N(k) = X_{pN}(pk) \), \( k = 0, 1, \ldots, N - 1 \)

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 \( x_n \), \( 0 \leq n \leq 15 \), with 16-point DFT \( X_k \), \( 0 \leq k \leq 15 \), 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 \( x_d(t) \) 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 \( n = 0, 1, \ldots, 7 \):
\[
x(n) = 1 + 2\sin\left(\frac{\pi n}{4}\right) - 2\sin\left(\frac{\pi n}{2}\right) + 2\sin\left(\frac{3\pi n}{4}\right) + 3(-1)^n
\]

Without performing any DFT or FFT computations, determine the 8-point DFT of this signal.

10.13 Let \( x(n) = \cos(\pi n/2) + 2\cos(\pi n/8) \), \( n = 0, 1, \ldots, 15 \). Without performing any actual DFT/FFT computations, determine the 16-point DFT of this 16-point signal. [Hint: Compute \( x(n) \) with the 16-point inverse DFT formula.]

10.14 Let \( x(n) = \cos(\pi n/2) + 2\cos(\pi n/8) \), \( n = 0, 1, \ldots, 31 \). 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:
\[
x(n) = 0.5 + 2\sin(0.5\pi n) + 1.5\cos(\pi n), \quad n = 0, 1, \ldots, 15
\]
a. Determine the DFT of \( X(\omega) \) of this finite sequence, and sketch it roughly versus \( \omega \) in the range \( 0 \leq \omega \leq 2\pi \). [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 \( X = Ax \) be the \( N \)-point DFT of the length-\( N \) signal \( x \) expressed in matrix form, where \( A \) is the \( N \times N \) DFT matrix defined by its matrix elements \( A_{kn} = W_N^k n, k, n = 0, 1, \ldots, N - 1 \). Show that the inverse of this matrix can be obtained essentially by conjugating \( A \), that is, \( A^{-1} = \frac{1}{N} A^* \).

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 \( x = [2, 1, 1, 2] \) and \( h = [1, -1, -1, 1] \). 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
\[
h = [2, 1, 1, 1, 2, 1, 1, 1], \quad x = [2, 1, 2, -3, 2, 1, 2, -3]
\]
in two ways:
10.14. PROBLEMS

a. Working exclusively in the time domain.

b. Using the formula:

\[ \tilde{y} = \text{IFFT} \{ \text{FFT}(h \cdot \text{FFT}(x)) \} \]

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 signal \( x = [6, 1, 0, 1, 6, 1, 0, 1] \).

b. Using the inverse DFT formula, express \( x \) 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 \( x \). These signals must not begin or end with zeros.

d. Compute the 4-point FFT of \( x \) by carrying out a single 4-point FFT.

10.21 Let \( A(k) \) be the \( N \)-point DFT of a real-valued signal \( a(n), n = 0, 1, \ldots, N - 1 \). Prove the symmetry property:

\[ A(k)^* = A(N - k), \quad k = 0, 1, \ldots, N - 1 \]

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

\[ A(k) = \frac{1}{2} \left[ X(k) + X(N - k)^* \right], \quad B(k) = \frac{1}{2} \left[ X(k) - X(N - k)^* \right] \]

for \( k = 0, 1, \ldots, N - 1 \). 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 \( x(n), n = 0, 1, \ldots, N - 1 \) 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

\[ y(n) = x(2n) + jx(2n + 1) = g(n) + jh(n), \quad n = 0, 1, \ldots, N - 1 \]

Then, compute the \( N/2 \)-point FFT of \( y(n) \), say, \( Y(k), k = 0, 1, \ldots, N/2 - 1 \), and extract the \( N/2 \)-point FFTs of \( g(n) \) and \( h(n) \) by

\[ G(k) = \frac{1}{2} \left[ Y(k) + Y(N/2 - k)^* \right], \quad H(k) = \frac{1}{2} \left[ Y(k) - Y(N/2 - k)^* \right] \]

for \( k = 0, 1, \ldots, N/2 - 1 \). And finally, construct the desired \( N \)-point FFT by

\[ X(k) = G(k) + W_N^k H(k), \quad X(k + N/2) = G(k) - W_N^k H(k) \]

for \( k = 0, 1, \ldots, N/2 - 1 \). 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 \texttt{fftreal.c} that implements the method of Problem 10.23. The routine must have inputs/output declarations:

```c
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 \texttt{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 \texttt{fft} called on the full input array as usual.

10.25 Consider the following \( N \)-point signal and its reverse:

\[ x = [x_0, x_1, \ldots, x_{N-1}] \]
\[ x_R = [x_{N-1}, \ldots, x_1, x_0] \]

Show that the \( z \)-transform and \( N \)-point DFT of the reversed signal can be expressed as:

\[ X_R(z) = z^{-(N-1)} X(z^{-1}) \]
\[ X_R(k) = W_N^{-k} X(N - k), \quad k = 0, 1, \ldots, N - 1 \]

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 \( x \) and its reverse as defined in Problem 10.25. Construct the concatenated signal of length \( 2N \):

\[ y = [x, x_R] = [x_0, x_1, \ldots, x_{N-1}, x_{N-1}, \ldots, x_1, x_0] \]

a. Show that its \( z \)-transform can be expressed in terms of the \( z \)-transform of \( x \):

\[ Y(z) = X(z) + z^{-N} X_R(z) = X(z) + z^{-(2N-1)} X(z^{-1}) \]

b. Let \( Y_k \) be the \( (2N) \)-point DFT of \( y \). Show that it can be expressed in the form:

\[ Y_k = 2e^{j\omega_k/2} C_k, \quad k = 0, 1, \ldots, 2N - 1 \]

where \( \omega_k = 2\pi k / (2N) = \pi k / N \) is the \( k \)th frequency for the \( (2N) \)-point DFT and \( C_k \) is one form of the discrete cosine transform of \( x_R \) given by:

\[ C_k = \sum_{n=0}^{N-1} x_n \cos (\omega_k (n + 1/2)) \quad \text{(10.14.1)} \]

\[ \text{[Hint: Evaluate part (a) at the (2N)th roots of unity and multiply by } z^{-1/2}.] \]

c. Using the results of Problem 10.21, show that \( C_k \) satisfies the symmetry property:

\[ C_{2N-k} = -C_k, \quad k = 0, 1, \ldots, 2N - 1 \]

In particular, show \( C_N = 0 \).
10.14. PROBLEMS

467

10. DFT/FFT ALGORITHMS

Show the inverse DCT:

\[
x_n = \frac{1}{N} \sum_{k=0}^{2N-1} C_k e^{i \omega_k (n+1/2)}, \quad n = 0, 1, \ldots, N - 1
\]

Using the symmetry property of part (c), show the alternative inverse DCT, which uses only the first \(N\) DCT coefficients \(C_k, k = 0, 1, \ldots, N - 1\):

\[
x_n = \frac{1}{N} \left[ C_0 + 2 \sum_{k=1}^{N-1} C_k \cos (\omega_k (n + 1/2)) \right], \quad n = 0, 1, \ldots, N - 1 \tag{10.14.2}
\]

Together, Eqs. (10.14.1) and (10.14.2) form a forward/inverse DCT pair. The relationship to the doubled signal \(y\) allows an efficient calculation using (2\(N\))-point FFTs [257-259].

10.27 a. Let \(X_N(k)\) denote the N-point DFT of a length-L sequence \(x(n), n = 0, 1, \ldots, L - 1\). Show the relationships:

\[
X_N(k) = X_{2N}(2k), \quad k = 0, 1, \ldots, N - 1
\]

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”

\[
x_0 = [x_0, x_1, x_2, x_3, x_4] \quad x_1 = [x_4, x_0, x_1, x_2, x_3] \quad x_2 = [x_3, x_4, x_0, x_1, x_2] \quad x_3 = [x_2, x_3, x_4, x_0, x_1] \quad x_4 = [x_1, x_2, x_3, x_4, x_0]
\]

Show that the 5-point DFT \(X_i(k)\) of \(x_i\) is related to the 5-point DFT \(X_0(k)\) of \(x_0\) by

\[
X_i(k) = W_5^k X_0(k), \quad \text{for } i = 1, 2, 3, 4
\]

Explain this result in terms of ordinary “linear” shifts of the original sequence \(x_0\).

10.29 Show that the following, successively shorter, signals all have the same 4-point DFT:

\[
\begin{array}{cccc}
x_0 & x_0 & x_0 & x_0 \\
x_1 & x_1 & x_1 & x_1 \\
x_2 & x_2 & x_2 & x_2 \\
x_3 & x_3 + x_7 & x_3 + x_7 & x_3 + x_7 \\
x_4 & x_4 & x_4 & x_4 \\
x_5 & x_5 & 0 & 0 \\
x_6 & x_6 & 0 & 0 \\
x_7 & 0 & 0 & 0 \\
\end{array}
\]

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

\[
x = [1, 1, 1, 1, 3, 3, 3, 3, 3, 3, 3, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 1, 1, 1]
\]

with the “short” filter

\[
h = [1, -1, -1, 1]
\]

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 \(0 \leq t \leq 1\) sec as follows (see also Fig. 1.9.1):

\[
x(t) = \begin{cases} 
\frac{1}{2} (1 - \sin 2\pi f_1 t), & 0 \leq t \leq 0.25 \\
\frac{1}{2} (1 - \sin 2\pi f_2 t), & 0.25 \leq t \leq 0.75 \\
t - 1, & 0.75 \leq t \leq 1
\end{cases}
\]

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:

\[
x_\text{a}(t) = A \sin (2\pi f_1 t) + B \sin (2\pi f_2 t)
\]

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 \(x_\text{a}(t)\) 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:

\[
\text{ovsave} \ h \ . \ dat \ N < x \ . \ dat > y \ . \ dat
\]

Like the program firfilt1t.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,

\[
\tilde{y} = \text{IFFT} (H \cdot \text{FFT}(x))
\]

where the FFT of the filter \(H = \text{FFT}(h)\) 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 firfilt1t.c must be identical, up to perhaps some last zeros.)
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 $h = [h_0, h_1, \ldots, h_{N-1}]$ in the case of FIR filters, or the numerator and denominator coefficient vectors $b = [b_0, b_1, \ldots, b_M]$, $a = [1, a_1, \ldots, a_M]$ 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.

A given desired ideal frequency response, say $D(\omega)$, being periodic in $\omega$ with period $2\pi$, need only be specified over one complete Nyquist interval $-\pi \leq \omega \leq \pi$. The corresponding impulse response, say $d(k)$, is related to $D(\omega)$ by the DTFT and inverse DTFT relationships:

$$D(\omega) = \sum_{k=-\infty}^{\infty} d(k) e^{-j\omega k} \quad d(k) = \int_{-\pi}^{\pi} D(\omega) e^{j\omega k} \frac{d\omega}{2\pi}$$  \hspace{1cm} (11.1.1)

In general, the impulse response $d(k)$ will be double-sided and infinite. For many ideal filter shapes, the $\omega$-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

$$D(\omega) = \begin{cases} 1, & \text{if } -\omega_c \leq \omega \leq \omega_c \\ 0, & \text{if } -\pi \leq \omega < -\omega_c \text{ or } \omega_c < \omega \leq \pi \end{cases}$$

Therefore, Eq. (11.1.1) gives:

$$d(k) = \int_{-\pi}^{\pi} D(\omega) e^{j\omega k} \frac{d\omega}{2\pi} = \int_{-\omega_c}^{\omega_c} e^{j\omega k} \frac{d\omega}{2\pi} = \left[ e^{j\omega k} \right]_{-\omega_c}^{\omega_c} = e^{j\omega_c k} - e^{-j\omega_c k} \frac{2\pi}{2\pi k}$$
which can be rewritten as

\[
\begin{align*}
\text{(lowpass filter)} & \quad d(k) = \frac{\sin(\omega_c k)}{\pi k}, \quad -\infty < k < \infty \\
\text{(highpass filter)} & \quad d(k) = \frac{\sin(\omega_h k)}{\pi k}, \quad -\infty < k < \infty \\
\text{(bandpass filter)} & \quad d(k) = \frac{\sin(\omega_b k) - \sin(\omega_a k)}{\pi k}, \quad -\infty < k < \infty \\
\text{(bandstop filter)} & \quad d(k) = \frac{\sin(\omega_b k)}{\pi k} - \frac{\sin(\omega_a k)}{\pi k}, \quad -\infty < k < \infty
\end{align*}
\]  

(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):

\[
d(0) = \frac{\omega_c}{\pi}
\]  

(11.1.3)

Similarly, we find for the highpass, bandpass, and bandstop filters of Fig. 11.1.1, defined over \( -\infty < k < \infty \):

\[
\begin{align*}
\text{(highpass filter)} & \quad d(k) = \frac{\sin(\omega_h k)}{\pi k} \\
\text{(bandpass filter)} & \quad d(k) = \frac{\sin(\omega_b k) - \sin(\omega_a k)}{\pi k} \\
\text{(bandstop filter)} & \quad d(k) = \frac{\sin(\omega_b k)}{\pi k} - \frac{\sin(\omega_a k)}{\pi k}
\end{align*}
\]  

(11.1.4)

Note that for the same values of the cutoff frequencies \( \omega_c, \omega_h, \omega_b, \omega_a \), 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):

\[
\begin{align*}
\text{(difference)} & \quad d_{LP}(k) + d_{HP}(k) = \delta(k) \quad \Leftrightarrow \quad D_{LP}(\omega) + D_{HP}(\omega) = 1 \\
\text{(difference)} & \quad d_{BP}(k) + d_{BS}(k) = \delta(k) \quad \Leftrightarrow \quad D_{BP}(\omega) + D_{BS}(\omega) = 1
\end{align*}
\]  

(11.1.5)

As we see below, such complementarity properties can be exploited to simplify the implementation of loudspeaker crossover 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 \( \pm 1 \) depending on the algebraic sign of its argument. The \( \omega \)-integrations in Eq. (11.1.1) give the impulse responses:

\[
\begin{align*}
\text{(differentiator)} & \quad d(k) = \frac{\cos(\pi k)}{k} - \frac{\sin(\pi k)}{\pi k^2} \\
\text{(Hilbert transformer)} & \quad d(k) = \frac{1 - \cos(\pi k)}{\pi k}
\end{align*}
\]  

(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 \( \omega \); the antisymmetric ones have \( D(\omega) \) which is imaginary and odd in \( \omega \). 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:

\[
d(k) = \int_{-\pi}^{\pi} D(\omega) e^{j\omega k} d\omega, \quad -M \leq k \leq M
\]  

(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:

\[
d = [d_{-M}, \ldots, d_{-2}, d_{-1}, d_0, d_1, d_2, \ldots, d_M]
\]  

(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:

\[
h = d = [h_0, \ldots, h_{M-2}, h_{M-1}, h_M, h_{M+1}, h_{M+2}, \ldots, h_{2M}]
\]  

(11.1.9)

where we defined \( h_0 = d_{-M}, h_1 = d_{-M+1}, \ldots, h_M = d_0, \ldots, h_{2M} = d_M \). Thus, the vectors \( d \) and \( h \) are the same, with the understanding that \( d \)'s origin is in its middle and \( h \)'s at its left. The definition of \( h \) may be thought of as time-delaying the double-sided sequence \( d(k), -M \leq k \leq M \), by \( M \) time units to make it causal:

\[
h(n) = d(n - M), \quad n = 0, 1, \ldots, N - 1
\]  

(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).
11. WINDOW METHOD

For example, the $N$-dimensional approximation to the ideal lowpass filter of Eq. (11.1.2) will be:

$$h(n) = d(n-M) = \frac{\sin(\omega_c(n-M))}{\pi(n-M)}, \quad n = 0, \ldots, M, \ldots, N-1$$  \hspace{1cm} (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:

$$y_n = \sum_{m=0}^{N-1} h_m x_{n-m}$$  \hspace{1cm} (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,

$$d(k) = \frac{\sin(\pi k/4)}{\pi k}, \quad \text{for} \quad -5 \leq k \leq 5$$

We find the numerical values:

$$h = d = \left[\begin{array}{ccccccccccc}
-\sqrt{2}/10\pi & 0 & \sqrt{2}/6\pi & 1/2\pi & \sqrt{2}/2\pi & 1/4\pi & \sqrt{2}/6\pi & 0 & -\sqrt{2}/10\pi
\end{array}\right]$$

For the differentiator filter, the second term, $\sin(\pi k)/\pi k^2$, vanishes for all values $k \neq 0$. Therefore, we find:

$$h = d = \left[\begin{array}{ccccccccccc}
1/3 & -1/2 & 1/3 & -1/2 & 1 & 0 & -1/2 & 1/3 & -1/2 & 1/5
\end{array}\right]$$

And, for the Hilbert transformer:

$$h = d = \left[\begin{array}{ccccccccccc}
-2/3\pi & 0 & -2/3\pi & 0 & -2/3\pi & 0 & 2/3\pi & 0 & 2/3\pi & 0
\end{array}\right]$$

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). \hfill \Box

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:

$$\hat{D}_M(\omega) = \sum_{k=-M}^{M} d(k) e^{-j\omega k}$$  \hspace{1cm} (11.1.13)

Replacing $z = e^{j\omega}$, we may also write it as the double-sided $z$-transform:

$$\hat{D}_M(z) = \sum_{k=-M}^{M} d(k) z^{-k}$$  \hspace{1cm} (11.1.14)

The final length-$N$ filter obtained by Eq. (11.1.10) will have transfer function:

$$H(z) = z^{-M} \hat{D}_M(z) = z^{-M} \sum_{k=-M}^{M} d(k) z^{-k}$$  \hspace{1cm} (11.1.15)

and frequency response:

$$H(\omega) = e^{-j\omega M} \hat{D}_M(\omega) = e^{-j\omega M} \sum_{k=-M}^{M} d(k) e^{-j\omega k}$$  \hspace{1cm} (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 $d = [d_{-3}, d_{-2}, d_{-1}, d_0, d_1, d_2, d_3]$ with truncated $z$-transform:

$$\hat{D}_M(z) = d_{-3} z^3 + d_{-2} z^2 + d_{-1} z + d_0 + d_1 z^{-1} + d_2 z^{-2} + d_3 z^{-3}$$

Delaying it by $M = 3$, we get the causal transfer function:

$$H(z) = z^{-3} \hat{D}_M(z) = z^{-3} (d_{-3} z^3 + d_{-2} z^2 + d_{-1} z + d_0 + d_1 z^{-1} + d_2 z^{-2} + d_3 z^{-3})$$

$$= d_{-3} + d_{-2} z^{-2} + d_{-1} z^{-3} + d_0 + d_1 z^{-1} + d_2 z^{-2} + d_3 z^{-3}$$

$$= h_0 + h_1 z^{-1} + h_2 z^{-2} + h_3 z^{-3} + h_4 z^{-4} + h_5 z^{-5} + h_6 z^{-6}$$

where we defined $h(n) = d(n-3), n = 0, 1, 2, 3, 4, 5, 6$. \hfill \Box

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 $\omega$. 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:

$$\hat{D}_M(\omega) = \text{sign}(\hat{D}_M(\omega)) |\hat{D}_M(\omega)| = e^{j\beta(\omega)} |\hat{D}_M(\omega)|$$

where $\beta(\omega) = [1 - \text{sign}(\hat{D}_M(\omega))]/2$, which is zero or one depending on the sign of $\hat{D}_M(\omega)$. It follows that $H(\omega)$ will be:

$$H(\omega) = e^{-j\omega M} \hat{D}_M(\omega) = e^{-j\omega M - j\beta(\omega)} |\hat{D}_M(\omega)|$$

Thus, its magnitude and phase responses will be:

$$|H(\omega)| = |\hat{D}_M(\omega)|, \quad \arg H(\omega) = -\omega M + \pi \beta(\omega)$$  \hspace{1cm} (11.1.17)
making the phase response piece-wise linear in \( \omega \) with 180° jumps at those \( \omega \) 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 \( e^{j\pi/2} \). Thus, we have

\[
H(\omega) = e^{-j\omega M} \hat{D}_M(\omega) = e^{-j\omega M} e^{j\pi/2} A(\omega) = e^{-j\omega M} e^{j\pi/2} |A(\omega)|
\]

where \( \alpha(\omega) = \left[ 1 - \text{sign}(A(\omega)) \right]/2 \), which gives for the magnitude and phase responses:

\[
|H(\omega)| = |A(\omega)|, \quad \arg H(\omega) = -\omega M + \frac{\pi}{2} + \pi \alpha(\omega) \quad (11.1.18)
\]

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 \( \hat{D}_M(\omega) \approx D(\omega) \)?

Intuitively one would expect that \( \hat{D}_M(\omega) \rightarrow D(\omega) \) as \( N \) increases. This is true for any \( \omega \) 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 \( \hat{D}_M(\omega) \approx D(\omega) \), we consider the design of an ideal lowpass filter of cutoff frequency \( \omega_c \approx 0 \).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):

\[
h(n) = d(n - 20) = \frac{\sin(0.3\pi(n - 20))}{\pi(n - 20)}, \quad n = 0, 1, \ldots, 40
\]

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 \( h(60) = d(0) = 0.3 \):

\[
h(n) = d(n - 60) = \frac{\sin(0.3\pi(n - 60))}{\pi(n - 60)}, \quad n = 0, 1, \ldots, 120
\]

The two impulse responses are plotted in Fig. 11.1.4. Note that the portion of the second response extending \( \pm 20 \) 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:

\[
H(\omega) = \sum_{n=0}^{N-1} h(n) e^{-j\omega n} \quad (11.1.19)
\]

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

\[
h(n) = w(n) d(n - M), \quad -\infty < n < \infty \quad (11.1.20)
\]

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):

\[
H(\omega) = \int_{-\pi}^{\pi} W(\omega - \omega') e^{-j\omega'M} D(\omega') \frac{d\omega'}{2\pi} \quad (11.1.21)
\]

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 \( -\omega_c \leq \omega' \leq \omega_c \), the frequency convolution integral becomes:

\[
H(\omega) = \int_{-\omega_c}^{\omega_c} W(\omega - \omega') e^{-j\omega'M} \frac{d\omega'}{2\pi} \quad (11.1.22)
\]

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 \( \omega \)'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 \( \omega \), we have \( \hat{D}_N(\omega) \to D(\omega) \) 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 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:

\[
w(n) = 0.54 - 0.46 \cos \left( \frac{2\pi n}{N-1} \right), \quad n = 0, 1, \ldots, N - 1
\]  

(11.1.23)

In particular, the Hamming windowed impulse response for a length-\( N \) lowpass filter will be, where \( N = 2M + 1 \) and \( n = 0, 1, \ldots, N - 1 \):

\[
h(n) = w(n)d(n - M) = \left[ 0.54 - 0.46 \cos \left( \frac{2\pi n}{N-1} \right) \right] \sin \left( \frac{\pi}{N} (n - M) \right) \quad (11.1.24)
\]

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 phenomenon 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.

The DTFT/IDTFT relationships relating an ideal impulse response \( d_k \) to the corresponding frequency response \( D(\omega) \) represent the Fourier series expansion of the periodic function \( D(\omega) \), with the \( d_k \) being the “Fourier series coefficients”,

\[
D(\omega) = \sum_{k=-\infty}^{\infty} d_k e^{-j\omega k} \quad \Leftrightarrow \quad d_k = \int_{-\pi}^{\pi} D(\omega) e^{j\omega k} \, d\omega = \frac{2\pi}{2\pi} \quad (11.2.1)
\]

Generally, the convergence of the infinite series (11.2.1) can be proved under the so-called Dirichlet conditions, which require that over one \( 2\pi \)-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,

\[
D(\omega) = \lim_{M \to \infty} \hat{D}_M(\omega), \quad \text{where} \quad \hat{D}_M(\omega) = \sum_{k=M}^{M} d_k e^{-j\omega k} \quad (11.2.2)
\]

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 \( \omega \)
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 (dis-
continuities 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,
\[
\frac{1}{2\pi} \int_{-\pi}^{\pi} |D(\omega)|^2 \, d\omega = \sum_{k=-\infty}^{\infty} |d_k|^2 \quad \text{(Parseval)} \tag{11.2.3}
\]

Moreover, one can prove that \(\hat{D}_M(\omega)\) converges to \(D(\omega)\) in the following mean-square-error
sense,
\[
\lim_{M \to \infty} \frac{1}{2\pi} \int_{-\pi}^{\pi} |D(\omega) - \hat{D}_M(\omega)|^2 \, d\omega = 0 \tag{11.2.4}
\]

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 approx-
imation 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, \(w_k, -M \leq k \leq M\). The windowed coefficients are,
\[
\hat{d}_k = w_k d_k = \begin{cases} d_k, & -M \leq k \leq M \\ 0, & \text{otherwise} \end{cases}
\]

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,
\[
\hat{D}_M(\omega) = \int_{-\pi}^{\pi} \frac{\sin(N(\omega - \omega'))}{\pi(\omega - \omega')} D(\omega') \, d\omega' \tag{11.2.5}
\]

Next, consider an ideal lowpass filter with cutoff frequency \(\omega_c\),
\[
D(\omega) = \begin{cases} 1, & |\omega| < \omega_c \\ 0.5, & \omega = \pm \omega_c \\ 0, & |\omega| > \omega_c \end{cases}
\]

It can be expressed compactly either in terms of the Heaviside unit-step function, or
in terms of the sigmoid function, as follows,
\[
\hat{D}_M(\omega) = \int_{-\omega_c}^{\omega_c} \frac{\sin(N(\omega - \omega'))}{\pi(\omega - \omega')} \, d\omega'
\]

Introducing the sine-integral, \(\text{Si}(x)\), function,\(^\dagger\)
\[
\text{Si}(x) = \int_{0}^{x} \frac{\sin v}{v} \, dv
\]

we may express \(\hat{D}_M(\omega)\) as the difference,
\[
\hat{D}_M(\omega) = \frac{1}{\pi} \text{Si}(N(\omega + \omega_c)/2) - \frac{1}{\pi} \text{Si}(N(\omega - \omega_c)/2)
\]

In fact, the two \(\text{Si}\) 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 \(\text{Si}\) function.

Fig. 11.2.1 below demonstrates how the function, \(\text{Si}(x)/\pi\) approximates the function,
\(\text{sign}(x)/2\), and how, \(\text{Si}(Nk/2)/\pi\), approximates, \(\text{sign}(x)/2\), even better, being a compressed version of \(\text{Si}(x)/\pi\).

\(^\dagger\)it can be evaluated by MATLAB's built-in function, \text{sinint}.
The maximum of the Si function occurs at,
\[
\frac{d}{dx} \text{Si}(x) = \frac{\sin x}{x} = 0 \quad \Rightarrow x = \pi
\]
resulting in the maximum value, \( \text{Si}(\pi) = 0.58949 \), shown on the left graph of the figure, whose deviation from the maximum value of the function \( \text{sign}(x)/2 \), that is, from 1/2, is the Gibbs overshoot,
\[
\text{Gibbs overshoot} = \frac{1}{\pi} \text{Si}(\pi) \cdot \frac{1}{2} = 0.58949 - 0.5 = 0.08949 \quad (11.2.6)
\]
The maximum shown on right graph occurs at \( Nx/2 = \pi \), or, \( x = 2\pi/N \), and the maximum value is still the same, \( \text{Si}(\pi)/\pi = 0.58949 \).

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 \( \delta_{\text{pass}}, \delta_{\text{stop}} \), as well as the transition width \( \Delta f \).

The passband/stopband frequencies \( \{f_{\text{pass}}, f_{\text{stop}}\} \) are related to the ideal cutoff frequency \( f_c \) and transition width \( \Delta f \) by

\[
f_c = \frac{1}{2} (f_{\text{pass}} + f_{\text{stop}}), \quad \Delta f = f_{\text{stop}} - f_{\text{pass}} \quad (11.3.1)
\]

Thus, \( f_c \) is chosen to lie exactly in the middle between \( f_{\text{pass}} \) and \( f_{\text{stop}} \). Eqs. (11.3.1) can be inverted to give:

\[
f_{\text{pass}} = f_c - \frac{1}{2} \Delta f, \quad f_{\text{stop}} = f_c + \frac{1}{2} \Delta f \quad (11.3.2)
\]

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:

\[
A_{\text{pass}} = 20 \log_{10} \left( \frac{1 + \delta_{\text{pass}}}{1 - \delta_{\text{pass}}} \right), \quad A_{\text{stop}} = -20 \log_{10} \delta_{\text{stop}} \quad (11.3.3)
\]

A simplified version of the passband equation can be obtained by expanding it to first order in \( \delta_{\text{pass}} \), giving:

\[
A_{\text{pass}} = 17.372 \delta_{\text{pass}} \quad (11.3.4)
\]

which is valid for small values of \( \delta_{\text{pass}} \). Eqs. (11.3.3) can be inverted to give:

\[
\delta_{\text{pass}} = 10^{A_{\text{pass}}/20} - 1, \quad \delta_{\text{stop}} = 10^{-A_{\text{stop}}/20} \quad (11.3.5)
\]

Thus, one can pass back and forth between the specification sets:

\[
\{f_{\text{pass}}, f_{\text{stop}}, A_{\text{pass}}, A_{\text{stop}}\} \leftrightarrow \{f_c, \Delta f, \delta_{\text{pass}}, \delta_{\text{stop}}\}
\]

Although \( \delta_{\text{pass}} \) and \( \delta_{\text{stop}} \) 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,

\[
\delta = \min(\delta_{\text{pass}}, \delta_{\text{stop}}) \quad (11.3.6)
\]

The designed filter will have passband and stopband ripple equal to \( \delta \). The value of \( \delta \) can also be expressed in dB:

\[
A = -20 \log_{10} \delta, \quad \delta = 10^{-A/20} \quad (11.3.7)
\]

In practice, the design is usually based on the stopband ripple \( \delta_{\text{stop}} \). 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
The Kaiser window is designed as follows:

\[
\text{w}(n) = \frac{I_0(\alpha \sqrt{1 - (n-M)^2/M^2})}{I_0(\alpha)}, \quad n = 0, 1, \ldots, N-1
\]  

(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:

\[
\text{w}(n) = \frac{I_0(\alpha \sqrt{n(2M-n)/M})}{I_0(\alpha)}, \quad n = 0, 1, \ldots, N-1
\]  

(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.

The Kaiser window parameters \([N, \alpha]\) are computable in terms of the filter specifications, namely, the ripple \(\delta\) and transition width \(\Delta f\). The design equations developed by Kaiser [265–267] are as follows. The shape parameter \(\alpha\) is calculated from:

\[
\alpha = \begin{cases} 0.1102(A - 8.7), & \text{if } A \geq 50 \\ 0.5842(A - 21)^{0.4} + 0.07886(A - 21), & \text{if } 21 < A < 50 \\ 0, & \text{if } A \leq 21 \end{cases}
\]  

(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:

\[
\Delta f = \frac{Df}{N-1} \quad \leftrightarrow \quad N-1 = \frac{Df}{\Delta f}
\]  

(11.3.11)
where the factor $D$ is computed also in terms of $A$ by

$$D = \begin{cases}  & A - 7.95, \quad \text{if } A > 21 \\ & 14.36, \quad \text{if } A \leq 21 \end{cases}$$  \hspace{1cm} (11.3.12)$$

The most practical range of these formulas is for $A \geq 50$ dB, for which they simplify to:

$$\alpha = 0.1102(A - 8.7), \quad D = \frac{A - 7.95}{14.36} \quad \text{for } A \geq 50 \text{ dB}$$  \hspace{1cm} (11.3.13)$$

To summarize, the steps for designing a lowpass filter are as follows. Given the specifications $\{f_{pass}, f_{stop}, A_{pass}, A_{stop}\}$:

1. Calculate $f_c$ and $\Delta f$ from Eq. (11.3.1). Then, calculate $\omega_c = 2\pi f_c / f_s$.
2. Calculate $\delta_{pass}$ and $\delta_{stop}$ from Eq. (11.3.5).
3. Calculate $\delta = \min(\delta_{pass}, \delta_{stop})$ and $A = -20 \log_{10} \delta$ in dB.
4. Calculate $\alpha$ 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 $w(n)$, $n = 0, 1, \ldots, N - 1$ from Eq. (11.3.8).
7. Calculate the windowed impulse response, for $n = 0, 1, \ldots, N - 1$:

$$h(n) = w(n) d(n - M) = w(n) \cdot \frac{\sin(\omega_c(n - M))}{\pi(n - M)}$$  \hspace{1cm} (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 $\Delta f' = f_{pass} - f_{stop}$ and to use the highpass response from Eq. (11.1.4). The highpass impulse response will be:

$$h(n) = w(n) d(n - M) = w(n) \left[ \delta(n - M) - \frac{\sin(\omega_c(n - M))}{\pi(n - M)} \right]$$

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:

$$h(n) = \delta(n - M) - w(n) \cdot \frac{\sin(\omega_c(n - M))}{\pi(n - M)}$$  \hspace{1cm} (11.3.15)$$

For the same value of $\omega_c$, the lowpass and highpass filters are complementary. The sum of Eqs. (11.3.14) and (11.3.15) gives:

$$h_{LP}(n) + h_{HP}(n) = \delta(n - M), \quad n = 0, 1, \ldots, N - 1$$  \hspace{1cm} (11.3.16)$$

which becomes in the $z$-domain:

$$H_{LP}(z) + H_{HP}(z) = z^{-M}$$  \hspace{1cm} (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,

$$\Delta f = \min(\Delta f_a, \Delta f_b)$$  \hspace{1cm} (11.3.18)$$

where the left and right transition widths are:

$$\Delta f_a = f_p - f_s, \quad \Delta f_b = f_s - f_p$$  \hspace{1cm} (11.3.19)$$

Figure 11.3.3 shows the case where the left transition width is the smaller one and, thus, defines $\Delta f$. 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:

$$f_a = f_p - \frac{1}{2} \Delta f, \quad f_b = f_p + \frac{1}{2} \Delta f$$  \hspace{1cm} (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:

$$f_a = f_s + \frac{1}{2} \Delta f, \quad f_b = f_s - \frac{1}{2} \Delta f$$  \hspace{1cm} (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 $n = 0, 1, \ldots, N - 1$:

$$h(n) = w(n) d(n - M) = w(n) \cdot \frac{\sin(\omega_p(n - M)) - \sin(\omega_s(n - M))}{\pi(n - M)}$$  \hspace{1cm} (11.3.22)$$

where $h(M) = (\omega_p - \omega_s)/\pi$, 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:

- $f_s = 20$ kHz
- $f_{pass} = 4$ kHz, $f_{stop} = 5$ kHz
- $A_{pass} = 0.1$ dB, $A_{stop} = 80$ dB
Solution: First, we calculate $\delta_{\text{pass}}$ and $\delta_{\text{stop}}$ from Eq. (11.3.5):

$$\delta_{\text{pass}} = \frac{10^{\delta_{\text{pass}}/10} - 1}{10^{\delta_{\text{pass}}/10} + 1} = 0.0058, \quad \delta_{\text{stop}} = 10^{-80/20} = 0.0001$$

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 $\alpha$ 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):

$$N = 1 + \frac{Df_s}{\Delta f} = 101.35 \Rightarrow N = 103, \quad M = \frac{1}{2}(N - 1) = 51$$

The windowed impulse response will be, for $n = 0, 1, \ldots, 102$:

$$h(n) = w(n)(n - M) = \frac{L_0(7.857, \sqrt{\pi(102 - n)/51})}{L_0(7.857)} \cdot \sin(0.45\pi(n - 51))$$

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 $A_{\text{pass}} = 17.3726 = 0.0017$ 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.

---

**Example 11.3.2: Bandpass Design.** Using the Kaiser window, design a bandpass digital filter with the following specifications:

$$f_s = 20 \text{ kHz}, \quad f_{\text{pa}} = 4 \text{ kHz}, \quad f_{\text{pb}} = 6 \text{ kHz}, \quad f_{\text{pa}} = 8 \text{ kHz}$$

$$A_{\text{pass}} = 0.1 \text{ dB}, \quad A_{\text{stop}} = 80 \text{ dB}$$

Solution: The parameters ($\delta_{\text{pass}}, \delta_{\text{stop}}, \delta, \alpha, D$) are the same as in the previous example. The two transition widths are:

$$\Delta f_a = f_{\text{pa}} - f_{\text{pa}} = 4 - 3 = 1 \text{ kHz}, \quad \Delta f_b = f_{\text{pb}} - f_{\text{pb}} = 8 - 6 = 2 \text{ kHz}$$

Therefore, the minimum width is $\Delta f = \min(\Delta f_a, \Delta f_b) = 1 \text{ kHz}$, and the filter length:

$$N = 1 + \frac{Df_s}{\Delta f} = 101.35 \Rightarrow N = 103, \quad M = \frac{1}{2}(N - 1) = 51$$

Using the standard definition of Eq. (11.3.20), we find for the left and right ideal cutoff frequencies:

$$f_a = f_{\text{pa}} - \frac{1}{2}\Delta f = 4 - 0.5 = 3.5 \text{ kHz}, \quad f_b = f_{\text{pb}} + \frac{1}{2}\Delta f = 6 + 0.5 = 6.5 \text{ kHz}$$

with the normalized values $\omega_a = 2\pi f_a/f_s = 0.35\pi, \quad \omega_b = 2\pi f_b/f_s = 0.65\pi$.

For the alternative definition of Eq. (11.3.21), we have $f_a = 3 + 0.5 = 3.5$ and $f_b = 8 - 0.5 = 7.5 \text{ kHz}$, 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.
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 $n = 0, 1, \cdots, N - 1$

$$h_{LP}(n) = w(n) d_f(n - M) = w(n) \frac{\sin(\omega_c(n - M))}{\pi (n - M)}$$

$$h_{HP}(n) = w(n) d_f(n - M) = w(n) \left[ \delta(n - M) - \frac{\sin(\omega_c(n - M))}{\pi (n - M)} \right]$$

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, $20 \log_{10} |H(\omega)|$.

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 long single filter that can be implemented will have length such that $NT_{\text{inst}} = 20000 / 44.1 ≈ 453$, resulting in the smallest implementable transition width of

$$\Delta f_{\text{min}} = \frac{D_f}{N} = \frac{\frac{D_f}{T_{\text{inst}}}}{L} = 0.386 \text{ kHz}$$

**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].
11.3. KAISER WINDOW

Fig. 11.3.7 Complementary implementation of two-way crossover filters.

\[ h_{LP}(n) = w(n)d_{LP}(n-M) = w(n) \left[ \sin(\omega_a(n-M)) \pi(n-M) \right] \]

\[ h_{MP}(n) = w(n)d_{MP}(n-M) = w(n) \left[ \sin(\omega_b(n-M)) - \sin(\omega_a(n-M)) \pi(n-M) \right] \]

\[ h_{HP}(n) = w(n)d_{HP}(n-M) = w(n) \left[ \delta(n-M) - \sin(\omega_b(n-M)) \pi(n-M) \right] \]

where, \( \omega_a = \frac{2\pi f_a}{f_s} = 0.136\pi \) and \( \omega_b = \frac{2\pi f_b}{f_s} = 0.317\pi \). 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. \( \square \)

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 \( f_a = 3 \text{ kHz} \), \( f_b = 7 \text{ kHz} \), \( f_c = 11 \text{ kHz} \), \( f_d = 15 \text{ kHz} \), defining the five frequency bands:

\[
\begin{align*}
[0, f_a] & \quad \text{band 1} \\
[f_a, f_b] & \quad \text{band 2} \\
[f_b, f_c] & \quad \text{band 3} \\
[f_c, f_d] & \quad \text{band 4} \\
[f_d, f_s/2] & \quad \text{band 5}
\end{align*}
\]

The designed filter impulse responses will be:

\[ h_1(n) = w(n) \left[ \sin(\omega_a(n-M)) \pi(n-M) \right] \]

\[ h_2(n) = w(n) \left[ \sin(\omega_b(n-M)) - \sin(\omega_a(n-M)) \pi(n-M) \right] \]

\[ h_3(n) = w(n) \left[ \sin(\omega_c(n-M)) - \sin(\omega_b(n-M)) \pi(n-M) \right] \]

\[ h_4(n) = w(n) \left[ \sin(\omega_d(n-M)) - \sin(\omega_c(n-M)) \pi(n-M) \right] \]

\[ h_5(n) = w(n) \left[ \delta(n-M) - \sin(\omega_d(n-M)) \pi(n-M) \right] \]

where, \( \omega_a = \frac{2\pi f_a}{f_s} = 0.136\pi \), \( \omega_b = \frac{2\pi f_b}{f_s} = 0.317\pi \), \( \omega_c = \frac{2\pi f_c}{f_s} = 0.499\pi \), \( \omega_d = \frac{2\pi f_d}{f_s} = 0.680\pi \). 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

\[ H_1(z) + H_2(z) + H_3(z) + H_4(z) + H_5(z) = z^{-M} \]
11.3. KAISER WINDOW

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.

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:

\[ I_0(x) = \sum_{k=0}^{\infty} \left( \frac{x}{2} \right)^k \frac{1}{k!} \]

(11.3.23)

The Kaiser window (11.3.8) requires the evaluation of \( I_0(x) \) over the range of argument \( 0 \leq x \leq \alpha \). The following C function \texttt{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].

```c
#include <math.h>
#define eps (1.E-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):

\[ S_n = \sum_{k=0}^{n} \left( \frac{x}{2} \right)^k \frac{1}{k!} \]

It is initialized to \( S_0 = 1 \) and satisfies the recursion, for \( n \geq 1 \):

\[ S_n = S_{n-1} + \left( \frac{x}{2} \right)^n \frac{1}{n!} \]
11.3. KAISER WINDOW

\[ S_n = S_{n-1} + D_n, \quad \text{where} \quad D_n = \left( \frac{x}{2n} \right)^2 \]

In turn, \( D_n \) itself satisfies a recursion, for \( n \geq 1 \):

\[ D_n = \left( \frac{x}{2n} \right)^2 D_{n-1} = T_n^2 D_{n-1}, \quad \text{where} \quad T_n = \frac{x}{2n} \]

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 \( \epsilon \) is a small number, such as, \( \epsilon = 10^{-9} \). The Kaiser window itself may be calculated by invoking the routine 10..c; for example:

\[
\text{IOa} = \text{IO}(\text{alpha}); \\
\text{for (n=0; n<N; n++)} \\
\quad n[n]=\text{IO}((\text{alpha} * \sqrt(n*(2*M-n))) / M) / \text{IOa};
\]

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 \( \Delta f_w \) of a window depends inversely on the data record length \( L \):

\[ \Delta f_w = \frac{cf_s}{L-1} \Leftrightarrow L-1 = \frac{cf_s}{\Delta f_w} \]  

(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 \( \Delta f_w \), 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 \( \Delta f_w \), the design equations determine the length \( L \) and shape parameter \( \alpha \) of the window. The length is determined from Eq. (11.3.24), where \( c \) is given in terms of \( R \) by:

\[ c = \frac{6(R + 12)}{155} \]  

(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 \( \Delta f_w \) to be half the base of the mainlobe, instead of the base itself. The window shape parameter \( \alpha \) can be calculated also in terms of \( \Delta f_w \) and \( L \), where \( \{L, \alpha\} \) have been determined, the window may be calculated by:

\[ w(n) = \frac{I_0 \left( \alpha \sqrt{1 - (n - M)^2/M^2} \right)}{I_0(\alpha)}, \quad n = 0, 1, \ldots, L - 1 \]  

(11.3.27)

where \( M = (L - 1)/2 \), and then applied to a length-\( L \) data record by

\[ x_L(n) = w(n)x(n), \quad n = 0, 1, \ldots, L - 1 \]  

(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 \( \delta \) 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:

\[ x(t) = A_1 \cos(2\pi f_1 t) + A_2 \cos(2\pi f_2 t) + A_3 \cos(2\pi f_3 t) \]

<table>
<thead>
<tr>
<th>Window</th>
<th>( R )</th>
<th>( c )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rectangular</td>
<td>−13 dB</td>
<td>1</td>
</tr>
<tr>
<td>Hamming</td>
<td>−40 dB</td>
<td>2</td>
</tr>
<tr>
<td>Kaiser</td>
<td>variable( R )</td>
<td>( R/12 )/155</td>
</tr>
</tbody>
</table>

Table 11.3.2 Relative sidelobe levels.
11.3. KAISSER WINDOW

where \( t \) is in msec. The relative amplitudes are

\[
A_1 = A_3 = 1, \quad A_2 = 10^{-50/20} = 0.0032
\]

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 side lobes of the window. Thus, we must use a window whose side lobe 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 \( \Delta f = (f_2 - f_1)/T = 0.167 \) kHz. The Kaiser window parameters are calculated as follows:

\[
\alpha = 0.12438(R + 6.3) = 9.490, \quad c = \frac{6(R + 12)}{155} = 3.174
\]

\[
L = 1 + \frac{cf}{\Delta f} = 191.45 \quad \Rightarrow \quad L = 193, \quad M = \frac{1}{2}(L - 1) = 96
\]

The length-\( L \) sampled signal is, for \( n = 0, 1, \ldots, L - 1 \):

\[
x(n) = A_1 \cos(2\pi f_1 n/f_s) + A_2 \cos(2\pi f_2 n/f_s) + A_3 \cos(2\pi f_3 n/f_s)
\]

The Kaiser and Hamming windowed signals will be, for \( n = 0, 1, \ldots, L - 1 \):

\[
x_K(n) = w(n)x(n) = \frac{I_0(\alpha\sqrt{n(M-\frac{M}{2})})}{I_0(\alpha)} \cdot x(n)
\]

\[
x_H(n) = w(n)x(n) = \left[ 0.54 - 0.46 \cos \left( \frac{2\pi n}{L - 1} \right) \right] \cdot x(n)
\]

The corresponding spectra are:

\[
X_K(f) = \sum_{n=0}^{L-1} x_K(n) e^{-2\pi j fn/f_s}, \quad X_H(f) = \sum_{n=0}^{L-1} x_H(n) e^{-2\pi j fn/f_s}
\]

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 \( \text{dft.m} \) 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 side lobes 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 \( \Delta f \). The width of the Hamming peaks is \( \Delta f = c f_s/(L - 1) \) with \( c = 2 \), or, \( \Delta f = 0.104 \) kHz.

\[
\text{Fig. 11.3.12} \quad \text{Kaiser and Hamming spectra.}
\]

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:

\[
\tilde{d}(k) = \frac{1}{N} \sum_{i=-M}^{M} D(\omega_i) e^{j\omega_i k}, \quad -M \leq k \leq M \quad (11.4.1)
\]

where \( N = 2M + 1 \). The approximation is essentially an inverse \( N \)-point DFT, with the DFT frequencies \( \omega_i \) spanning equally the interval \([-\pi, \pi]\), instead of the standard DFT interval \([0, 2\pi]\):

\[
\omega_i = \frac{2\pi i}{N}, \quad -M \leq i \leq M \quad (11.4.2)
\]

The forward DFT applied to Eq. (11.4.1) gives:

\[
D(\omega_l) = \sum_{k=-M}^{M} \tilde{d}(k) e^{-j\omega_l k} \quad (11.4.3)
\]

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 \( \tilde{d}(k) \):

\[
h(n) = w(n) \tilde{d}(n-M), \quad n = 0, 1, \ldots, N - 1 \quad (11.4.4)
\]

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, $\delta g = \sqrt{\delta_{pass}\delta_{stop}}$:

$$N - 1 = \frac{D_{fs}}{\Delta f}$$

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 $\omega$. 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 $\omega$.

In both cases, use Euler’s formula to write $D(\omega)$ in 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:

$$D(\omega) = \begin{cases} j\omega, & \text{if } |\omega| \leq \omega_c \\ 0, & \text{if } \omega_c < |\omega| \leq \pi \end{cases}$$

Show that the corresponding ideal impulse responses are given by:

$$d(k) = \begin{cases} \omega_c \cos(\omega_c k) \pi / k & \text{(differentiator)} \\ 1 - \cos(\omega_c k) \pi / k & \text{(hilbert transformer)} \end{cases}$$

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:

$$D(\omega) = \begin{cases} j\omega, & \text{if } \omega_a \leq |\omega| \leq \omega_b \\ 0, & \text{if } 0 \leq |\omega| < \omega_a \text{ or } \omega_b < |\omega| \leq \pi \end{cases}$$

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 $\omega_c$?

By choosing $\omega_c$ 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:

$$E_M = \int_{-\pi}^{\pi} |\hat{D}(\omega) - \hat{D}_M(\omega)|^2 \frac{d\omega}{2\pi} = \int_{-\pi}^{\pi} |D(\omega)|^2 \frac{d\omega}{2\pi} \sum_{k=-M}^{M} d(k)^2$$

Then, show the limit $E_M \to 0$ as $M \to \infty$.

11.7 Using the differentiator and Hilbert transformer filters and the result of Problem 11.6, show the infinite series:

$$\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6}, \quad \sum_{k=1}^{\infty} \frac{1}{k^4} = \frac{\pi^4}{90}, \quad \sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{24}$$

11.8 The ideal Hilbert transformer has frequency response $D(\omega) = -j\text{sign}(\omega)$, for $-\pi \leq \omega \leq \pi$. Show that it acts as a 90° phase shifter converting a cosinusoidal input into a sinusoidal output and vice versa, that is, show the input/output pairs:

$$\cos(\omega n) \frac{D}{\sin(\omega n)}, \quad \sin(\omega n) \frac{D}{\cos(\omega n)}$$

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 $\{z_0^1, z_0^2, z_0^{1+}\}$ 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 $h$ 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 $\omega$?

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 $\delta_{\text{pass}}$, 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 $\Delta f$ 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 $\Delta f$ between passband and stopband that you can demand if the stopband attenuation is to be 80 dB?

b. If the minimum transition width $\Delta f$ 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 $f_{\text{inst}}$. 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:

$$N_{\text{max}} = f_{\text{inst}} / f_s, \quad \Delta f_{\text{min}} = D f_s^2 / f_{\text{inst}}^2,$$

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 $\Delta f$ Hz and sampling rate $f_s$. If the filter is to be implemented on a DSP chip that has instruction rate $f_{\text{inst}}$, show that the maximum attainable stopband attenuation for the filter is given by:

$$A_{\text{max}} = 14.36 f_{\text{inst}} \Delta f + 7.95,$$

where we defined the normalized frequencies $f_{\text{inst}} = f_{\text{inst}} / f_s, \Delta F = \Delta f / f_s$.

11.18 Computer Experiment: Rectangular and Hamming Windows. For the lengths $N = 11, 41, 81, \text{ and } 121$, 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 $0 \leq f \leq f_s/2$. You may find useful the MATLAB routines k1h.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 Hilbert Transformer Design. Using the MATLAB routine kdiff, design a lowpass differentiator for the following values of the cutoff frequency, transition width, and stopband attenuation parameters ($\omega_c, \Delta \omega, A$):

$$\omega_c = 0.40 \pi, 0.80 \pi \quad \text{[rads/sample]}$$

$$\Delta \omega = 0.10 \pi, 0.05 \pi \quad \text{[rads/sample]}$$

$$A = 30, 60 \quad \text{[dB]}$$

For each of the eight cases, plot the magnitude response $|H(\omega)|$ of the designed filter over the interval $0 \leq \omega \leq \pi$.

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:

$$[B, S] = sg(d, N); \quad F = S' \ast S; \quad [B, G] = sfo(-2);$$

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 $0 \leq \omega \leq \pi$. Then replot only over the range $0 \leq \omega \leq 0.1 \pi$ and use vertical scales $[0, 1]$ 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 Hilbert transformer for the following values of the cutoff frequency, transition width, and stopband attenuation parameters ($\omega_c, \Delta \omega, A$):

$$\omega_c = 0.80 \pi, 1.00 \pi \quad \text{[rads/sample]}$$

$$\Delta \omega = 0.10 \pi, 0.05 \pi \quad \text{[rads/sample]}$$

$$A = 30, 60 \quad \text{[dB]}$$

For each of the eight cases, plot the magnitude response $|H(\omega)|$ of the designed filter over the interval $0 \leq \omega \leq \pi$. 
11.6 PROBLEMS

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.

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:

$$s = f(z)$$

(12.1.1)

The corresponding mapping between the physical digital frequency $\omega = 2\pi f / f_s$ and the equivalent analog frequency $^\dagger \Omega$ is obtained by replacing $s = j\Omega$ and $z = e^{j\omega}$ into Eq. (12.1.1), giving $j\Omega = f(e^{j\omega})$. The frequency $\Omega$ 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.

---

\[^\dagger\] Here, $\Omega$ 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.
12.1. BILINEAR TRANSFORMATION

The bilinear transformation is a particular case of Eq. (12.1.1) defined by:

$$s = f(z) = \frac{1 - z^{-1}}{1 + z^{-1}}$$  \hspace{1cm} (bilinear transformation) \hspace{1cm} (12.1.3)

The corresponding mapping of frequencies is obtained as follows:

$$j\Omega = \tan \left( \frac{\omega}{2} \right)$$  \hspace{1cm} (bilinear transformation) \hspace{1cm} (12.1.4)

Because of the nonlinear relationship between the physical frequency $\omega$ and the fictitious analog frequency $\Omega$, 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:

- **(lowpass)**  \hspace{1cm} $s = f(z) = \frac{1 - z^{-1}}{1 + z^{-1}}$
- **(highpass)**  \hspace{1cm} $s = f(z) = \frac{1 + z^{-1}}{1 - z^{-1}}$
- **(bandpass)**  \hspace{1cm} $s = f(z) = \frac{1 - 2cz^{-1} + z^{-2}}{1 - z^{-2}}$
- **(bandstop)**  \hspace{1cm} $s = f(z) = \frac{1 - z^{-2}}{1 - 2c + z^{-2}}$

with corresponding frequency maps:

- **(lowpass)**  \hspace{1cm} $\Omega = g(\omega) = \tan \left( \frac{\omega}{2} \right)$
- **(highpass)**  \hspace{1cm} $\Omega = g(\omega) = -\cot \left( \frac{\omega}{2} \right)$
- **(bandpass)**  \hspace{1cm} $\Omega = g(\omega) = \frac{c - \cos \omega}{\sin \omega}$
- **(bandstop)**  \hspace{1cm} $\Omega = g(\omega) = \frac{\sin \omega}{\cos \omega - c}$

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:

$$H(z) = H_a(f(z))$$  \hspace{1cm} (12.1.7)

The corresponding frequency responses also map in a similar fashion:

$$H(\omega) = H_a(\Omega)$$  \hspace{1cm} (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.

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,
which shows that

\[ \text{Re } s < 0 \iff |z| < 1 \quad \text{and} \quad \text{Re } s = 0 \iff |z| = 1 \]

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, \( 0 \leq f \leq f_c \), and the range of frequencies that are filtered out, that is, \( f_c \leq f \leq f_s/2 \). The digital cutoff frequency \( \omega_c \) in units of radians per sample is defined to be:

\[ \omega_c = \frac{2\pi f_c}{f_s} \]

By convention, \( \omega_c \) 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:

\[ \frac{|H(\omega_c)|^2}{|H(0)|^2} = \frac{1}{2} \Rightarrow -10 \log_{10} \left( \frac{|H(\omega_c)|^2}{|H(0)|^2} \right) = -10 \log_{10} \left( \frac{1}{2} \right) = 3 \text{ dB} \]

Assuming that \( H(z) \) is normalized to unity gain at DC, \( |H(0)| = 1 \), this condition reads equivalently:

\[ |H(\omega_c)|^2 = \frac{1}{2} \quad (12.2.1) \]

More generally, we may define \( \omega_c \) 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:

\[ A_c = -10 \log_{10}(G_c^2) = -20 \log_{10} G_c \quad (12.2.2) \]

which can be inverted to give:

\[ G_c = 10^{-A_c/20} \quad (12.2.3) \]

Thus, in this case, the defining condition for \( \omega_c \) is:

\[ |H(\omega_c)|^2 = G_c^2 = 10^{-A_c/10} \quad (12.2.4) \]

If \( A_c = 3 \text{ dB} \), 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.](image)

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:

\[ \Omega_c = \tan \left( \frac{\omega_c}{2} \right) = \tan \left( \frac{\pi f_c}{f_s} \right) \]

Then, we design a first-order analog filter and adjust its parameters so that its cutoff frequency is \( \Omega_c \). Figure 12.2.2 shows the transformation of the specifications. The analog filter’s transfer function is taken to be:

\[ H_a(s) = \frac{\alpha}{s + \alpha} \quad (12.2.5) \]

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 \( s = j\Omega \):

\[ H_a(\Omega) = \frac{\alpha}{j\Omega + \alpha} \Rightarrow |H_a(\Omega)|^2 = \frac{\alpha^2}{\Omega^2 + \alpha^2} \quad (12.2.6) \]

Because the design method satisfies Eq. (12.1.8) for the frequency and magnitude responses, we can determine the filter parameter \( \alpha \) by requiring the cutoff condition:
12.2. FIRST-ORDER LOWPASS AND HIGHPASS FILTERS

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:

$$H(s) = \frac{\alpha}{\frac{1 + e^{-j\omega}}{1 - e^{-j\omega}}}$$

and similarly for the magnitude response:

$$|H(\omega)|^2 = b^2 + \frac{2(1 + \cos \omega)}{1 - 2a \cos \omega + a^2} = \frac{\alpha^2}{\alpha^2 + \tan^2(\omega/2)}$$

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

$$\omega_c = \frac{2\pi f_c}{f_s} = \frac{2\pi \cdot 1 \text{ kHz}}{10 \text{ kHz}} = 0.2\pi \text{ rads/sample}$$

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, \( \omega_c \) corresponds to attenuation \( G_c^2 = 0.9 \). Using Eq. (12.2.8) we find:

\[ \alpha = \Omega_c \frac{G_c}{\sqrt{1 - G_c^2}} = 0.3249 \frac{0.7096}{\sqrt{1 - 0.9}} = 0.9748 \]

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.

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:

\[ \alpha = \Omega_c \frac{G_c}{\sqrt{1 - G_c^2}} = 1.9626 \frac{0.7096}{\sqrt{1 - 0.9}} = 5.8878 \]

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:

\[ H_a(s) = \frac{s}{s + \alpha} \]

we can transform it into a highpass digital filter by the bilinear transformation:

\[ H(z) = H_a(s) \bigg|_{s \rightarrow \frac{1 - z^{-1}}{1 + z^{-1}}} = \frac{1 - z^{-1}}{1 + z^{-1} + \alpha} \]

which gives:

\[ H(z) = b \frac{1 - z^{-1}}{1 - az^{-1}} \]

(12.2.15)

where its coefficients are computed in terms of \( \alpha \):

\[ a = \frac{1 - \alpha}{1 + \alpha}, \quad b = \frac{1}{1 + \alpha} = \frac{1 + a}{2} \]

(12.2.16)

To determine the parameter \( \alpha \), we require that at a given highpass cutoff frequency \( \omega_c \) the attenuation is \( A_c \) dB. That is, we demand

\[ |H(\omega_c)|^2 = G_c^2 = 10^{-A_c/10} \]

(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:

\[ H_a(\Omega) = \frac{j\Omega}{j\Omega + \alpha} \quad \Rightarrow \quad |H_a(\Omega)|^2 = \frac{\Omega^2}{\Omega^2 + \alpha^2} \]

(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 \( \alpha \):

\[ \alpha = \frac{\sqrt{1 - G_c^2}}{G_c} \Omega_c = \frac{\sqrt{1 - G_c^2}}{G_c} \tan \left( \frac{\omega_c}{2} \right) \]

(12.2.19)
12.2. FIRST-ORDER LOWPASS AND HIGHPASS FILTERS

In summary, given the specifications \{\omega_c, A_c\}, we compute the analog parameter \(\alpha\) 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:

\[
H(\omega) = b \frac{1 - e^{-j\omega}}{1 - ae^{-j\omega}} = \frac{\tan(\omega/2)}{\alpha + j\tan(\omega/2)}
\]

\[
|H(\omega)|^2 = b^2 \frac{2(1 - \cos \omega)}{1 - 2a \cos \omega + a^2} = \frac{\tan^2(\omega/2)}{\alpha^2 + \tan^2(\omega/2)}
\]

We note again that if \(\omega_c\) represents the 3-dB cutoff frequency corresponding to \(G_c = 1/2\), then Eq. (12.2.19) simplifies to:

\[
\alpha = \Omega_c = \tan\left(\frac{\omega_c}{2}\right)
\]

(12.2.20)

This and Eq. (12.2.12) imply that if the lowpass and highpass filters have the same 3-dB cutoff frequency \(\omega_c\), then they will have the same analog filter parameter \(\alpha\), and hence the same z-plane pole parameter \(a\). Thus, in this case, the analog filters will be:

\[
H_{LP}(s) = \frac{\alpha}{s + \alpha}, \quad H_{HP}(s) = \frac{s}{s + \alpha}
\]

(12.2.21)

and the corresponding digital filters:

\[
H_{LP}(z) = \frac{1 - a}{2} \frac{1 + z^{-1}}{1 - az^{-1}}, \quad H_{HP}(z) = \frac{1 + a}{2} \frac{1 - z^{-1}}{1 - az^{-1}}
\]

(12.2.22)

They are complementary filters in the sense that their transfer functions add up to unity:

\[
H_{LP}(s) + H_{HP}(s) = \frac{\alpha}{s + \alpha} + \frac{s}{s + \alpha} = \frac{s + \alpha}{s + \alpha} = 1
\]

and similarly,

\[
|H_{LP}(\omega)|^2 + |H_{HP}(\omega)|^2 = \frac{\alpha^2}{\alpha^2 + \tan^2(\omega/2)} + \frac{\tan^2(\omega/2)}{\alpha^2 + \tan^2(\omega/2)} = 1
\]

(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,

\[
|H_{LP}(\omega)|^2 + |H_{HP}(\omega)|^2 = \frac{\alpha^2}{\alpha^2 + \tan^2(\omega/2)} + \frac{\tan^2(\omega/2)}{\alpha^2 + \tan^2(\omega/2)} = 1
\]

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 prewarp 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:

\[
\alpha = \Omega_c \sqrt{\frac{1 - G_c^2}{G_c}} = 0.1083
\]

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. \(\square\)

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].
12.3. SECOND-ORDER PEAKING AND NOTCHING FILTERS

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 \( \Omega = \Omega_0 \):

\[
H_a(s) = \frac{s^2 + \Omega_0^2}{s^2 + \alpha s + \Omega_0^2}
\]

(12.3.2)

We will see below that the filter parameters \( \{ \alpha, \Omega_0 \} \) can be calculated from the given specifications by:

\[
\Omega_0 = \tan \left( \frac{\omega_0}{2} \right), \quad \alpha = \frac{\sqrt{1 - G_B^2}}{G_B} \left( 1 + \Omega_0^2 \right) \tan \left( \frac{\Delta \omega}{2} \right)
\]

(12.3.3)

Then, using the bilinear transformation \( s = \frac{(1 - z^{-1})}{(1 + z^{-1})} \), the filter \( H_a(s) \) is transformed into the digital filter \( H(z) \) as follows:

\[
H(z) = H_a(s) = \frac{s^2 + \Omega_0^2}{s^2 + \alpha s + \Omega_0^2} = \left( \frac{1 - z^{-1}}{1 + z^{-1}} \right)^2 + \Omega_0^2
\]

\[
\left( 1 - z^{-1} \right)^2 + \Omega_0^2 \left( 1 + z^{-1} \right)^2
\]

\[
\frac{1 - 2 \left( 1 - \Omega_0^2 \right)/\left( 1 + \Omega_0^2 + \alpha \right) z^{-1} + \left( 1 + \Omega_0^2 - \alpha \right) z^{-2}}{1 - 2 \left( 1 - \Omega_0^2 \right)/\left( 1 + \Omega_0^2 + \alpha \right) z^{-1} + \left( 1 + \Omega_0^2 - \alpha \right) z^{-2}}
\]

The coefficients of the digital filter can be simplified considerably by recognizing that \( \alpha \) 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

\[
\beta = \frac{\sqrt{1 - G_B^2}}{G_B} \tan \left( \frac{\Delta \omega}{2} \right)
\]

(12.3.4)

Using some trigonometry, we can write also

\[
\frac{1 - \Omega_0^2}{1 + \Omega_0^2} = \frac{1 - \tan^2 \left( \omega_0/2 \right)}{1 + \tan^2 \left( \omega_0/2 \right)} = \cos \omega_0
\]

Canceling several common factors of \( (1 + \Omega_0^2) \), we can write the transfer function \( H(z) \) in the simplified form:
\( H(z) = \left( \frac{1}{1 + \beta} \right) \frac{1 - 2 \cos \omega_0 z^{-1} + z^{-2}}{1 - \left( \frac{\cos \omega_0}{1 + \beta} \right) z^{-1} + \left( \frac{1 - \beta}{1 + \beta} \right) z^{-2}} \) \hspace{1cm} (12.3.5)

Defining the overall normalization gain by

\[
\begin{align*}
\alpha &= \frac{1}{1 + \beta} \\
\frac{1}{1 + \beta} &= \frac{1}{1 + \frac{1}{2}} = \frac{1}{2} \tan \left( \frac{\Delta \omega}{2} \right)
\end{align*}
\]

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:

\[
H(z) = b \frac{1 - 2 \cos \omega_0 z^{-1} + z^{-2}}{1 - 2b \cos \omega_0 z^{-1} + (2b - 1) z^{-2}} \hspace{1cm} \text{(notch filter)} \hspace{1cm} (12.3.7)
\]

This is the final design. It expresses the filter coefficients in terms of the design specifications \(\omega_0, \Delta \omega, \frac{G_b}{2}\). Note that the numerator has a notch at the desired frequency \(\omega_0\) 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 \(\omega_0\) and makes the analog filter’s notch correspond to the digital filter’s notch. The equation for \(\alpha\) can be derived as follows. Setting \(s = j \Omega\) in Eq. (12.3.2), we obtain the frequency and magnitude responses:

\[
H_a(\Omega) = \frac{-\Omega^2 + \Omega_0^2}{-\Omega^2 + j \alpha \Omega + \Omega_0^2} = |H_a(\Omega)|^2 = \frac{(\Omega^2 - \Omega_0^2)^2}{(\Omega^2 - \Omega_0^2)^2 + \alpha^2 \Omega^2}
\]

It is evident from these expressions that \(H_a(\Omega)\) has a notch at \(\Omega = \pm \Omega_0\). The analog bandwidth frequencies \(\Omega_1\) and \(\Omega_2\) are solutions of the equation \(|H_a(\Omega)|^2 = \frac{G_b}{2}\), that is,

\[
\frac{(\Omega^2 - \Omega_0^2)^2}{(\Omega^2 - \Omega_0^2)^2 + \alpha^2 \Omega^2} = \frac{G_b}{2}
\]

Eliminating the denominator and rearranging terms, we can write it as the quartic equation in \(\Omega\):

\[
\Omega^4 - (2\Omega_0^2 + \frac{G_b}{1 - G_b} \alpha^2)\Omega^2 + \Omega_0^4 = 0 \hspace{1cm} (12.3.9)
\]

It may be thought of as a quadratic equation in the variable \(x = \Omega^2\), that is,

\[
x^2 - (2\Omega_0^2 + \frac{G_b}{1 - G_b} \alpha^2)x + \Omega_0^4 = 0
\]

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:

\[
\Omega_1^2 + \Omega_2^2 = x_1 + x_2 = 2\Omega_0^2 + \frac{G_b}{1 - G_b} \alpha^2
\]

\[
\Omega_1 \Omega_2 = x_1 x_2 = \Omega_0^4 \hspace{1cm} (12.3.10)
\]

From the second equation, we obtain:

\[
\Omega_1 \Omega_2 = \Omega_0^2
\]

which states that \(\Omega_0\) is the geometric mean of the left and right bandwidth frequencies. Using this result in the first of (12.3.10), we obtain:

\[
\Omega_1^2 + \Omega_2^2 = 2\Omega_1 \Omega_2 + \frac{G_b}{1 - G_b} \alpha^2
\]

which allows us to solve for the analog bandwidth:

\[
\Delta \Omega^2 = (\Omega_2 - \Omega_1)^2 = \Omega_1^2 + \Omega_2^2 - 2\Omega_1 \Omega_2 = \frac{G_b}{1 - G_b} \alpha^2
\]

or,

\[
\Delta \Omega = \Omega_2 - \Omega_1 = \frac{G_b}{\sqrt{1 - G_b}} \alpha \hspace{1cm} (A_b \text{-dB width}) \hspace{1cm} (12.3.12)
\]

Solving for \(\alpha\), we have:

\[
\alpha = \frac{\sqrt{1 - G_b}}{G_b} \Delta \Omega \hspace{1cm} (12.3.13)
\]

Note that for the 3-dB case, \(G_b^2 = 1/2\), the parameter \(\alpha\) is equal to the 3-dB bandwidth:

\[
\alpha = \Delta \Omega \hspace{1cm} (3\text{-dB width}) \hspace{1cm} (12.3.14)
\]

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:

\[
\tan \left( \frac{\Delta \omega}{2} \right) = \tan \left( \frac{\omega_2 - \omega_1}{2} \right) = \frac{\tan (\omega_2/2) - \tan (\omega_1/2)}{1 + \tan (\omega_2/2) \tan (\omega_1/2)}
\]

\[
= \frac{\Omega_2 - \Omega_1}{1 + \Omega_1 \Omega_2} = \frac{\Delta \Omega}{1 + \Omega_0^2}
\]

where we used \(\Omega_1 \Omega_2 = \Omega_0^2\). Solving for \(\Delta \Omega\), we have:
\[ \Delta \Omega = (1 + \Omega_0^2) \tan \left( \frac{\Delta \omega}{2} \right) \]  

(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 \( \omega_0 \) or \( \Omega_0 \) tend to zero, the notch filter will behave as a highpass filter. The transfer functions \( H(z) \) and \( H_d(s) \) 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:

\[ H_d(s) = \frac{s^2 + \Omega_0^2}{s^2 + \alpha s + \Omega_0^2} \bigg|_{\theta_n=0} = \frac{s^2}{s^2 + \alpha s + \Omega_0^2} \]

(12.3.16)

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:

\[ H_a(s) = \frac{\alpha s}{s^2 + \alpha s + \Omega_0^2} \]

which has frequency and magnitude responses:

\[ H_a(\Omega) = \frac{j \alpha \Omega}{-\Omega^2 + j \alpha \Omega + \Omega_0^2} \Rightarrow |H_a(\Omega)|^2 = \frac{\alpha^2 \Omega^2}{(\Omega^2 - \Omega_0^2)^2 + \alpha^2 \Omega^2} \]

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 \( \Omega_1 \) and \( \Omega_2 \) 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:

\[ \Omega^4 - (2 \Omega_0^2 + \frac{1 - G_B^2}{G_B^2} \alpha^2) \Omega^2 + \Omega_0^4 = 0 \]

which is similar to Eq. (12.3.9). Its two solutions \( \Omega_1^2 \) and \( \Omega_2^2 \) 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

\[ \Delta \Omega = \Omega_2 - \Omega_1 = \sqrt{\frac{1 - G_B^2}{1 - G_B^2}} \alpha \Rightarrow \alpha = \sqrt{\frac{G_B}{1 - G_B^2}} \Delta \Omega \]

(12.3.17)

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):

\[ \Omega_0 = \tan \left( \frac{\omega_0}{2} \right), \quad \alpha = \frac{G_B}{\sqrt{1 - G_B^2}} \frac{1 + \Omega_0^2}{\tan \left( \frac{\Delta \omega}{2} \right)} \]

(12.3.17)

The digital filter is obtained by the bilinear transformation:

\[ H(z) = H_a(s) \bigg|_{s = \frac{\alpha s}{s^2 + \alpha s + \Omega_0^2}} = \frac{\alpha s}{s^2 + \alpha s + \Omega_0^2} \bigg|_{s = \frac{z}{1 + \beta}} \]

which can be written in the form:

\[ H(z) = \left( \frac{\beta}{1 + \beta} \right) \frac{1 - z^{-2}}{1 - 2 \cos \frac{\omega_0}{1 + \beta} z^{-1} + \frac{1 - \beta}{1 + \beta} z^{-2}} \]

(12.3.18)

where \( \beta \) is similar, but not identical, to that in Eq. (12.3.4):

\[ \beta = \frac{G_B}{\sqrt{1 - G_B^2}} \tan \left( \frac{\Delta \omega}{2} \right) \]

(12.3.19)

Defining the gain \( b \) as in Eq. (12.3.6):

\[ b = \frac{1}{1 + \beta} = \frac{1}{1 + \frac{G_B}{\sqrt{1 - G_B^2}} \tan \left( \frac{\Delta \omega}{2} \right)} \]

(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:
12.3. SECOND-ORDER PEAKING AND NOTCHING FILTERS

\[ H(z) = \frac{(1 - b)}{1 - 2b \cos \omega_0 z^{-1} + (2b - 1) z^{-2}} \]  
(peak filter) \( (12.3.21) \)

Note that the numerator vanishes at \( z = \pm 1 \), that is, at DC and the Nyquist frequency. For the 3-dB widths, we have \( G_B^2 = 1/2 \), and the parameters \( b \) or \( \beta \) are the same as those of the notch filter:

\[ \beta = \tan \left( \frac{\Delta \omega}{2} \right) , \quad b = \frac{1}{1 + \beta} = \frac{1}{1 + \tan(\Delta \omega/2)} \]  
(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:

\[ H_{\text{notch}}(z) + H_{\text{peak}}(z) = 1 \]  
(12.3.23)

Note, finally, that in the limit as \( \omega_0 \) or \( \Omega_0 \) 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:

\[ H_a(s) = \frac{\alpha s}{s^2 + \alpha s + \Omega_0^2} \bigg|_{\Omega_0=0} = \frac{\alpha s}{s^2 + \alpha s} = \frac{\alpha}{s + \alpha} \]

**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 case, determine also the corresponding complementary notch filter.

**Solution:** The digital frequencies in radians per sample are:

\[ \omega_0 = \frac{2\pi f_0}{f_s} = \frac{2\pi \cdot 1.75}{10} = 0.35\pi, \quad \Delta \omega = \frac{2\pi \Delta f}{f_s} = \frac{2\pi \cdot 0.5}{10} = 0.1\pi \]

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 \( A_B = 10 \text{ dB} \), 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:

\[ H(z) = \frac{0.1367(1 - z^{-2})}{1 - 0.7838z^{-1} + 0.7265z^{-2}}, \quad H(z) = \frac{0.0501(1 - z^{-2})}{1 - 0.8624z^{-1} + 0.8997z^{-2}} \]

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:

\[ |H(f)|^2 = \frac{\alpha^2 \tan^2(\pi f / f_s)}{(\tan^2(\pi f / f_s) - \Omega_0^2)^2 + \alpha^2 \tan^2(\pi f / f_s)} \]

The right graph of Fig. 12.3.3 shows the complementary peak and notch filters. The parameters \( \beta \) and \( b \) of the notch filter were already calculated above. Using Eq. (12.3.7), we find its transfer function:

\[ H(f) = \frac{0.9080(1 - z^{-2})}{1 - 0.7739z^{-1} + 0.7265z^{-2}} \]

The zeros of the denominator, \( 1 - 0.7739z^{-1} + 0.7265z^{-2} = 0 \), determine the poles of the transfer function. They are:

\[ p, p^* = 0.3919 \pm j0.7569 = 0.8524 e^{j0.3479\pi} \]

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:

\[ p, p^* = 0.8524 e^{j0.3479\pi} = 0.3870 \pm j0.7597 \]

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.

\[ \square \]
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 $\omega_0$ of the boost or cut, the filter gain $G$ at $\omega_0$, 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:

$$
\begin{align*}
G_0^2 < G_B^2 < G^2 & \quad \text{(boost)} \\
G^2 < G_B^2 < G_0^2 & \quad \text{(cut)}
\end{align*}
$$

(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 notchting 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 $G_0^2 < G_B^2 < G^2$, 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:

$$
G_B^2 = \frac{G_0^2 + G^2}{2}
$$

(12.4.2)

Another good choice is the geometric mean, $G_B^2 = G \cdot G_0$, 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, $G_B^2 = 2G_0^2$, 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 $G_B$ are as follows:

$$
G_B = \begin{cases}
\frac{G_0^2}{2}, & \text{if } G^2 > 2G_0^2 \\
2G_0^2, & \text{if } G^2 < 2G_0^2
\end{cases}
$$

(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:

$$
H(z) = G_0 H_{\text{notch}}(z) + G H_{\text{peak}}(z)
$$

(12.4.4)

At $\omega_0$ 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:

$$
H(z) = \frac{G_0 + G \beta}{1 + \beta} - 2 \left( \frac{G_0 \cos \omega_0}{1 + \beta} \right) z^{-1} + \frac{G_0 - G \beta}{1 + \beta} z^{-2}
$$

(12.4.5)

The parameter $\beta$ is a generalization of Eqs. (12.3.4) and (12.3.19) and is given by:
12.4. PARAMETRIC EQUALIZER FILTERS

$$\beta = \frac{G_B^2 - G_0^2}{G^2 - G_B^2} \tan \left( \frac{\Delta \omega}{2} \right)$$ (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 $\beta$ does not depend on the $G_i$'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 $H(z)$:

$$H_a(s) = G_0 H_{\text{notch}}(s) + G H_{\text{peak}}(s) = \frac{G_0 s^2 + \Omega_0^2 + Gs}{s^2 + \alpha s + \Omega_0^2}$$ (12.4.7)

Then, the bandwidth condition $|H_a(\Omega)|^2 = G_B^2$ can be stated as:

$$|H_a(\Omega)|^2 = \frac{G_B^2 (\Omega^2 - \Omega_0^2)^2 + G^2 \alpha^2 \Omega^2}{(\Omega^2 - \Omega_0^2) + \alpha^2 \Omega^2} = G_B^2$$ (12.4.8)

It can be cast as the quartic equation:

$$\Omega^4 - (2\Omega_0^2 + \frac{G^2 - G_B^2 \alpha^2}{G_B^2 - G_0^2}) \Omega^2 + \Omega_0^4 = 0$$

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 $\alpha$ and the analog bandwidth $\Delta \Omega = \Omega_2 - \Omega_1$:

$$\alpha = \frac{G_B^2 - G_0^2}{G^2 - G_B^2} \Delta \Omega = \frac{G_B^2 - G_0^2}{G^2 - G_B^2} (1 + \Omega_0^2) \tan \left( \frac{\Delta \omega}{2} \right) = (1 + \Omega_0^2) \beta$$

This defines $\beta$. 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 $G_H(z)$.

(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 $10^{0.3} = 1.9953 = 2$. Therefore, in case (a), a boost gain of 9 dB above the reference $G_0$ corresponds to the value:

$$G = 10^{0.3/20} G_0 = 2.8184, \quad G^2 = 7.9433 \quad \text{(instead of 8)}$$

The bandwidth level is defined to be 3 dB below the peak, that is, $A_B = 9 - 3 = 6$ dB, and therefore:

$$G_B = 10^{-3/20} G_0 = 10^{0.3/20} G_0 = 1.9953, \quad G_B^2 = 3.9811$$

With these values of $[G_0, G, G_B, \omega_0, \Delta \omega]$, we calculate the value of $\beta$ from Eq. (12.4.6):

$$\beta = \frac{G_B^2 - G_0^2}{G^2 - G_B^2} \tan \left( \frac{\Delta \omega}{2} \right) = \frac{3.9811 - 1}{7.9433 - 3.9811} \tan \left( \frac{0.1\pi}{2} \right) = 0.1374$$

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:

$$G_B = 10^{0.3/20} G_0 = 10^{0.3/20} G_0 = 1.4125, \quad G_B^2 = 1.9953$$

From Eq. (12.4.6), we calculate $\beta = 0.0648$, and from Eq. (12.4.5) the filter:

$$H_a(z) = \frac{1.1106 - 0.8527z^{-1} + 0.7677z^{-2}}{1 - 0.8527z^{-1} + 0.8783z^{-2}}$$

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:

$$G_B = 10^{0.3/20} G_0 = 10^{-6/20} G_0 = 0.5012, \quad G_B^2 = 0.2512$$
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, and –6 dB, whose intersections with the magnitude responses define the corresponding bandwidths \( \Delta \omega \).

Finally, in case (f), we have a 2-dB cut, giving the values:

\[
A_B \Delta \omega = -3 \text{ dB: } G_B = 0.7943, \quad G_B^2 = \frac{G_B^2}{2} = 0.8155
\]

corresponding to \( A_B = 10 \log_{10}(G_B^2) = -0.886 \) dB. The parameter \( \beta \) 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 \( \omega_1 \) 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 \( \omega_2 \) 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 \( \omega_0 \).

(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 \( \omega_1, \omega_2 \).

**Solution:** Assuming \( G_B = 1 \), 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 \( \omega_0 \) and \( \omega_2 \). 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 \( \omega_1 \):

\[
\tan\left(\frac{\omega_1}{2}\right) = \frac{\Omega_1}{\Omega_2} = 0.9653 \Rightarrow \omega_1 = 0.4887\pi
\]
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 $\omega_0$ and $\omega_1$ and calculate $\omega_2$:

$$\Omega_1 = \tan\left(\frac{\omega_1}{2}\right) = 1, \quad \tan\left(\frac{\omega_2}{2}\right) = \Omega_2 = \frac{\Omega_0}{\Omega_1} = 1.8944 \implies \omega_2 = 0.6908\pi$$

where $\Omega_0$ 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:

$$H_b(z) = \frac{0.9436 + 0.4787z^{-1} + 0.6506z^{-2}}{1 + 0.4787z^{-1} + 0.5492z^{-2}}$$

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.

![Graph showing magnitude responses](image)

**Fig. 12.4.3** Parametric EQ filters.

For case (c), we are given $\omega_1$ and $\omega_2$. 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:

$$\tan(\omega_0/2) = \Omega_0 = \sqrt{\Omega_1 \Omega_2} = 1.4009 \implies \omega_0 = 0.6053\pi$$

Using the calculated $\omega_0$ 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 $\omega_0$ 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 $\omega_1$ and $\omega_2$ are obtained by solving the system of equations:

$$\Omega_1 \Omega_2 = \Omega_0^2 = 1.8944, \quad \Omega_1 - \Omega_2 = \Delta \Omega = (1 + \Omega_0^2)\tan(\Delta \omega/2) = 0.9404$$

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 $\beta$ 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 $\omega_0$, 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**:

$$H_{lp}(z) = \left(\frac{G_0 + G_B}{1 + \beta} - \frac{G_0 - G_B}{1 + \beta}\right) z^{-1}$$

(12.4.9)

where $\beta$ is still given by Eq. (12.4.6), but with $\Delta \omega$ replaced by the filter’s cutoff frequency $\omega_c$ and with $G_B$ replaced by the defining level $G_c$ of the cutoff frequency:

$$\beta = \frac{G_c^2 - G_0^2}{G_c^2 - G_0^2} \tan\left(\frac{\omega_c}{2}\right)$$

(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**:
\[ H_{\text{HP}}(z) = \frac{\left( G_0 + G \beta \right) z^{-1} + \left( G_0 - G \beta \right) z^{-1}}{1 + \frac{1 - \beta}{1 + \beta}} \] (12.4.11)

It can also be obtained from Eq. (12.4.9) by the replacement \( z \rightarrow -z \). The parameter \( \beta \) is obtained from Eq. (12.4.6) by the replacements \( G_B \rightarrow G_c \) and \( \Delta \omega \rightarrow \pi - \omega_c \). The latter is necessary because \( \Delta \omega \) is measured from the center frequency \( \omega_0 = \pi \), whereas \( \omega_c \) is measured from the origin \( \omega = 0 \). Noting that \( \tan\left( (\pi - \omega_c)/2 \right) = \cot(\omega_c/2) \), we have:

\[ \beta = \frac{G_c^2 - G_0^2}{G_c^2 - G_0^2} \cot\left( \frac{\omega_c}{2} \right) \] (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).

\[ |H(\omega)|^2 = G_0^2 + G^2 \Omega^2 \frac{\beta^2}{\Omega^2 + \beta^2} = G_c^2, \quad |H_{\text{HP}}(\Omega)|^2 = \frac{G_0^2 + G^2 \Omega^2}{1 + \beta^2 \Omega^2} = G_c^2 \]

### 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](image-url)

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 \( \Omega_0 \rightarrow 0 \) and for the highpass case, the limit \( \Omega_0 \rightarrow \infty \). 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):

\[ H_{\text{LP}}(s) = \frac{G_0 s + G \beta}{s + \beta}, \quad H_{\text{HP}}(s) = \frac{G_0 + G \beta s}{1 + \beta s} \] (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 \( \beta \), resulting into the design equations (12.4.10) and (12.4.12):

\[ \beta = \frac{G_c^2 - G_0^2}{G_c^2 - G_0^2} \cot\left( \frac{\omega_c}{2} \right) \]

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 \( \Omega_0 \rightarrow 0 \) and for the highpass case, the limit \( \Omega_0 \rightarrow \infty \). 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):

\[ H_{\text{LP}}(s) = \frac{G_0 s + G \beta}{s + \beta}, \quad H_{\text{HP}}(s) = \frac{G_0 + G \beta s}{1 + \beta s} \] (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 \( \beta \), resulting into the design equations (12.4.10) and (12.4.12):
12. IIR DIGITAL FILTER DESIGN

### 12.1 IIR Filter Design

A digital filter is designed by specifying its frequency response at some points, usually even at the Nyquist frequency. The filter is implemented using a discrete-time system, where the input and output are both sequences of numbers. The design process involves specifying the desired frequency response and selecting a filter structure that meets these specifications.

#### 12.1.1 Filter Specifications

- **Center Frequency**: $\omega_0$ in radians per sample
- **Bandwidth**: $D\omega$ in radians per sample
- **Numerator Coefficients**: $b = [b_0, b_1, b_2]$ (for a second-order filter)
- **Denominator Coefficients**: $a = [1, a_1, a_2]$ (for a second-order filter)
- **Nyquist-Frequency Gain**: $G_1$ (at $\omega = \pi$)

#### 12.1.2 Filter Design Equations

**Design Equations**

$$\Omega_0 = \tan \left( \frac{\omega_0}{2} \right)$$

$$\Delta \Omega = \left( 1 + \frac{G_B^2 - G_0^2}{G_B^2 - G_1^2} \right) \frac{G_1^2 - G_B^2}{G_1^2 - G_0^2} \tan \left( \frac{\Delta \omega}{2} \right)$$

$$W^2 = \frac{G_1^2 - G_0^2}{G_1^2 - G_1^2} \Omega_0^2$$

$$A = \sqrt{\frac{C + D}{G_1^2 - G_0^2}}, \quad B = \sqrt{\frac{G_1^2 C + G_D^2 D}{G_1^2 - G_0^2}}$$

**Example Equation**

$$C = (\Delta \Omega)^2 |G_B^2 - G_1^2| - 2W^2 \left( |G_B^2 - G_0^2| \left| G_B^2 - G_0^2 \right| (G_B^2 - G_1^2) \right)$$

$$D = 2W^2 \left( |G_1^2 - G_0^2| - \sqrt{(G_1^2 - G_0^2)(G_1^2 - G_1^2)} \right)$$

**Transfer Function**

$$H(z) = \frac{G_1 + G_0W^2 B}{1 + W^2 A} - 2 \left( \frac{1 - W^2}{1 + W^2 A} \right) z^{-1} + \left( \frac{1 + W^2 - A}{1 + W^2 A} \right) z^{-2}$$

#### 12.1.3 Design Process

1. Specify the frequency response at the Nyquist frequency
2. Use the above equations to calculate filter coefficients
3. Implement the filter using the calculated coefficients

#### 12.1.4 Design Examples

The design examples can be found in [296].

---

### 12.5 Comb Filters

Comb filters are used to implement periodic comb or notch filters. They can be used to design periodic and notched responses with adjustable gains and peak widths. The lowpass and highpass shelving filters of the previous section can be turned into comb filters by replicating the transformation of Eq. (15.11.7), that is, $z \rightarrow z^{-D}$, 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 $H_{LP}(z)$ of Eq. (12.4.9) becomes the comb filter:

$$H(z) = \frac{b - cz^{-D} - c^{-1} a z^{-D}}{1 - a z^{-D}}$$

This transfer function can also be obtained from the analog lowpass shelving filter $H_{LP}(s)$ of Eq. (12.4.13) by the generalized bilinear transformation:

$$s = \frac{1 - z^{-D}}{1 + z^{-D}}, \quad \Omega = \tan \left( \frac{\omega D}{2} \right) = \tan \left( \frac{\pi D}{f_s} \right)$$

The DC peak of the lowpass filter $H_{LP}(z)$ 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 $D$th root-of-unity frequencies $\omega_k = 2\pi k / D$, $k = 0, 1, \ldots, D - 1$. Thus, the full width $\Delta \omega$ of the individual replicas can be obtained by the substitution $2\omega_c / D \rightarrow \Delta \omega$, or $\omega_c \rightarrow D\Delta \omega / 2$.

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 $\beta$ can be expressed as:

$$\beta = \frac{G_B - G_0^2}{G_1^2 - G_0^2} \tan \left( \frac{D \Delta \omega}{4} \right)$$

---

### 12.1.5 Additional Notes

Additional details on this method, including derivations, bandwidth definitions, and several design examples, can be found in [296].
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 notchting 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 $G_B = 1/2$, that is, 3 dB.

The replicating transformation $z \rightarrow z^D$ can also be applied to the highpass shelving filter $H_{HP}(z)$ 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 $\pi/D$, placing them between the peaks of the comb (12.5.1), that is, at the odd-multiple frequencies $\omega_k = (2k + 1)\pi/D$, $k = 0, 1, \ldots, D - 1$. The parameter $\beta$ 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 filter. 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 $\omega_k = 2\pi k/D = 0.2\pi k$, $k = 0, 1, \ldots, 9$. 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, $0 \leq \omega \leq 2\pi$. ☐
12.6. \textit{Higher-Order Filters}

that can be tolerated in the passband and $A_{\text{pass}}$ 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 $A_{\text{pass}}$ so that the passband becomes flatter, (2) increasing $A_{\text{stop}}$ so that the stopband becomes deeper, and (3) moving $f_{\text{stop}}$ closer to $f_{\text{pass}}$ 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 $A(f) = -10 \log_{10} |H(f)|^2$:

$$0 \leq A(f) \leq A_{\text{pass}}, \quad \text{for} \quad 0 \leq f \leq f_{\text{pass}}$$
$$A(f) \geq A_{\text{stop}}, \quad \text{for} \quad f_{\text{stop}} \leq f \leq f_{\text{s}}/2$$

Equivalently, in absolute units, the design specifications are:

$$1 \geq |H(f)|^2 \geq \frac{1}{1 + \epsilon_{\text{pass}}^2}, \quad \text{for} \quad 0 \leq f \leq f_{\text{pass}}$$
$$|H(f)|^2 \leq \frac{1}{1 + \epsilon_{\text{stop}}^2}, \quad \text{for} \quad f_{\text{stop}} \leq f \leq f_{\text{s}}/2$$

where $\{\epsilon_{\text{pass}}, \epsilon_{\text{stop}}\}$ are defined in terms of $\{A_{\text{pass}}, A_{\text{stop}}\}$ as follows:

$$|H(f_{\text{pass}})|^2 = \frac{1}{1 + \epsilon_{\text{pass}}^2} = 10^{-A_{\text{pass}}/10},$$

$$|H(f_{\text{stop}})|^2 = \frac{1}{1 + \epsilon_{\text{stop}}^2} = 10^{-A_{\text{stop}}/10}. $$

The quantities $\{\epsilon_{\text{pass}}, \epsilon_{\text{stop}}\}$ control the depths of the passband and stopband. They can be written in the equivalent forms:

$$\epsilon_{\text{pass}} = \sqrt{10^{A_{\text{pass}}/10} - 1} \quad \Leftrightarrow \quad A_{\text{pass}} = 10 \log_{10} (1 + \epsilon_{\text{pass}}^2)$$
$$\epsilon_{\text{stop}} = \sqrt{10^{A_{\text{stop}}/10} - 1} \quad \Leftrightarrow \quad A_{\text{stop}} = 10 \log_{10} (1 + \epsilon_{\text{stop}}^2) \tag{12.6.4}$$

The specifications of the equivalent analog filter are $\{\Omega_{\text{pass}}, \Omega_{\text{stop}}, A_{\text{pass}}, A_{\text{stop}}\}$, or, $\{\Omega_{\text{pass}}, \Omega_{\text{stop}}, \epsilon_{\text{pass}}, \epsilon_{\text{stop}}\}$, where the analog frequencies are obtained by prewarping the digital frequencies:

$$\Omega_{\text{pass}} = \tan \left( \frac{\omega_{\text{pass}}}{2} \right), \quad \Omega_{\text{stop}} = \tan \left( \frac{\omega_{\text{stop}}}{2} \right) \tag{12.6.5}$$

where

$$\omega_{\text{pass}} = \frac{2\pi f_{\text{pass}}}{f_{\text{s}}}, \quad \omega_{\text{stop}} = \frac{2\pi f_{\text{stop}}}{f_{\text{s}}} \tag{12.6.6}$$

The parameters $\{\epsilon_{\text{pass}}, \epsilon_{\text{stop}}\}$ are useful in the design of both Butterworth and Chebyshev filters. In the next section, we begin with the Butterworth case.

12.7 \textit{Analog Lowpass Butterworth Filters}

Analog lowpass Butterworth filters are characterized by just two parameters: the filter order $N$ and the 3-dB normalization frequency $\Omega_0$. Their magnitude response is simply:

$$|H(\Omega)|^2 = \frac{1}{1 + \left( \frac{\Omega}{\Omega_0} \right)^{2N}} \tag{12.7.1}$$

and the corresponding attenuation in decibels:

$$A(\Omega) = -10 \log_{10} |H(\Omega)|^2 = 10 \log_{10} \left[ 1 + \left( \frac{\Omega}{\Omega_0} \right)^{2N} \right] \tag{12.7.2}$$

Note that, as $N$ increases for fixed $\Omega_0$, 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:

$$A(\Omega_{\text{pass}}) = 10 \log_{10} \left[ 1 + \left( \frac{\Omega_{\text{pass}}}{\Omega_0} \right)^{2N} \right] = A_{\text{pass}} = 10 \log_{10} (1 + \epsilon_{\text{pass}}^2)$$

$$A(\Omega_{\text{stop}}) = 10 \log_{10} \left[ 1 + \left( \frac{\Omega_{\text{stop}}}{\Omega_0} \right)^{2N} \right] = A_{\text{stop}} = 10 \log_{10} (1 + \epsilon_{\text{stop}}^2) \tag{12.6.3}$$

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 $\Omega_0$, we rewrite them in the form:
In this case, Eq. (12.7.1) can be written in terms of the stopband parameters:

\[
|H(\Omega)|^2 = \frac{1}{1 + \varepsilon_{\text{stop}}^2 \left( \frac{\Omega}{\Omega_{\text{stop}}} \right)^{2N} + \varepsilon_{\text{stop}}^2}
\]  

(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 \( s = \jmath \Omega \) and noting that \( (H(\Omega))^* = H^*(\Omega) \), we may write Eq. (12.7.1) in terms of the variable \( s \):

\[
H(s)H^*(-s) = \frac{1}{1 + \left( \frac{s}{\jmath \Omega_0} \right)^{2N}}
\]  

(12.7.10)

Setting \( H(s) = \frac{1}{D(s)} \), we have

\[
D(s)D^*(-s) = 1 + (-1)^N \left( \frac{s}{\jmath \Omega_0} \right)^{2N}
\]  

(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

\[
1 + (-1)^N \left( \frac{s}{\jmath \Omega_0} \right)^{2N} = 0 \Rightarrow s^{2N} = (-1)^N 1 \Omega_0^{2N}
\]

are given by

\[
s_i = \Omega_0 e^{\jmath \theta_i}, \quad \theta_i = \frac{\pi}{2N} (N - 1 + 2i), \quad i = 1, 2, \ldots, N, 2N
\]  

(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 \( i = 1, 2, \ldots, N \). Because \( |s_i| = \Omega_0 \), all of the zeros lie on a circle of radius \( \Omega_0 \), 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, \( s_1, s_2^* \), \( i = 1, 2, \ldots, K \). In this case, \( D(s) \) will factor into second-order sections as follows:

\[\text{The notation } H^*(-s) \text{ denotes complex conjugation of the filter coefficients and replacement of } s \text{ by } -s, \text{ for example, } H^*(-s) = \sum a_i (-s)^n \text{ if } H(s) = \sum a_i s^n.\]
12.7. ANALOG LOWPASS BUTTERWORTH FILTERS

\[ H_0(s) = \begin{cases} 1, & \text{if } N = 2K \\ 1 + \frac{s}{\Omega_0}, & \text{if } N = 2K + 1 \end{cases} \quad (12.7.16) \]

**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 \( \Omega_0 \), \( s \) must be replaced by \( s/\Omega_0 \) in each table entry.

The coefficients of \( s \) of the second-order sections are the cosine factors, \(-2\cos \theta_i\), of Eq. (12.7.11). For example, in the case \( N = 7 \), we have \( K = 3 \) and the three \( \theta \)'s are calculated from Eq. (12.7.14):

\[ \theta_i = \frac{i\pi}{14} (6 + 2i) = \frac{8\pi}{14}, \quad \frac{10\pi}{14}, \quad \frac{12\pi}{14}, \quad \text{for } i = 1, 2, 3 \]

\[ -2\cos \theta_i = 0.4450, 1.2470, 1.8019 \]

The corresponding Butterworth filters \( H(s) \) of orders 1–7 are obtained as the inverses of the table entries. \( \square \)

<table>
<thead>
<tr>
<th>( N )</th>
<th>( K )</th>
<th>( \theta_1, \theta_2, \ldots, \theta_K )</th>
<th>( D(s) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td>(1 + s)</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>( \frac{3\pi}{4} )</td>
<td>(1 + 1.4142s + s^2)</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>( \frac{4\pi}{6} )</td>
<td>(1 + s(1 + s + s^2))</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>( \frac{5\pi}{8} ), ( \frac{7\pi}{8} )</td>
<td>(1 + 0.7654s + s^2)(1 + 1.8478s + s^2)</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>( \frac{6\pi}{10} ), ( \frac{8\pi}{10} )</td>
<td>(1 + s)(1 + 0.6180s + s^2)(1 + 1.6180s + s^2)</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>( \frac{7\pi}{12} ), ( \frac{9\pi}{12} ), ( \frac{11\pi}{12} )</td>
<td>(1 + 0.5176s + s^2)(1 + 1.4142s + s^2)(1 + 1.9319s + s^2)</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>( \frac{8\pi}{14} ), ( \frac{10\pi}{14} ), ( \frac{12\pi}{14} )</td>
<td>(1 + s)(1 + 0.4450s + s^2)(1 + 1.2470s + s^2)(1 + 1.8019s + s^2)</td>
</tr>
</tbody>
</table>

Table 12.7.1 Butterworth polynomials.

**Example 12.7.2:** Determine the \( 2^N \) possible \( N \)th 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), \( D(s)D^*(-s) = 1 + (-1)^2s^2 \). They are:

\[ D(s) = \frac{1 + s^2}{2} \]

\[ D^*(-s) = \frac{1 + s^2}{2} \]

\[ D(s)D^*(-s) = \frac{(1 + s^2)(1 + s^2)}{4} = 1 + s^2 \]

\( \square \)
Chapter 12. IIR Digital Filter Design

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:

\[ H_i(z) = \left| \begin{array}{c} 1 \\ 1 - 2 \frac{\Omega_0}{\Omega_0 + 1} \cos \theta_i + \Omega_0^2 \end{array} \right| \frac{G_i(1 + z^{-1})^2}{1 + a_{i1}z^{-1} + a_{i2}z^{-2}} \]  \hspace{1cm} (12.8.1)

where the filter coefficients \( G_i, a_{i1}, a_{i2} \) are easily found to be:

\[ G_i = \frac{\Omega_0^2}{1 - 2\Omega_0 \cos \theta_i + \Omega_0^2} \]

\[ a_{i1} = \frac{2 (\Omega_0^2 - 1)}{1 - 2\Omega_0 \cos \theta_i + \Omega_0^2} \]

\[ a_{i2} = \frac{1 + 2\Omega_0 \cos \theta_i + \Omega_0^2}{1 - 2\Omega_0 \cos \theta_i + \Omega_0^2} \]

for \( i = 1, 2, \ldots, K \). If \( N \) is odd, then there is also a first-order section:

\[ H_0(z) = \frac{1}{1 + \frac{\Omega_0}{\Omega_0 + 1} \cos \theta_0 + \Omega_0^2} = \frac{G_0(1 + z^{-1})}{1 + a_{01}z^{-1}} \]  \hspace{1cm} (12.8.3)

where

\[ G_0 = \frac{\Omega_0}{\Omega_0 + 1}, \quad a_{01} = \frac{\Omega_0 - 1}{\Omega_0 + 1} \]  \hspace{1cm} (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:

\[ H(z) = H_0(z)H_1(z)H_2(z) \cdots H_K(z) \]  \hspace{1cm} (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 \( \Omega_0 \) by

\[ \Omega_0 = \tan \left( \frac{\omega_0}{2} \right) = \tan \left( \frac{\pi f_0}{f_s} \right) \Rightarrow f_0 = \frac{f_s}{\pi} \arctan(\Omega_0) \]  \hspace{1cm} (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:

\[ |H(\omega)|^2 \leq \frac{1}{1 + (\Omega/\Omega_0)^{2N}} = \frac{1}{1 + (\tan(\omega/2)/\Omega_0)^{2N}} \]  \hspace{1cm} (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):

\[ \frac{4G_i}{1 + a_{i1} + a_{i2}} = 1 \quad \text{and} \quad \frac{2G_0}{1 + a_{01}} = 1 \]

In summary, the design steps for a lowpass digital filter with given specifications \( \{f_{\text{pass}}, f_{\text{stop}}\}, \{\omega_{\text{pass}}, \omega_{\text{stop}}\} \) and the corresponding prewarping 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 \( \Omega_0 \) 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 \( \{ \epsilon_{\text{pass}}, \epsilon_{\text{stop}} \} \):

\[
\epsilon_{\text{pass}} = \sqrt{10^{0.5/10} - 1} = \sqrt{10^{0.05/10} - 1} = 0.3493
\]

\[
\epsilon_{\text{stop}} = \sqrt{10^{10/10} - 1} = \sqrt{10^{2/10} - 1} = 3
\]

Then, Eq. (12.7.4) gives:

\[
N_{\text{exact}} = \frac{\ln(\epsilon_{\text{pass}})}{\ln(\epsilon_{\text{stop}}/\epsilon_{\text{pass}})} = \frac{\ln(3/0.3493)}{\ln(1/0.7265)} = 6.73 \Rightarrow N = 7
\]

Thus, there is one first-order section \( H_0(z) \) and three second-order sections. Eq. (12.7.7) gives for \( \Omega_0 \) and its value in Hz:

\[
\Omega_0 = \frac{\Omega_{\text{pass}}}{\epsilon_{\text{pass}}} = \frac{0.7265}{0.3493} = 2.08443
\]

\[
f_0 = \frac{f_s}{\pi} \arctan(\Omega_0) = \frac{20}{\pi} \arctan(0.8443) = 4.4640 \text{ kHz}
\]

The Butterworth angles \( \theta_1, \theta_2, \theta_3 \) were calculated in Example 12.7.1. The SOS coefficients are calculated from Eqs. (12.8.4) and (12.8.2):

<table>
<thead>
<tr>
<th>( i )</th>
<th>( G_i )</th>
<th>( a_{0,i} )</th>
<th>( a_{1,i} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.4578</td>
<td>0.0844</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.3413</td>
<td>-0.2749</td>
<td>0.6402</td>
</tr>
<tr>
<td>2</td>
<td>0.2578</td>
<td>-0.2076</td>
<td>0.2386</td>
</tr>
<tr>
<td>3</td>
<td>0.2204</td>
<td>-0.1775</td>
<td>0.0592</td>
</tr>
</tbody>
</table>

resulting in the transfer function:

\[
H(z) = H_0(z) H_1(z) H_2(z) H_3(z)
\]

\[
= \frac{0.4578(1 + z^{-1})}{1 - 0.0844z^{-1}} \cdot \frac{1 - 0.2749z^{-1} + 0.6402z^{-2}}{1 - 0.2578(1 + z^{-1})^2} \cdot \frac{1 - 0.2749z^{-1} + 0.6402z^{-2}}{1 - 0.2204(1 + z^{-1})^2}
\]

It can be implemented in the time domain by the routines `lhbutt.m` or `filter.m`. 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 \( \omega \) expressed in kHz, \( \omega = 2\pi f/f_s \):

\[
|H(f)|^2 \approx \frac{1}{1 + (\tan(\pi f/f_s)/\omega_0)^{2N}} = \frac{1}{1 + (\tan(\pi f/20)/0.8443)^{10}}
\]

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, \( A(f_{\text{stop}}) = 10.68 \text{ dB} \).

**Fig. 12.8.1** Digital lowpass Butterworth filters.

The second filter has more stringent specifications. The desired passband attenuation is \( A_{\text{pass}} = -10 \log_{10}(0.98) = 0.9877 \text{ dB} \), and the stopband attenuation \( A_{\text{stop}} = -10 \log_{10}(0.02) = 16.9807 \text{ dB} \). With these values, we find the design parameters \( \{ \epsilon_{\text{pass}}, \epsilon_{\text{stop}} \} = \{ 0.1429, 7 \} \) 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 `1butter.m` of Appendix C:
12.9 DIGITAL HIGHPASS FILTERS

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 \( N \).

\[ s = \frac{1 + z^{-1}}{1 - z^{-1}}, \quad \Omega = -\cot \left( \frac{\omega}{2} \right), \quad \omega = \frac{2\pi f}{f_s} \]  \hspace{1cm} (12.9.1)

\[ \Omega_{\text{pass}} = \cot \left( \frac{\Omega_{\text{pass}}}{2} \right) = \cot \left( \frac{\pi f_{\text{pass}}}{f_s} \right) \]  \hspace{1cm} (12.9.2)

According to Eq. (12.9.1), we should have used \( \Omega_{\text{pass}} = -\cot(\omega_{\text{pass}}/2) \). However, as far as the determination of the parameters \( N \) and \( \Omega_0 \) 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 \( \Omega \).

Using Eqs. (12.7.4)–(12.7.7), we determine the parameters \( a_i \) and \( \Omega_0 \) 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:

\[ H_i(z) = \frac{1}{1 - 2 \frac{s}{\Omega_0} \cos \theta_i + \frac{s^2}{\Omega_0^2} \bigg| \frac{s - 1 - z^{-1}}{s + 1 + z^{-1}} \bigg.} = G_i(1 - z^{-1})^2 \]  \hspace{1cm} (12.9.3)

where the filter coefficients \( G_i, a_{i1}, a_{i2} \) are easily found to be

\[ G_i = \frac{\Omega_0^2}{1 - 2 \Omega_0 \cos \theta_i + \Omega_0^2} \]  \hspace{1cm} (12.9.4)

\[ a_{i1} = -\frac{2(\Omega_0^2 - 1)}{1 - 2 \Omega_0 \cos \theta_i + \Omega_0^2} \]  \hspace{1cm} (12.9.4)

\[ a_{i2} = \frac{1 + 2 \Omega_0 \cos \theta_i + \Omega_0^2}{1 - 2 \Omega_0 \cos \theta_i + \Omega_0^2} \]  \hspace{1cm} (12.9.4)

for \( i = 1, 2, \ldots, K \). If \( N \) is odd, then there is also a first-order section given by

\[ H_0(z) = \frac{1}{1 + \frac{s}{\Omega_0} \bigg| \frac{s - 1 - z^{-1}}{s + 1 + z^{-1}} \bigg.} = G_0(1 - z^{-1}) \]  \hspace{1cm} (12.9.5)

where

\[ G_0 = \frac{\Omega_0}{\Omega_0 + 1}, \quad a_{i0} = -\frac{\Omega_0 - 1}{\Omega_0 + 1} \]  \hspace{1cm} (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

\[ H(z) = H_0(z)H_1(z)H_2(z) \cdots H_K(z) \]  \hspace{1cm} (12.9.7)
12.9 DIGITAL HIGHPASS FILTERS

The 3-dB frequency \( f_0 \) of the designed filter may be calculated from:

\[
\Omega_0 = \cot \left( \frac{\omega_0}{2} \right) = \cot \left( \frac{\pi f_0}{f_s} \right) \quad \Rightarrow \quad f_0 = \frac{f_s}{\pi} \arctan \left( \frac{1}{\Omega_0} \right)
\]

and the magnitude response from:

<table>
<thead>
<tr>
<th>( H(\omega) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \frac{1}{1 + (\cot(\omega/2)/\Omega_0)^2} )</td>
</tr>
</tbody>
</table>

Note the similarities and differences between the highpass and lowpass cases: The coefficients \( G_1 \) and \( a_{12} \) are the same, but \( a_{11} \) 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 \( z \rightarrow -z \).

**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 \( 1 - \frac{|H(f)|}{|H|} \geq 0.98 \) in the passband, and \( |H(f)|^2 \leq 0.02 \) in the stopband.

**Solution:** The digital frequencies and their prewarped versions are:

\[ \omega_{\text{pass}} = \frac{2 \pi f_{\text{pass}}}{f_s} = \frac{2 \pi \cdot 5}{20} = 0.5 \pi, \quad \Omega_{\text{pass}} = \cot \left( \frac{\omega_{\text{pass}}}{2} \right) = 1 \]
\[ \omega_{\text{stop}} = \frac{2 \pi f_{\text{stop}}}{f_s} = \frac{2 \pi \cdot 4}{20} = 0.4 \pi, \quad \Omega_{\text{stop}} = \cot \left( \frac{\omega_{\text{stop}}}{2} \right) = 1.3764 \]

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:

\[ N_{\text{exact}} = \frac{\ln(\Omega_{\text{stop}}/\varepsilon_{\text{pass}})}{\ln(\Omega_{\text{stop}}/\Omega_{\text{pass}})} = \frac{\ln(1/0.3493)}{\ln(1/1.3764)} = 6.73 \quad \Rightarrow \quad N = 7 \]

Thus, there is one first-order section \( H_0(z) \) and three second-order sections. Eq. \((12.7.7)\) gives for \( \Omega_0 \):

\[ \Omega_0 = \frac{\Omega_{\text{pass}}}{(10^{\varepsilon_{\text{pass}}/10} - 1)^{1/2N}} = \frac{\Omega_{\text{pass}}}{\varepsilon_{\text{pass}}^{1/2N}} = \frac{1}{(0.3493)^{1/7}} = 1.1621 \]

The SOS coefficients are calculated from Eqs. \((12.9.4)\) and \((12.9.6)\):

\[
\begin{array}{ccc}
  i & G_i & a_{11} & a_{12} \\
  0 & 0.5375 & -0.0750 & \\
  1 & 0.4709 & -0.2445 & 0.6393 \\
  2 & 0.3554 & -0.1845 & 0.2372 \\
  3 & 0.3039 & -0.1577 & 0.0577 \\
\end{array}
\]

resulting in the transfer function:

\[
H(z) = H_0(z) H_1(z) H_2(z) H_3(z) \\
= \frac{0.5375(1 - z^{-1})}{1 - 0.0750z^{-1}} \cdot \frac{0.4709(1 - z^{-1})^2}{1 - 0.2445z^{-1} + 0.6393z^{-2}} \cdot \frac{0.3554(1 - z^{-1})^2}{1 - 0.1845z^{-1} + 0.2372z^{-2}} \cdot \frac{0.3039(1 - z^{-1})^2}{1 - 0.1577z^{-1} + 0.0577z^{-2}}
\]

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:

\[
\begin{array}{ccc}
  i & G_i & a_{11} & a_{12} \\
  0 & 0.5374 & -0.0747 & \\
  1 & 0.5131 & -0.2655 & 0.7870 \\
  2 & 0.4252 & -0.2200 & 0.4807 \\
  3 & 0.3677 & -0.1903 & 0.2806 \\
  4 & 0.3300 & -0.1708 & 0.1493 \\
  5 & 0.3062 & -0.1584 & 0.0663 \\
  6 & 0.2930 & -0.1516 & 0.0203 \\
\end{array}
\]

The magnitude responses of the two designs are shown in Fig. 12.9.2.

**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 band-pass specifications and their analog equivalents.

\[
\begin{align*}
\Omega_{\text{pass}} &= \frac{c - \cos \omega_{\text{pa}}}{\sin \omega_{\text{pa}}} \\
\Omega_{\text{pass}} &= \frac{c - \cos \omega_{\text{pb}}}{\sin \omega_{\text{pb}}}
\end{align*}
\]  

where \( \omega_{\text{pa}} = 2\pi f_{\text{pa}}/f_s \) and \( \omega_{\text{pb}} = 2\pi f_{\text{pb}}/f_s \). By adding them, we solve for \( c \). Then, inserting the computed value of \( c \) into one or the other we find \( \Omega_{\text{pass}} \). The resulting solution is:

\[
c = \frac{\sin(\omega_{\text{pa}} + \omega_{\text{pb}})}{\sin \omega_{\text{pa}} + \sin \omega_{\text{pb}}}, \quad \Omega_{\text{pass}} = \left| \frac{c - \cos \omega_{\text{pb}}}{\sin \omega_{\text{pb}}} \right| \tag{12.10.2}
\]

Note that for \( \omega_{\text{pa}}, \omega_{\text{pb}} \) in the interval \([0, \pi]\), the above expression for \( c \) implies \(|c| \leq 1\), as required for stability. Next, we compute the two numbers:

\[
\Omega_{\text{sa}} = \frac{c - \cos \omega_{\text{sa}}}{\sin \omega_{\text{sa}}}, \quad \Omega_{\text{sb}} = \frac{c - \cos \omega_{\text{sb}}}{\sin \omega_{\text{sb}}}
\]

where \( \omega_{\text{sa}} = 2\pi f_{\text{sa}}/f_s \) and \( \omega_{\text{sb}} = 2\pi f_{\text{sb}}/f_s \).

Ideally, the stopband of the digital filter should map exactly onto the stopband of the analog filter so that we should have \( \Omega_{\text{sa}} = -\Omega_{\text{stop}} \) and \( \Omega_{\text{sb}} = -\Omega_{\text{stop}} \). 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 \( \Omega \), it is enough to choose the smallest of the two stopbands defined above. Thus, we define:

\[
\Omega_{\text{stop}} = \min(\Omega_{\text{sa}}, \Omega_{\text{sb}}) \tag{12.10.3}
\]

With the computed values of \( \Omega_{\text{pass}} \) and \( \Omega_{\text{stop}} \), we proceed to compute the Butterworth parameters \( N \) and \( \Omega_0 \) and the corresponding SOs of Eq. (12.7.15). Because \( s \) is quadratic in \( z \), the substitution of \( s \) into these SOs will give rise to fourth-order sections in \( z \):

\[
H_i(z) = \frac{1}{1 - 2\frac{s}{\Omega_0} \cos \theta_i + \left(\frac{s}{\Omega_0}\right)^2} |_{s = 1 - 2z^{-1}z^{-2}} \tag{12.10.4}
\]

where, for \( i = 1, 2, \ldots, K \):

\[
G_i = \frac{\Omega_0^2}{1 - 2\Omega_0 \cos \theta_i + \Omega_0^2}
\]

\[
a_{11} = \frac{4c(\Omega_0 \cos \theta_1 - 1)}{1 - 2\Omega_0 \cos \theta_1 + \Omega_0^2} \\
a_{12} = \frac{2(2c^2 + 1 - \Omega_0^2)}{1 - 2\Omega_0 \cos \theta_1 + \Omega_0^2} \\
a_{13} = \frac{-4c(\Omega_0 \cos \theta_1 + 1)}{1 - 2\Omega_0 \cos \theta_1 + \Omega_0^2} \\
a_{14} = \frac{1 + 2\Omega_0 \cos \theta_1 + \Omega_0^2}{1 - 2\Omega_0 \cos \theta_1 + \Omega_0^2}
\]  

\[
a_{21} = \frac{-4c(\Omega_0 \cos \theta_1 - 1)}{1 - 2\Omega_0 \cos \theta_1 + \Omega_0^2} \\
a_{22} = \frac{2(2c^2 + 1 - \Omega_0^2)}{1 - 2\Omega_0 \cos \theta_1 + \Omega_0^2} \\
a_{23} = \frac{4c(\Omega_0 \cos \theta_1 + 1)}{1 - 2\Omega_0 \cos \theta_1 + \Omega_0^2} \\
a_{24} = \frac{1 - 2\Omega_0 \cos \theta_1 + \Omega_0^2}{1 - 2\Omega_0 \cos \theta_1 + \Omega_0^2}
\]  

\[
a_{31} = \frac{4c(\Omega_0 \cos \theta_1 - 1)}{1 - 2\Omega_0 \cos \theta_1 + \Omega_0^2} \\
a_{32} = \frac{-2(2c^2 + 1 - \Omega_0^2)}{1 - 2\Omega_0 \cos \theta_1 + \Omega_0^2} \\
a_{33} = \frac{-4c(\Omega_0 \cos \theta_1 + 1)}{1 - 2\Omega_0 \cos \theta_1 + \Omega_0^2} \\
a_{34} = \frac{1 + 2\Omega_0 \cos \theta_1 + \Omega_0^2}{1 - 2\Omega_0 \cos \theta_1 + \Omega_0^2}
\]  

\[
a_{41} = \frac{-4c(\Omega_0 \cos \theta_1 - 1)}{1 - 2\Omega_0 \cos \theta_1 + \Omega_0^2} \\
a_{42} = \frac{-2(2c^2 + 1 - \Omega_0^2)}{1 - 2\Omega_0 \cos \theta_1 + \Omega_0^2} \\
a_{43} = \frac{4c(\Omega_0 \cos \theta_1 + 1)}{1 - 2\Omega_0 \cos \theta_1 + \Omega_0^2} \\
a_{44} = \frac{1 - 2\Omega_0 \cos \theta_1 + \Omega_0^2}{1 - 2\Omega_0 \cos \theta_1 + \Omega_0^2}
\]
If \( N \) is odd, then there is also a first-order section in \( s \) which becomes a second-order section in \( z \):

\[
H_0(z) = \left( \frac{1}{1 + \frac{\Omega_0}{\Omega_0}} \right) G_0(1 - z^{-2}) \frac{1}{1 + a_{01}z^{-1} + a_{02}z^{-2}}
\]

where

\[
G_0 = \frac{\Omega_0}{1 + \Omega_0}, \quad a_{01} = -\frac{2c}{1 + \Omega_0}, \quad a_{02} = \frac{1 - \Omega_0}{1 + \Omega_0}
\]

(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:

\[
c = \frac{\cos \omega_0}{\sin \omega_0} = \pm \Omega_0
\]

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:

\[
\tan \left( \frac{\Omega_0 a_0}{2} \right) = \tan \left( \frac{\pi f_{pa}}{f_s} \right) = \sqrt{\Omega_0^2 + 1 - c^2} - \Omega_0
\]

\[
\tan \left( \frac{\Omega_0 a_1}{2} \right) = \tan \left( \frac{\pi f_{pb}}{f_s} \right) = \sqrt{\Omega_0^2 + 1 - c^2} + \Omega_0
\]

(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 \( 1 - |H(f)|^2 \geq 0.02 \) in the passband, and \( |H(f)|^2 \leq 0.02 \) 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 \( \Omega_{pass} \):

\[
c = \frac{\sin(\omega_{pa} + \omega_{pb})}{\sin \omega_{pa} + \sin \omega_{pb}} = 0.6180, \quad \Omega_{pass} = \left| \frac{c - \cos \omega_{pb}}{\sin \omega_{pb}} \right| = 0.3249
\]

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 \( \Omega_{stop} = \min(\Omega_{sa}, \Omega_{sb}) = 0.4674 \). The analog filter with the specifications \{\( \Omega_{pass}, \Omega_{stop}, A_{pass}, A_{stop} \)\} has parameters \{\( \varepsilon_{pass}, \varepsilon_{stop} \)\} = {0.3493, 3} and:

\[
N_{exact} = 5.92, \quad N = 6, \quad \Omega_0 = 0.3872
\]

The left-right 3-dB frequencies are calculated from Eq. (12.10.8) to be: \( f_{sb} = 1.8689 \text{ kHz} \), \( f_{pa} = 4.2206 \text{ kHz} \). The coefficients of the three fourth-order sections of the digital filter are (computed by the MATLAB function bpsbut.m):

\[
\begin{array}{cccccc}
 i & G_i & a_0 & a_2 & a_4 & a_6 \\
 1 & 0.1110 & -2.0142 & 2.3906 & -1.6473 & 0.7032 \\
 2 & 0.0883 & -1.8551 & 1.9017 & -1.0577 & 0.3549 \\
 3 & 0.0790 & -1.7897 & 1.7009 & -0.8154 & 0.2118
\end{array}
\]

The magnitude response can be calculated from:

\[
|H(\omega)|^2 = \frac{1}{1 + \left( \frac{\Omega}{\Omega_{pass}} \right)^2} = \frac{1}{1 + \left( \frac{c - \cos \omega}{\sin \omega_{pass}} \right)^2}
\]

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, \( \Omega_{stop} = |\Omega_{sa}| \), 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 $A_{\text{pass}} = -10 \log_{10}(0.98) = 0.0877 \text{ dB}$, and $A_{\text{stop}} = -10 \log_{10}(0.02) = 16.9897 \text{ dB}$ and $(f_{\text{pass}}, f_{\text{stop}}) = [0.1429, 7]$. The design has the same $c, \Omega_{\text{pass}},$ and $\Omega_{\text{stop}}$, 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 $f_{\text{sa}} = 1.8677 \text{ kHz}, f_{\text{sb}} = 4.2228 \text{ kHz}$. The digital filter coefficients of the second- and fourth-order sections are:

<table>
<thead>
<tr>
<th>$i$</th>
<th>$G_i$</th>
<th>$a_{i1}$</th>
<th>$a_{i2}$</th>
<th>$a_{i3}$</th>
<th>$a_{i4}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.2794</td>
<td>-0.8907</td>
<td>0.4411</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.1193</td>
<td>-2.0690</td>
<td>2.5596</td>
<td>-1.8526</td>
<td>0.8249</td>
</tr>
<tr>
<td>2</td>
<td>0.1021</td>
<td>-1.9492</td>
<td>2.1915</td>
<td>-1.4083</td>
<td>0.5624</td>
</tr>
<tr>
<td>3</td>
<td>0.0907</td>
<td>-1.8694</td>
<td>1.9460</td>
<td>-1.1122</td>
<td>0.3874</td>
</tr>
<tr>
<td>4</td>
<td>0.0834</td>
<td>-1.8186</td>
<td>1.7900</td>
<td>-0.9239</td>
<td>0.2762</td>
</tr>
<tr>
<td>5</td>
<td>0.0794</td>
<td>-1.7904</td>
<td>1.7033</td>
<td>-0.8193</td>
<td>0.2144</td>
</tr>
</tbody>
</table>

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 $\cos \theta_i = c$. In Hz, we have $f_c = f_s \arccos(c)/(2\pi) = 2.8793 \text{ kHz}$. 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, $0 \leq f \leq f_{\text{pa}}$ and $f_{pb} \leq f \leq f_{pb}/2$, and one stopband $f_{sa} \leq f \leq f_{sb}$.

The bandstop version of the bilinear transformation and the corresponding frequency mapping were given in Eqs. (12.1.5) and (12.1.6):

$$s = \frac{1 - z^{-2}}{1 - 2cz^{-1} + z^{-2}}, \quad \Omega = \frac{\sin \omega}{\cos \Omega - c}, \quad \omega = \frac{2\pi f}{f_s} (12.11.1)$$

The design steps are summarized as follows. First, we compute the digital frequencies in radians per sample:

$$\omega_{\text{pa}} = \frac{2\pi f_{\text{pa}}}{f_s}, \quad \omega_{\text{pb}} = \frac{2\pi f_{\text{pb}}}{f_s}, \quad \omega_{\text{sa}} = \frac{2\pi f_{\text{sa}}}{f_s}, \quad \omega_{\text{sb}} = \frac{2\pi f_{\text{sb}}}{f_s}$$

Then, we calculate $c$ and $\Omega_{\text{pass}}$ by requiring:

$$-\Omega_{\text{pass}} = \frac{\sin \omega_{\text{pa}}}{\cos \omega_{\text{pa}} - c}, \quad \Omega_{\text{pass}} = \frac{\sin \omega_{\text{pb}}}{\cos \omega_{\text{pb}} - c}$$

which may be solved as follows:

$$c = \frac{\sin(\omega_{\text{pa}} + \omega_{\text{pb}})}{\sin \omega_{\text{pa}} + \sin \omega_{\text{pb}}}, \quad \Omega_{\text{pass}} = \frac{1}{\left| \frac{\sin \omega_{\text{pb}}}{\cos \omega_{\text{pb}} - c} \right|}$$

Next, we compute the two possible stopbands:

$$\Omega_{\text{sa}} = \frac{\sin \omega_{\text{sa}}}{\cos \omega_{\text{sa}} - c}, \quad \Omega_{\text{sb}} = \frac{\sin \omega_{\text{sb}}}{\cos \omega_{\text{sb}} - c}$$

and define:

$$\Omega_{\text{stop}} = \min(|\Omega_{\text{sa}}|, |\Omega_{\text{sb}}|)$$

Then, use the analog specifications $\{\Omega, \Omega_{\text{pass}}, A_{\text{pass}}, A_{\text{stop}}\}$ to compute the Butterworth parameters $\{N, \Omega_0\}$. And finally, transform the analog filter sections into fourth-order sections by Eq. (12.11.1):

$$H_i(z) = \frac{1}{1 - 2 \frac{\Omega_i}{\Omega_0} \cos \theta_i + \frac{\Omega_i^2}{\Omega_0^2} \left(1 - z^{-2}\right)}$$

where the coefficients are given for $i = 1, 2, \ldots, K$:

$$G_i = \frac{\Omega_i^2}{1 - 2 \Omega_0 \cos \theta_i + \Omega_0^2}, \quad a_{i1} = \frac{4c\Omega_0 (\cos \theta_i - \Omega_0)}{1 - 2\Omega_0 \cos \theta_i + \Omega_0^2}, \quad a_{i2} = \frac{2(2c^2\Omega_0^2 + \Omega_0^2 - 1)}{1 - 2\Omega_0 \cos \theta_i + \Omega_0^2},$$

$$a_{i3} = \frac{4c\Omega_0 (\cos \theta_i - \Omega_0)}{1 - 2\Omega_0 \cos \theta_i + \Omega_0^2}, \quad a_{i4} = \frac{1 + 2\Omega_0 \cos \theta_i + \Omega_0^2}{1 - 2\Omega_0 \cos \theta_i + \Omega_0^2}$$

If $N$ is odd, we also have a second-order section in $z$:

$$H_0(z) = \frac{1}{1 + \frac{\Omega_0}{\Omega_0} \left(1 - z^{-2}\right)} = \frac{G_0 (1 - 2cz^{-1} + z^{-2})}{1 + a_{01} z^{-1} + a_{02} z^{-2}}$$
where

\[ G_0 = \frac{\Omega_0}{1 + \Omega_0}, \quad a_{01} = -\frac{2c\Omega_0}{1 + \Omega_0}, \quad a_{02} = -\frac{1 - \Omega_0}{1 + \Omega_0} \]

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:

\[
\tan \left( \frac{\omega_{pa}}{2} \right) = \tan \left( \frac{\pi f_{pa}}{f_s} \right) = \frac{\sqrt{1 + \Omega_0^2(1-c^2)} - 1}{\Omega_0(1+c)}
\]

\[
\tan \left( \frac{\omega_{pa}}{2} \right) = \tan \left( \frac{\pi f_{pa}}{f_s} \right) = \frac{\sqrt{1 + \Omega_0^2(1-c^2)} + 1}{\Omega_0(1+c)}
\]

**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 ... 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 \(A_{\text{stop}} = -10 \log_{10}(0.02) = 16.9897\) dB. The design has the same \(c\), \(\Omega_{\text{pass}}\), and \(\Omega_{\text{stop}}\), 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 \(f_{pa} = 1.6194\) kHz, \(f_{pb} = 4.5212\) kHz. The digital filter coefficients of the second- and fourth-order sections are:

<table>
<thead>
<tr>
<th>(i)</th>
<th>(G_i)</th>
<th>(a_{i1})</th>
<th>(a_{i2})</th>
<th>(a_{i3})</th>
<th>(a_{i4})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.6952</td>
<td>-0.9172</td>
<td>0.3904</td>
<td>0.3904</td>
<td>0.3904</td>
</tr>
<tr>
<td>1</td>
<td>0.7208</td>
<td>-2.0876</td>
<td>2.4192</td>
<td>-1.7164</td>
<td>0.7187</td>
</tr>
<tr>
<td>2</td>
<td>0.5751</td>
<td>-1.9322</td>
<td>1.9301</td>
<td>-1.1026</td>
<td>0.3712</td>
</tr>
<tr>
<td>3</td>
<td>0.5045</td>
<td>-1.8570</td>
<td>1.6932</td>
<td>-0.8053</td>
<td>0.2029</td>
</tr>
</tbody>
</table>

The magnitude response of the designed filter is shown in the left graph of Fig. 12.11.2.

![Digital Butterworth filters](image)

**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 \(c\), \(\Omega_{\text{pass}}\), and \(\Omega_{\text{stop}}\), 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 \(f_{pa} = 1.6194\) kHz, \(f_{pb} = 4.5212\) kHz. The digital filter coefficients of the second- and fourth-order sections are:

<table>
<thead>
<tr>
<th>(i)</th>
<th>(G_i)</th>
<th>(a_{i1})</th>
<th>(a_{i2})</th>
<th>(a_{i3})</th>
<th>(a_{i4})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.6951</td>
<td>-0.9171</td>
<td>0.3902</td>
<td>0.3902</td>
<td>0.3902</td>
</tr>
<tr>
<td>1</td>
<td>0.7208</td>
<td>-2.0876</td>
<td>2.4192</td>
<td>-1.7164</td>
<td>0.7187</td>
</tr>
<tr>
<td>2</td>
<td>0.5751</td>
<td>-1.9322</td>
<td>1.9301</td>
<td>-1.1026</td>
<td>0.3712</td>
</tr>
<tr>
<td>3</td>
<td>0.5045</td>
<td>-1.8570</td>
<td>1.6932</td>
<td>-0.8053</td>
<td>0.2029</td>
</tr>
</tbody>
</table>

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 \(A_{\text{stop}} = 19.67\) 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. $\Box$
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.](image)

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 \( \{ \epsilon_{\text{pass}}, \epsilon_{\text{stop}} \} \) 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:

\[
|H(\Omega)|^2 = \frac{1}{1 + \epsilon_{\text{pass}}^2 C_N^2 \left( \frac{\Omega}{\Omega_{\text{pass}}} \right)}
\]  

(12.12.1)

and, for the type 2 case:

\[
|H(\Omega)|^2 = \frac{C_N^2 \left( \frac{\Omega_{\text{stop}}}{\Omega} \right)}{C_N^2 \left( \frac{\Omega_{\text{stop}}}{\Omega} \right) + \epsilon_{\text{stop}}^2}
\]  

(12.12.2)

where \( C_N(x) \) is the Chebyshev polynomial [301] of degree \( N \), defined by

\[
C_N(x) = \begin{cases} 
\cos(N \cos^{-1}(x)), & \text{if } |x| \leq 1 \\
\cosh(N \cosh^{-1}(x)), & \text{if } |x| > 1 
\end{cases}
\]

(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 \( \theta \) to be imaginary, say \( \theta = j\beta \), so that \( x = \cos(j\beta) = \cosh(\beta) \) and

\[
C_N(x) = \cos(N \theta) = \cosh(Nj\beta) = \cosh(N\beta) = \cosh(N \cosh^{-1} x)
\]

Using trigonometric identities, it can be shown that \( \cos(N \theta) \) is expressible as an \( N \)th 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

\[
\begin{align*}
\cos(2\theta) &= 2 \cos^2 \theta - 1 & \quad & C_2(x) = 2x^2 - 1 \\
\cos(3\theta) &= 4 \cos^3 \theta - 3 \cos \theta & \quad & C_3(x) = 4x^3 - 3x \\
\cos(4\theta) &= 8 \cos^4 \theta - 8 \cos^2 \theta + 1 & \quad & C_4(x) = 8x^4 - 8x^2 + 1
\end{align*}
\]

The following routine cheby.c returns the value of the \( N \)th order Chebyshev polynomial for non-negative values of \( x \) and can be used in the numerical evaluation of the magnitude responses:

\[
\text{/* cheby.c - Chebyshev polynomial } C_N(x) */
\]

#include <math.h>

double cheby(int N, double x);
int N;

double cheby(int N, double x)
{
    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:

\[
\cosh(N \cosh^{-1} x) = \cos \left( N \ln \left( x + \sqrt{x^2 - 1} \right) \right)
\]
12.12. CHEBYSHEV FILTER DESIGN

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 \( 0 \leq \Omega \leq \Omega_{\text{pass}} \), we take \( 0 \leq x \leq 1 \), 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 \( 1/\left(1 + \varepsilon_{\text{pass}}^2\right) \). 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 \( N \)),} \\
|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, \( |H(\Omega)|^2 \leq 1/(1 + \varepsilon_{\text{stop}}^2) \) for \( \Omega \geq \Omega_{\text{stop}} \). 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:

\[
\frac{1}{1 + \varepsilon_{\text{stop}}^2} \cosh^2 \left( N \cosh^{-1} \left( \frac{\Omega_{\text{stop}}}{\Omega_{\text{pass}}} \right) \right) = \frac{1}{1 + \varepsilon_{\text{stop}}^2}
\]

which gives:

\[
\cosh \left( N \cosh^{-1} \left( \frac{\Omega_{\text{stop}}}{\Omega_{\text{pass}}} \right) \right) = \frac{\varepsilon_{\text{stop}}}{\varepsilon_{\text{pass}}} \quad \Rightarrow \quad \cosh \left( N \cosh^{-1} w \right) = e
\]

where, as in Eq. (12.7.5), we used the stopband to passband ratios:

\[
e = \frac{\varepsilon_{\text{stop}}}{\varepsilon_{\text{pass}}} = \sqrt{\frac{10^{\varepsilon_{\text{stop}}/10} - 1}{10^{\varepsilon_{\text{pass}}/10} - 1}}, \quad w = \frac{\Omega_{\text{stop}}}{\Omega_{\text{pass}}} \tag{12.12.5}
\]

Thus, solving for \( N \), we find:

\[
N_{\text{exact}} = \cosh^{-1} e = \frac{\ln(e + \sqrt{e^2 - 1})}{\cosh^{-1} w} = \frac{\ln(w + \sqrt{w^2 - 1})}{\cosh^{-1} w} \tag{12.12.6}
\]

The final value of \( N \) is obtained by rounding \( N_{\text{exact}} \) 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, \( |H^2(\Omega_{\text{stop}})|^2 < 1/(1 + \varepsilon_{\text{stop}}^2) \). The 3-dB frequency can be calculated by requiring \( |H(\Omega)|^2 = 1/2 \), which can be solved to give:

\[
\frac{1}{1 + \varepsilon_{\text{pass}}^2} C_N^2 \left( \frac{\Omega_{\text{3dB}}}{\Omega_{\text{pass}}} \right) = \frac{1}{2} \Rightarrow \cosh \left( N \cosh^{-1} \left( \frac{\Omega_{\text{3dB}}}{\Omega_{\text{pass}}} \right) \right) = \frac{1}{\varepsilon_{\text{pass}}}
\]

or,

\[
\tan \left( \frac{\pi f_{\text{3dB}}}{f_s} \right) = \Omega_{\text{3dB}} = \Omega_{\text{pass}} \cosh \left( \frac{1}{N} \cosh^{-1} \left( \frac{1}{\varepsilon_{\text{pass}}} \right) \right) \tag{12.12.7}
\]

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

\[
s_i = \Omega_{\text{pass}} \sinh a \cos \theta_i + j\Omega_{\text{pass}} \cosh a \sin \theta_i, \quad i = 1, 2, \ldots, K \tag{12.12.8}
\]

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 \( \theta_i \) are the same as the Butterworth angles of Eq. (12.7.11):

\[
\theta_i = \frac{\pi}{2N} (N - 1 + 2i), \quad i = 1, 2, \ldots, K \tag{12.12.12}
\]

The second-quadrant values of these angles place the \( s_i \) in the left-hand \( s \)-plane. The second-order sections are then:

\[
H_i(s) = \frac{1}{s^2 - 2\Omega_0 \cos \theta_i s + \Omega_i^2}, \quad i = 1, 2, \ldots, K \tag{12.12.14}
\]

The first-order factor \( H_0(s) \) is defined by

\[
H_0(s) = \begin{cases} 
\sqrt{\frac{1 + \varepsilon_{\text{pass}}^2}{\Omega_0^2}} & \text{if } N \text{ is even, } N = 2K \\
\frac{1}{s + \Omega_0} & \text{if } N \text{ is odd, } N = 2K + 1
\end{cases} \tag{12.12.15}
\]

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:

\[
H(s) = H_0(s)H_1(s)H_2(s) \cdots H_K(s) \tag{12.12.16}
\]
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:

\[
1 + \varepsilon_{\text{pass}}^2 \cosh^2(N \cosh^{-1}(-js/\Omega_{\text{pass}})) = 0,
\]

or

\[
\cosh(N \cosh^{-1}(-js/\Omega_{\text{pass}})) = \frac{\pm j}{\varepsilon_{\text{pass}}}
\quad (12.12.17)
\]

Replacing \( \theta_i = \phi_i + \pi/2 \), where \( \phi_i = (2i-1)\pi/(2N) \), into Eq. (12.12.8), we find

\[-js_i/\Omega_{\text{pass}} = \cosh a \sin \theta_i - j \sinh a \cos \theta_i = \cosh a \cos \theta_i + j \sinh a \sin \theta_i \]

Using the trigonometric identity

\[
\cosh(x + y) = \cosh x \cos y + j \sinh x \sin y
\]

we find

\[-js_i/\Omega_{\text{pass}} = \cosh(a + j\phi_i) \Rightarrow \cosh^{-1}(-js_i/\Omega_{\text{pass}}) = a + j\phi_i
\]

and therefore,

\[
\cosh(N \cosh^{-1}(-js_i/\Omega_{\text{pass}})) = \cosh(Na + jN\phi_i)
= \cosh(Na) \cos(N\phi_i) + j \sinh(Na) \sin(N\phi_i) = \frac{\pm j}{\varepsilon_{\text{pass}}}
\]

where we used \( \cos(N\phi_i) = \cos((2i-1)\pi/2) = 0 \), \( \sin(N\phi_i) = \sin((2i-1)\pi/2) = \pm 1 \), 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:

\[
H(z) = H_0(z)H_1(z)H_2(z)\cdots H_K(z)
\quad (12.12.18)
\]

where \( H_i(z) \) are the transformations of Eq. (12.12.14):

\[
H_i(z) = \frac{G_i(1 + z^{-1})^2}{1 + a_{i1}z^{-1} + a_{i2}z^{-2}}, \quad i = 1, 2, \ldots, K
\quad (12.12.19)
\]

where the coefficients are computed in terms of the definitions Eq. (12.12.13):

\[
G_i = \frac{\Omega_0^2 + \Omega_i^2}{1 - 2\Omega_0 \cos \theta_i + \Omega_0^2 + \Omega_i^2}
\]

\[
a_{i1} = \frac{2(\Omega_0^2 + \Omega_i^2 - 1)}{1 - 2\Omega_0 \cos \theta_i + \Omega_0^2 + \Omega_i^2}
\]

\[
a_{i2} = \frac{1 + 2\Omega_0 \cos \theta_i + \Omega_0^2 + \Omega_i^2}{1 - 2\Omega_0 \cos \theta_i + \Omega_0^2 + \Omega_i^2}
\quad (12.12.20)
\]

The first-order factor is given by

\[
H_0(z) = \begin{cases} \sqrt{\frac{1}{1 + \varepsilon_{\text{pass}}^2}} & \text{if } N \text{ is even} \\ \frac{G_0}{1 + a_{01}z^{-1}} & \text{if } N \text{ is odd} \end{cases}
\quad (12.12.21)
\]

where

\[
G_0 = \frac{\Omega_0}{\Omega_0 + 1}, \quad a_{01} = \frac{\Omega_0 - 1}{\Omega_0 + 1}
\quad (12.12.22)
\]

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):

\[
\tan \left( \frac{\pi f_{\text{pass}}}{f_s} \right) = \Omega_{\text{pass}} = \Omega_{\text{pass}} \cosh \left( \frac{1}{N} \cosh^{-1} \left( \frac{1}{\varepsilon_{\text{pass}}} \right) \right)
\]

which gives \( f_{\text{pass}} = 4.2729 \) kHz. The actual stopband attenuation is larger than \( A_{\text{stop}} \) because of the increased value of \( N \). We calculate:

\[
A(\Omega_{\text{stop}}) = 10 \log_{10} \left( 1 + \varepsilon_{\text{pass}}^2 C_K(\Omega_{\text{stop}} / \Omega_{\text{pass}}) \right) = 14.29 \text{ dB}
\]

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:

\[
H(z) = 0.9441 \cdot \frac{0.3091(1 + z^{-1})^2}{1 - 0.4830z^{-1} + 0.7194z^{-2}} \cdot \frac{0.1043(1 + z^{-1})^2}{1 - 0.9044z^{-1} + 0.3177z^{-2}}
\]

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,

\[
|H(f)|^2 = \frac{1}{1 + \varepsilon_{\text{pass}}^2 \text{cheby}(N, \tan(\pi f / f_s) / \varepsilon_{\text{pass}})}^2
\]

For the more stringent specifications, we have \( \varepsilon_{\text{pass}} = 0.08774 \) dB and \( A_{\text{stop}} = 16.9897 \) dB. We calculate the parameters:
12.12. CHEBYSHEV FILTER DESIGN

565

12. IIR DIGITAL FILTER DESIGN

566

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 \( a_i \) coefficients. We have for the two specification sets:

\[
\begin{array}{ccc}
\bar{i} & \bar{G_i} & \bar{a_1} & \bar{a_2} \\
0 & 0.9441 & 0 & 0.8989 \\
1 & 0.4405 & -0.5026 & 0.7095 \\
2 & 0.1618 & 0.5843 & 0.2314 \\
3 & & & \\
\end{array}
\]

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. \( \square \)

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 \( |H| = 1/\sqrt{1 + \epsilon_{\text{stop}}^2} \). At large frequencies, \( \Omega \rightarrow \infty \), we have \( x \rightarrow 0 \). 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 \( 1/\sqrt{1 + \epsilon_{\text{stop}}^2} \) for even \( N \).

The zero frequency \( \Omega = 0 \) corresponds to the limit \( x \rightarrow \infty \) 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

\[
\text{Fig. 12.12.2} \quad \text{Digital lowpass Chebyshev type 1 filters.}
\]

\[
\text{Fig. 12.12.3} \quad \text{Digital highpass Chebyshev type 1 filters.}
\]
\(|H(0)|^2 = 1\). Thus, type 2 filters are always normalized to unity at DC. The filter order \(N\) can be determined by requiring the passband specification:

\[
|H(\Omega_{\text{pass}})|^2 = \frac{C_N^2 \left( \Omega_{\text{stop}} / \Omega_{\text{pass}} \right)^2 + \varepsilon_{\text{stop}}^2}{1 + \varepsilon_{\text{pass}}^2} = 1
\]

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 \(|H(\Omega_{\text{db}})|^2 = 1/2\) which gives the solution:

\[
\Omega_{\text{db}} = \frac{\Omega_{\text{stop}}}{\cosh \left( \frac{1}{N} \cosh^{-1}(\varepsilon_{\text{stop}}) \right)} \quad (12.12.23)
\]

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 \(\Omega = -js\):

\[
\cosh^2 \left( N \cosh^{-1} \left( \frac{\Omega_{\text{stop}}}{-js} \right) \right) = 0, \quad \cosh^2 \left( N \cosh^{-1} \left( \frac{j\Omega_{\text{stop}}}{-js} \right) \right) + \varepsilon_{\text{stop}}^2 = 0
\]

The zeros are the conjugate pairs \(\{z_i, z_i^*\}\):

\[
z_i = \frac{j \Omega_{\text{stop}}}{\sin \theta_i}, \quad i = 1, 2, \ldots, K
\]

where \(N = 2K\) or \(N = 2K + 1\). The poles are essentially the reciprocals of the type 1 poles, that is, the pairs \(\{s_i, s_i^*\}\):

\[
s_i = \frac{\Omega_{\text{stop}}}{\sinh a \cos \theta_i + j \cosh a \sin \theta_i}, \quad i = 1, 2, \ldots, K
\]

\[
s_0 = -\frac{\Omega_{\text{stop}}}{\sinh a}
\]

In the odd-\(N\) case, there is also a real pole at

\[
\theta_i = \frac{\Omega_{\text{stop}}}{\sinh a}, \quad i = 1, 2, \ldots, K
\]

where the parameter \(a\) is the solution of

\[
\sinh(Na) = \varepsilon_{\text{stop}} \quad (12.12.27)
\]

that is,

\[
a = \frac{1}{N} \sinh^{-1}(\varepsilon_{\text{stop}}) = \frac{1}{N} \ln \left( \varepsilon_{\text{stop}} + \sqrt{\varepsilon_{\text{stop}}^2 + 1} \right)
\]

The angles \(\theta_i\) 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:

\[
H_i(s) = \frac{1 - \frac{s}{z_i}}{1 - \frac{s}{z_i^*}} \quad (12.12.29)
\]

For convenience, we define the parameters:

\[
\Omega_0 = \frac{\Omega_{\text{stop}}}{\sinh a}, \quad \Omega_i = \frac{\Omega_{\text{stop}}}{\sin \theta_i}, \quad i = 1, 2, \ldots, K
\]

Then, we may express the second-order sections in the form:

\[
H_i(s) = \frac{1 + \Omega_i^{-2}s^2}{1 - 2\Omega_0^{-1} \cos \theta_i s + (\Omega_i^{-2} + \Omega_0^{-2})s^2}, \quad i = 1, 2, \ldots, K
\]

The first-order factor \(H_0(s)\) is defined by

\[
H_0(s) = \begin{cases} 1, \quad \text{if} \ N \text{ is even} \\ \frac{s}{\Omega_0 + s}, \quad \text{if} \ N \text{ is odd} \end{cases}
\]

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:

\[
H_i(z) = \frac{G_i(1 + b_{1i}z^{-1} + z^{-2})}{1 + a_{1i}z^{-1} + a_{2i}z^{-2}}, \quad i = 1, 2, \ldots, K
\]

where the coefficients are computed in terms of the definitions in Eq. (12.12.30):

\[
G_i = \frac{1 + \Omega_i^{-2}}{1 - 2\Omega_0^{-1} \cos \theta_i + \Omega_i^{-2} + \Omega_0^{-2}}, \quad b_{1i} = 2 \frac{1 - \Omega_i^{-2}}{1 + \Omega_i^{-2}}
\]

\[
a_{1i} = \frac{2(1 - \Omega_0^{-2} - \Omega_i^{-2})}{1 - 2\Omega_0^{-1} \cos \theta_i + \Omega_i^{-2} + \Omega_0^{-2}}, \quad a_{2i} = \frac{1 + 2\Omega_i^{-2} \cos \theta_i + \Omega_i^{-2} + \Omega_0^{-2}}{1 - 2\Omega_0^{-1} \cos \theta_i + \Omega_i^{-2} + \Omega_0^{-2}}
\]

The first-order factor, if present, is given by

\[
H_0(z) = \frac{G_0(1 + z^{-1})}{1 + a_{10}z^{-1}}
\]

where

\[
G_0 = \frac{\Omega_0}{\Omega_0 + 1}, \quad a_{10} = \frac{\Omega_0 - 1}{\Omega_0 + 1}
\]

The overall digital transfer function is then:

\[
H(z) = H_0(z) H_1(z) H_2(z) \cdots H_K(z)
\]

Example 12.12.3: Redesign the two filters of Example 12.12.1 using a type 2 Chebyshev design.
Solution: The values of \( \varepsilon_{\text{pass}} \), \( \varepsilon_{\text{stop}} \), \( N_{\text{exact}} \), \( N \) 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:

\[
\tan \left( \frac{\pi f_{3dB}}{f_s} \right) = \Omega_{3dB} = \frac{\Omega_{\text{stop}}}{\cos \left( \frac{\pi}{N} \cos^{-1}(\varepsilon_{\text{stop}}) \right)}
\]

which gives the values \( f_{3dB} = 4.7009 \) 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:

\[
H(z) = \frac{0.7612(1 + 0.1580z^{-1} + z^{-2})}{1 - 0.8615z^{-1} + 0.7043z^{-2}} = \frac{0.5125(1 + 1.4890z^{-1} + z^{-2})}{1 + 0.5823z^{-1} + 0.2328z^{-2}}
\]

For the second specification set, we have three sections with coefficients:

<table>
<thead>
<tr>
<th>( i )</th>
<th>( G_i )</th>
<th>( b_{i1} )</th>
<th>( a_{i1} )</th>
<th>( b_{i2} )</th>
<th>( a_{i2} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.8137</td>
<td>0.0693</td>
<td>-0.1180</td>
<td>0.8017</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.6381</td>
<td>0.6667</td>
<td>0.2492</td>
<td>0.4524</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.4955</td>
<td>1.7489</td>
<td>0.6742</td>
<td>0.1834</td>
<td></td>
</tr>
</tbody>
</table>

The designed magnitude responses are shown in Fig. 12.12.4.

![Fig. 12.12.4 Digital lowpass Chebyshev type 2 filters.](image)

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:

\[
H_i(z) = G_i \frac{1 + b_{i1}z^{-1} + b_{i2}z^{-2} + b_{i3}z^{-3} + z^{-4}}{1 + a_{i1}z^{-1} + a_{i2}z^{-2} + a_{i3}z^{-3} + a_{i4}z^{-4}}, \quad i = 1, 2, \ldots, K
\]

(12.12.38)

where by symmetry, the numerator coefficients of \( z^{-1} \) and \( z^{-3} \) are the same. The coefficients are given by:

\[
G_i = \frac{1 + \Omega_i^{-2}}{1 - 2\Omega_i^{-1} \cos \theta_i + \Omega_i^{-2}}
\]

\[
b_{i1} = -\frac{4c\Omega_i^{-2}}{1 + \Omega_i^{-2}}, \quad b_{i2} = \frac{2(2c^2 + 1) - \Omega_i^{-2}}{1 + \Omega_i^{-2}}
\]

\[
a_{i1} = -\frac{4c(1 - \Omega_i^{-1} \cos \theta_i)}{1 - 2\Omega_i^{-1} \cos \theta_i + \Omega_i^{-2}}, \quad a_{i2} = \frac{2(2c^2 + 1 - \Omega_i^{-2} - \Omega_i^{-2})}{1 - 2\Omega_i^{-1} \cos \theta_i + \Omega_i^{-2} + \Omega_i^{-2}}
\]

\[
a_{i3} = -\frac{4c(1 + \Omega_i^{-1} \cos \theta_i)}{1 - 2\Omega_i^{-1} \cos \theta_i + \Omega_i^{-2} + \Omega_i^{-2}}, \quad a_{i4} = \frac{1 + 2\Omega_i^{-1} \cos \theta_i + \Omega_i^{-2} + \Omega_i^{-2}}{1 - 2\Omega_i^{-1} \cos \theta_i + \Omega_i^{-2} + \Omega_i^{-2}}
\]

For the first set, there are two fourth-order sections with coefficients:

<table>
<thead>
<tr>
<th>( i )</th>
<th>( G_i )</th>
<th>( b_{i1} )</th>
<th>( b_{i2} )</th>
<th>( a_{i1} )</th>
<th>( a_{i2} )</th>
<th>( a_{i3} )</th>
<th>( a_{i4} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.7334</td>
<td>-1.9684</td>
<td>2.4015</td>
<td>-1.9604</td>
<td>2.2956</td>
<td>-1.6758</td>
<td>0.7697</td>
</tr>
<tr>
<td>2</td>
<td>0.3677</td>
<td>-0.9922</td>
<td>0.2187</td>
<td>-1.4221</td>
<td>0.8671</td>
<td>-0.4101</td>
<td>0.1813</td>
</tr>
</tbody>
</table>

Example 12.12.4: Redesign the bandpass digital filters of Example 12.10.1 using a type 2 Chebyshev analog prototype.

Solution: The prewarp 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:
For the second set, there are three fourth-order sections:

<table>
<thead>
<tr>
<th>i</th>
<th>$G_i$</th>
<th>$b_{i1}$</th>
<th>$b_{i2}$</th>
<th>$a_{i1}$</th>
<th>$a_{i2}$</th>
<th>$a_{i3}$</th>
<th>$a_{i4}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.7840</td>
<td>-2.0032</td>
<td>2.4793</td>
<td>-2.0118</td>
<td>2.4414</td>
<td>-1.8265</td>
<td>0.8501</td>
</tr>
<tr>
<td>2</td>
<td>0.5858</td>
<td>-1.7205</td>
<td>1.8472</td>
<td>-1.7286</td>
<td>1.6780</td>
<td>-1.1223</td>
<td>0.5094</td>
</tr>
<tr>
<td>3</td>
<td>0.3159</td>
<td>-0.5802</td>
<td>-0.7026</td>
<td>-1.3122</td>
<td>0.3868</td>
<td>-0.1879</td>
<td>0.0904</td>
</tr>
</tbody>
</table>

The designed magnitude responses are shown in Fig. 12.12.5.

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:

<table>
<thead>
<tr>
<th>i</th>
<th>$G_i$</th>
<th>$b_{i1}$</th>
<th>$b_{i2}$</th>
<th>$a_{i1}$</th>
<th>$a_{i2}$</th>
<th>$a_{i3}$</th>
<th>$a_{i4}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.8727</td>
<td>-2.3644</td>
<td>3.1438</td>
<td>-2.2003</td>
<td>2.6963</td>
<td>-1.9264</td>
<td>0.7924</td>
</tr>
<tr>
<td>2</td>
<td>0.7442</td>
<td>-2.5872</td>
<td>3.6287</td>
<td>-2.2339</td>
<td>2.6565</td>
<td>-1.6168</td>
<td>0.5323</td>
</tr>
</tbody>
</table>

For the second set, there are three fourth-order sections:

<table>
<thead>
<tr>
<th>i</th>
<th>$G_i$</th>
<th>$b_{i1}$</th>
<th>$b_{i2}$</th>
<th>$a_{i1}$</th>
<th>$a_{i2}$</th>
<th>$a_{i3}$</th>
<th>$a_{i4}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.9074</td>
<td>-2.3417</td>
<td>3.0945</td>
<td>-2.2171</td>
<td>2.7626</td>
<td>-2.0326</td>
<td>0.8601</td>
</tr>
<tr>
<td>2</td>
<td>0.8009</td>
<td>-2.4708</td>
<td>3.3754</td>
<td>-2.2137</td>
<td>2.6612</td>
<td>-1.7441</td>
<td>0.6441</td>
</tr>
<tr>
<td>3</td>
<td>0.7412</td>
<td>-2.6149</td>
<td>3.6889</td>
<td>-2.2524</td>
<td>2.6929</td>
<td>-1.6241</td>
<td>0.5328</td>
</tr>
</tbody>
</table>

The designed magnitude responses are shown in Fig. 12.12.6.

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 $\omega_0$ and $\Delta \omega$. Show that its transient (e-level) time constant is given by:

$$n_{\text{eff}} = \frac{\ln \epsilon}{\ln \rho}, \quad \rho = \sqrt{\frac{1 + \beta}{1 - \beta}}$$

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, $\Delta \omega = 2(1 - R)$. Show also that $n_{\text{eff}} = 2\ln \epsilon/\Delta \omega$.

12.3 Verify the complementarity properties for the filters of Eqs. (12.3.7) and (12.3.21):

$$H_{\text{match}}(z) + H_{\text{peak}}(z) = 1, \quad |H_{\text{match}}(\omega)|^2 + |H_{\text{peak}}(\omega)|^2 = 1,$$

when $G_0^2 = 1/2$.

12.4 Consider the three choices for the bandwidth reference gain in parametric equalizer filters:

- $G_0^2 = (G_0 + G_1)/2$ (arithmetic mean), $G_0^2 = G_0 G$ (geometric mean), and $G_0^2 = 2G_0 G^2/(G_0^2 + G^2)$ (harmonic mean). For each case, discuss how the design parameter $\beta$ 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 $|H_{\text{boost}}(\omega)|^2 |H_{\text{cut}}(\omega)|^2 = G_0^2$. Show that the more general weighted geometric mean choice $G_0^2 = G_0^{-2} G_1 G^{-1}$ 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.)
12.13. PROBLEMS

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:

\[ y = \text{combfilt}(a, b, c, D, w, x); \]  
(C version)

\[ \{y, w\} = \text{combfilt}(a, b, c, D, w, x); \]  
(MATLAB version)

where \( w \) 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 \( \Omega_0 \) of an analog Butterworth filter are determined by solving the two specification equations \( A(\Omega_{\text{pass}}) = A_{\text{pass}} \) and \( A(\Omega_{\text{stop}}) = A_{\text{stop}} \). The resulting filter order is then rounded up to the next integer value \( N \).

Using this slightly larger \( N \), show that if \( \Omega_0 \) is found from the passband specification, that is, by solving \( A(\Omega_{\text{pass}}) = A_{\text{pass}} \), then the stopband specification is more satisfied, that is, \( A(\Omega_{\text{stop}}) > A_{\text{stop}} \).

Similarly, show that if we find \( \Omega_0 \) from the stopband specification \( A(\Omega_{\text{stop}}) = A_{\text{stop}} \), then the passband specification is more 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: \( f_{\text{pass}} = 10 \text{ kHz} \), \( A_{\text{pass}} = 3 \text{ dB} \), \( f_{\text{stop}} = 15 \text{ kHz} \), \( A_{\text{stop}} = 35 \text{ dB} \). 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 \( \cot(\alpha/2) \). Sketch the magnitude response over the frequency interval \( 0 \leq f \leq 20 \text{ kHz} \).

12.13 Show that the generalized bilinear transformation

\[ s = \frac{1 - 2cz^{-1} + z^{-2}}{1 - z^{-2}} \]  
(12.13.1)

maps the left-hand \( z \)-plane onto the inside of the unit circle of the \( z \)-plane, provided the real constant \( c \) is such that \( |c| \leq 1 \).

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 \( |c| \leq 1 \), regardless of the values of \( \omega_1 \) and \( \omega_2 \) 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 \( 0 \leq f \leq 30 \text{ kHz} \).

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 \( \lambda_j \) and the quantities \( \{\Omega_0, \Omega_1\} \). Show that they satisfy the following relationship, which is used in the second-order Chebyshev sections (12.12.14):

\[ |\lambda_j|^2 = \Omega_0^2 + \Omega_1^2 \]

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 bpccheb2.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:
12.13. PROBLEMS

\[ y = \text{cas4}(K, B, A, W, x); \]  \quad \text{(C version)}
\[ [y, W] = \text{cas4}(K, B, A, W, x); \]  \quad \text{(MATLAB version)}

It must call \( K \) times a single fourth-order section routine, like the \( \text{sos} \) routine. Then, write C and/or MATLAB filtering routines, like \( \text{casfilt} \) 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 \( \text{lhbutt.m} \) and \( \text{bpsbutt.m} \) 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 \( \text{roots} \)). Then, using the routines \( \text{cas.c} \) or \( \text{cas.m} \) (or \( \text{cas4.c} \), \( \text{cas4.m} \) 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 \( \text{lhcheb1} \), \( \text{lhcheb2} \), \( \text{bpcheb2} \), and \( \text{bscheb2} \) 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 \( \text{cas2can} \) and \( \text{dtft} \). For example, the frequency response of a type 2 bandstop design can be computed as follows:

\[
[A, B, P] = \text{bscheb2}(fs, fpa, fpb, fsa, fsb, Apass, Asstop);
\]
\[
a = \text{cas2can}(A); \quad \text{direct-form denominator}
\]
\[
b = \text{cas2can}(B); \quad \text{direct-form numerator}
\]
\[
w = (0: NF-1) \times \pi / NF; \quad \text{NF frequencies over} \ [0, \pi]
\]
\[
H = \text{dtft}(b, w) ./ \text{dtft}(a, w); \quad \text{compute} \ H(\omega) = N(\omega)/D(\omega)
\]

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:

<table>
<thead>
<tr>
<th>Filter Type</th>
<th>Smaller Order</th>
<th>Narrower Transition</th>
<th>More Linear Phase</th>
<th>Over the Passband</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bessel</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Butterworth</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chebyshev</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elliptic</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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:

\[
G_p = \frac{1}{\sqrt{1 + \varepsilon_p^2}} = 10^{-A_p/20}, \quad G_s = \frac{1}{\sqrt{1 + \varepsilon_s^2}} = 10^{-A_s/20} \quad (13.1.1)
\]

which can be inverted to give:

\[
A_p = -20 \log_{10} G_p = 10 \log_{10} (1 + \varepsilon_p^2) \quad \Rightarrow \quad \varepsilon_p = \sqrt{G_p^2 - 1} = \sqrt{10^{-A_p/10} - 1}
\]
\[
A_s = -20 \log_{10} G_s = 10 \log_{10} (1 + \varepsilon_s^2) \quad \Rightarrow \quad \varepsilon_s = \sqrt{G_s^2 - 1} = \sqrt{10^{-A_s/10} - 1} \quad (13.1.2)
\]

† Bessel filters have, by design, the most linear phase over their passband, discussed briefly in Sec. 14.4.4.
13.1 INTRODUCTION

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, $\varepsilon_s = \varepsilon_pk_1^{-1}$, we can rewrite:

$$|H(\Omega)|^2 = \frac{1}{1 + \varepsilon_pN/w^2} = \frac{1}{1 + \varepsilon_s^2/C_N(\Omega_s/\Omega)} \quad (13.1.6)$$

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:

$$|H(\Omega_p)|^2 = \frac{1}{1 + \varepsilon_pN/k^2} = \frac{1}{1 + \varepsilon_s^2/C_N(\Omega_s/\Omega)} \quad (13.1.7)$$

$$|H(\Omega_s)|^2 = \frac{1}{1 + \varepsilon_pN/k^2} = \frac{1}{1 + \varepsilon_s^2/C_N(\Omega_s/\Omega)}$$

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 $N,k,k_1$:

$$F_N(k^{-1}) = k_1^{-1}$$

(13.1.8)

In particular, we find that the degree equation takes the following forms in the Butterworth and both Chebyshev cases:

$$k^{-N} = k_1^{-1} \Rightarrow N = \ln(k_1^{-1})/\ln(k^{-1}) = \ln(\varepsilon_s/\varepsilon_p)/\ln(\Omega_s/\Omega_p)$$

(13.1.9)

$$C_N(k^{-1}) = k_1^{-1} \Rightarrow N = \text{acosh}(k_1^{-1})/\text{acosh}(k^{-1}) = \text{acosh}(\varepsilon_s/\varepsilon_p)/\text{acosh}(\Omega_s/\Omega_p)$$

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 $\Omega_p,\Omega_s$ and $\varepsilon_s,\varepsilon_p$, 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 $2\pi$ to lie within $[-\pi, \pi]$.) The specifications were as follows:

$$f_p = 4, \quad G_p = 0.95, \quad A_p = -20\log_{10}G_p = 0.4455 \text{ dB}$$

$$f_s = 4.5, \quad G_s = 0.05, \quad A_s = -20\log_{10}G_s = 26.0206 \text{ dB}$$

(13.1.10)

$^1$The Chebyshev-2 is $F_N(1) = 1$ that provides the desired relationship among $N,k,k_1$. 

---

Fig. 13.1.1 Brick wall specifications for a Butterworth filter.
13.1. INTRODUCTION

Butterworth, $N = 35$

Chebyshev−1, $N = 10$

Chebyshev−2, $N = 10$

Elliptic, $N = 5$

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:

$$k = \frac{\Omega_p}{\Omega_s} = \frac{f_p}{f_s} = 0.8889$$

$$k_1 = \frac{\varepsilon_p}{\varepsilon_s} = \frac{\sqrt{10^{4}\omega_{10}^2 - 1}}{\sqrt{10^{4}\omega_{10}^2 - 1}} = 0.0165$$

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

$$C_N(w) \rightarrow \frac{1}{k_1 C_N(k^{-1}W^{-1})}$$

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:

$$F_N(w) = \frac{1}{k_1 F_N(k^{-1}W^{-1})}$$

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:

$$|H(\Omega)|^2 = \frac{1}{1 + \varepsilon_p^2 F_N(\Omega/\Omega_p)} = \frac{1}{1 + \varepsilon_s^2 / F_N(\Omega_s/\Omega)}$$

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, \( w = \text{sn}(z, k) \), is defined indirectly through the elliptic integral:
\[
    w = \sin \phi, \quad z = \int \frac{d\theta}{\sqrt{1-k^2 \sin^2 \theta}} = \int \frac{dt}{\sqrt{(1-t^2)(1-k^2 t^2)}} \quad (13.2.1)
\]
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\(^1\) and is assumed to be a real number in the interval, \( 0 \leq k \leq 1 \). Thus, writing, \( \phi = \phi(z, k) \), the function \( \text{sn} \) is defined as:
\[
    w = \text{sn}(z, k) = \sin \phi(z, k) \quad (13.2.2)
\]
The three related elliptic functions, \( \text{cn}, \text{dn}, \text{cd} \), are defined in terms of \( \text{sn} \) by:
\[
    w = \text{cn}(z, k) = \cos \phi(z, k) = \cos[\arcsin(\text{sn}(z, k))] \\
    w = \text{dn}(z, k) = \sqrt{1-k^2 \sin^2(z, k)} \\
    w = \text{cd}(z, k) = \frac{\text{cn}(z, k)}{\text{dn}(z, k)} \quad (13.2.3)
\]
In filter design, only the functions \( \text{sn} \) and \( \text{cd} \) are needed. In the limits \( k = 0 \) and \( k = 1 \), we obtain the trigonometric and hyperbolic functions, respectively:
\[
    \text{sn}(z, 0) = \sin z, \quad \text{sn}(z, 1) = \tanh z \\
    \text{cn}(z, 0) = \cos z, \quad \text{cn}(z, 1) = \sech z \\
    \text{dn}(z, 0) = 1, \quad \text{dn}(z, 1) = \sech z \\
    \text{cd}(z, 0) = \cosh z, \quad \text{cd}(z, 1) = 1 \quad (13.2.4)
\]
The functions \( \text{sn}, \text{cn}, \text{dn}, \text{cd} \) satisfy the following properties, where \( k' = (1-k^2)^{1/2} \):
\[
    \text{sn}^2(z, k) + \text{cn}^2(z, k) = 1 \\
    k^2 \text{sn}^2(z, k) + \text{dn}^2(z, k) = 1 \\
    \text{dn}^2(z, k) - k^2 \text{cn}^2(z, k) = k^2 \\
    k^2 \text{sn}^2(z, k) + \text{cn}^2(z, k) = \text{dn}^2(z, k) \quad (13.2.5)
\]
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 \( K \) or simply \( K(k) \):
\[
    K = \int_0^{\pi/2} \frac{d\theta}{\sqrt{1-k^2 \sin^2 \theta}} \quad (complete \ elliptic \ integral) \quad (13.2.6)
\]
It follows from the definitions (13.2.2) and (13.2.3) that, \( \text{sn}(K, k) = 1 \), and, \( \text{cd}(K, k) = 0 \). Associated with an elliptic modulus \( k \), we may define the complementary modulus\(^1\).

\(^1\)In some discussions, as well as in MATLAB, the parameter \( m = k^2 \) is used instead of \( k \).
13. JACOBIAN ELLIPTIC FUNCTIONS

13.2. JACOBIAN ELLIPTIC FUNCTIONS

In particular, setting \( z = 0 \) in (13.2.11), or, \( z = K' \) in (13.2.12), we obtain:

\[
\text{cd} \left( jK', k \right) = \frac{1}{k}
\]

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 \( S, C, N, D \).

An elliptic function \( pq(z, k) \) 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,

\[
\text{sn}, \text{sd}, \text{sc}, \text{cd}, \text{cs}, \text{dn}, \text{dc}, \text{ds}, \text{ns}, \text{nd}, \text{nc}
\]

Each function \( pq(z, k) \) has a simple zero at corner \( p \) and a simple pole at corner \( q \) of the fundamental rectangle. For example, \( \text{sn}(z, k) \) has a zero at the point \( S, z = 0 \), and a pole at the point \( N, z = jK' \). Similarly, \( \text{cd}(z, k) \) has a zero at the point \( C, z = K \), and a pole at the point \( D, z = K + jK' \). Moreover, the following relationships hold:

\[
\text{pq}(z, k) = \frac{1}{\text{qp}(z, k)} \quad \text{pq}(z, k) = \frac{\text{pr}(z, k)}{\text{qr}(z, k)}
\]

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), \( \text{cd}(z, k) = \text{cn}(z, k) / \text{dn}(z, k) \).

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 \( \text{cd}(z, k) \), shown in Fig. 13.2.5, are given follows, where \( n, m \) are arbitrary integers (positive, negative, or zero):

\[
\text{zeros:} \quad z = K + 2mK + 2njK' = (2m + 1)K + 2njK' \\
\text{poles:} \quad z = K + jK' + 2mK + 2njK' = (2m + 1)K + (2n + 1)jK'
\]

The functions, \( w = \text{cd}(z, k) \), and, \( w = \text{sn}(z, k) \), 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 \( pq(z, k) \), such a region

**Fig. 13.2.4** The fundamental rectangle.
13. JACOBIAN ELLIPTIC FUNCTIONS

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 \( \text{cd}(z, k) \) and \( \text{sn}(z, k) \) functions are centered at the points \( C \) and \( S \), respectively, and are defined by:

\[
\begin{align*}
\text{cd}(z, k): & \quad 0 \leq \text{Re } z \leq 2K, \quad -K' \leq \text{Im } z \leq K' \quad \text{(fundamental regions)} \\
\text{sn}(z, k): & \quad -K \leq \text{Re } z \leq K, \quad -K' \leq \text{Im } z \leq K' \quad \text{(period rectangles)}
\end{align*}
\]

(13.2.16)

Recalling that the periods of \( \text{cd} \) and \( \text{sn} \) are \( 4K \) and \( 2K' \), we have doubled-up the fundamental regions in Figs. 13.2.6 and 13.2.7 to cover one complete period rectangle, that is,

\[
\begin{align*}
\text{cd}(z, k): & \quad 0 \leq \text{Re } z \leq 4K, \quad -K' \leq \text{Im } z \leq K' \\
\text{sn}(z, k): & \quad -K \leq \text{Re } z \leq 3K, \quad -K' \leq \text{Im } z \leq K' 
\end{align*}
\]

(13.2.17)

Of particular interest to filter design is the property that for the function, \( w = \text{cd}(z, k) \), the path around the fundamental rectangle \( C \rightarrow S \rightarrow N \rightarrow D \) 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 \( 0 \leq u \leq 1 \), get mapped as follows:

\[
\begin{align*}
\text{path } C \rightarrow S, \quad 0 \leq u \leq 1, & \quad z = K - Ku \Rightarrow 0 \leq w \leq 1, \quad \text{passband} \\
\text{path } S \rightarrow N, \quad 0 \leq u \leq 1, & \quad z = jK'u \Rightarrow 1 \leq w \leq 1/k, \quad \text{transition region} \\
\text{path } N \rightarrow D, \quad 0 \leq u \leq 1, & \quad z = Ku + jK' \Rightarrow 1/k \leq w \leq \infty, \quad \text{stopband}
\end{align*}
\]

(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 \( D \rightarrow N \rightarrow S \rightarrow C \) covers the negative \( w \)-axis.

To verify these properties, we note that for the first segment \( C \rightarrow S \), the argument \( z = K - Ku \) is real and the values of \( w = \text{cd}(K - Ku, k) \) will vary over the interval \( 0 \leq w \leq 1 \), as seen in Fig. 13.2.2. For the segment \( S \rightarrow N \), using property (13.2.12), we have \( w = \text{cd}(jK', k) = 1/\text{dn}(uK', k') \), which increases from \( w = 1 \) at \( u = 0 \) to the value \( w = \text{cd}(jK', k) = 1/\text{dn}(K', k') = 1/k \) at \( u = 1 \). Finally, for the segment \( N \rightarrow D \), we use the property (13.2.11) to get:

\[
\begin{align*}
w & = \text{cd}(Ku + jK', k) = \frac{1}{k \text{cd}(Ku, k)}
\end{align*}
\]
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 \( \text{cd} \) function [309]:

\[
F_N(w) = \text{cd}(N\nu K_1, k_1), \quad w = \text{cd}(uK, k)
\]

(13.3.1)

where \( w = \text{cd}(uK, k) \) may be inverted to give \( u \) as a function of \( w \), that is, \( uK = \text{cd}^{-1}(w, k) \). 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 \( k \to 1^\circ \) of the elliptic case:

\[
C_N(w) = \cos(N\pi t/2), \quad w = \cos(u\pi t/2)
\]

(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:

\[
\begin{bmatrix}
N & K' \\
K & K_1'
\end{bmatrix} = \begin{bmatrix}
K' \\
K_1'
\end{bmatrix}
\]

(degree equation) (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:

\[
k^{-1}w^{-1} = k^{-1} \text{cd}(uK + jK', k) = \text{cd} \left( \frac{u + jK'}{K}, k \right)
\]

(13.3.4)

and using Eq. (13.2.11) again, applied with respect to the modulus \( k_1 \), we have:

\[
\frac{1}{k_1} F_N(w) = \frac{1}{k_1} \text{cd}(N\nu K_1, k_1) = \text{cd}(N\nu K_1 + jK_1', k_1)
\]

(13.3.5)

Comparing Eqs. (13.3.4) and (13.3.5), we conclude that in order to satisfy the constraint, \( F_N(k^{-1}w^{-1}) = \left[ k_1 F_N(w) \right]^{-1} \), the following identity must be satisfied for all \( u \):

\[
\text{cd} \left( N\nu K_1 + jNK_1'/K, k_1 \right) = \text{cd}(N\nu K_1 + jK_1', k_1) \Rightarrow \frac{NK_1}{K} = K_1'
\]

(13.3.6)

from which Eq. (13.3.3) follows. We will see below that condition (13.1.8) that was obtained earlier,

\[
F_N(k^{-1}) = k_1^{-1}
\]

(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:

\[
C \to S, \quad w = \text{cd}(K - Ku, k), \quad F_N(w) = \text{cd}(NK_1 - NK_1u, k_1)
\]

(13.3.8)

\[
S \to N, \quad w = \text{cd}(juK', k), \quad F_N(w) = \text{cd}(juK_1, k_1) = 1/ \text{cd}(uK_1', k_1)
\]

\[
N \to D, \quad w = \text{cd}(Ku + jK', k), \quad F_N(w) = \text{cd}(NK_1u + jK_1', k_1) = \left[ k_1 \text{cd}(NK_1u, k_1) \right]^{-1}
\]

(13.3.9)

For the path \( C \to S, F_N(w) \) is equiripple and bounded by \( |F_N(w)| \leq 1 \). For the path \( S \to N \), we have, using the degree equation (13.3.3):

\[
w = \text{cd}(juK'/K, k) \Rightarrow F_N(w) = \text{cd}(jNK_1(juK'/K, k_1)) = \text{cd}(juK_1', k_1) = \frac{1}{\text{cd}(uK_1', k_1)}
\]

and therefore, \( F_N(w) \) is an increasing function taking the values \( 1 \leq |F_N(w)| \leq 1/k_1 \). Finally, for \( N \to D \), we have, using \( w = \text{cd}(Ku + jK', k) = \text{cd}((u + jK'/K)K, k) \) and the degree equation:

\[
F_N(w) = \text{cd}(jNK_1(u + jK'/K, k_1)) = \text{cd}(NK_1u + jK_1', k_1) = \frac{1}{k_1 \text{cd}(NK_1u, k_1)}
\]

Thus, the inverse \( 1/F_N(w) = k_1 \text{cd}(NK_1u, k_1) \) is equiripple and remains bounded in the interval, \( |1/F_N(w)| \leq 1/k_1 \). 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,

\[
\cos \left( \frac{N\pi r}{2} \right) = \sum_{i=1}^{L} \frac{w^2 - \zeta_i^2}{1 - \zeta_i^2}, \quad i = 1, 2, \ldots, L
\]

resulting in the \( N \)th degree polynomial \( C_N(w) \):

\[
C_N(w) = [w]^r \prod_{i=1}^{L} \frac{w^2 - \zeta_i^2}{1 - \zeta_i^2}
\]

(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 (13.3.10) with \( i = 1 \) and \( i = N \):

\[
F_N(w) = \cd (uK, k) = \cd (uK + 2K, k) = \cd ((u + 2)K, k)
\]

\[
F_N(-w) = \cd (NuK, k_1) = \cd (NuK_1 + 2NK_1, k_1)
\]

\[
= (-1)^N \cd (NuK_1, k_1) = (-1)^N F_N(w)
\]

The zeros of \( F_N(w) \) are obtained by solving the equation, \( \cd (NuK_1, k_1) = 0 \). It follows from Eq. (13.2.15) that:

\[
\cd (NuK_1, k_1) = 0 \Rightarrow NuK_1 = (2i - 1)K_1 \quad \text{or},
\]

\[
u_i = \frac{2i - 1}{N}, \quad i = 1, 2, \ldots, L
\]

(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, \( w_i = \cd (u_i K, k) \), and we denote them by:

\[
\zeta_i = \cd (u_i K, k), \quad i = 1, 2, \ldots, L
\]

(13.3.13)

Because of the relationship \( F_N(k^{-1}w^{-1}) = [k_1 F_N(w)]^{-1} \), the frequencies \( w_i = (k_i \zeta_i)^{-1} \) 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 \( F_N(1) = 1 \):

\[
F_N(w) = [w]^r \prod_{i=1}^{L} \frac{1 - k^2 \zeta_i^2}{1 - \zeta_i^2}
\]

(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 \( N, k \):

\[
k_1^{-1} = [k^{-1}]^r \prod_{i=1}^{L} \frac{1 - k^2 \zeta_i^2}{1 - \zeta_i^2}
\]

(13.3.15)

where we used the property of the modular equations—known as the modular equations—derived 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].

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 moduli \( q, q_1 \), corresponding to the moduli \( k, k_1 \), defined by:

\[
q = e^{-\pi k'/k}, \quad q_1 = e^{-\pi k_1'/k_1}
\]

(13.3.17)

Exponentiating the degree equation (13.3.3), we have:

\[
q_1 = q^N \quad \text{or} \quad q = q_1^{1/N}
\]

(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:

\[
k = 4\sqrt{q} \left[ \sum_{m=0}^{\infty} q^m \frac{m+1}{1 + 2 \sum_{m=1}^{\infty} q^m} \right]^2
\]

(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 \( w = \text{cd}(z, k) \) and \( w = \text{sn}(z, k) \) 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 \( k_0 = k \):

\[
k_n = \left( \frac{k_{n-1}}{1 + k_{n-1}} \right)^2, \quad n = 1, 2, \ldots, M
\]

(13.4.1)

where \( k_{n-1} = (1 - k_n^2)^{1/2} \). 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 \( 0 \leq k \leq 0.999 \), the recursion may be stopped at \( M = 5 \), with all subsequent \( k_n \) being smaller than \( 10^{-15} \), while for \( k \leq 0.99 \), 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:

\[
k_{n-1} = \frac{2\sqrt{k_n}}{1 - k_n}, \quad n = M, M-1, \ldots, 1
\]

(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 \( k_n \):

\[
K_{n-1} = (1 + k_n)K_n
\]

(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:

\[
K = (1 + k_1)(1 + k_2) \cdots (1 + k_M)K_M, \quad K_M = \frac{\pi}{2}
\]

(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 \( 0 \leq k \leq k_{\text{max}} \), where \( k_{\text{max}} = (1 - k_{\text{min}}^2)^{1/2} \), with \( k_{\text{min}} = 10^{-6} \). For \( k \) in the range \( k_{\text{max}} < k \leq 1 - \epsilon \), where \( \epsilon \) is the machine epsilon, one may use the expansion:

\[
K = \frac{L}{2} + \left( \frac{L - 1}{2} \right) k^2, \quad L = -\ln \left( \frac{k}{2} \right), \quad k' = (1 - k^2)^{1/2}
\]

(13.4.5)
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:

\[
H_a(s) = H_0 \left[ \frac{1}{1 - s/P_{ai}} \right] \prod_{i=1}^{L} \left[ \frac{1 + B_{i1}s + B_{i2}s^2}{1 + A_{i1}s + A_{i2}s^2} \right]^{\epsilon treatment value}\]

(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 \([F]^t\) 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:

\[
H_0 = \begin{cases} 
1, & \text{Butterworth and Chebyshev-2} \\
G_{p,L}^{-1/2}, & \text{Chebyshev-1 and Elliptic} 
\end{cases}
\]

(13.5.2)

where \(G_p = (1 + \epsilon_p^2)^{-1/2}\) is the passband gain. The variable \(s\) must be replaced by \(s = j\Omega = j\omega_0\) to get the filter’s frequency response. Multiplying the second-order factors, we may write the transfer function in the form:

\[
H(s) = H_0 \left[ \frac{1}{1 - s/A_{i1}s} \right] \prod_{i=1}^{L} \left[ \frac{1 + B_{i1}s + B_{i2}s^2}{1 + A_{i1}s + A_{i2}s^2} \right]^{\epsilon treatment value}
\]

(13.5.3)

where the numerator and denominator coefficients are given by

\[
\begin{align*}
[1, B_{i1}, B_{i2}] &= \left[ 1, -2 \text{Re}\left( \frac{1}{z_{ai}} \right), \frac{1}{|z_{ai}|^2} \right] \\
[1, A_{i1}, A_{i2}] &= \left[ 1, -2 \text{Re}\left( \frac{1}{p_{ai}} \right), \frac{1}{|p_{ai}|^2} \right] \\
[1, A_{01}] &= \left[ 1, -\frac{1}{P_{a0}} \right]
\end{align*}
\]

(13.5.4)

Because the magnitude response corresponding to Eq. (13.5.1) is given by

\[
|H(\Omega)|^2 = \frac{1}{1 + \epsilon_p^2 F_N(\omega)} , \quad \omega = \Omega \Omega_p,
\]

(13.5.5)

it follows that the zeros \(z_{ai}\) will arise from the poles of \(F_N(\omega)\), and the poles \(p_{ai}\) will arise from the zeros of the denominator, that is, \(1 + \epsilon_p^2 F_N(\omega) = 0\). We saw in the previous section that the poles of \(F_N(\omega)\) occur at the normalized frequencies \(\omega_i = (k\xi_i)^{-1}\). Therefore, taking into account the normalization factor \(\Omega_p\), the denormalized \(s\)-plane zeros \(z_{ai} = \Omega_p(j\omega_i)\) will be:

\[
z_{ai} = \Omega_p(jk\xi_i)^{-1}, \quad i = 1, 2, \ldots, L \quad \text{(s-plane zeros)}
\]

(13.5.6)

The poles \(p_{ai}\) are found by solving the equation: \(1 + \epsilon_p^2 F_N^2(\omega) = 0\), or,

\[
F_N(\omega) = \pm j \frac{1}{\epsilon_p}
\]

(13.5.7)

The complex-frequency solutions \(\omega_i\) of (13.5.7) determine the denormalized poles by setting, \(p_{ai} = \Omega_p j\omega_i\). The resulting left-hand \(s\)-plane poles \(p_{ai}\) are found to be:

\[
p_{ai} = \Omega_p j\text{cd}(u_i - j\nu_0)K,k), \quad i = 1, 2, \ldots, L\]

(13.5.8)

where the \(u_i\) are the same as in Eq. (13.3.12), and \(\nu_0\) is the real-valued solution of the equation:

\[
\text{sn}(j\nu_0NK_1,k_1) = j \frac{1}{\epsilon_p} \Rightarrow \nu_0 = -\frac{j}{NK_1} \text{sn}^{-1} \left( \frac{1}{\epsilon_p}, k_1 \right)
\]

(13.5.9)

As noted earlier in Fig. 13.2.6, the bottom two quadrants of the fundamental rectangle on the \(z\)- or \(s\)-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 \(u_1 = 1\) (which corresponds to the index \(i = L + 1\)):

\[
p_{a0} = \Omega_p j\text{cd}(1 - j\nu_0)K,k)
\]

(13.5.10)

To verify that \(w_i = \text{cd}(u_i - j\nu_0)K,k)\) 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:

\[
F_N(w_i) = \text{cd}(u_i - j\nu_0)NK_1,k_1)
\]

(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.

\[
\begin{align*}
fp &= 4; \quad fs = 4.5; \quad Gp = 0.95; \quad Gs = 0.05; \quad \% \text{filter specifications} \\
wp &= 2*\pi*fp; \quad ws = 2*\pi*fs; \\
ep &= \text{sqrt}(0.5); \quad es = \text{sqrt}(0.5) \quad \% \text{ripples} \quad \epsilon_p = 0.3287, \quad \epsilon_s = 0.09750 \\
k &= wp/ws; \quad \% k = 0.88888 \\
k1 &= ep/es; \quad \% k_1 = 0.01454 \\
[K,Kp] &= \text{ellipk}(k); \quad \% \text{elliptic integrals} \quad K = 2.2353, \quad K' = 1.6646 \\
[K1,K1p] &= \text{ellipk}(k1); \quad \% K = 1.5709, \quad K' = 5.4937 \\
Nexact &= (K1p/K1)/(Kp/K); \quad N = \text{ceil}(Nexact); \quad \% \text{Nexact} = 4.8604, \quad N = 5
\end{align*}
\]
13.6. Design Example

Elliptic filter design

The calculated zeros and poles of the filter are, for \( N = 5 \) and \( L = 2 \):

\[
\begin{align*}
    z_1 &= 28.0265j, & p_0 &= -15.1717 \\
    z_2 &= 36.7945j, & p_1 &= -1.0115 + 25.4353j \\
    z_3 &= -1.0115 - 25.4353j, & p_2 &= -6.2951 + 21.4113j
\end{align*}
\]

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:

\[
B = \begin{bmatrix}
1 & 0 & 0 \\
1 & 0.00127 & 0.00154 \\
1 & 0.00794 & 0.00201
\end{bmatrix}, \quad A = \begin{bmatrix}
1 & 0.06591 & 0 \\
1 & 0.00312 & 0.00154 \\
1 & 0.02528 & 0.00201
\end{bmatrix}
\]

Thus, the transfer function will be, with \( s = 2\pi jf \):

\[
H(s) = \frac{1}{1 + 0.06591 s} \frac{1 + 0.00127 s^2}{1 + 0.00794 s^2} \frac{1 + 0.00074 s^2}{1 + 0.02528 s + 0.00201 s^2}
\]

The following function `ellipap2.m` incorporates the above design steps and serves as a substitute for MATLAB's built-in function `ellipap`.

The filter order was determined by calculating the exact value of \( N \) that satisfies the degree equation (13.3.3), that is, \( \frac{N}{k_1} = \frac{K_1}{K} \), 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.
13.7 BUTTERWORTH AND CHEBYSHEV DESIGNS

Filter. Indeed, since \( \Omega_0^2N = \Omega_p^2N / \varepsilon_p^2 \), the magnitude response can be written as

\[
|H(\Omega)|^2 = \frac{1}{1 + \varepsilon_p^2 \Omega^2 / \varepsilon_p^2} = \frac{1}{1 + \left( \frac{\Omega}{\Omega_0} \right)^{2N}} = \frac{1}{1 + \left( \frac{\Omega}{\Omega_0} \right)^{2N}}.
\]

In the Chebyshev-1 case, we have:

\[
p_a i = \Omega_p j \cos\left((u_i - jv_0)\pi/2\right), \quad i = 1, 2, \ldots, L
\]

\[
p_a 0 = \Omega_p j \cos\left((1 - jv_0)\pi/2\right) = -\Omega_p \sinh(v_0\pi/2)
\]

where \( v_0 \) is obtained from the solution of:

\[
\sinh(Nv_0\pi/2) = \frac{1}{\varepsilon_p} \Rightarrow v_0 = \frac{\sinh(1/\varepsilon_p)}{N\pi/2}
\]

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,

\[
|H(\Omega)|^2 = \frac{1}{1 + \varepsilon_p^2 k_1^{-2} / C_N^{-1} (k^{-1}w^{-1})} = \frac{C_N^{-1} (k^{-1}w^{-1})}{C_N^{-1} (k^{-1}w^{-1}) + \varepsilon_p^2 k_1^{-2}}
\]

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 \( w \rightarrow k^{-1}w^{-1} \):

\[
k^{-1} z_{ai}^{-1} = \Omega_p^{-1} j \cos\left((u_i - jv_0)\pi/2\right), \quad i = 1, 2, \ldots, L
\]

\[
k^{-1} p_{ai}^{-1} = \Omega_p^{-1} j \cos\left((u_i - jv_0)\pi/2\right), \quad i = 1, 2, \ldots, L
\]

\[
k^{-1} p_{a0} = -\Omega_p^{-1} j \cos\left((1 - jv_0)\pi/2\right)
\]

where \( v_0 \) is the solution of the equation:

\[
\sinh(Nv_0\pi/2) = \varepsilon_p k_1^{-1} = \varepsilon_s \Rightarrow v_0 = \frac{\sinh(\varepsilon_s)}{N\pi/2}
\]

13.7.3 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 \( k_1, k_1 \) 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:

\[
1 + \varepsilon_p^2 F_N^2(w) = 0 \Rightarrow F_N^2(w) = -\frac{1}{\varepsilon_p}
\]

(13.7.1)

As in the elliptic case, we define the quantities:

\[
r = \text{mod}(N, 2), \quad L = \frac{N - r}{2}, \quad u_i = \frac{2i - 1}{N}, \quad i = 1, 2, \ldots, L
\]

(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 \( u_i = 1 \)):

\[
p_{ai} = \Omega_p \varepsilon_p^{-1/N} j^e^{u_i\pi/2}, \quad i = 1, 2, \ldots, L
\]

\[
p_{a0} = \Omega_p \varepsilon_p^{-1/N} j^{\pi/2} = -\Omega_p \varepsilon_p^{-1/N}
\]

(Butterworth)

(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.
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)

% selectivity and discrimination parameters
k = Wp/Ws; k1 = ep/es;
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);
\% 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*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);
p0 = -Wp * sinh(v0*pi/2);
\% case 2
v0 = asinh(es) / (N*pi/2);
za = Wp ./ (j*k*cos(u*pi/2));
p0 = -Wp / (k*sinh(v0*pi/2));
\% case 3
v0 = -j * asne(j/ep, k1) / N;
za = Wp ./ (j*k*cos(u-v0)*pi/2);
p0 = -Wp / (k*sinh(v0*pi/2));
end
B = [ones(L+1,1), zeros(L+1,2)];
A = [ones(L,1), -2*real(1/pa), abs(1/pa).^2];
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,
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:

```matlab
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);

s = j*(1:1000); % s-domain
H = ones(len(s));
H = H .* (B(1,1) + B(1,2)*s + B(1,3)*s.^2) ./(A(1,1) + A(1,2)*s + A(1,3)*s.^2);
plot(s,abs(H),'r');xtick(0:1:10); ylim([0,1.1]); ytick(0:0.1:1); grid off;line([0,fp],[1,1]); line([fp,fp],[1,1]); line([0,fp],[Gp,Gp]); line([fp,fs],[Gp,0]); line([fs,10],[Gs,Gs]); line([fs,fs],[Gp,Gp]);
```

**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:

\[
s' = F(s)
\]
where

\[ \Omega'_{s1} = \Omega_{s1} - \frac{\Omega_0^2}{\Omega_{s1}}, \quad \Omega'_{s2} = \Omega_{s2} - \frac{\Omega_0^2}{\Omega_{s2}} \]  

(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 \( \Omega \) to the lowpass frequency \( \Omega' \):

\[ \Omega' = \Omega - \frac{\Omega_0^2}{\Omega} \]

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:

\[ \Omega'_p = \Omega_{p2} - \Omega_{p1}, \quad \Omega'_0 = \Omega_{p1}\Omega_{p2} \]

Once \( \Omega_0 \) 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, \( \Omega'_s \) 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 \( \Omega_0 \) are selected on the basis of the stopband interval:

\[ \Delta\Omega = \Omega_{s2} - \Omega_{s1}, \quad \Omega_0 = \sqrt{\Omega_{s1}\Omega_{s2}} \]  

(13.8.8)

Then, the corresponding LP parameters are:

\[ s' = \frac{1}{s + \frac{\Omega_0^2}{s}}, \quad \Omega'_p = \max(|\Omega'_{p1}|, |\Omega'_{p2}|), \quad \Omega'_s = \frac{1}{\Delta\Omega} \]  

(13.8.9)

where

\[ \Omega'_{p1} = \frac{1}{\Omega_{p1} - \frac{\Omega_0^2}{\Omega_{p1}}}, \quad \Omega'_{p2} = \frac{1}{\Omega_{p2} - \frac{\Omega_0^2}{\Omega_{p2}}} \]  

(13.8.10)

In all three cases, once the equivalent frequencies \( \Omega'_p, \Omega'_s, \Delta\Omega \) 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. 13.8.1 shows some design examples. To clarify the design steps, the following code fragment implements the elliptic bandpass example:

```python
Wp1 = 2*pi*fp1;
Wp2 = 2*pi*fp2;
K = Wp1/Wp2;
K1 = Wp1/Ws;

f0 = sqrt(Wp1/Ws);
eps = Wp1/f0;

k = eps/K;
eps1 = eps/k;
eps2 = eps/k;
```

For highpass designs, define:

\[ s' = \frac{1}{s}, \quad \Omega'_p = \frac{1}{\Omega_p}, \quad \Omega'_s = \frac{1}{\Omega_s} \]  

(13.8.4)

For bandpass designs, the bandwidth and center frequency of the passband are:

\[ \Delta\Omega = \Omega_{p2} - \Omega_{p1}, \quad \Omega_0 = \sqrt{\Omega_{p1}\Omega_{p2}} \]  

(13.8.5)

Then, the corresponding LP parameters are:

\[ s' = s + \frac{\Omega_0^2}{s}, \quad \Omega'_p = \Delta\Omega, \quad \Omega'_s = \min(|\Omega'_{s1}|, |\Omega'_{s2}|) \]  

(13.8.6)
13.8. HIGHPASS, BANDPASS, AND BANDSTOP ANALOG FILTERS

\[ W_s1 = 2\pi f_s1; \]
\[ W_s2 = 2\pi f_s2; \]
\[ G_p = 0.95; \]
\[ G_s = 0.05; \]
\[ A_p = -20\log_{10}(G_p); \]
\[ A_s = -20\log_{10}(G_s); \]
\[ W_p = W_p2 - W_p1; \]
\[ W_0 = \sqrt{W_p1 \cdot W_p2}; \]
\[ W_1 = W_s1 - W_0^2/W_s1; \]
\[ W_2 = W_s2 - W_0^2/W_s2; \]
\[ W_s = \min(\text{abs}(\{W_1,W_2\})); \]
\[ \text{type} = 3; \]
\[ [N,B,A] = \text{lpa}(W_p,W_s,A_p,A_s,\text{type}); \]
\[ f = \text{linspace}(0,10,1001); \]
\[ s = j*2\pi f/s; \]
\[ s = s + W_0^2/s; \]
\[
\text{figure; plot}(f,abs(H),'r'); \text{xtick}(0:1:10); \text{ylim}([0,1.1]); \text{ytick}(0:0.1:1); \text{grid off;}
\]
\[ \text{line}([fpl,fpl],[1,1],'	ext{'LineStyle'}','-'); \]
\[ \text{line}([fpl,fpl],[1,1.05],'	ext{'LineStyle'}','-'); \]
\[ \text{line}([fp2,fp2],[1,1.05],'	ext{'LineStyle'}','-'); \]
\[ \text{line}([fpl,fp1],[Gp,Gp]); \text{line}([fpl,fp1],[Gp,0]); \text{line}([fp2,fp2],[Gp,0]); \]
\[ \text{line}([0,fs1],[Gs,Gs]); \text{line}([fs1,fs1],[4*Gs,Gs]); \]
\[ \text{line}([fs2,10],[Gs,Gs]); \text{line}([fs2,fs2],[4*Gs,Gs]); \]

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 \( s = f(z) \):

\[
H(z) = H_a(s) \bigg|_{s = f(z)} = H_a(f(z)) \tag{13.9.1}
\]

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 \( \Omega \) is the equivalent analog frequency and \( \omega = 2\pi f/f_s \), the digital frequency, are as follows:

- **Lowpass (LP)**
  \[
s = \frac{1 - z^{-1}}{1 + z^{-1}}, \quad \Omega = \tan \left( \frac{\omega}{2} \right)
\]

- **Highpass (HP)**
  \[
s = \frac{1 + z^{-1}}{1 - z^{-1}}, \quad \Omega = -\cot \left( \frac{\omega}{2} \right)
\]

- **Bandpass (BP)**
  \[
s = \frac{1 - 2c_0z^{-1} + z^{-2}}{1 - z^{-2}}, \quad \Omega = \frac{c_0 - \cos \omega}{c_0 - \cos \omega}
\]

- **Bandstop (BS)**
  \[
s = \frac{1 - z^{-2}}{1 - 2c_0z^{-1} + z^{-2}}, \quad \Omega = -\frac{\sin \omega}{c_0 - \cos \omega}
\]

where \( c_0 = \cos \omega_0 \), with \( \omega_0 \) 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:

\[
H_{lp}(z) = H_a(s) \bigg|_{s = \frac{1 - z^{-1}}{1 + z^{-1}}} = H_a \left[ \frac{1}{1 - s/p_{ai}} \right]^{L} \prod_{i=1}^{L} \left[ \frac{1 - s/z_{ai}}{1 - s/p_{ai}} \right]^{L} \bigg|_{s = \frac{1 - z^{-1}}{1 + z^{-1}}} \tag{13.10.2}
\]

After mapping the analog \( s \)-plane poles and zeros \( \{p_i, z_{ai}\} \) 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:

\[
H_{lp}(z) = H_a \left[ \frac{1 + z^{-1}}{1 - p_0z^{-1}} \right]^{L} \prod_{i=1}^{L} \left| G_i \right|^2 \left[ \frac{1 - z_iZ^{-1} \left( 1 - z_i^2Z^{-1} \right)}{(1 - p_iZ^{-1} \left( 1 - p_i^2Z^{-1} \right))} \right]^{L} \tag{13.10.3}
\]

Noting that the inverse of Eq. (13.10.1) is

\[
\frac{s}{1 - s} = \frac{1 + s}{1 + s}
\]

the digital poles and zeros are computed by:

\[
p_0 = \frac{1 + p_{ai}}{1 - p_{ai}}, \quad p_i = \frac{1 + p_{ai}}{1 - p_{ai}}, \quad z_i = \frac{1 + z_{ai}}{1 - z_{ai}}, \quad i = 1, 2, \ldots, L \tag{13.10.4}
\]

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:

\[
\frac{1 - s/z_{ai}}{1 - s/p_{ai}} = G_i \left( \frac{1 - z_i^{-1}}{1 - p_i^{-1}} \right), \quad \frac{1 - s/p_{ai}}{1 - s/z_{ai}} = G_i \left( \frac{1 - z_i^{-1}}{1 - p_i^{-1}} \right) \tag{13.10.6}
\]

The zeros of the Butterworth and Chebyshev-1 cases are simply \( z_i = -1 \) as follows by setting \( z_{ai} = -1 \) 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 \( z^{-1} \rightarrow -z^{-1} \) in the lowpass case. Replacing the lowpass poles and zeros \( \{p_i, z_i\} \) by their negatives, one obtains the highpass transfer function:

\[
H_{hp}(z) = H_0 \left[ \frac{1}{1 - s/p_{ai}} \right]^{L} \prod_{i=1}^{L} \left[ \left( 1 - s/z_{ai} \right) \left( 1 - s/p_{ai} \right) \right]^{L} \bigg|_{s = \frac{1 - z^{-1}}{1 + z^{-1}}} \tag{13.10.8}
\]

where the highpass digital poles and zeros are now given by:

\[
p_0 = \frac{1 + p_{ai}}{1 - p_{ai}}, \quad p_i = \frac{1 + p_{ai}}{1 - p_{ai}}, \quad z_i = \frac{1 + z_{ai}}{1 - z_{ai}}, \quad i = 1, 2, \ldots, L \tag{13.10.9}
\]

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:

\[
\begin{array}{cccc}
\text{LP analog} & \text{LP digital} & \text{HP/HP digital} \\
H_a(s) & H_0 & H(z) \\
\end{array}
\tag{13.10.11}
\]

For bandpass designs, the required transformations take the two-step form:

\[
s = \frac{1 - z^{-1}}{1 + z^{-1}} = \frac{1 - 2c_0z^{-1} + z^{-2}}{1 - z^{-2}}, \quad z^{-1} = \frac{z^{-1}(c_0 - z^{-1})}{1 - c_0z^{-1}} \tag{13.10.12}
\]
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 \( H_{BP}(z) \) by the mapping \( \hat{z} \to z \), that is, the transfer functions are related by:

\[
H_{BP}(z) = \hat{H}(\hat{z}) \bigg|_{\hat{z} = z^{-1}c_0^{-1}z^{-1}} = H_a(s) \bigg|_{s = \frac{1 - z^{-1}}{1 - c_0z^{-1}}} \tag{13.10.13}
\]

For bandpass 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:

\[
H_{BS}(z) = \hat{H}(\hat{z}) \bigg|_{\hat{z} = z^{-1}c_0z^{-1}} = H_a(s) \bigg|_{s = \frac{1 - z^{-1}}{1 + c_0z^{-1}}} \tag{13.10.15}
\]

The bandpass and bandstop cases can be combined into one by defining:

\[
s = \frac{1 - \hat{z}^{-1}}{1 + \hat{z}^{-1}}, \quad \hat{z}^{-1} = \frac{q^{-\frac{1}{2}}(c_0 - z^{-1})}{1 - c_0z^{-1}}, \quad q = \begin{cases} +1, & \text{BP case} \\ -1, & \text{BS case} \end{cases} \tag{13.10.16}
\]

The transfer functions will be related by:

\[
H(z) = \hat{H}(\hat{z}) \bigg|_{\hat{z} = z^{-1}c_0z^{-1}} = H_a(s) \bigg|_{s = \frac{1 - z^{-1}}{1 + c_0z^{-1}}} \tag{13.10.17}
\]

Setting, \( s = j\Omega, \hat{z} = e^{j\omega} \), and, \( z = e^{j\omega} \), we find the corresponding frequency relationships:

\[
\Omega = \tan \left( \frac{\omega}{2} \right) = \begin{cases} \frac{c_0 - \cos \omega}{\sin \omega}, & \text{if } q = 1 \\ -\frac{\sin \omega}{c_0 - \cos \omega}, & \text{if } q = -1 \end{cases} \tag{13.10.18}
\]

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):

\[
p_{ai} = \frac{1 - \hat{p}_i}{1 + \hat{p}_i} \Rightarrow \hat{p}_i = \frac{1 + p_{ai}}{1 - p_{ai}} \tag{13.10.19}
\]

then, the poles pairs \( p_i^+ \) are obtained as the two solutions for \( p_i \) of the equation:

\[
p_i = q \frac{p_i(c_0 - p_i)}{1 - c_0p_i} \Rightarrow p_i = \frac{1}{2} \left[ c_0(1 + q\hat{p}_i) \pm \sqrt{c_0^2(1 + q\hat{p}_i)^2 - 4q\hat{p}_i} \right] \tag{13.10.20}
\]

Because \( \hat{z} \) remains invariant under the substitution \( z \to \hat{z}' = (c_0 - z)/(1 - c_0z) \), that is,

\[
\hat{z} = \frac{q' (c_0 - z)}{1 - c_0z}, \quad \text{with } \hat{z}' = \frac{c_0 - z}{1 - c_0z}, \quad z = \frac{c_0 - \hat{z}'}{1 - c_0\hat{z}'} \tag{13.10.21}
\]

it follows that the pair \( p_i, p_i^- \) can be constructed, alternatively, by:

\[
p_i^+ = \frac{1}{2} \left[ c_0(1 + q\hat{p}_i) + \sqrt{c_0^2(1 + q\hat{p}_i)^2 - 4q\hat{p}_i} \right], \quad p_i^- = \frac{c_0 - p_i^+}{1 - c_0p_i^+} \tag{13.10.22}
\]

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:

\[
\frac{1 - s/z_{ai}}{1 - s/p_{ai}} = G_i \frac{1 - \hat{z}_i}{1 - \hat{p}_i} = G_i \left( \frac{1 - z_i^- z_i^+}{1 - p_i^- p_i^+} \right), \quad G_i = 1 - \hat{p}_i \tag{13.10.23}
\]

It follows that the transfer function can be expressed in the equivalent factored forms:

\[
H(z) = H_0 \prod_{i=1}^{L} \left[ \frac{1 - s/z_{ai}}{1 - s/p_{ai}} \right] = H_0 \prod_{i=1}^{L} \left[ \frac{(1 - z_i^- z_i^+)}{(1 - p_i^- p_i^+)} \right] \tag{13.10.24}
\]

The poles \( p_{ai}^0 \) arise from the mapping of the analog pole \( p_{a0} \):

\[
p_0 = \frac{1 + p_{a0}}{1 - p_{a0}} \Rightarrow p_{0i}^0 = \frac{1}{2} \left[ c_0(1 + q\hat{p}_0) \pm \sqrt{c_0^2(1 + q\hat{p}_0)^2 - 4q\hat{p}_0} \right] \tag{13.10.25}
\]

Because \( p_{a0} \) is real, \( p_{0i}^0 \) is either both real-valued or conjugate pairs. The zeros \( z_{0i}^0 \) arise from mapping the analog zero \( z_{a0} = \infty \) to \( z_0 = -1 \), and are given in the two cases \( q = \pm 1 \) by:

\[
z_0 = -1 \Rightarrow z_{0i}^0 = \frac{1}{2} \left[ c_0(1 + q\hat{z}_0) \pm \sqrt{c_0^2(1 + q\hat{z}_0)^2 - 4q\hat{z}_0} \right] = \begin{cases} \pm 1, & \text{if } q = 1 \\ e^{j\pi}, & \text{if } q = -1 \end{cases} \tag{13.10.26}
\]

Thus, the corresponding numerator second-order factor becomes:

\[
(1 - z_{0i}^0 z_i^-)(1 - z_{0i}^0 z_i^+) = \begin{cases} 1 - z^{-2}, & \text{if } q = 1 \\ 1 - 2c_0z^{-1} + z^{-2}, & \text{if } q = -1 \end{cases} \tag{13.10.27}
\]
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:

\[
H(z) = H_0 \left[ \prod_{i=1}^L \frac{|G_i|^2 (1 - z_i^- z^{-1}) (1 - z_i^+ z^{-1}) (1 - z_i^- z^{-1}) (1 - z_i^+ z^{-1})}{(1 - p_i^- z^{-1})(1 - p_i^+ z^{-1})(1 - p_i^- z^{-1})(1 - p_i^+ z^{-1})} \right].
\]

(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 \( z = \pm z \). 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, \( p_i^- = z_i^- = \pm 1 \).

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 \( z \)-plane unit circle, to poles inside the \( z \)-plane unit circle. These follow from the relationships:

\[
\text{Re}(s) = \frac{1 - |z|^2}{|z + 1|^2}, \quad 1 - |z|^2 = (1 - |z|^2) \frac{|c_0 - z|^2 + s_0}{|1 - c_0 z|^2}, \quad s_0 = \sin \omega_0
\]

(13.10.29)

which show that \( \text{Re}(s) \leq 0 \) is equivalent to \( |z| \geq 1 \) and to \( |z| \leq 1 \).

### 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:

\[
\Omega_p = \tan \left( \frac{\omega_{\text{pass}}}{2} \right), \quad \Omega_s = \tan \left( \frac{\omega_{\text{stop}}}{2} \right)
\]

(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:

\[
\Omega_p = \cot \left( \frac{\omega_{\text{pass}}}{2} \right), \quad \Omega_s = \cot \left( \frac{\omega_{\text{stop}}}{2} \right)
\]

(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:

\[
\omega_{p1} = 2 \pi f_{p1} / f_s, \quad \omega_{p2} = 2 \pi f_{p2} / f_s, \quad \omega_{s1} = 2 \pi f_{s1} / f_s, \quad \omega_{s2} = 2 \pi f_{s2} / f_s
\]

\[
\cos \omega_0 = \frac{\sin (\omega_{p1} + \omega_{p2})}{\sin \omega_{p1} + \sin \omega_{p2}}, \quad \Omega_{p1} = \frac{\cos \omega_0 - \cos \omega_{s2}}{\sin \omega_{s1}}, \quad \Omega_{s1} = \frac{\cos \omega_0 - \cos \omega_{s1}}{\sin \omega_{s2}}
\]

\[
\Omega_p = \tan \left( \frac{\omega_{p2} - \omega_{p1}}{2} \right), \quad \Omega_s = \tan \left( \frac{\omega_{s2} - \omega_{s1}}{2} \right)
\]

(13.11.3)

These choices match the passband specifications. An alternative choice that matches the stopband specifications is as follows:

\[
\cos \omega_0 = \frac{\sin (\omega_{s1} + \omega_{s2})}{\sin \omega_{s1} + \sin \omega_{s2}}, \quad \Omega_{p1} = \frac{\cos \omega_0 - \cos \omega_{p1}}{\sin \omega_{p1}}, \quad \Omega_{p2} = \frac{\cos \omega_0 - \cos \omega_{p2}}{\sin \omega_{p2}}
\]

\[
\Omega_p = \max (|\Omega_{p1}|, |\Omega_{p2}|), \quad \Omega_s = \tan \left( \frac{\omega_{s2} - \omega_{s1}}{2} \right)
\]

(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:

\[
\omega_{p1} = 2 \pi f_{p1} / f_s, \quad \omega_{p2} = 2 \pi f_{p2} / f_s, \quad \omega_{s1} = 2 \pi f_{s1} / f_s, \quad \omega_{s2} = 2 \pi f_{s2} / f_s
\]
\[
\cos \omega_0 = \frac{\sin(\omega_p^1 + \omega_p^2)}{\sin \omega_p^1 + \sin \omega_p^2}, \quad \Omega_{s1} = \frac{\sin \omega_1}{\cos \omega_0 - \cos \omega_1}, \quad \Omega_{s2} = \frac{\sin \omega_2}{\cos \omega_0 - \cos \omega_2}
\]
\[
\Omega_p = \cot\left(\frac{\omega_p^2 - \omega_p^1}{2}\right), \quad \Omega_s = \min(|\Omega_{s1}|, |\Omega_{s2}|)
\]

Alternatively, we may match the stopband specifications:

\[
\cos \omega_0 = \frac{\sin(\omega_s^1 + \omega_s^2)}{\sin \omega_s^1 + \sin \omega_s^2}, \quad \Omega_{p1} = \frac{\sin \omega_1}{\cos \omega_0 - \cos \omega_1}, \quad \Omega_{p2} = \frac{\sin \omega_2}{\cos \omega_0 - \cos \omega_2}
\]
\[
\Omega_p = \max(|\Omega_{p1}|, |\Omega_{p2}|), \quad \Omega_s = \cot\left(\frac{\omega_s^2 - \omega_s^1}{2}\right)
\]

### 13.12 MATLAB Implementation and Examples

Once the parameters \(\Omega_p, \Omega_s\) 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:

```matlab
% 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 = 'LP', 'HP', 'BP', 'BS', For lowpass, highpass, bandpass, bandstop designs
% match = 'p', 's', if omitted or 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/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 `butter`, `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):

```matlab
% 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 2N
% 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 = 'LP', 'HP', 'BP', 'BS', For lowpass, highpass, bandpass, bandstop designs
% 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 'sos', (2L+1)x3 for 'dir', (2N+1)x1 for 'dir'
% w0 = center freq (radians/sample) of the wp 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,As,ws,3,shape,coeffs) match at wp, equiripple at Ap,As
% [B,A,w0,] = dfdes(N,Ap,As,wp,3,shape,coeffs) match at wp, equiripple at Ap,As
% [B,A,w0,] = dfdes(N,Ap,As,wp,4,shape,coeffs) match at wp, equiripple at Ap,As
% [B,A,w0,] = dfdes(N,Ap,As,wp,5,shape,coeffs) match at wp, equiripple at Ap,As
% note that N = 2L+r, r=0,1, L = floor(N/2) = number of SOSs of LP analog prototype
% for example, the elliptic designs may be generated by the code fragment:
```

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):

- **LP case:** \(f_s = 20\), \(f_{\text{pass}} = 4.0\), \(f_{\text{stop}} = 4.5\)
- **HP case:** \(f_s = 20\), \(f_{\text{pass}} = 4.5\), \(f_{\text{stop}} = 4.0\)
- **BP case:** \(f_s = 20\), \([f_{\text{p1}}, f_{\text{p2}}] = [3.0, 6.0]\), \([f_{\text{s1}}, f_{\text{s2}}] = [2.5, 6.5]\)
- **BS case:** \(f_s = 20\), \([f_{\text{p1}}, f_{\text{p2}}] = [2.5, 6.5]\), \([f_{\text{s1}}, f_{\text{s2}}] = [3.0, 6.0]\)
13.12. MATLAB IMPLEMENTATION AND EXAMPLES

\[ G_p = 0.95; \quad A_p = -20^\circ \log_{10}(G_p); \]
\[ G_s = 0.05; \quad A_s = -20^\circ \log_{10}(G_s); \]
\[ f = \text{linspace}(0,10,1001); \]
\[ w = 2\pi f / \text{fsamp}; \]
\[ \text{% -------- LP } \]
\[ [N, Ad, wd] = \text{dford}(wp, ws, Ap, As, 3,'s'); \]
\[ [B, A] = \text{dfdes}(N, Ad, wd, 3, 'LP', 'sos'); \]
\[ H = \text{fresp}(B, A, w); \]
\[ \text{figure; plot(f, abs(H), 'r');} \]

\[ \text{% -------- HP } \]
\[ [N, Ad, wd] = \text{dford}(wp, ws, Ap, As, 3,'s'); \]
\[ [B, A] = \text{dfdes}(N, Ad, wd, 3, 'HP', 'sos'); \]
\[ H = \text{fresp}(B, A, w); \]
\[ \text{figure; plot(f, abs(H), 'r');} \]

\[ \text{% -------- BP } \]
\[ [N, Ad, wd] = \text{dford}(wp, ws, Ap, As, 3, 's'); \]
\[ [B, A] = \text{dfdes}(N, Ad, wd, 3, 'BP', 'sos'); \]
\[ H = \text{fresp}(B, A, w); \]
\[ \text{figure; plot(f, abs(H), 'r');} \]

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 \text{fresp.m}, borrowed from [329]. The designed transfer function coefficients and coefficient matrices are as follows:

**LP case:**

\[
B = \begin{bmatrix}
0.3204 & 0.3204 & 0 \\
0.8591 & -0.2363 & 0.8591 \\
0.4534 & 0.1260 & 0.4534 \\
\end{bmatrix}, \quad A = \begin{bmatrix}
1 & -0.3593 & 0 \\
1 & -0.4436 & 0.9255 \\
1 & -0.5547 & 0.5821 \\
\end{bmatrix}, \quad N = 5
\]

\[
H_{LP}(z) = \frac{0.3204(1 + z^{-1}) - 0.8591z^{-1} + 0.4534z^{-2}}{1 - 0.3593z^{-1} - 0.4436z^{-2} + 0.5547z^{-3} + 0.5821z^{-4}}
\]

**HP case:**

\[
B = \begin{bmatrix}
0.4317 & -0.4317 & 0 \\
0.8986 & -0.5866 & 0.8986 \\
0.5615 & -0.6118 & 0.5615 \\
\end{bmatrix}, \quad A = \begin{bmatrix}
1 & 0.1366 & 0 \\
1 & -0.4582 & 0.9257 \\
1 & -0.1727 & 0.5621 \\
\end{bmatrix}, \quad N = 5
\]

\[
H_{HP}(z) = \frac{0.4317 + 0.8986z^{-1} + 0.5615z^{-2}}{1 - 0.4317z^{-1} - 0.5866z^{-2} + 0.6118z^{-3} + 0.5615z^{-4}}
\]

**BS case:**

\[
B = \begin{bmatrix}
0.9500 & 0 & 0 \\
0.8161 & -1.1771 & 0.8161 \\
0.4017 & -0.7171 & 0.4017 \\
\end{bmatrix}, \quad A = \begin{bmatrix}
1 & 0 & 0 \\
1 & -1.2501 & 0.9253 \\
1 & -0.8124 & 0.6129 \\
\end{bmatrix}, \quad N = 4
\]

\[
B = \begin{bmatrix}
0.9500 & 0 & 0 \\
0.9081 & -1.0417 & 0.9081 \\
0.6221 & -0.4912 & 0.6221 \\
\end{bmatrix}, \quad A = \begin{bmatrix}
1 & 0 & 0 \\
1 & -1.2399 & 0.9239 \\
1 & -1.0384 & 0.5163 \\
\end{bmatrix}, \quad N = 4
\]

The functions \text{lp}, \text{dford}, and \text{dfdes} are available in the ISP toolbox.

### 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 \(z^{-1}\) variable. These realizations are also generated by the function \text{dfdes} and have transfer function:

\[
H(z) = H_0 \left[ \frac{G_0}{1 - p_l z^{-1}} \right]^{L} \prod_{l=1}^{L} \left[ \frac{(G_l)^2}{(1 - p_l z^{-1})} \right] \left[ \frac{1}{(1 - p_l^* z^{-1})} \right]
\]

where \(\hat{z}^{-1}\) must be replaced by

\[
\hat{z}^{-1} = q \frac{z^{-1} - (c_0 z^{-1})}{1 - c_0 z^{-1}}
\]

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 \(\omega_0\). The hat-coefficients depend only on the desired bandwidth and attenuations, and not on \(\omega_0\). Thus, one can design a lowpass filter and shift it to any center frequency \(\omega_0\) transforming it into a bandpass (or bandstop) filter.

The MATLAB function \text{dfdes} calculates (with argument \text{coefs} set to ‘hso’) the hat-coefficients from the order \(N\) of the analog prototype and a prescribed frequency.
13.13. FREQUENCY-SHIFTED REALIZATIONS

Fig. 13.12.1 Digital LP, HP, BP, and BS filters.

The bandwidth \( \Delta \omega_d = \omega_{d2} - \omega_{d1} \) is used internally by digital filter design tools to calculate the analog design parameter \( \Omega_d = \tan(\Delta \omega_d/2) \). Then, \( \Omega_d \) is mapped to the passband parameter \( \Omega_p \), which is used to design the analog prototype filter.

To design a bandpass filter with a given bandwidth \( \Delta \omega_d \) and center frequency \( \omega_0 \), one may start by designing a lowpass digital filter with cutoff frequency \( \omega_{d2} = \Delta \omega_d \) matched at level \( A_d \), and then shift it to \( \omega_0 \). Since \( \omega_0 \) and \( \Delta \omega_d \) are given, the bandedge frequencies \( [\omega_{d1}, \omega_{d2}] \) cannot be independently specified, but may be calculated by

\[
\omega_{d1} = \frac{c_0 + \omega_{d2}(\Omega_d^2 + s_0^2)}{1 + \Omega_d^2}, \quad \cos \omega_{d2} = \frac{c_0 - \Omega_d(\Omega_d^2 + s_0^2)}{1 + \Omega_d^2} \tag{13.13.3}
\]

where \( \Omega_d = \tan(\Delta \omega_d/2) \), and \( c_0, s_0 = \sin \omega_0 \). 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 \( \omega_0 \) is calculated by

\[
\cos \omega_0 = \frac{\sin(\omega_{d1} + \omega_{d2})}{\sin \omega_{d1} + \sin \omega_{d2}} \tag{13.13.4}
\]
13.13. FREQUENCY-SHIFTED REALIZATIONS

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, $\omega_d = \pi - \Delta \omega_p$, 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.

The passband frequencies satisfy $f_{p1} = f_{p2} - f_{p1} = 3$ 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, $\Delta f_s = \Delta f_{stop} = 4$ kHz. The resulting filter order was $N = 6$. The hat-second-order section coefficients were:

$$
\hat{B} = \begin{bmatrix}
1 & 0 \\
0.6796 & -0.4558 & 0.6796 \\
0.4768 & -0.0352 & 0.4768 \\
0.2919 & 0.4366 & 0.2919 \\
\end{bmatrix}, \quad \hat{A} = \begin{bmatrix}
1 & 0 \\
-0.8721 & 0.7755 \\
1 & -0.4583 & 0.3767 \\
1 & -0.0335 & 0.0539 \\
\end{bmatrix}
$$

The $c_0$ parameter was equal to 0.3090. The MATLAB code used to generate Fig. 13.13.3 was:

```matlab
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,w0); % frequency response of shifted LP filter
figure; plot(f,abs(HLP),'r');
figure; plot(f,abs(HBP),'r');
```

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 $\omega_0$ 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.

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:

\[ \hat{B} = \begin{bmatrix} 0.6843 & -0.3796 & 0.6843 \\ 0.4830 & 0.0262 & 0.4830 \\ 0.3065 & 0.4749 & 0.3065 \end{bmatrix}, \quad \hat{A} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & -0.7805 & 0.7695 \\ 1 & -0.3760 & 0.3683 \\ 1 & 0.0340 & 0.0539 \end{bmatrix} \]

The MATLAB code used to generate Fig. 13.13.6 was:

```matlab
Lp, Cheby-2, matched passband
BP, shifted to \([f_{p1},f_{p2}] = [2,6]\)

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:

\[ \hat{B} = \begin{bmatrix} 0.8043 & 0.9141 & 0.8043 \\ 0.6460 & 0.9598 & 0.6460 \\ 0.5565 & 1.0698 & 0.5565 \end{bmatrix}, \quad \hat{A} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0.7548 & 0.7680 \\ 1 & 0.8176 & 0.4342 \\ 1 & 0.9320 & 0.2508 \end{bmatrix} \]

The 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:

\[ f_0 = 3.2982 \text{ kHz}, \quad c_0 = 0.5095, \quad [f_{p1},f_{p2}] = [1.6476, 5.6476] \text{kHz} \]

\[ \Delta f_p = 4, \quad \Delta f_s = 3 \]

\[ [N,Ad,wd] = dford(wp,ws,Ap,As,2,'p'); % match passband
\]

\[ [Bh,Ah] = dfdes(N,Ad,wd,2,'LP','hsos'); % output hat-sos sections \]
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, $f_s' = 4f_s$.

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 $f_s$.

---

†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 \( f_c < f_s/2 \) and then sampled at rate \( f_s \) and quantized. This operation is shown in Fig. 14.1.2.

\[
x_a(t) \rightarrow \text{prefilter with } f_c < f_s/2 \rightarrow \text{A/D converter} \rightarrow x(f) \rightarrow f_s
\]

**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 \( 1/T \)).

\[
\begin{align*}
\text{original spectrum } X_a(f) & \rightarrow X(f) \\
\text{spectral images} & \rightarrow f_s, f_s/2, -f_s/2, -f_s
\end{align*}
\]

**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 reconstruc-

\[
\begin{align*}
\text{digital signal} & \rightarrow \text{D/A converter} \\
\text{staircase analog signal} & \rightarrow \text{lowpass anti-image postfilter} \\
\text{analog output} & \rightarrow f_s < f_s/2
\end{align*}
\]

**Fig. 14.1.4** Analog reconstruction of sampled signal.

The passband of the postfilter extends up to \( f_{\text{pass}} = f_s/8 = f_s/2 \), but its stopband need only begin at \( f_{\text{stop}} = f_s/16 - f_s/8 = f_s/16 \). 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 \( f_s/4 = 4f_s/4 \). It is evident from this figure that the higher the sampling rate, the more closely the staircase

\[
\begin{align*}
\text{ideal reconstructor} & \rightarrow \text{sharp transition} \\
\text{postfilter passband} & \rightarrow f_s/8 \\
\text{postfilter stopband} & \rightarrow f_s/16 \\
\text{Nyquist interval} & \rightarrow f_s/8, f_s/4
\end{align*}
\]

**Fig. 14.1.6** Spectrum of signal resampled at high rate \( 4f_s \), and postfilter requirements.

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, \( f_s' = 4f_s \).
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 \([-\frac{f'_s}{2}, \frac{f'_s}{2}]\), 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.

A digital lowpass FIR filter with cutoff frequency \( \frac{f'_s}{8} \) 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.

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.

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.

Fig. 14.1.11 Filter calculates missing samples from the surrounding low-rate samples.
14.1. INTERPOLATION AND OVERSAMPLING

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:

\[
X = 0.75C + 0.25D \\
Y = 0.50C + 0.50D \\
Z = 0.25C + 0.75D
\] (14.1.1)

Indeed, the straight line connecting \(C\) and \(D\) is parametrized as \(C + (D - C)t/T\), for \(0 \leq t \leq T\). 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:

\[
X = -0.18B + 0.90C + 0.30D - 0.13E \\
Y = -0.21B + 0.64C + 0.64D - 0.21E \\
Z = -0.13B + 0.30C + 0.90D - 0.18E
\] (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:

\[
X = 0.10A - 0.18B + 0.90C + 0.30D - 0.13E + 0.08F \\
Y = 0.13A - 0.21B + 0.64C + 0.64D - 0.21E + 0.13F \\
Z = 0.08A - 0.13B + 0.30C + 0.90D - 0.18E + 0.10F
\] (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.](image)

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 \(L\), \(n' = nL\),

\[
x_{\text{up}}(nL) = x(n)
\] (14.2.1)

The \(L - 1\) intermediate samples between \(x_{\text{up}}(nL)\) and \(x_{\text{up}}(nL + L)\) are zero:

\[
x_{\text{up}}(nL + i) = 0, \quad i = 1, 2, \ldots, L - 1
\] (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:

\[
x_{\text{up}}(n') = \begin{cases} 
  x(n), & \text{if } n' = nL \\
  0, & \text{otherwise}
\end{cases}
\] (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 \(i = 0, 1, \ldots, L - 1\).

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.
14. INTERPOLATION, DECIMATION, AND OVERSAMPLING

14.2. INTERPOLATION FILTER DESIGN

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 $f_s'$,

\[ f_c = \frac{f_s}{2} = \frac{f_s'}{2L} \quad (14.2.4) \]

and expressed in units of the digital frequency, $\omega' = 2\pi f'/f_s'$:

\[ \omega_c' = \frac{2\pi f_c}{f_s'} = \frac{\pi}{L} \quad (14.2.5) \]

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:

\[
d(k') = \int_{-\pi}^{\pi} D(\omega') e^{j\omega'k'} d\omega' = \int_{-\pi/L}^{\pi/L} L e^{j\omega'k'} d\omega' = \frac{\sin(\pi k'/L)}{\pi k'/L} \quad (14.2.6)
\]

A causal version of the filter may be obtained by delaying it by $LM$ samples:

\[
h(n') = d(n' - LM) = \frac{\sin(\pi (n' - LM)/L)}{\pi (n' - LM)/L}, \quad n' = 0, 1, \ldots, N - 1
\]

And a windowed version is obtained by:

\[
h(n') = w(n') d(n' - LM), \quad n' = 0, 1, \ldots, N - 1
\]

where $w(n')$ is an appropriate length-$N$ window, such as a Hamming window:

\[
w(n') = 0.54 - 0.46 \cos \left( \frac{2\pi n'}{N - 1} \right), \quad n' = 0, 1, \ldots, N - 1
\]

or a Kaiser window. The output of the ideal FIR interpolation filter is obtained by the convolution of the upsampled input $x_{up}(n')$ with the impulse response $d(k')$:

\[
y_{up}(n') = \sum_{k' = -LM}^{LM} d(k') x_{up}(n' - k'), \quad n' = 0, 1, \ldots, N - 1
\]

14.2.2 Polyphase Form

The interpolated values between the low-rate samples $x_{up}(nL)$ and $x_{up}(nL + L)$, that is, the values at the high-rate time instants $n' = nL + i$, are calculated by the filter as follows:

\[
y_{up}(nL + i) = \sum_{k' = -LM}^{LM} d(k') x_{up}(nL + i - k'), \quad i = 0, 1, \ldots, L - 1
\]

Writing uniquely $k' = kL + j$, with $0 \leq j \leq L - 1$, and replacing the single summation over $k'$ by a double summation over $k$ and $j$, we find

\[
y_{up}(nL + i) = \sum_{k' = -M}^{M-1} \sum_{j = 0}^{L-1} d(kL + j) x_{up}(nL + i - kL - j)
\]

To be precise, for the case $i = 0$, the summation over $k$ should be over the range $-M \leq k \leq M$. But as we will see shortly, the term $k = M$ does not contribute to the sum. Defining the $i$th polyphase subfilter by\(^1\)

\[
d_i(k) = d(kL + i), \quad -M \leq k \leq M - 1
\]

for $i = 0, 1, \ldots, L - 1$, we can rewrite the $i$th interpolated sample value as:

\(^1\)For $i = 0$, the range of $k$ is $-M \leq k \leq M$. 

Fig. 14.2.2 Low-rate samples with respect to the slow and fast time scales.

An FIR approximation to the ideal interpolator is obtained by truncating $d(k')$ to finite length, say $N = 2LM + 1$:

\[
d(k') = \frac{\sin(\pi k'/L)}{\pi k'/L}, \quad -LM \leq k' \leq LM
\]
14. INTERPOLATION, DECIMATION, AND OVERSAMPLING

\[ y_{up}(nL + i) = \sum_{k=0}^{M-1} \sum_{j=0}^{L-1} d_j(k) x_{up}(nL - kL + i - j) \]

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

\[ x_{up}(nL - kL + i - j) = 0, \quad \text{if } i \neq j \]

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

\[ y_{up}(nL + i) = \sum_{k=0}^{M-1} d_i(k) x_{up}(nL - kL), \quad i = 0, 1, \ldots, L - 1 \]  \hspace{1cm} (14.2.11)

or, in terms of the low-rate samples:

\[ y_i(n) = \sum_{k=0}^{M-1} d_i(k) x(n - k), \quad i = 0, 1, \ldots, L - 1 \]  \hspace{1cm} (14.2.12)

where we set \( y_i(n) = y_{up}(nL + i) \). Thus, the \( i \)th interpolated value, \( y_{up}(nL + i) \), is computed by the \( i \)th polyphase subfilter, \( d_i(k) \), which has length \( 2M \) and is acting 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.

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 \( x_{up}(n') \) 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,

\[ R = N (L f_s) = N L f_s \]  \hspace{1cm} (direct form)

\[ R = L (2M) f_s = N f_s \]  \hspace{1cm} (polyphase form)

where in the direct form we have a single filter of length \( N \) operating at rate \( L f_s \) 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. 14.2.4](image)

Interpolator uses \( M \) low-rate samples before and after \( y_{up}(nL + i) \).

![Fig. 14.2.5](image)

Polyphase subfilter implementation of digital interpolator.

This can be seen more formally, as follows. Let \( \zeta^{-1} \) denote the unit delay with respect to the high rate \( L f_s \) and let \( z^{-1} \) denote the low-rate delay. Since \( L \) high-rate delays equal one low-rate delay, that is, \( L T' = T \), we will have:

\[ z = \zeta^L \quad \iff \quad \zeta = z^{1/L} \]  \hspace{1cm} (14.2.13)

The \( \zeta \)-transform of the high-rate filter output \( y_{up}(n') \) can be expressed in terms of the \( z \)-transforms of the low-rate output signals \( y_i(n) \) as follows. Writing uniquely \( n' = nL + i \) with \( 0 \leq i \leq L - 1 \), we have

\[ Y_{up}(\zeta) = \sum_{n'=-\infty}^{\infty} y_{up}(n') \zeta^{-n'} = \sum_{i=0}^{L-1} \sum_{n=-\infty}^{\infty} y_i(n) \zeta^{-nL-i} = \sum_{i=0}^{L-1} \zeta^{-i} \sum_{n=-\infty}^{\infty} y_i(n) \zeta^{-Ln}, \quad \text{or,} \]

†Actually, there are \( L - 1 \) subfilters of length \( (2M) \) and one (the filter \( d_0 \)) of length \( (2M + 1) \), giving rise to \( R = (L - 1) (2M) f_s + (2M + 1)f_s = N f_s \), where \( N = 2LM + 1 \).
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 $\mathbf{z}$-transform of the high-rate filter (14.2.6) and the $\mathbf{z}$-transforms of its polyphase subfilters (14.2.10), justifying the realization of Fig. 14.2.5:

\[
Y_{\text{up}}(\zeta) = \sum_{i=0}^{L-1} \zeta^{-i} Y_i(\zeta L) = \sum_{i=0}^{L-1} z^{-i/L} Y_i(z) \tag{14.2.14}
\]

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:

\[
d_0(k) = d(kL) = \frac{\sin(\pi k)}{\pi k} = \delta(k), \quad -M \leq k \leq M
\]

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 $y_{\text{up}}(nL) = x(nL)$. 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:

\[
h_i(n) = d_i(n-M) = d((n-M)L + i) = d(nL + i - LM)
\]

for $n = 0, 1, \ldots, 2M - 1$. For the windowed case, we have:

\[
h_i(n) = d(nL + i - LM)w(nL + i), \quad n = 0, 1, \ldots, 2M - 1
\]

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 $k = m - M$,

\[
y_i(n) = \sum_{m=-P}^{P} h_i(m)x(M+n-m)
\]

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 $\mathbf{z}$-transforms. Definition (14.2.16) reads in the $\mathbf{z}$-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 $D_i(z) = z^M H_i(z)$, we obtain the $\mathbf{z}$-domain equivalent of Eq. (14.2.18):

\[
Y_i(z) = D_i(z)X(z) = H_i(z)(z^M X(z))
\]

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

\[
w_m(n) = x(M+n-m), \quad m = 0, 1, \ldots, P
\]

Then, Eq. (14.2.18) can be written in the dot-product notation of Chapter 4:

\[
y_i(n) = \text{dot}(P,h_i,w(n)) = \sum_{m=0}^{P} h_i(m)w_m(n), \quad i = 0, 1, \ldots, L-1
\]

After computing the outputs of the $L$ subfilters, the internal state $w$ may be updated to the next time instant by a call to the routine delay, which shifts the contents:

\[
w_m(n+1) = w_{m-1}(n), \quad m = 1, 2, \ldots, P
\]

This leads to the following sample processing algorithm for the polyphase form:

1. Initialize the internal state vector $w(n) = [w_0(n), w_1(n), \ldots, w_P(n)]$ by filling it with the first $M$ low-rate input samples, $x(0), x(1), \ldots, x(M-1)$, that is, at time $n = 0$ start with

\[
w(0) = [0, x_{M-1}, x_{M-2}, \ldots, x_0, 0, 0, \ldots, 0]^{T}
\]

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:

\[
Y_{\text{up}}(\zeta) = \sum_{i=0}^{L-1} \zeta^{-i} Y_i(\zeta L) = \sum_{i=0}^{L-1} z^{-i/L} Y_i(z)
\]
14.2. INTERPOLATION FILTER DESIGN

14. INTERPOLATION, DECIMATION, AND OVERSAMPLING

and

\[ X_{\text{up}}(\omega') = \sum_{n} x_{\text{up}}(n') e^{-j\omega' n'}, \quad X(\omega) = \sum_{n} x(n) e^{-j\omega n} \]

Similarly, using Eq. (14.2.13), their z-transforms will be related by

\[ X_{\text{up}}(z) = X(z) = X(z^L) \]

where the slow and fast z variables are related to the corresponding digital frequencies by

\[ z = e^{j\omega} = e^{2\pi njf_s}, \quad \zeta = e^{j\omega'} = e^{2\pi nf/L f_s} \]

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:

\[ X'(f) = \sum_{n} x_a(n'T') e^{-2\pi nf'/f_s'} = \frac{1}{T} \sum_{m} X_a(f - mf_s') \quad (14.2.22) \]

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 remove these replicas, that is, 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,

\[ X'(f) = D(f) X(f) \quad \text{(ideal interpolation)} \quad (14.2.23) \]

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

\[ \frac{1}{T} X_a(f) + \text{replicas} = \frac{1}{T} D(f) X_a(f) + \frac{1}{T} D(f) \sum_{m=1}^{L-1} X_a(f - mf_s) + \text{replicas} \]

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 \( [f_s/2, L f_s - f_s/2] \). The filter \( D(f) \) is required to remove these replicas, that is,

\[ D(f) = 0, \quad f_s/2 \leq |f| \leq L f_s - f_s/2 \]

The same relationship can be expressed in terms of the digital frequencies as:

\[ X_{\text{up}}(\omega') = X(\omega) = X(\omega'L) \]

where

\[ \omega = \frac{2\pi f}{f_s}, \quad \omega' = \frac{2\pi f}{f_s'} = 2\pi f'T', \quad \omega = \omega'L \]

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 \( x(n) \) the low-rate input sample, \( x(M + n), n = 0, 1, \ldots, \) and processing it by the algorithm:

\[ \text{for each low-rate input sample } x \text{ do:} \]

\[ w_0 = x \]

\[ \text{for } i = 0, 1, \ldots, L - 1 \text{ compute:} \]

\[ y_i = \text{dot}(P, h_i, w) \]

\[ \text{delay}(P, w) \]

\[ \text{(14.2.20)} \]

14.2.3 Frequency Domain Characteristics

The upsampling 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. (14.2.21) and (14.2.22) and \( T = LT' \), we have

\[ X_{\text{up}}(f) = \sum_{n} x_{\text{up}}(n') e^{-2\pi jnf'/f_s'} = \frac{1}{T} \sum_{m} x_{\text{up}}(n - mT/L) e^{-2\pi jnf'/f_s'} = \frac{1}{T} \sum_{m} x_a(f - mf_s') \]

Thus,

\[ X_{\text{up}}(f) = X(f) = \sum_{m} x_a(f - mf_s) \quad (14.2.21) \]

The same relationship can be expressed in terms of the digital frequencies as:

\[ X_{\text{up}}(\omega') = X(\omega) = X(\omega'L) \]

\[ \omega = \frac{2\pi f}{f_s}, \quad \omega' = \frac{2\pi f}{f_s'} = 2\pi f'T', \quad \omega = \omega'L \]
14.2. INTERPOLATION FILTER DESIGN

as shown in Fig. 14.2.3. Similarly, within the low-rate Nyquist interval $-f_s/2 \leq f \leq f_s/2$, the filter must satisfy:

$$
D(f) = \frac{1}{T}X_a(f) = \frac{1}{T}D(f)X_a(f) \quad \Rightarrow \quad D(f) = \frac{T}{T'} = L
$$

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 $[-f_s'/2, f_s'/2]$:

$$
D(f) = \begin{cases} 
L, & \text{if } |f| \leq \frac{f_s}{2} \\
0, & \text{if } \frac{f_s}{2} < |f| \leq f_s'
\end{cases}
$$

(ideal interpolator) \hspace{1cm} (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 $f_s$:

$$
h(t) = \frac{\sin(\pi t/T)}{\pi t/T}
$$

(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

$$
d(k') = h(k'T') = \frac{\sin(\pi k'T'/T)}{\pi k'T'/T} = \frac{\sin(\pi k'/L)}{\pi k'/L}
$$

and using $x_a(nLT') = x_a(nT) = x(n)$, we obtain $h(n'T' - nLT') = d(n' - nL)$ and the filtering equation:

$$
x'(n') = \sum_n d(n' - nL)x(n) = \sum_m d(n' - m')x_a(m')
$$

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.)

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 interpolator filter is operating at the fast rate $f_s'$. For any length-$N$ window $w(n')$, the interpolator’s impulse response is computed by:

$$
h(n') = w(n')d(n' - LM), \quad n' = 0, \ldots, N - 1 = 2LM
$$

(14.2.27)

The $L$ length-$(2M)$ polyphase subfilters are defined in terms of $h(n')$ as follows. For $i = 0, 1, \ldots, L - 1$:

$$
h_i(n) = h(nL + i), \quad n = 0, 1, \ldots, 2M - 1
$$

(14.2.28)

For a Kaiser window design, we start by specifying the desired stopband attenuation $A$ in dB, and desired transition width $\Delta f$ about the ideal cutoff frequency:

$$
f_c = \frac{f_s'}{2L} = \frac{f_s}{2}
$$

so that the passband and stopband frequencies are:

$$
f_{pass} = f_c - \frac{1}{2}\Delta f, \quad f_{stop} = f_c + \frac{1}{2}\Delta f
$$

The Kaiser window parameters are calculated by:

$$
\delta = 10^{-A/20}
$$

$$
D = \frac{A - 7.95}{14.36}
$$

$$
\alpha = 0.1102(A - 8.7) \quad (\text{because, typically, } A > 50 \text{ dB})
$$

$$
N - 1 \geq \frac{Df_s'}{\Delta f} = \frac{Df_s}{\Delta f} = \frac{DL}{\Delta F}
$$

(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 \( n' = 0, 1, \ldots, N - 1 \):

\[
w(n') = \frac{I_0(\alpha \sqrt{1 - (n' - LM)^2 / (LM)^2})}{I_0(\alpha)} = \frac{I_0(\alpha \sqrt{n' (2LM - n') / LM})}{I_0(\alpha)}
\]

The frequency response of the designed filter may be computed by:

\[
H(f) = \sum_{n'=0}^{N-1} h(n') e^{-2\pi j fn'/fs} = \sum_{n'=0}^{N-1} h(n') e^{-2\pi j fn'(1/Ls)}
\]

The designed filter \( h(n') \) can be implemented in its direct or polyphase forms.

![Fig. 14.2.7](image-url) 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](image-url) 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 \( \Delta f_s \), which is typically very narrow. The remaining stages have much wider transition widths and therefore smaller filter lengths.

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 \( f_s' = L_0 f_s = 2f_s \) 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 \( \Delta f_s \) 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:

\[
H_0(f) = \sum_{n'=0}^{N_0-1} h_0(n') e^{-2\pi j fn'(2f_s)}, \quad H_1(f) = \sum_{n'=0}^{N_1-1} h_1(n') e^{-2\pi j fn'(4f_s)}
\]

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:

\[
R_{\text{multi}} = N_0 f_s + N_1 (2f_s) \simeq \left( \frac{2D}{\Delta F} + 8D \right) f_s = \frac{2D}{\Delta F} (1 + 4\Delta F) f_s
\]
14.2. INTERPOLATION FILTER DESIGN

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:

\[ R_{\text{single}} = Nf_s = \frac{4D}{\Delta F}f_s \]

The relative performance of the multistage versus the single stage designs will be:

\[ \frac{R_{\text{multi}}}{R_{\text{single}}} = \frac{1 + 4\Delta F}{2} = \frac{1}{2} + 2\Delta F \]

(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:

\[ \frac{1}{2} + 2\Delta F < 1 \quad \Leftrightarrow \quad \Delta F < \frac{1}{4} \]

which is usually satisfied because typical values of \( \Delta F \) 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 8\( f_s \). For the 2\( \times 2 \times 2 \) 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:

\[ R_{\text{multi}} = N_0f_s + N_1(2f_s) + N_2(4f_s) = \left( \frac{2D}{\Delta F} + 8D + \frac{32D}{3} \right)f_s \]

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:

\[ R_{\text{single}} = Nf_s = \frac{8D}{\Delta F}f_s \]

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\( \times 4 \) and 4\( \times 2 \) cases:

\[ \frac{R_{\text{multi}}}{R_{\text{single}}} = \frac{1}{4} + \frac{2}{3}\Delta F \quad (2 \times 4 \text{ case}) \]

\[ \frac{R_{\text{multi}}}{R_{\text{single}}} = \frac{1}{2} + \frac{4}{3}\Delta F \quad (4 \times 2 \text{ case}) \]

Comparing the three multistage cases, it appears that the 2\( \times 4 \) case is more efficient than the 2\( \times 2 \times 2 \) case, which is more efficient than the 4\( \times 2 \) 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 \( H_0, H_1, \ldots, H_{K-1} \) that increase the sampling rate successively by the factors \( L_0, L_1, \ldots, L_{K-1} \). The sampling rate at the input of the \( i \)th 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, \( F_{i}f_s = L_iF_{i-1}f_s \), where:

\[
F_i = L_iF_{i-1} = L_0L_1 \cdots L_i, \quad i = 0, 1, \ldots, K - 1
\]

We set \( F_{-1} = 1 \), so that \( F_0 = L_0 \). The total interpolation factor will be:

\[
L = F_{K-1} = L_0L_1 \cdots L_{K-1}
\]

For a Kaiser design, we assume a given transition width \( \Delta f \) about the ideal cutoff frequency of the \( i \)th stage interpolator \( \omega_{c,i} \). The specifications of the \( i \)th stage are shown in Fig. 14.2.7, with \( L = L_0 \). The ideal cutoff frequencies of these filters will be:

\[
f_{ci} = \frac{F_{i-1}f_s}{2L_i} \Rightarrow \omega_{c,i} = \frac{2\pi f_{ci}}{F_{i}f_s} = \frac{\pi}{L_i}
\]

for \( i = 0, 1, \ldots, K - 1 \). 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:

\[
f_{\text{pass},0} = f_{c0} - \frac{1}{2}\Delta f_0 = \frac{f_s}{2} - \frac{1}{2}\Delta f, \quad f_{\text{stop},0} = f_{c0} + \frac{1}{2}\Delta f_0 = \frac{f_s}{2} + \frac{1}{2}\Delta f
\]

and for \( i = 1, 2, \ldots, K - 1 \):

\[
f_{\text{pass},i} = f_{ci} - \frac{1}{2}\Delta f_i = \frac{f_s}{2} - \frac{1}{2}\Delta f, \quad f_{\text{stop},i} = f_{ci} + \frac{1}{2}\Delta f_i = F_{i-1}f_s - \frac{f_s}{2}
\]

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 \( i = 1, 2, \ldots, K - 1 \):

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.

Note that the transition widths remain the same, but the ideal cutoff frequencies also shift down by \( \Delta f/2 \), that is, for \( i = 1, 2, \ldots, K - 1 \):

\[
f_{ci} = \frac{1}{2}(f_{\text{pass},i} + f_{\text{stop},i}) = \frac{1}{2}F_{i-1}f_s - \frac{1}{2}\Delta f \Rightarrow \omega_{c,i} = 2\pi f_{ci}/F_{i}f_s = \pi/L_i - \pi\Delta f/F_i
\]

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 \( i \)th stage are shown in Fig. 14.2.11.

![Fig. 14.2.11](image)

The windowed impulse responses of the filters will be, for \( i = 0, 1, \ldots, K - 1 \):

\[
h_i(n') = d(L_i, n' - L_iM_i)w(\alpha_i, N_i, n'), \quad n' = 0, 1, \ldots, N_i - 1
\]

where \( d(L_i, k') \) is the ideal interpolation filter with cutoff frequency\(^1\) \( \omega_{c,i} = \pi/L_i \)

\[\text{for the alternative case, replace } \omega_{c,i}' \text{ by its shifted version } \omega_{c,i}' = \pi/L_i - \pi\Delta F/F_i\]

\[\text{Note that the transition widths remain the same, but the ideal cutoff frequencies also shift down by } \Delta f/2, \text{ that is, for } i = 1, 2, \ldots, K - 1\]

\[f_{ci} = \frac{1}{2}(f_{\text{pass},i} + f_{\text{stop},i}) = \frac{1}{2}F_{i-1}f_s - \frac{1}{2}\Delta f \Rightarrow \omega_{c,i}' = 2\pi f_{ci}/F_{i}f_s = \pi/L_i - \pi\Delta f/F_i\]

The windowed impulse responses of the filters will be, for \( i = 0, 1, \ldots, K - 1 \):

\[
h_i(n') = d(L_i, n' - L_iM_i)w(\alpha_i, N_i, n'), \quad n' = 0, 1, \ldots, N_i - 1
\]

where \( d(L_i, k') \) is the ideal interpolation filter with cutoff frequency\(^1\) \( \omega_{c,i}' = \pi/L_i \)
14. INTERPOLATION, DECIMATION, AND OVERSAMPLING

14.2. INTERPOLATION FILTER DESIGN

\[
d(L_i, k') = \sin(\omega_0' k') / \omega_0' k' = \sin(\pi k'/L_i) / \pi k'/L_i
\]

and \(w(\alpha_i, N_i, n')\) is the corresponding Kaiser window:

\[
w(\alpha_i, N_i, n') = \frac{I_0(\alpha_i \sqrt{1 - (n'/L_i M_i)^2}/(L_i M_i)^2)}{I_0(\alpha_i)}
\]

The polyphase subfilters of each \(H_i\) are defined by

\[
h_{ij}(n) = h(n L_i + j), \quad j = 0, 1, \ldots, L_i - 1, \quad n = 0, 1, \ldots, P_i
\]

where \(P_i = 2M_i - 1\). Finally, compute the frequency responses of the individual stages:

\[
H_i(f) = \sum_{n=0}^{N_i-1} h_i(n') e^{-2\pi j n'/(F_i f_i)}, \quad i = 0, 1, \ldots, K - 1
\]

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

\[
1 - \delta_i \leq \left| \frac{H_i(f)}{L_i} \right| \leq 1 + \delta_i
\]

where the DC gain \(L_i\) has been factored out. Multiplying these inequalities together, we obtain the bounds:

\[
\prod_{i=0}^{K-1} (1 - \delta_i) \leq \left| \frac{H_0(f) H_1(f) \cdots H_{K-1}(f)}{L_0 L_1 \cdots L_{K-1}} \right| \leq \prod_{i=0}^{K-1} (1 + \delta_i)
\]

For small \(\delta_i\) we may use the approximation

\[
\prod_{i=0}^{K-1} (1 \pm \delta_i) \approx 1 \pm \sum_{i=0}^{K-1} \delta_i
\]

to get the total passband ripple of the cascaded filter

\[
1 - \delta_{\text{tot}} \leq \left| \frac{H_{\text{tot}}(f)}{L} \right| \leq 1 + \delta_{\text{tot}}, \quad \text{where} \quad \delta_{\text{tot}} = \sum_{i=0}^{K-1} \delta_i
\]

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 \(\delta_i\)'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,

\[
R_{\text{multi}} = \sum_{i=0}^{K-1} R_i, \quad \text{where} \quad R_i = N_i F_{i-1} f_i, \quad i = 0, 1, \ldots, K - 1
\]

These follow from the observation that the \(i\)th filter operates at rate \(F_{i} f_{i}\) and would have computational rate \(R_i = N_i F_{i} f_{i}\). By comparison, the single-stage design will have Kaiser length and polyphase computational rate:

\[
N - 1 = \frac{D(L f_s)}{\Delta f}, \quad R_{\text{single}} = N f_s
\]

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:

\[
\frac{R_{\text{multi}}}{R_{\text{single}}} = \frac{N_0 + \sum_{i=1}^{K-1} N_i F_{i-1}}{N}
\]

(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:

\[
N_0 \approx \frac{D L_0}{\Delta F}, \quad N_i \approx \frac{D F_i}{F_{i-1}}, \quad N \approx \frac{D L}{\Delta F}
\]

to obtain the simplified expression:

\[
\frac{R_{\text{multi}}}{R_{\text{single}}} = \frac{L_0 L_{i-1}}{L} + \frac{\Delta F}{L_{i-1} - 1} \sum_{i=1}^{K-1} \frac{F_i F_{i-1}}{(F_i - 1) L_i}
\]

(14.2.34)

For a two-stage design with \(L = L_0 L_1, F_1 = L_1,\) and \(F_0 = L_0\), we have

\[
\frac{R_{\text{multi}}}{R_{\text{single}}} = \frac{L_0 L_{i-1}}{L} + \frac{F_1 F_0}{(F_0 - 1) L} \Delta F = \frac{1}{L_1} + \frac{L_0}{L_0 - 1} \Delta F
\]

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:

\[
\frac{1}{L_1} + \frac{L_0}{L_0 - 1} \Delta F < 1 \quad \iff \quad \Delta F < \left(1 - \frac{1}{L_0}\right) \left(1 - \frac{1}{L_1}\right)
\]
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 $2 \times 2 \times 2$ 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,

$$d(k') = h(k'T')$$  \hspace{1cm} (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

$$y_{up}(n') = \sum_m d(n' - mL)x(m)$$  \hspace{1cm} (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 $x(n)$

$$y_{up}(n') = \sum_m d(n' - m')x_{up}(m') = \sum_{k'} d(k')x_{up}(n' - k')$$  \hspace{1cm} (14.3.3)

Two of the most common interpolators are the hold and linear interpolators resulting from the sample/hold and linear analog reconstructor having impulse responses:

$$h(t) = \begin{cases} 1, & \text{if } 0 \leq t < T \\ 0, & \text{otherwise} \end{cases}$$

and

$$h(t) = \begin{cases} 1 - \frac{|t|}{T}, & \text{if } |t| \leq T \\ 0, & \text{otherwise} \end{cases}$$

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:

- **(hold)**
  $$d(k') = \begin{cases} 1, & \text{if } 0 \leq k' \leq L - 1 \\ 0, & \text{otherwise} \end{cases} = u(k') - u(k' - L)$$

- **(linear)**
  $$d(k') = \begin{cases} 1 - \frac{|k'|}{L}, & \text{if } |k'| \leq L - 1 \\ 0, & \text{otherwise} \end{cases}$$

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 $L = 8$.](image)

The filtering operations of these interpolators are very simple. The hold interpolator holds each low-rate sample constant for 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 $x_{up}(nL + i - kL - j) = 0$, if $i \neq j$, to get

$$y_{up}(nL + i) = \sum_k d_i(k)x(n - k)$$  \hspace{1cm} (14.3.4)

where $d_i(k)$ are the corresponding polyphase subfilters:

$$d_i(k) = d(kL + i), \quad i = 0, 1, \ldots, L - 1$$

The summation over $k$ must be determined for each case. In the hold case, we have the restriction on $k$:

$$0 \leq k' \leq L - 1 \Rightarrow 0 \leq kL + i \leq L - 1 \Rightarrow 0 \leq k \leq 1 - \frac{1 + i}{L}$$

Because $0 \leq i \leq L - 1$, the only allowed value of $k$ in that range is $k = 0$. Similarly, in the linear case, we have:

$$|k'| \leq L - 1 \Rightarrow |kL + i| \leq L - 1 \Rightarrow -1 + \frac{1 - i}{L} \leq k \leq 1 - \frac{1 + i}{L}$$

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:
14.3. LINEAR AND HOLD INTERPOLATORS

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 $z$-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

$$D_i(z) = 1 \quad \Rightarrow \quad D_i(\zeta^L) = 1$$

Then, it follows from Eq. (14.2.15)

$$D(\zeta) = \sum_{i=0}^{L-1} \zeta^{-i}D_i(\zeta^L) = \sum_{i=0}^{L-1} \zeta^{-i} = 1 - \frac{\zeta^{-L}}{1 - \zeta^{-1}}$$

Setting $\zeta = e^{j\omega_0'} = e^{2\pi f/L_f}$, we obtain the frequency response of the hold interpolator:

$$D(f) = \frac{1 - e^{-j\omega_0'/2}}{1 - e^{-j\omega_0'/2}} = \frac{\sin(L\omega_0'/2)}{\sin(\omega_0'/2)}$$

Similarly, the low-rate $z$-transform of $d_i(k)$ for the linear case is:

$$D_i(z) = 1 + \frac{i}{L}z \quad \Rightarrow \quad D_i(\zeta^L) = 1 + \frac{i}{L} (\zeta^L - 1)$$

From Eq. (14.2.15) we find:

$$D(\zeta) = \sum_{i=0}^{L-1} \zeta^{-i}D_i(\zeta^L) = \sum_{i=0}^{L-1} \zeta^{-i} + \frac{\zeta^L - 1}{L} \sum_{i=0}^{L-1} i\zeta^{-i}$$

**Figure 14.3.2** 8-fold hold and linear interpolators.
and using the identity:

\[
\sum_{i=0}^{L-1} i\zeta^{-i} = \frac{(1 - \zeta^{-L})}{(1 - \zeta^{-1})^2} - \frac{L\zeta^{-L}}{1 - \zeta^{-1}}
\]

we obtain (see also Problem 5.11):

\[
D(\zeta) = \frac{1}{L} \left( 1 - \frac{\zeta^{-1}}{(1 - \zeta^{-1})^2} \right) - \frac{\zeta^{-L}}{1 - \zeta^{-1}}
\]

which leads to the frequency response:

\[
D(f) = \frac{1}{L} \left| \sin \left( \frac{L\omega'}{2} \right) \right|^2 = \frac{1}{L} \left| \sin \left( \frac{\pi f / f_s}{L} \right) \right|^2
\]

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 multi-stage implementations of interpolators, especially in the latter stages that have higher sampling rates. Some example designs are discussed in Section 14.4.5.

### 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:

\[
d(k') = \frac{\sin (\pi k' / 4)}{\pi k' / 4}, \quad -8 \leq k' \leq 8
\]

or, numerically,

\[
h = d = [0, -0.13, -0.21, -0.18, 0, 0.30, 0.64, 0.90, 0.90, 0.64, 0.30, 0.18, -0.21, -0.13, 0]
\]

where \( h \) is the causal version, with time origin shifted to the left of the vector, and \( d \) 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 \( i = 0, 1, 2, 3 \),

\[
d_i(k) = d(4k + i), \quad -2 \leq k \leq 1
\]

They are extracted from \( h \) by taking every fourth entry, starting with the \( i \)th entry:

\[
\begin{align*}
h_0 &= d_0 = [0, 0, 1, 0] \\
h_1 &= d_1 = [-0.13, 0.30, 0.90, -0.18] \\
h_2 &= d_2 = [-0.21, 0.64, 0.64, -0.21] \\
h_3 &= d_3 = [-0.18, 0.90, 0.30, -0.13]
\end{align*}
\]

The interpolated samples between \( x(n) = x_{up}(4n) \) and \( x(n + 1) = x_{up}(4n + 4) \) 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 \( \{x_{up}(4n + 8), x_{up}(4n + 4), x_{up}(4n), x_{up}(4n - 4)\} \). Equations (14.2.12) can be cast in a compact matrix form:

\[
\begin{bmatrix}
y_{up}(4n) \\
y_{up}(4n + 1) \\
y_{up}(4n + 2) \\
y_{up}(4n + 3)
\end{bmatrix} =
\begin{bmatrix}
0 & 0 & 1 & 0 \\
-0.13 & 0.30 & 0.90 & -0.18 \\
-0.21 & 0.64 & 0.64 & -0.21 \\
-0.18 & 0.90 & 0.30 & -0.13
\end{bmatrix}
\begin{bmatrix}
x_{up}(4n + 8) \\
x_{up}(4n + 4) \\
x_{up}(4n) \\
x_{up}(4n - 4)
\end{bmatrix}
\]

These results can be understood more intuitively using the LTI form of convolution, that is, superimposing the full length-17 symmetric impulse response \( d \) 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.
14. DESIGN EXAMPLES

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 $x_{up}(4n + 8)$ will contribute an amount $-0.13x_{up}(4n + 8)$, the sample $x_{up}(4n + 4)$ will contribute an amount $0.30x_{up}(4n + 4)$, the sample $x_{up}(4n)$ will contribute $0.90x_{up}(4n)$, and the sample $x_{up}(4n - 4)$ an amount $-0.18x_{up}(4n - 4)$. The interpolated value is built up from these four contributions:

$$y_{up}(4n + 1) = -0.13x_{up}(4n + 8) + 0.30x_{up}(4n + 4) + 0.90x_{up}(4n) - 0.18x_{up}(4n - 4)$$

Similarly, it should be evident from Fig. 14.4.2 that $y_{up}(4n) = x_{up}(4n)$, 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:

$$y_{up}(4n + 1) = \sum_{k'} d(4n + 1 - k') x_{up}(k')$$

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.

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:

$$y_{up}(4n + 1) = -0.18x_{up}(4n - 4) + 0.90x_{up}(4n) + 0.30x_{up}(4n + 4) - 0.13x_{up}(4n + 8)$$

The Hamming windowed version of the filter is obtained by multiplying the full length-17 filter response $h$ by a length-17 Hamming window. The resulting impulse response becomes:

$$h = [0, -0.02, -0.05, -0.07, 0, 0.22, 0.55, 0.87, 1, 0.87, 0.55, 0.22, 0, -0.07, -0.05, -0.02, 0]$$

The polyphase interpolation equations become in this case:

$$\begin{bmatrix}
  u_p(4n) \\
  u_p(4n + 1) \\
  u_p(4n + 2) \\
  u_p(4n + 3)
\end{bmatrix} =
\begin{bmatrix}
  0 & 0 & 1 & 0 \\
  -0.02 & 0.22 & 0.87 & -0.07 \\
  -0.05 & 0.55 & 0.55 & -0.05 \\
  -0.07 & 0.87 & 0.22 & -0.02
\end{bmatrix}
\begin{bmatrix}
  x_{up}(4n + 8) \\
  x_{up}(4n + 4) \\
  x_{up}(4n) \\
  x_{up}(4n - 4)
\end{bmatrix}$$

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,

$$H(\zeta) = H_0(\zeta^4) + \zeta^{-1}H_1(\zeta^4) + \zeta^{-2}H_2(\zeta^4) + \zeta^{-3}H_3(\zeta^4) = H_0(z) + z^{-1/4}H_1(z) + z^{-2/4}H_2(z) + z^{-3/4}H_3(z)$$

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:

$$h_i = [h_{i0}, h_{i1}, h_{i2}, h_{i3}], \quad H_i(z) = h_{i0} + h_{i1}z^{-1} + h_{i2}z^{-2} + h_{i3}z^{-3}$$
for \( i = 0, 1, 2, 3 \), where \( h_i \) 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 timescale:

\[
x(n) = \{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1\}
\]

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.
14.4. DESIGN EXAMPLES

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 \( f_c = f_s'/2L = 20 \) kHz. Therefore, the passband and stopband frequencies of the filter will be

\[
\begin{align*}
\text{passband:} & \quad f_{\text{pass}} = 20 - \frac{5}{2} = 17.5 \text{ kHz}, \\
\text{stopband:} & \quad f_{\text{stop}} = 20 + \frac{5}{2} = 22.5 \text{ kHz}
\end{align*}
\]

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 \( \delta = 10^{-8} \), passband attenuation \( A_{\text{pass}} = 0.0017 \) dB, and the following values for the Kaiser window parameters \( D \) and \( \alpha \):

\[
D = \frac{A - 7.95}{14.36} = 5.017, \quad \alpha = 0.1102(A - 8.7) = 7.857
\]

For a \( 2 \times 2 \) 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 \( H_1 \):

\[
\begin{align*}
N_0 - 1 = \frac{DL_0}{\Delta F} = 2D = 80.27, \\
N_1 - 1 = \frac{DF_1}{F_0 - 1} = 4D = 20.07
\end{align*}
\]

which get rounded up to the values:

\[
N_0 = 85 = 2L_0M_0 + 1 = 4M_0 + 1 \quad \Rightarrow \quad M_0 = 21 \\
N_1 = 25 = 2L_1M_1 + 1 = 4M_1 + 1 \quad \Rightarrow \quad M_1 = 6
\]

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.

†See Ref. [356] for actual DSP chips with comparable design characteristics.
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

$$N = 169 = 2LM + 1 = 8M + 1 \quad \Rightarrow \quad M = 21$$

Its magnitude response $H(f)$ is shown in Fig. 14.4.14 together with the magnitude response of the combined multistage filter $H_{\text{tot}}(f) = H_0(f)H_1(f)$. Again, we have normalized them to their DC values, namely, $H(f)/L$, and $H_{\text{tot}}(f)/L$.

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:

$$\frac{R_{\text{multi}}}{R_{\text{single}}} = \frac{N_0 + N_1\Delta F}{N} = \frac{85 + 25 \cdot 2}{169} = 0.80$$

which compares well with the approximate result of Eq. (14.2.30).

Finally, as we saw earlier, the passband of the total filter $H_{\text{tot}}(f) = H_0(f)H_1(f)$ tends to be worse than the passbands of the individual factors. Let $\delta$ be the common passband ripple, as calculated from Eq. (14.2.29). Then, the two individual filters will satisfy within their passbands:

$$1 - \delta \leq \left| \frac{H_0(f)}{L_0} \right| \leq 1 + \delta, \quad 1 - \delta \leq \left| \frac{H_1(f)}{L_1} \right| \leq 1 + \delta$$

Multiplying the two inequalities, we find for the total filter

$$(1 - \delta)^2 \leq \left| \frac{H_0(f)H_1(f)}{L_0L_1} \right| \leq (1 + \delta)^2$$

or, approximately if $\delta$ is small,

$$1 - 2\delta \leq \left| \frac{H_{\text{tot}}(f)}{L} \right| \leq 1 + 2\delta$$

Thus, the passband ripple is effectively doubled, $\delta_{\text{tot}} = 2\delta$. Taking logs of both sides, we obtain the following bounds for the passband attenuations in dB:

$$-8.7\delta \leq 20 \log_{10} \left| \frac{H_0(f)}{L_0} \right| \leq 8.7\delta, \quad -8.7\delta \leq 20 \log_{10} \left| \frac{H_1(f)}{L_1} \right| \leq 8.7\delta$$

$$-8.7(2\delta) \leq 20 \log_{10} \left| \frac{H_{\text{tot}}(f)}{L} \right| \leq 8.7(2\delta)$$

where we used the small-\(\delta\) approximation:

$$20 \log_{10}(1 + \delta) = 8.7\delta$$

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 $H_{\text{tot}}(f)$ is less than the worst-case ripple $\delta_{\text{tot}} = 2\delta$. 
In units of the high-rate digital frequency \( \omega' = 2\pi f'/f_s' \), Eq. (14.4.5) becomes:

\[
D(\omega') = \begin{cases} 
  L \left( \frac{\omega'}{2} \right), & \text{if } |\omega'| \leq \frac{\pi}{L} \\
  0, & \text{if } \frac{\pi}{L} < |\omega'| \leq \pi 
\end{cases}
\]  

(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:

\[
\tilde{d}(k') = \frac{1}{N} \sum_{i=-LM}^{LM} D(\omega_i') e^{i\omega_i'k'}, \quad -LM \leq k' \leq LM
\]

(14.4.8)

where \( \omega_i' \) are the N DFT frequencies spanning the symmetric Nyquist interval \([-\pi, \pi]\):

\[
\omega_i' = \frac{2\pi i}{N}, \quad -LM \leq i \leq LM
\]

The designed causal windowed filter will be

\[
h(n') = \tilde{d}(n' - LM) w(n'), \quad 0 \leq n' \leq N - 1
\]

(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 \( \Delta f \), and then determine the filter length \( N \) and the window parameter \( \alpha \). Because the filter is sloping upwards in the passband, to achieve a true attenuation 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:

\[
\tilde{d}(k') = \frac{1}{N} \sum_{i=-LM}^{LM} D(\omega_i) \cos(\omega_i'k'), \quad -LM \leq k' \leq LM
\]

and because \( D(\omega') \) is even in \( \omega' \):

\[
\tilde{d}(k') = \frac{1}{N} \left[ D(\omega_0) + 2 \sum_{i=1}^{LM} D(\omega_i) \cos(\omega_i'k') \right], \quad -LM \leq k' \leq LM
\]

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

\[
0 < \omega_i' < \frac{\pi}{L} \quad \Rightarrow \quad 0 < \frac{2\pi i}{N} < \frac{\pi}{L} \quad \Rightarrow \quad 0 < i < \frac{N}{2L} = M + 1\frac{1}{2L}
\]

Thus, the summation can be restricted over \( 1 \leq i \leq M \), giving

\[
\tilde{d}(k') = \frac{L}{N} \left[ 1 + 2 \sum_{i=1}^{M} \frac{\alpha_i/2}{\sin(\omega_i/2)} \cos(\omega_i'k') \right], \quad -LM \leq k' \leq LM
\]
14.4. DESIGN EXAMPLES

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

$$D = \frac{A - 7.95}{14.36} = 5.017, \quad N - 1 \geq \frac{D \Delta F}{\Delta F} = 80.3$$

which rounds up to $N = 85$. Fig. 14.4.16 shows the designed filter together with the inverse DAC response $1/|H_{dac}(f)|$ 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.

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.

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:

$$D = \frac{A - 7.95}{14.36} = 3.625, \quad N - 1 \geq \frac{D \Delta F}{\Delta F} = 115.98 \quad \Rightarrow \quad N = 121$$

The designed filter and its passband are shown in Fig. 14.4.18.

![Fig. 14.4.16 2-fold interpolator/equalizer designed with $A = 80$ dB.](image)

![Fig. 14.4.17 2-fold interpolator/equalizer redesigned with $A = 90$ dB.](image)

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.

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$^\dagger$ and its stopband begins at the left edge of the first spectral image, that is,

$$f_{\text{pass}} = f_s/2 - \Delta f/2 \quad \text{and} \quad f_{\text{stop}} = Lf_s - f_s/2 - \Delta f/2.$$  

$^\dagger$If the interpolator is not ideal but has transition width $\Delta f$ 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. 14.4.19 Interpolation filter equalizes DAC and postfilter responses.](image)
14.4. DESIGN EXAMPLES

14. INTERPOLATION, DECIMATION, AND OVERSAMPLING

passband
postfilter spectral images removed by postfilter

Fig. 14.4.20 Analog anti-image postfilter specifications.

\[
\begin{align*}
&f_{\text{pass}} = \frac{f_s}{2}, \quad f_{\text{stop}} = f_s' - f_{\text{pass}} = Lf_s - \frac{f_s}{2}
\end{align*}
\]

At \( f_{\text{stop}} \), the DAC already provides a certain amount of attenuation given by:

\[
|H_{\text{dac}}(f_{\text{stop}})| = \frac{\sin(\pi f_{\text{stop}}/f_s')}{\pi f_{\text{stop}}/f_s'} = \frac{\sin(\pi/2L)}{\pi - \pi/2L}
\]

which, for large \( L \), becomes approximately:

\[
|H_{\text{dac}}(f_{\text{stop}})| \approx \frac{\sin(\pi/2L)}{\pi - \pi/2L} \approx \frac{1}{2L}
\]

or, in dB

\[
A_{\text{dac}} \approx 20 \log_{10}(2L) \quad (14.4.10)
\]

The analog postfilter must supply an additional amount of attenuation \( A_{\text{stop}} \), raising the total attenuation at \( f_{\text{stop}} \) to a desired level, say \( A_{\text{tot}} \) dB:

\[
A_{\text{tot}} = A_{\text{dac}} + A_{\text{stop}}
\]

For example, suppose \( f_s = 40 \) kHz and \( L = 4 \), and we require the total suppression of the replicas to be more than \( A_{\text{tot}} = 60 \) dB. The stopband frequency will be \( f_{\text{stop}} = Lf_s - f_s/2 = 160 - 20 = 140 \) kHz. At that frequency the DAC will provide an attenuation \( A_{\text{dac}} = 20 \log_{10}(8) = 18 \) dB, and therefore the postfilter must provide the rest:

\[
A_{\text{stop}} = A_{\text{tot}} - A_{\text{dac}} = 60 - 18 = 42 \text{ dB}
\]

Suppose we use a third-order Butterworth filter with magnitude response [298–300]:

\[
|H_{\text{post}}(f)|^2 = \frac{1}{1 + \left(\frac{f}{f_0}\right)^6}
\]

and attenuation in dB:

\[
A_{\text{post}}(f) = -10 \log_{10} |H_{\text{post}}(f)|^2 = 10 \log_{10} \left[ 1 + \left(\frac{f}{f_0}\right)^6 \right] \quad (14.4.11)
\]

where \( f_0 \) is the 3-dB normalization frequency to be determined. Then, the requirement that at \( f_{\text{stop}} \) the attenuation be equal to \( A_{\text{stop}} \) gives:

\[
A_{\text{stop}} = 10 \log_{10} \left[ 1 + \left(\frac{f_{\text{stop}}}{f_0}\right)^6 \right]
\]

which can be solved for \( f_0 \):

\[
f_0 = f_{\text{stop}} \left[ 10^{\frac{42}{10} - 1} \right]^{-1/6} = 140 \cdot \left[ 10^{\frac{42}{10} - 1} \right]^{-1/6} = 28 \text{ kHz}
\]

The third-order Butterworth analog transfer function of the postfilter will be:

\[
H_{\text{post}}(s) = \frac{1}{1 + 2 \left(\frac{s}{\Omega_0}\right) + 2 \left(\frac{s}{\Omega_0}\right)^2 + \left(\frac{s}{\Omega_0}\right)^3} \quad (14.4.12)
\]

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 \( f_{\text{pass}} = f_s/2 \) can be computed from Eqs. (14.4.11) and (14.4.4):

\[
A_{\text{post}}(f_{\text{pass}}) = 10 \log_{10} \left[ 1 + \left(\frac{f_{\text{pass}}}{f_0}\right)^6 \right] = 10 \log_{10} \left[ 1 + \frac{20}{28} \right] = 0.54 \text{ dB}
\]

\[
A_{\text{dac}}(f_{\text{pass}}) = -20 \log_{10} \left[ \frac{\sin(\pi/2L)}{\pi/2L} \right] = 0.22 \text{ dB}
\]

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:

\[
D(\omega') = \begin{cases} 
L \left[ \frac{\omega'/2}{\sin(\omega'/2)} \right] \left[ 1 + \left(\frac{\omega'}{\omega_0}\right)^6 \right]^{1/2}, & \text{if } |\omega'| \leq \frac{\pi}{L} \\
0, & \text{if } \frac{\pi}{L} < |\omega'| \leq \pi 
\end{cases} \quad (14.4.13)
\]

where \( \omega_0 = 2\pi f_0/f_s' \). The impulse response coefficients will be calculated by:

\[
\tilde{d}(k) = \frac{L}{N} \left[ 1 + 2 \sum_{i=1}^{M} \frac{\omega_i'/2}{\sin(\omega_i'/2)} \right] \left[ 1 + \left(\frac{\omega_i'}{\omega_0}\right)^6 \right]^{1/2} \cos(\omega_i'k)
\]

for \(-LM \leq k' \leq LM \). These coefficients must be weighted by an appropriate window, as in Eq. (14.4.9).
14.4. DESIGN EXAMPLES

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 specifications at $f_{\text{stop}} = 140$ kHz. The DAC response vanishes (i.e., it has infinite attenuation) at $f_{\text{stop}} = 160$ 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 $1/(H_{\text{dac}}(f)H_{\text{post}}(f))$.

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 $f_{\text{interp}} = 160$ 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.](image)

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]:

$$H_{\text{post}}(s) = \frac{15}{1 + 15\left(\frac{s}{\Omega_0}\right) + 6\left(\frac{s}{\Omega_0}\right)^2 + \left(\frac{s}{\Omega_0}\right)^3}$$

$|H_{\text{post}}(f)|^2 = \frac{225}{225 + 45\left(\frac{f}{f_0}\right)^2 + 6\left(\frac{f}{f_0}\right)^4 + \left(\frac{f}{f_0}\right)^6}$

where $\Omega_0 = 2\pi f_0$ and $f_0$ is related to the 3-dB frequency of the filter by $f_{3\text{dB}} = 1.75 f_0$

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 $f_{3\text{dB}} = 28$ kHz, we find the normalization frequency $f_0 = f_{3\text{dB}}/1.75 = 16$ 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:

$$A_{\text{post}}(f_{\text{pass}}) = 1.44 \text{ dB}, \quad A_{\text{post}}(f_{\text{stop}}) = 33 \text{ dB}$$

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:

$$D = \frac{A - 7.95}{14.36} = 2.998$$

$$N - 1 \geq \frac{DL}{\Delta F} = 95.93 \Rightarrow N = 97$$

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 reconstrcuting DAC and postfilter may also be equalized by the first stage. The overall system is shown in Fig. 14.4.24.
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:

$$|H_{\text{post}}(f)| = \frac{1}{\left[1 + \left(\frac{f}{f_0}\right)^6\right]^{3/2}} = \frac{1}{\left[1 + \left(\frac{\omega'}{\omega_0}\right)^6\right]^{3/2}} = D_{\text{post}}(\omega')$$

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:

$$D(\omega') = \begin{cases} \frac{L_0}{N_0} & \text{if } |\omega'| \leq \frac{\pi}{L_0} \\ 0 & \text{if } \frac{\pi}{L_0} < |\omega'| \leq \pi \end{cases} \quad (14.4.16)$$

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:

$$\tilde{d}(k) = \frac{L_0}{N_0} \left[ 1 + 2 \sum_{i=1}^{M_0} D_{\text{lin}}(\omega_i) D_{\text{dac}}(\omega_i) D_{\text{post}}(\omega_i) \cos(\omega_i'k) \right]$$

for $-L_0M_0 \leq k' \leq L_0M_0$. 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 $0 \leq \omega' \leq \pi/L_0$ 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 $0 \leq \omega' \leq \pi/L_0$, we can approximate the linear interpolator response by its analog equivalent ($\sin x/x$) response, which is independent of $L_1$:

$$D_{\text{lin}}(\omega') = \frac{\sin(\omega'/2)}{L_1\sin(\omega'/2L_1)} = \left| \frac{\sin(\omega'/2)}{\omega'/2} \right|^2$$

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
14.4. DESIGN EXAMPLES

\[ A_{\text{lin}}(f_{\text{stop}}) = -20 \log_{10} \left| \frac{\sin(\omega'_{\text{stop}}/2)}{L_1 \sin(\omega'_{\text{stop}}/2L_1)} \right|^2 \]

\[ A_{\text{dac}}(f_{\text{stop}}) = -20 \log_{10} \left| \frac{\sin(\omega'_{\text{stop}}/2L_1)}{\omega'_{\text{stop}}/2L_1} \right| \]

\[ A_{\text{post}}(f_{\text{stop}}) = 10 \log_{10} \left[ 1 + \left( \frac{f_{\text{stop}}}{f_0} \right)^6 \right] \]

From these equations, one may determine the 3-dB frequency \( f_0 \) of the postfilter in order that the total attenuation at \( f_{\text{stop}} \) be equal to a desired level, say \( A_{\text{tot}} \).

\[ A_{\text{tot}} = A_{\text{lin}}(f_{\text{stop}}) + A_{\text{dac}}(f_{\text{stop}}) + A_{\text{post}}(f_{\text{stop}}) \]

Similarly, one can calculate the attenuations of the linear interpolator, DAC, and postfilter at the edge of the passband, that is, at \( f_{\text{pass}} = f_s/2 \) or \( \omega'_{\text{pass}} = \pi/L_0 \)

\[ A_{\text{lin}}(f_{\text{pass}}) = -20 \log_{10} \left| \frac{\sin(\omega'_{\text{pass}}/2)}{L_1 \sin(\omega'_{\text{pass}}/2L_1)} \right|^2 \]

\[ A_{\text{dac}}(f_{\text{pass}}) = -20 \log_{10} \left| \frac{\sin(\omega'_{\text{pass}}/2L_1)}{\omega'_{\text{pass}}/2L_1} \right| \]

\[ A_{\text{post}}(f_{\text{pass}}) = 10 \log_{10} \left[ 1 + \left( \frac{f_{\text{pass}}}{f_0} \right)^6 \right] \]

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 \( L_1 f_0 \), 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 \( f_{\text{last}} = L_1 L_0 f_s - f_s/2 \),

\[ \omega'_{\text{last}} = \frac{2\pi f_{\text{last}}}{L_0 f_s} = \frac{\pi(2L_1 L_0 - 1)}{L_0} \]

the attenuations of the linear interpolator, DAC, and postfilter are

\[ A_{\text{lin}}(f_{\text{last}}) = -20 \log_{10} \left| \frac{\sin(\omega'_{\text{last}}/2)}{L_1 \sin(\omega'_{\text{last}}/2L_1)} \right|^2 \]

\[ A_{\text{dac}}(f_{\text{last}}) = -20 \log_{10} \left| \frac{\sin(\omega'_{\text{last}}/2L_1)}{\omega'_{\text{last}}/2L_1} \right| \]

\[ A_{\text{post}}(f_{\text{last}}) = 10 \log_{10} \left[ 1 + \left( \frac{f_{\text{last}}}{f_0} \right)^6 \right] \]

As a concrete design example,\(^1\) 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

\[ D = \frac{A - 7.95}{14.36} = 3.625 \]

\[ N_0 - 1 \geq \frac{D L_0}{\Delta F} = 115.98 \Rightarrow N_0 = 121 \]

The linear interpolator and DAC will have frequency responses:

\[ |H_1(f)| = \left| \frac{\sin(\pi f/4f_s)}{4 \sin(\pi f/16f_s)} \right|^2, \quad |H_{\text{dac}}(f)| = \left| \frac{\sin(\pi f/16f_s)}{\pi f/16f_s} \right| \]

It is evident that \( |H_1(f)| \) vanishes at all multiples of \( 4f_s \) which are not multiples of \( 16f_s \), whereas \( |H_{\text{dac}}(f)| \) vanishes at all non-zero multiples of \( 16f_s \).

Using Eqs. (14.4.17), we find that the requirement that at \( f_{\text{stop}} = L_0 f_s - f_s/2 = 140 \) 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 \( f_{\text{pass}} = 20 \) kHz, \( f_{\text{stop}} = 140 \) kHz, and \( f_{\text{last}} = 620 \) kHz, or equivalently, the normalized ones \( \omega'_{\text{pass}} = \pi/4, \omega'_{\text{stop}} = \pi/4, \) and \( \omega'_{\text{last}} = 31\pi/4 \).

<table>
<thead>
<tr>
<th>( f_{\text{pass}} )</th>
<th>( f_{\text{stop}} )</th>
<th>( f_{\text{last}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 kHz</td>
<td>140 kHz</td>
<td>620 kHz</td>
</tr>
<tr>
<td>( A_{\text{lin}} )</td>
<td>( 0.421 )</td>
<td>( 0.421 )</td>
</tr>
<tr>
<td>( A_{\text{dac}} )</td>
<td>( 0.014 )</td>
<td>( 0.014 )</td>
</tr>
<tr>
<td>( A_{\text{post}} )</td>
<td>( 0.018 )</td>
<td>( 0.018 )</td>
</tr>
<tr>
<td>( A_{\text{tot}} )</td>
<td>( 0.453 )</td>
<td>( 0.453 )</td>
</tr>
</tbody>
</table>

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 \( H_0(f), H_1(f), H_{\text{dac}}(f), H_{\text{post}}(f) \). It also shows the total interpolator response \( H_0(f) H_1(f) \), which removes the replicas at multiples of \( L_0 f_s \) 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 \( H_1(f) H_{\text{dac}}(f) H_{\text{post}}(f) \). It also shows the magnified passband of the
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 $mf_s = 40 \text{ kHz}$, 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.

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,

$$H_{\text{eq}}(f) = H_1(f)H_{\text{dac}}(f)H_{\text{post}}(f)$$

so that the overall reconstructor is:

$$H_{\text{rec}}(f) = H_0(f)H_{\text{eq}}(f)$$

Assuming that $H_0$ and $H_{\text{rec}}$ are normalized to unity at DC by dividing out a factor $L_0$, we write the corresponding attenuations in dB:

$$A_{\text{rec}}(f) = -20 \log_{10} |H_{\text{rec}}(f)|$$
$$A_0(f) = -20 \log_{10} |H_0(f)|$$
$$A_{\text{eq}}(f) = -20 \log_{10} |H_{\text{eq}}(f)|$$

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 $f_{\text{stop}}$ should be at least $A$ dB, that is, $A_{\text{rec}}(f_{\text{stop}}) \geq A$, and therefore

$$A_0(f_{\text{stop}}) + A_{\text{eq}}(f_{\text{stop}}) \geq A$$

(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:

$$A_0(f_{\text{stop}}) = A_0(f_{\text{pass}})$$
14. DESIGN EXAMPLES

But at frequency \( f_{\text{pass}} \), the filter \( H_0(f) \) is designed to equalize the total filter \( H_{\text{eq}}(f) \), that is,

\[
|H_0(f_{\text{pass}})| = \frac{1}{|H_{\text{eq}}(f_{\text{pass}})|} \Rightarrow \quad A_0(f_{\text{stop}}) = A_0(f_{\text{pass}}) = -A_{\text{eq}}(f_{\text{pass}})
\]

Therefore, we rewrite the design condition (14.4.20) as:

\[
A_{\text{eq}}(f_{\text{stop}}) - A_{\text{eq}}(f_{\text{pass}}) \geq A \quad (14.4.21)
\]

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—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 \( A_{\text{eq}} \) 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

\[
A_{\text{post}}(f_{\text{stop}}) - A_{\text{post}}(f_{\text{pass}}) \geq A - A_d
\]

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

\[
10 \log_{10} \left[ \frac{1 + (f_{\text{stop}}/f_0)^{2N_b}}{1 + (f_{\text{pass}}/f_0)^{2N_b}} \right] \geq A - A_d \quad (14.4.23)
\]

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 \( A_{\text{post}}(f_{\text{stop}}) \) can be ignored because it is much smaller than the stopband term \( A_{\text{post}}(f_{\text{pass}}) \), 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 \( f_{\text{stop}} \), 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

\[
20N_b \log_{10} \left( \frac{f_{\text{stop}}}{f_{\text{pass}}} \right) \geq A - A_d \quad \Rightarrow \quad N_b \geq \frac{A - A_d}{20 \log_{10} \left( \frac{f_{\text{stop}}}{f_{\text{pass}}} \right)}
\]

Because \( f_{\text{stop}}/f_{\text{pass}} = 2L_0 - 1 \), we can rewrite this condition as

\[
N_b \geq \frac{A - A_d}{20 \log_{10}(2L_0 - 1)} \quad (14.4.24)
\]

If Eq. (14.4.24) is satisfied, then the solution of Eq. (14.4.23) for \( f_0 \) is

\[
f_0 = f_{\text{pass}} \left[ \frac{(2L_0 - 1)^{2N_b} - 1}{10^{(A-A_d)/10} - 1} \right]^{1/2N_b} \quad (14.4.25)
\]

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 nearest integer above each curve. As \( L_0 \) increases, separating the \( L_0f_s \) 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 \( N_b \geq (60 - 33.123)/20 \log_{10}(8 - 1) = 1.59 \).

---

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 \( L \)th 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:

\[
X_{\text{down}}(n) = X'(n') \mid n' = nL = X'(nL) \tag{14.5.1}
\]

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 \([-fs/2, fs/2]\) by a factor of \(L\) to the new interval \([-fs/2, fs/2]\). Thus, if the signal had frequency components outside the new Nyquist interval, aliasing would occur and \(x_{\text{down}}(n) \neq x(n)\).

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:

\[
X(f) = X_{\text{down}}(f) = \frac{1}{L} \sum_{m=0}^{L-1} X'(f - mf_s) \tag{14.5.3}
\]

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.

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.

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:

\[
y_{\text{down}}(n) = y'(nL), \quad Y_{\text{down}}(f) = \frac{1}{L} \sum_{m=0}^{L-1} Y'(f - mf_s) \tag{14.5.4}
\]

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:

\[
h(n') = w(n') d(n' - LM), \quad \text{where } d(k') = \sin(\pi k'/L) / \pi k'
\]
where \( n' = 0, 1, \ldots, N - 1 \), and \( N = 2LM + 1 \). A Kaiser window \( w(n') \) may be used. The downsampled output is obtained by:

\[
y_{\text{down}}(n) = y'(n) = \sum_{m'=0}^{N-1} h(m') x'(n - m')
\]

(14.5.5)

Because only every \( L \)th output of the filter is needed, the overall computational rate is reduced by a factor of \( L \), that is,

\[
R = \frac{1}{L} N f_s' = N f_s
\]

(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 \( w = [w_0, w_1, \ldots, w_{N-1}] \) the \( N \)-dimensional internal state vector of the filter, we may state this filtering/downsampling algorithm as follows:

\[
\text{for each high-rate input sample } x' \text{ do:} \\
\qquad w_0 = x' \\
\text{for every } L \text{th input compute:} \\
\qquad y_{\text{down}} = \text{dot}(N - 1, h, w) \\
\qquad \text{delay}(N - 1, w)
\]

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:

\[
H(\zeta) = \frac{1}{L} \frac{1 - \zeta^{-L}}{1 - \zeta^{-1}} = \frac{1}{L} \left[ 1 + \zeta^{-1} + \zeta^{-2} + \cdots + \zeta^{-(L-1)} \right]
\]

(14.5.8)

where \( \zeta^{-1} \) is one high-rate delay. Thus, the decimator is a simple FIR averaging filter that averages \( L \) successive high-rate samples:

\[
y_{\text{down}}(n) = \frac{x'(n) + x'(n-1) + x'(n-2) + \cdots + x'(n-L+1)}{L}
\]

(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 \( f_{\text{Nyq}} = f_s/2 \). 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 \( f_s' = L f_s \). The decimation filter then reduces the bandwidth of the sampled signal to \( f_{\text{Nyq}}/2 \). The sharp cutoffs at the Nyquist frequency \( f_{\text{Nyq}}/2 \) 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 \([f_s/2, L f_s - f_s/2]\). But because of periodicity, it cannot remove any frequencies in the range \([L f_s - f_s/2, f_s/2]\). 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:

\[
f_{\text{pass}} = \frac{f_s}{2}, \quad f_{\text{stop}} = L f_s - \frac{f_s}{2}
\]

(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 \( N \).)

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.
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:

$$f_s' = \frac{L}{M} f_s$$  \hspace{1cm} (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 $T'$:

$$f_s'' = Lf_s = Mf_s', \hspace{1cm} T'' = \frac{T}{L} = \frac{T'}{M}$$  \hspace{1cm} (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 $f_c''/2L = f_s/2$ and the decimation filter $f_c''/2M = f_s'/2$. Thus, the cutoff frequency of the common filter must be chosen to be the minimum of the two:

$$f_c = \frac{1}{2} \min(f_s, f_s')$$  \hspace{1cm} (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:

$$f_c = \min\left(1, \frac{L}{M} f_s'\right) = \min\left(M f_s, 1\right) \frac{f_s'}{2} = \min\left(1, \frac{1}{M} f_s''\right) \frac{f_s'}{2}$$

In units of the high-rate digital frequency $\omega'' = 2\pi f_s''$, we have:

$$\omega_c'' = \frac{2\pi f_c}{f_s''} = \min\left(\frac{\pi}{L}, \frac{\pi}{M}\right) = \frac{\pi}{\max(L, M)}$$  \hspace{1cm} (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 $L f_s$. When $f_s' < f_s$, it acts as an antialiasing prefilter for the downsampler, making sure that the downshifted 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 $n'' = 0, 1, \ldots, N - 1$:

$$h_i(n'') = w(n'') d(n'' - LK), \quad d(k'') = L \sin\left(\frac{\omega_c'' k''}{\pi}\right)$$  \hspace{1cm} (14.6.5)

where $w(n'')$ is any desired length-$N$ window. Its $L$ polyphase subfilters of length $2K$ are defined for $i = 0, 1, \ldots, L - 1$:

$$h_i(n) = h(Ln + i), \quad n = 0, 1, \ldots, 2K - 1$$  \hspace{1cm} (14.6.6)

Next, we discuss the time-domain operation and implementation of the converter. The input signal $x(n)$ is upsamped to the high rate $f_s''$. Then, the upsampled input $x_{up}(n'')$ is filtered, generating the interpolated output $y_{up}(n'')$, 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:

$$y_{up}(n'') = \sum_{m=0}^{N-1} h(m'') x_{up}(n'' - m'') \quad \text{and} \quad y(n') = y_{up}(Mn')$$

The interpolation operation can be implemented efficiently in its polyphase realization. Setting $n'' = Ln + i$, with $i = 0, 1, \ldots, L - 1$, we obtain the $i$th sample interpolated between the input samples $x(n)$ and $x(n + 1)$, from Eq. (14.2.18):

\footnote{Here, we use $K$ instead of $M$ to avoid confusion with the downsampling factor $M$.}
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 \( w = [w_0, w_1, \ldots, w_p] \), which is used by all polyphase subfilters before it is updated.

Because the downsampled output is the only \( M \)th 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 \( 0 \leq i < L - 1 \). It follows that the downsampled output will be the \( i \)th interpolated value arising from the current input \( x(n) \) and computed as the output of the \( i \)th polyphase subfilter \( h_i \):

\[
y(n') = y_{up}(Mn') = y_{up}(Ln + i) = y_i(n)
\]

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:

\[
T_{block} = LT' = LMT'' = MT
\]

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, \( R = 2K/LT_{block} = N/MT_{block} = Nf_s/M \). Equivalently, we have one polyphase output in each output period \( T' \), \( R = 2K/T' = 2Kf_s' \). Thus,

\[
R = \frac{Nf_s}{M} = \frac{Nf_s'}{L} = 2Kf_s'
\]

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 \( T_{block} = 15T'' = 3T = 5T' \) 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.

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 \( h_1 \) with current input \( x_0 \). After this operation, the input sample \( x_0 \) is no longer needed and the delay-line \( w \) 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 \( h_1 \), whereas the output \( y_3 \) is the fourth \( (i = 4) \) interpolated value, or the output of the \( h_1 \). After this operation, the delay-line \( w \) 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 the \( h_1 \) 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:

\[
\text{for each input block } \{x_0, x_1, x_2\} \text{ do: }
\]

\[
\begin{align*}
w_0 &= x_0 \\
y_0 &= \text{dot}(P, h_0, w) = x_0 \\
y_1 &= \text{dot}(P, h_1, w) \\
\text{delay}(P, w) \\
w_0 &= x_1 \\
y_2 &= \text{dot}(P, h_1, w) \\
y_3 &= \text{dot}(P, h_1, w) \\
\text{delay}(P, w) \\
w_0 &= x_2 \\
y_4 &= \text{dot}(P, h_2, w) \\
\text{delay}(P, w)
\end{align*}
\]

The outputs \( \{y_0, y_1, y_2, y_3, y_4\} \) were computed by the five polyphase filters \( \{h_0, h_1, h_2\} \).
h₁,h₂,h₃} 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 the polyphase subfilters \( \{h₁, h₂, h₃\} \). The inputs \( \{x₂, x₃\} \) 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:

\[
\text{for each input block } \{x₀, x₁, x₂, x₃\} \text{ do:} \\
w₀ = x₀ \\
y₀ = \text{dot}(P, h₀, w) = x₀ \\
delay(P, w) \\
w₁ = x₁ \\
y₁ = \text{dot}(P, h₁, w) \\
delay(P, w) \\
w₂ = x₂ \\
y₂ = \text{dot}(P, h₂, w) \\
delay(P, w) \\
w₃ = x₃ \\
y₃ = \text{dot}(P, h₃, w) \\
delay(P, w) \\
\]

The inner loop ensures that the output time index \( m \) lies between the two input times \( L / M < Mm < L(n + 1) / M \), 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 \( f_s'' = 3f_s / 5 \). 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 \( f_s'' / 2 \leq f \leq f_s'' / 2 \).

In the time domain, each block of five input samples \( \{x₀, x₁, x₂, x₃, x₄\} \) generates a block of three output samples \( \{y₀, y₁, y₂\} \). The solution of Eq. (14.6.12) gives the polyphase selector sequences, for \( m = 0, 1, 2, \):

\[
\begin{align*}
\text{for each input block } \{x₀, x₁, x₂, x₃, x₄\} \text{ do:} \\
w₀ &= x₀ \\
y₀ &= \text{dot}(P, h₀, w) = x₀ \\
\text{delay}(P, w) \\
w₁ &= x₁ \\
y₁ &= \text{dot}(P, h₁, w) \\
\text{delay}(P, w) \\
w₂ &= x₂ \\
y₂ &= \text{dot}(P, h₂, w) \\
\text{delay}(P, w) \\
w₃ &= x₃ \\
y₃ &= \text{dot}(P, h₃, w) \\
\text{delay}(P, w) \\
w₄ &= x₄ \\
y₄ &= \text{dot}(P, h₄, w) \\
\text{delay}(P, w) \\
\end{align*}
\]

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 \( N = 64 \times 2^{16} = 2^{22} \approx 4 \times 10^6 \) and is realized in its polyphase form. Thus, there are \( L = 2^{16} \) polyphase filters of length \( 2^K = 64 \). The input delay-line buffer also has length 64. The computational rate of the chip is only \( R = (2K)f_s' = 64f_s' \) MAC/sec, where \( f_s' \) is the output rate.

The chip has a polyphase filter selector (like the quantity \( l_m \)) 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:

\[
f_c' = \rho f_s' = \frac{\omega''}{f_s} = \frac{2\pi f_c'}{f_s} = \rho \frac{\pi}{L}
\]

If \( \rho < 1 \), the corresponding ideal impulse response is:

\[
d_\rho(k'') = L \frac{\sin(\omega''k'')}{\pi k''} = \rho \frac{\sin(\pi \rho k''/L)}{\pi \rho k''/L}
\]

The fixed filter has response corresponding to \( \rho = 1 \):

\[
d(k'') = \frac{\sin(\pi k''/L)}{\pi k''/L} , \quad -LK \leq k'' \leq LK
\]

It follows that \( d_\rho(k'') \) will be the “stretched” version of \( d(k'') \):

\[
d_\rho(k'') = \rho d(\rho k'') \quad (14.6.15)
\]

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, \(-LK \leq \rho k'' \leq LK\), we must have in Eq. (14.6.15):

\[\frac{-1}{\rho}LK \leq k'' \leq \frac{1}{\rho}LK \]

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.
14.7. NOISE SHAPING QUANTIZERS

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.

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:

$$H(\zeta) = \frac{\zeta^{-1}}{1 - \zeta^{-1}} \quad (14.7.1)$$

where $\zeta^{-1}$ denotes a high-rate unit delay. The numerator delay $\zeta^{-1}$ is necessary to make the feedback loop computable.

Working with $\zeta$-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 $Y'(\zeta)$.

$$Y'(\zeta) = \frac{H(\zeta)}{1 + H(\zeta)} X'(\zeta) + \frac{1}{1 + H(\zeta)} E'(\zeta) \quad (14.7.2)$$

It can be written in the form:

$$Y'(\zeta) = H_s(\zeta) X'(\zeta) + H_{NS}(\zeta) E'(\zeta) \quad (14.7.3)$$

where the noise shaping transfer function $H_{NS}(\zeta)$ and the transfer function for the input $H_s(\zeta)$ are defined as:

$$H_s(\zeta) = \frac{H(\zeta)}{1 + H(\zeta)}, \quad H_{NS}(\zeta) = \frac{1}{1 + H(\zeta)} \quad (14.7.4)$$

Inserting $H(\zeta)$ from Eq. (14.7.1), we find for the first-order case:

$$H_s(\zeta) = \zeta^{-1}, \quad H_{NS}(\zeta) = 1 - \zeta^{-1} \quad (14.7.5)$$

Thus, $H_{NS}(\zeta)$ is a simple highpass filter, and $H_s(\zeta)$ an allpass plain delay. The I/O equation (14.7.3) becomes:

$$Y'(\zeta) = \zeta^{-1} X'(\zeta) + (1 - \zeta^{-1}) E'(\zeta) \quad (14.7.6)$$

or, in the time domain:

$$y'(n') = x'(n' - 1) + e(n') \quad (14.7.7)$$

where we defined the filtered quantization noise:
14.7. NOISE SHAPING QUANTIZERS

\[ \varepsilon(n') = e'(n') - e'(n' - 1) \Rightarrow \varepsilon(\zeta) = (1 - \zeta^{-1})e'(\zeta) \quad (14.7.8) \]

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:

\[ y(n) = x(n) + e(n) \quad (14.7.9) \]

where \( e(n) \) is modeled as white noise over \([−fs/2, fs/2] \).

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 \([−fs/2, fs/2] \). 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:

\[ \sigma_x^2 = \sigma_e^2 \int \frac{1}{fs} |H_{NS}(f)|^2 df \quad (14.7.10) \]

Setting \( fs' = Lfs \) and \( \sigma_x/\sigma_e = 2^{-B}/2^{-B'} = 2^{-DB}, \) where \( \Delta B = B - B' \), we obtain the desired relationship between \( L \) and \( \Delta B \) given by Eq. (2.2.10).

Higher-order delta-sigma quantizers have highpass noise shaping transfer functions of the form:

\[ H_{NS}(\zeta) = (1 - \zeta^{-1})^p \quad (14.7.11) \]

where \( p \) is the order. The input/output equations for such quantizers are still of the form of Eq. (14.7.3), where \( H_{NS}(\zeta) \) is typically a multiple delay. The frequency and magnitude responses of \( H_{NS}(\zeta) \) are obtained by setting \( \zeta = e^{2\pi if/fs'} \):

\[ H_{NS}(f) = \left(1 - e^{-2\pi if/fs'}\right)^p, \quad |H_{NS}(f)|^2 = 2 \sin \left(\frac{\pi f}{fs}L\right)^2 (14.7.12) \]

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:

\[ Q(x) = \text{sign}(x) = \begin{cases} +1, & \text{if } x \geq 0 \\ -1, & \text{if } x < 0 \end{cases} \quad (14.7.13) \]

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:

![Block Diagram of Delta-Sigma Quantizer](plot.png)

**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:

<table>
<thead>
<tr>
<th>( x )</th>
<th>( w_2 )</th>
<th>( y )</th>
<th>( v )</th>
<th>( w_0 )</th>
<th>( x )</th>
<th>( w_2 )</th>
<th>( y )</th>
<th>( v )</th>
<th>( w_0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>0.0</td>
<td>1.0</td>
<td>-0.6</td>
<td>-0.6</td>
<td>-0.2</td>
<td>0.0</td>
<td>1.0</td>
<td>-1.2</td>
<td>-1.2</td>
</tr>
<tr>
<td>0.4</td>
<td>-0.6</td>
<td>-1.0</td>
<td>1.4</td>
<td>0.8</td>
<td>-0.2</td>
<td>-1.2</td>
<td>-1.0</td>
<td>0.8</td>
<td>-0.4</td>
</tr>
<tr>
<td>0.4</td>
<td>0.8</td>
<td>1.0</td>
<td>-0.6</td>
<td>0.2</td>
<td>-0.2</td>
<td>-0.4</td>
<td>0.4</td>
<td>1.0</td>
<td>-1.2</td>
</tr>
<tr>
<td>0.4</td>
<td>0.2</td>
<td>1.0</td>
<td>-0.6</td>
<td>-0.4</td>
<td>-0.2</td>
<td>0.4</td>
<td>1.0</td>
<td>-1.2</td>
<td>-0.8</td>
</tr>
<tr>
<td>0.4</td>
<td>-0.4</td>
<td>-1.0</td>
<td>1.4</td>
<td>1.0</td>
<td>-0.2</td>
<td>-0.8</td>
<td>1.0</td>
<td>0.8</td>
<td>0.0</td>
</tr>
<tr>
<td>0.4</td>
<td>1.0</td>
<td>1.0</td>
<td>-0.6</td>
<td>0.4</td>
<td>-0.2</td>
<td>0.0</td>
<td>0.4</td>
<td>1.0</td>
<td>-1.2</td>
</tr>
<tr>
<td>0.4</td>
<td>0.4</td>
<td>1.0</td>
<td>-0.6</td>
<td>-0.2</td>
<td>-0.2</td>
<td>-1.2</td>
<td>0.8</td>
<td>0.0</td>
<td>-0.4</td>
</tr>
<tr>
<td>0.4</td>
<td>-0.2</td>
<td>-1.0</td>
<td>1.4</td>
<td>1.2</td>
<td>-0.2</td>
<td>-0.4</td>
<td>0.4</td>
<td>1.0</td>
<td>-1.2</td>
</tr>
<tr>
<td>0.4</td>
<td>1.2</td>
<td>1.0</td>
<td>-0.6</td>
<td>0.6</td>
<td>-0.2</td>
<td>0.4</td>
<td>1.0</td>
<td>-1.2</td>
<td>-0.8</td>
</tr>
<tr>
<td>0.4</td>
<td>0.6</td>
<td>1.0</td>
<td>-0.6</td>
<td>0.0</td>
<td>-0.2</td>
<td>-0.8</td>
<td>1.0</td>
<td>0.8</td>
<td>0.0</td>
</tr>
</tbody>
</table>

The average of the ten successive values of \( y \) are in the two cases, \( y = 0.4 \) and \( y = -0.2 \). 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:

\[ x'(n) = 0.5 \sin(2\pi fn'/fs'), \quad n' = 0, 1, \ldots, N_{\text{tot}} - 1 \]

We choose the values \( f_0 = 8.82 \text{ kHz}, f_s = 44.1 \text{ kHz}, L = 10, \) and \( N_{\text{tot}} = 200 \) samples. The fast rate is \( fs'/f_s = 10 \times 44.1 = 441 \text{ kHz}, \) 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:

\[ h(n') = w(n') \sin(\pi (n' - LM)/L) / \pi (n' - LM), \quad n' = 0, 1, \ldots, N - 1 \]

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
14.7. NOISE SHAPING QUANTIZERS

Fig. 14.7.3 Magnitude responses of decimation filters.

The Kaiser method was used to design 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,

$$|H(f)| = \left| \frac{\sin(\pi f / f_c)}{L \sin(\pi f / 10 f_c)} \right|$$

which does not vanish everywhere between $[f_c / 2, 10 f_c - f_c / 2]$, although it does vanish at the multiples $m f_c$, $m = 1, 2, \ldots, 9$.

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.

Fig. 14.7.5 Decimator filter output for rectangular and Kaiser designs.

The outputs of the quantizers $y'$ (this time) 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,

$$|H(f)| = \left| \frac{\sin(\pi f / f_c)}{L \sin(\pi f / 10 f_c)} \right|$$

which does not vanish everywhere between $[fs / 2, 10 fs - fs / 2]$, although it does vanish at the multiples $m fs$, $m = 1, 2, \ldots, 9$.

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.

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 $\zeta$-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:

$$Y'(\zeta) = X'(\zeta) + (1 - H(\zeta))E'(\zeta) = X'(\zeta) + H_{\text{NS}}(\zeta)E'(\zeta)$$  \hspace{1cm} (14.7.14)

Thus, the effective noise shaping filter is

$$H_{\text{NS}}(\zeta) = 1 - H(\zeta)$$  \hspace{1cm} (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:

$$H(\zeta) = \zeta^{-1} \Rightarrow H_{\text{NS}}(\zeta) = (1 - \zeta^{-1})$$

$$H(\zeta) = 2\zeta^{-1} - \zeta^{-2} \Rightarrow H_{\text{NS}}(\zeta) = (1 - \zeta^{-1})^2$$

Noise shaping requantizers are based on the same principle of feedback quantization as delta-sigma A/D converters. Therefore, the tradeoff between $L$ and $AB$ 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 $z = \zeta^{1/4}$ 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):

$$D(\zeta) = D_0(\zeta^{1/4}) + \zeta^{-1}D_1(\zeta^{1/4}) + \zeta^{-2}D_2(\zeta^{1/4}) + \zeta^{-3}D_3(\zeta^{1/4})$$

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 x 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,

$$y_{\text{up}}(3n + i) = a_ix(n + 2) + b ix(n + 1) + c_ix(n) + d_ix(n - 1)$$

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:

$$x(n) = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0\}$$

$$x(n) = \sin(2\pi F_1n), \quad n = 0, 1, \ldots, 24$$
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 \( y_{\text{int}}(n') \), \( n' = 0, 1, \ldots, 99 \), 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× 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:

\[
2 \times 4 = 4 \times 2 = 2 \times 2 = 8
\]

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 \( 0 \leq f \leq 320 \) kHz. Normalize them to 0 dB at DC. Plot also the total response of the stages, that is, \( H_{\text{tot}}(f) = H_0(f)H_1(f), \) (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 \( H_{\text{tot}}(f) \) 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 \( R_{\text{poly}}/R_{\text{single}} \) 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× 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 \( 0 \leq f \leq 176.4 \) 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 \( 0 \leq f \leq 20 \) kHz. On a separate graph, plot their phase response over the range \( 0 \leq f \leq 160 \) 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 \( f_s/L_1 \) into a staircase DAC, \( D_{\text{DAC}} \), and then into a final analog postfilter \( H_{\text{post}} \). Such a system is used, for example, in the Philips Bitstream 1-bit DAC system for CD players [535], with \( L_0 = 4, L_1 = 32, L_2 = 2 \).

a. Write expressions for the magnitude responses \( |H_0(f)|, |H_2(f)|, |H_{\text{DAC}}(f)| \), 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 the frequency range \( 0 \leq f \leq 16f_s \), 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 \( H_{\text{DAC}} \), and at the output of \( H_{\text{post}} \). What transition width did you choose for \( H_{\text{post}} \)?

d. Sketch the time-domain signals at the input and output of \( H_2 \) and the output of \( H_{\text{DAC}} \). 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 property:

\[
\frac{1}{L} \sum_{m=0}^{L-1} D(f − mf_s) = 1
\]

for all \( f \), where \( f_s' = f_s' / L \).

### 14.12 Consider the sampling of an analog signal \( x_n(t) \) at the two sampling rates \( f_s \) and \( f_s' = Lf_s \). The corresponding digital signals are \( x(nT) = x_n(nT) \) and \( x'(nT') = x_n'(nT') \). 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, \( x(n) = x_n(nT) = x_n(nLT') = x'(nT') \). The spectra of \( x(n) \) and \( x'(n') \) are given by the Poisson summation formulas:

\[
X(f) = \frac{1}{T} \sum_{k=−\infty}^{\infty} X_d(f − kf_s), \quad X'(f) = \frac{1}{T'} \sum_{k=−\infty}^{\infty} X_d(f − k'f_s')
\]

Using the change of variables \( k = k'L + m \), where \( m = 0, 1, \ldots, L−1 \), show that the spectrum of the downsampled signal is given by the discrete-time version of the Poisson summation formula:

\[
X(f) = \frac{1}{L} \sum_{m=0}^{L-1} X'(f − mf_s)
\]

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

\[
X(\omega) = \frac{1}{L} \sum_{m=0}^{L-1} X'(\omega' − \frac{2\pi m}{L})
\]
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:

$$x_{up}(n') = \sum_{n=-\infty}^{\infty} x'(nL) \delta(n' - nL) = s'(n') x'(n'), \quad \text{where} \quad s'(n') = \sum_{n=-\infty}^{\infty} \delta(n' - nL)$$

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:

$$s'(n') = \sum_{n=-\infty}^{\infty} \delta(n' - nL) = \frac{1}{L} \sum_{m=0}^{L-1} e^{2\pi j mn'/L}$$

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 $A_{\text{stop}} \text{dB}$ is given approximately by:

$$L = 0.94 \cdot 10^{A_{\text{stop}}/60} + 0.5$$

Make a plot of the above formula versus $A_{\text{stop}}$ in the range $20 < A_{\text{stop}} < 100$ 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 $\{A_{\text{pass}}, A_{\text{stop}}, f_{\text{pass}}, f_{\text{stop}}\}$, as shown in Fig. 14.5.4, is given by:

$$N = \left\lfloor \ln \left( \frac{10^{A_{\text{pass}}/50} - 1}{10^{A_{\text{stop}}/50} - 1} \right) \right\rfloor / 2 \ln (2L - 1)$$

Determine $N$ for the values $A_{\text{pass}} = 0.1 \text{ dB}$, $A_{\text{stop}} = 60 \text{ dB}$, $L = 16$. Round $N$ up to the next integer, say $N_0$. For what range of $L$s 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 $N$? 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 $l_m, n_m$, 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 $L, M, \text{stopband attenuation}$, 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, $x(t) = \sin(2\pi t/100)$ where $0 \leq t \leq 10$ 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:

$$x(n) = \sin(2\pi n/30), \quad n = 0, 1, \ldots, 29$$
$$x'(n') = \sin(2\pi n'/50), \quad n' = 0, 1, \ldots, 49$$
$$x''(n'') = \sin(2\pi n''/150), \quad n'' = 0, 1, \ldots, 149$$

Design a 5/3 converter that has $A = 30 \text{ dB}$ 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.

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 $H_0(\zeta)$ and $H_{NS}(\zeta)$? 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
14.8 PROBLEMS

701

14. INTERPOLATION, DECIMATION, AND OVERSAMPLING

\[ H_x(\zeta) = \zeta^{-1}, \quad H_{NS}(\zeta) = (1 - \zeta^{-1})^2 \]

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 \texttt{fir} or \texttt{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. A \textit{-fold “comb” decimator is defined by:}

\[ H(\zeta) = \frac{1}{L} \frac{1 - \zeta^{-L}}{1 - \zeta^{-1}} \]

It is similar to a linear interpolator normalized by \(L\). For \(L = 10\), determine its impulse response \(h\). 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.](image)

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

\[ Y'(\zeta) = X'(\zeta) + (1 - \zeta^{-1})E_2'(\zeta) \]

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 \(H_1(\zeta)\) and \(H_2(\zeta)\) in terms of the loop filters \(H_1(\zeta)\) and \(H_2(\zeta)\). Then, determine \(H_1(\zeta)\) and \(H_2(\zeta)\) such that

\[ H_x(\zeta) = 1, \quad H_{NS}(\zeta) = (1 - \zeta^{-1})^2 \]

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. 14.8.2 Alternative model of first-order delta-sigma quantizer.](image)

14.25 An alternative discrete-time model for the second-order \(\Delta\Sigma\) quantizer is obtained by removing the delay \(\zeta^{-1}\) 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:

\[ H_x(\zeta) = \zeta^{-1}, \quad H_{NS}(\zeta) = (1 - \zeta^{-1})^2 \]

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.](image)

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

\[ Y'(\zeta) = X'(\zeta) + (1 - \zeta^{-1})E_2'(\zeta) \]

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 \(H_1(\zeta)\) and \(H_2(\zeta)\) in terms of the loop filters \(H_1(\zeta)\) and \(H_2(\zeta)\). Then, determine \(H_1(\zeta)\) and \(H_2(\zeta)\) such that

\[ H_x(\zeta) = 1, \quad H_{NS}(\zeta) = (1 - \zeta^{-1})^2 \]

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. 14.8.2 Alternative model of first-order delta-sigma quantizer.](image)

14.25 An alternative discrete-time model for the second-order \(\Delta\Sigma\) quantizer is obtained by removing the delay \(\zeta^{-1}\) 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:

\[ H_x(\zeta) = \zeta^{-1}, \quad H_{NS}(\zeta) = (1 - \zeta^{-1})^2 \]
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:

\[
\begin{align*}
Y_1' &= X' + DE_1' \\
Y_2' &= -E_1' + DE_2' \quad \Rightarrow \quad Y' = X' + DE_3' \\
Y_3' &= -E_2' + DE_3'
\end{align*}
\]

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

### 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:

\[
x(n) = s(n) + v(n)
\]  

(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:

\[
y'(n) = y_s(n) + y_v(n)
\]  

(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:

\[
\begin{align*}
y_s(n) &= s(n) \\
y_v(n) &= 0
\end{align*}
\]

(15.1.3)

\(^\dagger\)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:

\[
\begin{align*}
Y_s(\omega) &= H(\omega)S(\omega) = S(\omega) \\
Y_v(\omega) &= H(\omega)V(\omega) = 0
\end{align*}
\]  

(15.1.4)

The first requires that \( H(\omega) \neq 0 \) for all \( \omega \) for which the signal spectrum is nonzero, \( S(\omega) \neq 0 \). The second requires that \( H(\omega) = 0 \) at all \( \omega \) for which the noise spectrum is nonzero, \( V(\omega) \neq 0 \).

These two conditions can be met simultaneously if the signal and noise spectra separately 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.](image)

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 \( \omega \) such that both \( S(\omega) \neq 0 \) and \( V(\omega) \neq 0 \) and therefore the conditions (15.1.4) would require \( H(\omega) = 1 \) at all \( \omega \) for the same \( \omega \).

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.

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 \( \sigma_{yv}^2 = E[y_v(n)^2] \), we have the definition,

\[
R = \frac{\sigma_{yv}^2}{\sigma_v^2} = \int_{-\pi}^{\pi} |H(\omega)|^2 \frac{d\omega}{2\pi} = \sum h_n^2 \tag{NRR} \]

(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:

\[
R = \frac{\sigma_{yv}^2}{\sigma_v^2} = \int_{-\omega_c}^{\omega_c} \frac{d\omega}{2\pi} = \frac{2\omega_c}{2\pi} = \frac{\omega_c}{\pi} \tag{15.1.6}
\]

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:

\[
R = \frac{\sigma_{yv}^2}{\sigma_v^2} = \frac{\omega_b - \omega_a}{\pi} \tag{15.1.7}
\]

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:

\[
\text{SNR}_{\text{in}} = \frac{E[s(n)^2]}{E[v(n)^2]}, \quad \text{SNR}_{\text{out}} = \frac{E[y_v(n)^2]}{E[y_v(n)^2]} \]

![Fig. 15.1.2 Signal enhancement filter with partial noise reduction.](image)
Therefore, the relative improvement in the SNR introduced by the filter will be:

\[
\frac{\text{SNR}_{\text{out}}}{\text{SNR}_{\text{in}}} = \frac{E[y_s(n)^2]}{E[y_v(n)^2]} \cdot \frac{E[v(n)]^2}{E[s(n)]^2} = \frac{1}{R}
\]

If the desired signal is not changed by the filter, \(y_s(n) = s(n)\), then

\[
\frac{\text{SNR}_{\text{out}}}{\text{SNR}_{\text{in}}} = \frac{1}{R} \tag{15.1.8}
\]

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 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:

\[
y_s(n) = \int_{-\omega_c}^{\omega_c} S(\omega) e^{j \omega (n-D)} \frac{d\omega}{2\pi} = \int_{-\omega_c}^{\omega_c} S(\omega) e^{j \omega n} \frac{d\omega}{2\pi} = s(n-D)
\]

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

\[x(n) = s(n) + v(n)\]

where \(v(n)\) is zero-mean white Gaussian noise of variance \(\sigma_v^2\). 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 \(\lambda\) 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 \(\alpha\) of the filter:

\[
|H(z)|_{z=1} = \frac{\alpha}{1 - \lambda} = 1 \quad \Rightarrow \quad \alpha = 1 - \lambda
\]

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

\[
\mathcal{R} = \frac{\sigma_s^2}{\sigma_v^2} = \sum_{n=0}^{\infty} h_n^2 = \alpha^2 \sum_{n=0}^{\infty} \lambda^{2n} = \frac{\alpha^2}{1 - \lambda^2} = \frac{(1 - \lambda)^2}{1 - \lambda^2} = \frac{1 - \lambda^2}{1 + \lambda}
\]

This ratio is always less than one because \(\lambda\) is restricted to the range \(0 < \lambda < 1\). To achieve high noise reduction, \(\lambda\) must be chosen near one. But, then the filter’s transient time constant, given by Eq. (6.3.12), will become large:

\[
n_{\text{eff}} = \frac{\ln \epsilon}{\ln \lambda} \rightarrow \infty \quad \text{as} \quad \lambda \rightarrow 1
\]

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 \(\omega_c\) can be calculated by requiring that \(|H(\omega_c)|^2\) drops by 1/2, that is,

\[
|H(\omega_c)|^2 = \frac{\alpha^2}{1 - 2\lambda \cos \omega_c + \lambda^2} = \frac{1}{2}
\]
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 \( \cos x \approx 1 - x^2/2 \) and solve for \( \omega_c \) approximately:

\[
\omega_c \approx 1 - \lambda
\]

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 \( \alpha = 1 - \lambda \)

\[
y(n) = \alpha y(n-1) + \alpha x(n)
\]

and implemented with the sample processing algorithm, where \( w_1(n) = y(n-1) \)

\[
\text{for each input sample } x \text{ do: }
\begin{align*}
y &= \alpha w_1 + \alpha x \\
w_1 &= y
\end{align*}
\]

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 \texttt{gran} of Appendix A.1. The figure on the left corresponds to \( \alpha = 0.90 \), which has 1-percent time constant and NRR:

\[
n_{\text{eff}} = \frac{\ln(0.01)}{\ln(0.90)} = 44, \quad R = \frac{1 - 0.90}{1 + 0.90} = \frac{1}{19}
\]

It corresponds to an improvement of the SNR by, \( 10 \log_{10}(1/R) = 12.8 \) dB. The right figure has \( \lambda = 0.98 \), with a longer time constant and smaller NRR:

\[
n_{\text{eff}} = \frac{\ln(0.01)}{\ln(0.98)} = 228, \quad R = \frac{1 - 0.98}{1 + 0.98} = \frac{1}{99}
\]

and an SNR improvement by, \( 10 \log_{10}(1/R) = 20 \) 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:

\[
y(n) = \alpha \sum_{m=0}^{n} \lambda^m x(n-m) = \alpha \{x(n) + \lambda x(n-1) + \lambda^2 x(n-2) + \cdots + \lambda^n x(0)\}
\]

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 are averaged out. The closer \( \lambda \) is to 1, the more equal weighting the terms get, resulting in more effective averaging of the noise. The \textit{exponential weighting} de-emphasizes the higher frequencies present in \( s(n) \). Because of the exponential weights, this filter is also called an \textit{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 \( \omega_c \), 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:

\[
H(z) = \alpha \frac{1 + z^{-1}}{1 - \lambda z^{-1}} \implies |H(\omega)|^2 = \frac{2\alpha^2(1 + \cos \omega)}{1 - 2\lambda \cos \omega + \lambda^2}
\]

(15.2.2)

where \( \alpha \) is fixed by requiring unity gain at DC:

\[
H(1) = \frac{2\alpha}{1 - \lambda} = 1 \implies \alpha = \frac{1 - \lambda}{2}
\]
15.2. IIR EXPONENTIAL SMOOTHER

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 $\lambda$. 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

$$A_0 = -\frac{\alpha}{\lambda}, \quad A_1 = \frac{\alpha(1 + \lambda)}{\lambda}$$

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

$$R = \sum_n h_n^2 = h_0^2 + \sum_{n=1}^\infty h_n^2 = \alpha^2 + A_1^2 \frac{\lambda^2}{1 - \lambda^2} = 1 - \lambda^2$$

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 $\omega_c$ in terms of $\lambda$:

$$\cos \omega_c = \frac{2\lambda}{1 + \lambda^2}$$

(15.2.3)

Conversely, we can solve for $\lambda$ in terms of $\omega_c$:

$$\lambda = \frac{1 - \sin \omega_c}{\cos \omega_c} = \frac{1 - \tan(\omega_c/2)}{1 + \tan(\omega_c/2)}$$

(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 $z \rightarrow -z$ 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 $\sigma_v^2$. Because, the signal band is now at the Nyquist frequency $\omega = \pi$, we may use a first-order highpass IIR filter:

$$H(z) = \frac{\alpha}{1 + \lambda z^{-1}} \quad H(\omega) = \frac{\alpha}{1 + \lambda e^{-j\omega}}, \quad |H(\omega)|^2 = \frac{\alpha^2}{1 + 2\alpha \cos \omega + \lambda^2}$$

where $0 < \lambda < 1$. The gain $\alpha$ is fixed such that $H(\pi) = 1$, or equivalently $H(z) = 1$ at $z = e^{j\pi} = -1$, which gives the condition:

$$H(z) \big|_{z=-1} = \frac{\alpha}{1 - \lambda} = 1 \quad \Rightarrow \quad \alpha = 1 - \lambda$$

The impulse response is now, $h_n = \alpha(-\lambda)^n u(n)$. The corresponding NRR can be calculated as in the previous example:

$$R = \sum_n h_n^2 = \alpha^2 \sum_{n=0}^\infty (-\lambda)^{2n} = \frac{\alpha^2}{1 - \lambda^2} = \frac{1 - \lambda}{1 + \lambda}$$

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 $\omega_c$ is the same as in the previous example.

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:

$$n_{eff} = \frac{\ln(0.01)}{\ln(0.97)} = 151, \quad R = \frac{1 - 0.97}{1 + 0.97} = \frac{3}{197} = 0.0101$$

which corresponds to an SNR improvement by, $10 \log_{10}(1/R) = 19.96$ dB.
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 \( z = Re^{j\omega_0} \), 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:

\[
H(z) = \frac{G}{1 + a_1 z^{-1} + a_2 z^{-2}}, \quad h_n = \frac{G}{\sin \omega_0} R^n \sin(\omega_0 n + \omega_0) u(n)
\]

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:

\[
\mathcal{R} = \sum_{n=0}^{\infty} h_n^2 = \frac{(1 - R)(1 + R^2)(1 - 2R \cos(2\omega_0) + R^2)}{(1 + R)(1 - 2R \cos(2\omega_0) + R^4)}
\]

(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 \textsc{gran}. The output was computed by the sample processing algorithm of the filter:

\[
\text{for each input sample } x \text{ do: } \\
y = -d_1 w_1 - d_2 w_2 + G x \\
W_2 = W_1 \\
W_1 = y
\]

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, \( n_{\text{ph}}(\omega_0) = -\arg H(\omega_0) / \omega_0 \).

For the given numerical values, we find, \( n_{\text{ph}}(\omega_0) = 3.95 \) 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:

\[
H(z) |_{z=1} = h_0 + h_1 + h_2 + h_3 = 1 \quad (15.5.1)
\]

The NRR of this filter will be simply:

\[
\mathcal{R} = \sum h_n^2 = h_0^2 + h_1^2 + h_2^2 + h_3^2 \quad (15.5.2)
\]

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
\[ R = h_0^2 + h_1^2 + h_2^2 + (h_0 + h_1 + h_2 - 1)^2 \]

The minimization of this expression can be carried out easily by setting the partial derivatives of \( R \) to zero and solving for the \( h \)’s:
\[
\frac{\partial}{\partial h_0} R = 2h_0 + 2(h_0 + h_1 + h_2 - 1) = 2(h_0 - h_3) = 0 \\
\frac{\partial}{\partial h_1} R = 2h_1 + 2(h_0 + h_1 + h_2 - 1) = 2(h_1 - h_3) = 0 \\
\frac{\partial}{\partial h_2} R = 2h_2 + 2(h_0 + h_1 + h_2 - 1) = 2(h_2 - h_3) = 0
\]

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:
\[ R_{\text{min}} = \left( \frac{1}{4} \right)^2 + \left( \frac{1}{4} \right)^2 + \left( \frac{1}{4} \right)^2 + \left( \frac{1}{4} \right)^2 = 4 \cdot \left( \frac{1}{4} \right)^2 = \frac{1}{4} \]

The I/O equation for this optimum smoothing filter becomes:
\[ y(n) = \frac{1}{N} \{ x(n) + x(n-1) + x(n-2) + x(n-3) \} \]

More generally, the optimum length-\( N \) FIR filter with unity DC gain and minimum NRR is the filter with equal weights:
\[ h_n = \frac{1}{N}, \quad n = 0, 1, \ldots, N-1 \] (15.5.3)

and I/O equation:
\[ y(n) = \frac{1}{N} \{ x(n) + x(n-1) + x(n-2) + \cdots + x(n-N+1) \} \] (15.5.4)

Its NRR is:
\[ R = h_0^2 + h_1^2 + \cdots + h_{N-1}^2 \cdot \left( \frac{1}{N} \right)^2 = \frac{1}{N} \]

(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 \( n_{\text{eff}} \) 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 \( \lambda = 0.90 \), then \( N = n_{\text{eff}} = 44 \) as in Example 15.2. But then, the NRR of the FIR smoother will be \( R = 1/N = 1/44 \), which is better than that of the IIR filter, \( R = 1/19 \). 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.

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:
\[ R = \frac{1 - \lambda}{1 + \lambda} = \frac{1}{N} \quad \Rightarrow \quad N = \frac{1 + \lambda}{1 - \lambda}, \quad \Rightarrow \quad \lambda = \frac{N-1}{N+1} \] (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 \( n_{\text{eff}} \) and \( N \) can be derived as follows. Using the small-\( \lambda \) approximation \( \ln (1 + \lambda) / (1 - \lambda) = 2 \lambda \), we have for large \( N \):
\[ \ln (1/\lambda) = \ln \left( \frac{1 + 1/N}{1 - 1/N} \right) \right) \approx 2x \]

It follows that,
\[ n_{\text{eff}} = \frac{\ln (1/\epsilon)}{\ln (1/\lambda)} \approx N \frac{1}{2} \ln \left( \frac{1}{\epsilon} \right) \]
Typically, the factor $(\ln(1/\epsilon))/2$ is greater than one, resulting in a longer IIR time constant $n_{\text{eff}}$ than $N$. For example, we have:

\[
\begin{align*}
    n_{\text{eff}} &= 1.15 N, \quad \text{if } \epsilon = 10^{-1} \quad (10\% \text{ time constant}) \\
    n_{\text{eff}} &= 1.50 N, \quad \text{if } \epsilon = 5 \cdot 10^{-2} \quad (5\% \text{ time constant}) \\
    n_{\text{eff}} &= 2.30 N, \quad \text{if } \epsilon = 10^{-2} \quad (1\% \text{ or } 40 \text{ dB time constant}) \\
    n_{\text{eff}} &= 3.45 N, \quad \text{if } \epsilon = 10^{-3} \quad (0.1\% \text{ or } 60 \text{ dB time constant})
\end{align*}
\]

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:

\[
H(z) = \frac{1}{N} \left( 1 + z^{-1} + z^{-2} + \cdots + z^{-(N-1)} \right) = \frac{1 - z^{-N}}{1 - z^{-1}} \tag{15.5.7}
\]

Setting, $z = e^{j\omega}$, we obtain the frequency response:

\[
H(\omega) = \frac{1}{N} \frac{1 - e^{-jN\omega}}{1 - e^{-j\omega}} = \frac{1}{N} \frac{\sin(N\omega/2)}{\sin(\omega/2)} e^{-jN\omega(N-1)/2} \tag{15.5.8}
\]

which has a linear phase response. The transfer function (15.5.7) has zeros at the $N$th roots of unity, except at $z = 1$, that is,

\[z_k = e^{j\omega_k}, \quad \omega_k = \frac{2\pi k}{N}, \quad k = 1, 2, \ldots, N - 1\]

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 $180^\circ$ 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.](image)

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:

\[
\left| \frac{1}{N} \frac{\sin(N\pi/2N)}{\sin(\pi/2N)} \right|^2 \approx \left| \frac{1}{N} \frac{\sin(\pi/2)}{\sin(\pi/2N)} \right|^2 = \left| \frac{1}{2} \right|^2
\]

where we used the approximation $\sin(\pi/2N) \approx \pi/2N$, for large $N$. In decibels, we have $-10 \log_{10}(1/2) = 3.9$ dB. (Thus, $\omega_c$ 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,

\[
\left| \frac{\sin(\pi x)}{\pi x} \right|^2 = \frac{1}{2}
\]

with solution, $x \approx 0.443$, resulting in the following useful approximation for the 3-dB cutoff frequency of the FIR averager,

\[
\omega_c = 0.443 \frac{2\pi}{N} = \frac{0.886\pi}{N} \tag{15.5.10}
\]

In MATLAB, the solution of Eq. (15.5.9) can be obtained by the command,

\[x = \text{fzero}(@x \sin(x), \sqrt{2} - 0.5, 0.5); \quad \% \text{resulting in } x = 0.44294\ldots\]

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 $n(n)$, 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 $\omega_c$ 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 \( \omega_{\text{max}} \), then we may choose \( N \) such that \( \omega_c \geq \omega_{\text{max}} \), which gives \( N \leq \pi/\omega_{\text{max}} \) and in units of Hz, \( N \leq f_s/2f_{\text{max}} \).

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:

\[
y(n) = y(n-1) + \frac{1}{N} (x(n)-x(n-N)) \quad \text{(direct form)} \tag{15.5.11}
\]

and the canonical realization by the system of equations:

\[
w(n) = x(n) + w(n-1) \\
y(n) = \frac{1}{N} (w(n)-w(n-N)) \quad \text{(canonical form)} \tag{15.5.12}
\]

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 \( (x(n)-x(n-N))/N \) 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:

\[
H(z) = \frac{1}{N} \sum_{n=0}^{N-1} (\rho^N z^{-n})^{-1} = 1 - \frac{z^{-1}}{1 + (1 - \rho^{-1})z^{-1}}
\]

where \( 0 < \rho \leq 1 \). If so desired, the filter may be renormalized to unity gain at DC resulting in

\[
H(z) = \frac{1}{\rho^N} \frac{1 - \rho^N z^{-N}}{1 - \rho z^{-1}}
\]

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 \( \rho \), then we can set approximately \( \rho^N \approx 0 \). 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:

\[
H(z) = \sum_n h_n z^{-n} \quad \to \quad H(-z) = \sum_n h_n (-z)^{-n} = \sum_n (-1)^n h_n z^{-n}
\]

and to the change in the impulse response:

\[
h_n \to (-1)^n h_n
\]

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:

\[
h_n = (-1)^n \frac{1}{N}, \quad n = 0, 1, \ldots, N-1 \tag{15.6.1}
\]

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,

\[
H(-1) = \sum_{n=0}^{N-1} (-1)^n h_n = 1 \tag{15.6.2}
\]

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 \( R \), 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,
15.7. NOISE REDUCTION, TIME CONSTANT, GROUP DELAY

\[ R = \sum_{n=0}^{\infty} h^2(n) = \int_{-\pi}^{\pi} |H(\omega)|^2 \frac{d\omega}{2\pi} = \int_{u.c.} H(z)H(z^{-1}) \frac{dz}{2\pi jz} \] (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, \(|p|_{\text{max}}\), determines the effective time constant, given as follows in time samples or in seconds,

\[ n_{\text{eff}} = \frac{\ln(c)}{\ln|p|_{\text{max}}} \Rightarrow t_{\text{eff}} = \frac{1}{f_s} n_{\text{eff}} \] (15.7.2)

where, \( f_s = 1/T \) is the sampling rate, and \( c \) is a small user-defined constant, such as, \( c = 10^{-2} \), or, \( c = 10^{-3} \). These two choices define the so-called 40-dB and 60-dB time constants, since in dB the values of \( c \) are, \( \epsilon_{\text{dB}} = 20 \log_{10}(c) = -40 \), and \( \epsilon_{\text{dB}} = -60 \) 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,\(^1\)

\[
\begin{align*}
H(\omega) &= |H(\omega)| e^{j\theta(\omega)} \\
n_{\text{ph}}(\omega) &= -\frac{\theta(\omega)}{\omega} \\
n_{\text{gr}}(\omega) &= -\theta'(\omega) = -\frac{d\theta(\omega)}{d\omega}
\end{align*}
\] (15.7.3)

The group delay is also known as the filter lag. It can be shown that \( n_{\text{gr}}(\omega) \) is expressible directly in terms of \( H(\omega) \), or in terms of \( h(n) \), by,

\[
n_{\text{gr}}(\omega) = -\theta'(\omega) = -\text{Re} \left[ \frac{\sum_{n=0}^{\infty} h(n)e^{-j\omega n}}{\sum_{n=0}^{\infty} h(n)e^{-j\omega n}} \right]
\] (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 sine wave 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,

\[
y(n) = |H(\omega_0)| \cdot \cos(\omega_0 n + \theta(\omega_0)) \cdot p(n - n_{\text{gr}}(\omega_0))
\] (15.7.5)

that is, the envelope \( p(n) \) gets delayed by the group delay at \( \omega_0 \), whereas the sinusoid is modified in magnitude by the magnitude response, \( |H(\omega_0)| \), and shifted in phase by the phase delay at frequency \( \omega_0 \).

To derive this result, consider the complex sinusoid \( e^{j\omega_0 n} \) being modulated by \( p(n) \), and its DTFT,

\[
x(n) = e^{j\omega_0 n} p(n) \Rightarrow X(\omega) = P(\omega - \omega_0)
\]

where \( P(\omega) \) is the DTFT of \( p(n) \),

\[
P(\omega) = \sum_{n} p(n)e^{-j\omega n} \Rightarrow p(n) = \int_{-\pi}^{\pi} P(\omega)e^{j\omega n} \frac{d\omega}{2\pi}
\]

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,

\[
y(n) = \int_{-\pi}^{\pi} Y(\omega)e^{j\omega n} \frac{d\omega}{2\pi} = \int_{-\pi}^{\pi} H(\omega)P(\omega - \omega_0)e^{j\omega n} \frac{d\omega}{2\pi}
\] (15.7.6)

Since \( P(\omega) \) is highly concentrated at DC, it follows that \( P(\omega - \omega_0) \) will be highly concentrated about \( \omega_0 \). Therefore inside the integral of Eq. (15.7.6), we may expand \( H(\omega) \) in the neighborhood of \( \omega_0 \), that is, to the lowest order in the variable, \( \omega - \omega_0 \), and assume the following approximations for the magnitude and phase responses,

\[
|H(\omega)| \approx |H(\omega_0)| \\
\theta(\omega) \approx \theta(\omega_0) + (\omega - \omega_0) \theta'(\omega_0) = \theta(\omega_0) - (\omega - \omega_0) n_{\text{gr}}(\omega_0)
\]

Thus, the output signal will be in the time domain,

\[
y(n) = \int_{-\pi}^{\pi} H(\omega_0)P(\omega - \omega_0)e^{j\omega n} \frac{d\omega}{2\pi}
\]

\[
= \int_{-\pi}^{\pi} |H(\omega)|e^{j\theta(\omega_0) - j(\omega - \omega_0)n_{\text{gr}}(\omega_0)} P(\omega - \omega_0)e^{j\omega n} \frac{d\omega}{2\pi}
\]

\[
= |H(\omega_0)|e^{j\theta(\omega_0)}e^{j\omega_0 n} \int_{-\pi}^{\pi} e^{-j(\omega - \omega_0)n_{\text{gr}}(\omega_0)} P(\omega - \omega_0)e^{j\omega n} \frac{d\omega}{2\pi}
\]

\[
= |H(\omega_0)|e^{j\theta(\omega_0)}e^{j\omega_0 n} \int_{-\pi}^{\pi} P(\omega - \omega_0)e^{j\omega_0(n - n_{\text{gr}}(\omega_0))} \frac{d\omega}{2\pi}
\]

\[
= |H(\omega_0)|e^{j\omega_0 n + j\theta(\omega_0)} p\left(n - n_{\text{gr}}(\omega_0)\right)
\]

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).

---

\(^1\)We assume that the angle \( \theta(\omega) \) is reduced modulo-2\( \pi \) to the standard interval \([-\pi, \pi]\)
15.7. NOISE REDUCTION, TIME CONSTANT, GROUP DELAY

A particularly important case is the filter lag at DC, $\omega = 0$, denoted by, $\hat{n} = n_{gr}(0)$, and given as follows as a special case of Eq. (15.7.4),

$$\hat{n} = n_{gr}(0) = \sum_{n=0}^{\infty} nh(n)$$ (filter lag) (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,

$$\hat{n} = \sum_{n=0}^{\infty} nh(n)$$ (15.7.8)

where we used the assumed fact that,

$$H(\omega)_{\omega=0} = \sum_{n=0}^{\infty} h(n) = 1$$

The DC filter lag $\hat{n}$ 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, $\hat{n} = 0$, 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 $\hat{n}$, 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 $\hat{n}$, that is,

$$x(n) = a + bn \longrightarrow H \longrightarrow y(n) = a + b(n - \hat{n})$$ (15.7.9)

Passing through a cascade of two identical filters $H(z)$, the lag will be doubled,

$$x(n) = a + bn \longrightarrow H \longrightarrow H \longrightarrow y(n) = a + b(n - 2\hat{n})$$

By forming the new filter,

$$H_a(z) = 2H(z) - H^2(z)$$ (zero-lag filter) (15.7.10)

the lag can be eliminated,

$$x(n) = a + bn \longrightarrow H_a \longrightarrow y(n) = a + bn$$

Indeed, the output of $2H(z)$ will be, $2[a + b(n - \hat{n})]$, and upon subtracting the output of $H^2(z)$, we obtain,

$$y(n) = 2[a + b(n - \hat{n})] - [a + b(n - 2\hat{n})] = a + bn$$

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, $v(n)$,

$$x(n) = s(n) + v(n) = \cos(\omega_0 n) + v(n)$$ (15.8.1)

The desired signal $s(n)$ can be extracted with the help of a peaking resonator filter with center frequency at $\omega_0$. 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,

$$H(z) = \frac{p}{\beta z - 1} = \frac{1 - \beta z^{-2}}{\beta z^{-1} - 1}$$ (15.8.2)

where $\beta$ is related to the peak’s 3-dB width, say, $\Delta \omega$ in rads/sample, by,

$$\beta = \tan\left(\frac{\Delta \omega}{2}\right)$$ (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,

$$p_{\pm} = \frac{\cos \omega_0 \pm j \sin^2 \omega_0 - \beta^2}{1 + \beta}$$ (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,

$$|p_{\pm}| = \sqrt{1 - \frac{\beta}{1 + \beta}}$$ (15.8.5)
726

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, \(0 \leq \omega \leq \pi\). 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, \(\omega_L, \omega_R\), on the graph and connect them with a straight-line segment to indicate the 3-dB width.

(c) Calculate and plot the group delay \(n_{gr}(\omega)\) 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,

\[
\text{load } v;
\]

Then, construct the time signal, \(x(n), n = 0, 1, \ldots, N-1\), 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, \texttt{filter},

\[
y = \text{filter}(\text{num},\text{den},x);
\]

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.
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),

$$\hat{R} = \frac{\text{std}^2(y_v)}{\text{std}^2(v)}$$

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,

$$p(n) = \begin{cases} 0.5 - 0.5 \cos \left( \frac{2\pi n}{M} \right), & 0 \leq n \leq M \\ 0, & \text{otherwise} \end{cases}$$

With $M = 1000$, calculate and plot the magnitude spectrum $|P(\omega)|$ of the pulse $p(n)$ over the narrow interval $|\omega| \leq 0.1\pi$ 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 $N = 1100$,

$$x(n) = \cos(\omega_0 n) \cdot p(n), \quad 0 \leq n \leq N - 1$$

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,

$$y(n) = \cos(\omega_0 n) \cdot p(n - n_{\text{gr}}(\omega_0))$$

To visualize this result better, add the delayed envelope signal $p(n - n_{\text{gr}}(\omega_0))$ to the graph of $y(n)$. Additionally, calculate and plot the error difference signal,

$$y(n) - \cos(\omega_0 n) \cdot p(n - n_{\text{gr}}(\omega_0)), \quad 0 \leq n \leq N - 1$$

**Example Graphs**

![Magnitude response, $\Delta \omega = 0.01\pi$](image1.png)

![Group delay, $n_{\text{gr}}(\omega)$](image2.png)

![Impulse response, $h(n)$, $n_{\text{eff}} = 293$](image3.png)

![Noisy sinusoid, $x(n)$](image4.png)

![Filtered noisy sinusoid](image5.png)

![Noise component, $v(n)$](image6.png)

![Filtered noise, $y_v(n)$](image7.png)
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 $\omega_0$ rads/sample, and 3-dB width $\Delta\omega$,

\[ H_{\text{notch}}(z) = \frac{1}{1 + \beta}, \quad H_{\text{peak}}(z) = \frac{\beta}{1 + \beta}, \quad \beta = \frac{1 - 2 \cos(\omega_0) z^{-1} + z^{-2}}{1 - 2 \cos(\omega_0) \frac{1}{1 + \beta} z^{-1} + \frac{1 - \beta}{1 + \beta} z^{-2}} \]

The peaking filter is the same as in Eq. (15.8.2). The parameter $\beta$ 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),

\[ H_{\text{notch}}(z) + H_{\text{peak}}(z) = 1 \]
\[ H_{\text{notch}}(\omega) + H_{\text{peak}}(\omega) = 1 \]
\[ |H_{\text{notch}}(\omega)|^2 + |H_{\text{peak}}(\omega)|^2 = 1 \]
\[ h_{\text{notch}}(n) + h_{\text{peak}}(n) = \delta(n) \]
\[ R_{\text{notch}} + R_{\text{peak}} = 1 \]
\[ R_{\text{notch}} = \frac{1}{1 + \beta} \]
\[ R_{\text{peak}} = \frac{\beta}{1 + \beta} \]

**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:

\[ x_a(t) = \begin{cases} 
\cos(2\pi f_1 t), & 0 \leq t < 5 \text{ sec} \\
\cos(2\pi f_2 t), & 5 \leq t < 10 \text{ sec} \\
\cos(2\pi f_3 t), & 10 \leq t < 15 \text{ sec}
\end{cases} \]

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 $t_n = n T$. In MATLAB, you could construct the vector of times as follows, with $T_{\text{max}} = 15$ sec,

\[ t_n = 0 : T : T_{\text{max}}; \]

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 \( 0 \leq f \leq 10 \) Hz. Add to the graph the values corresponding to 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.

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 \( |H_{\text{notch}}(f_3)| \).

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.

Repeat parts (c,d,e) by redesigning the filters to have the shorter 3-dB width of \( \Delta f = 0.3 \) Hz, and comment on the tradeoff between filter width and time constant.

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, \( R_{\text{peak}} \), is very small, while that of the notch, \( R_{\text{notch}} \), 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,

\[ h(n) = \frac{1}{N}, \quad 0 \leq n \leq N - 1 \]

\[ H(z) = \frac{1}{N} \left( 1 + z^{-1} + z^{-2} + \cdots + z^{-(N-1)} \right) = \frac{1 - z^{-N}}{1 - z^{-1}} \quad (15.10.1) \]

\[ y_n = \frac{1}{N} (x_n + x_{n-1} + x_{n-2} + \cdots + x_{n-N+1}) \]

and for EMA,

\[ h(n) = (1 - \lambda)^n, \quad 0 \leq n < \infty \]

\[ H(z) = \frac{1 - \lambda}{1 - \lambda z^{-1}} \quad (15.10.2) \]

\[ y_n = \lambda y_{n-1} + (1 - \lambda) x_n \]

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 \( \lambda \) of the EMA case is a forgetting factor such that \( 0 < \lambda < 1 \), which is usually specified in terms of an equivalent SMA length \( N \) given by,

\[ N = \frac{1 + \lambda}{1 - \lambda} \quad \Leftrightarrow \quad \lambda = \frac{N - 1}{N + 1} \quad (15.10.3) \]

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 result in longer lags, and for EMA, 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,

\[ n = \frac{N - 1}{2} = \frac{\lambda}{1 - \lambda} \quad \Leftrightarrow \quad R = \frac{1}{N} = \frac{1 - \lambda}{1 + \lambda} \quad (15.10.5) \]

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,

\[ n = \frac{N - 1}{2} = \frac{\lambda}{1 - \lambda} \quad \Leftrightarrow \quad R = \frac{1}{N} = \frac{1 - \lambda}{1 + \lambda} \quad (15.10.5) \]

The tradeoff is evident, with \( R \) decreasing and \( \lambda \) increasing with \( N \), or equivalently, with increasing \( \lambda \) towards unity. The time constants of the two filters are as follows, where in the SMA case it is simply the filter order,

\[ n_{\text{eff, SMA}} = N - 1, \quad n_{\text{eff, EMA}} = \frac{\ln \epsilon}{\ln \lambda} \quad (15.10.6) \]

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 \( \tau \) time steps. The impulse response of that filter is, for \( n = 0, 1, \ldots, N - 1 \),

\[ h_{\tau}(n) = h_{\tau}(n) + \tau h_{\tau}(n) = \frac{2(2N - 1 - 2n)}{N(N + 1)} + \tau \frac{6(N - 1 - 2n)}{N(N^2 - 1)} \quad (15.10.7) \]

This filter has negative group-delay at DC equal to \(-\tau\), that is, \( n = -\tau \), thus, it has a positive time advance of \(+\tau\) corresponding to predicting ahead by \( \tau \) units.

Actually, the time “advance” \( \tau \) can be non-integer, positive, or negative. Positive \( \tau \)s correspond to prediction or forecasting, negative \( \tau \)s 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:
For $N = 24$ and $\lambda$ defined through Eq. (15.10.3), filter $s(n)$ through an SMA and an EMA filter, and plot the corresponding outputs, $y_{ema}(n)$ and $y_{ema}(n)$. 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 $\lambda$, and plot the corresponding outputs, $y_{pma}(n)$ and $y_{dema}(n)$. 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, $h_p(n) = h_{pm}(n) + \tau h_b(n)$, where $h_{pm}(n)$ represents the zero-lag portion. The part, $h_b(n)$, is a slope-tracking filter that tracks the changing slope of the input;

$$h_b(n) = \frac{6(N - 1 - 2n)}{N(N^2 - 1)}, \quad n = 0, 1, \ldots, N - 1, \quad \text{(PMA slope filter)} \quad (15.10.15)$$

Similarly, in the DEMA case, the corresponding slope filter is constructed by the following rule, derived in Chap. 25,

$$H_b(z) = \frac{1 - \lambda}{\lambda} \left[H_{ema}(z) - H_{ema}^2(z)\right], \quad \text{(DEMA slope filter)} \quad (15.10.16)$$

The signal that represents the slope of the piecewise linear input $s(n)$ is obtained by inspection from Eq. (15.10.14),

$$s_b(n) = \begin{cases} 0.8, & 0 \leq n < 75 \\ -0.3, & 75 \leq n < 225 \\ 0.8, & 225 \leq n \leq 300 \end{cases} \quad (15.10.17)$$

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

$$x(n) = s(n) + v(n) \quad (15.10.18)$$

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**
  - Input
  - EMA
  - SMA

- **Slope filter outputs**
  - Input slope
  - PMA slope
  - DEMA slope

- **Noisy input and outputs**
  - Noisy input
  - EMA
  - SMA

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.

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:

$$H_{\text{notch}}(\omega) + H_{\text{comb}}(\omega) = 1 \quad (15.11.1)$$
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., \( f_1 = 60 \text{ Hz} \)), 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:

\[
\omega_1 = \frac{2\pi f_1}{f_s} = \frac{2\pi 60}{1000} = 1.2\pi \text{ radians/sample}
\]

Assuming a Q-factor of 60 for the notch filter, we have a 3-dB width:

\[
\Delta f = \frac{f_1}{Q} = 1 \text{ Hz} \quad \Rightarrow \quad \Delta \omega = \frac{2\pi \Delta f}{f_s} = 0.002\pi \text{ radians/sample}
\]

Using the design equations (16.2.22) and (16.2.23), we find the notch filter:

\[
H(z) = 0.99687 \cdot \frac{1 - 1.85955z^{-1} + z^{-2}}{1 - 1.85373z^{-1} + 0.99374z^{-2}}
\]

Figure 15.11.3 shows three beats of a simulated ECG with 60 Hz noise generated by

\[
x(n) = s(n) + 0.5 \cos(\omega_1 n), \quad n = 0, 1, \ldots, 1500
\]

The ECG signal \( x(n) \) 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 \( a_2 = R^2 \):

\[
R^2 = 0.99374 \quad \Rightarrow \quad R = 0.99686
\]

which gives for the 1% time constant \( n_{\text{eff}} = \ln(0.01) / \ln(R) = 1464 \) samples. In seconds, this is \( \tau = n_{\text{eff}}T = 1464/1000 = 1.464 \text{ sec} \), where \( T = 1/f_s = 1 \text{ msec} \).}

**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 \( D \)th roots-of-unity frequencies:

\[
f_k = kf_1 = k \frac{f_s}{D}, \quad \omega_k = k \omega_1 = \frac{2\pi k}{D}, \quad k = 0, 1, \ldots, D - 1 \quad (15.11.2)
\]

In this case, we can use the general procedure discussed in Section 6.4.3. The notch polynomial having as roots the \( D \)th roots of unity, \( z_k = e^{j\omega_k} = e^{2\pi jk/D} \) is given by

\[
N(z) = 1 - z^{-D}. \quad \text{The corresponding multi-notch filter is obtained by sharpening the zeros, that is, putting poles behind the zeros by the replacement } z \rightarrow z/D:
\]

\[
H_{\text{notch}}(z) = \frac{bN(z)}{N(z^D)} = b \left( \frac{1 - z^{-D}}{1 - az^{-D}} \right), \quad b = \frac{1 + a}{2} \quad (15.11.3)
\]

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 $\Delta\omega = 2\pi f_s/\Delta f$, of the notch dips. Using the bilinear transformation method\(^1\) of Chapter 12, we obtain the following design equations, for a given $\Delta\omega$:

$$\beta = \tan \left( \frac{D\Delta\omega}{4} \right), \quad a = \frac{1 - \beta}{1 + \beta}, \quad b = \frac{1}{1 + \beta} \quad (15.11.4)$$

Because $a$ must be in the interval $0 \leq a < 1$, we find the restriction $0 < \beta \leq 1$, which translates into the following bound on the desired 3-dB width: $D\Delta\omega/4 \leq \pi/4$, or

$$\Delta\omega \leq \frac{\pi}{D} \quad (15.11.5)$$

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:

$$|H_{\text{notch}}(\omega)|^2 = \frac{\tan^2(\omega D/2)}{\tan^2(\omega D/2) + \beta^2} \quad (15.11.6)$$

**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 $0 < \Delta\omega \leq \pi/D = 0.1\pi$.

Using Eqs. (15.11.4), we design the filter for the following three values of $\Delta\omega$:

- $\Delta\omega = 0.1\pi \quad \Rightarrow \quad \beta = 1 \quad a = 0 \quad b = 0.5$
- $\Delta\omega = 0.05\pi \quad \Rightarrow \quad \beta = 0.4142 \quad a = 0.4142 \quad b = 0.7071$
- $\Delta\omega = 0.0125\pi \quad \Rightarrow \quad \beta = 0.0985 \quad a = 0.8207 \quad b = 0.9103$

corresponding to the three transfer functions:

$$H_{\text{notch}}(z) = 0.5(1 - z^{-10}) \quad H_{\text{notch}}(z) = 0.7071 \frac{1 - z^{-10}}{1 - 0.4142z^{-10}} \quad H_{\text{notch}}(z) = 0.9103 \frac{1 - z^{-10}}{1 - 0.8207z^{-10}}$$

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:

$$x(n) = s(n) + v(n), \quad n = 0, 1, \ldots, 1199$$

\(^1\)Here, the highpass analog filter $s/(s + \beta)$ is transformed by $s = (1 - z^{-D})/(1 + z^{-D})$. 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:

$$v(n) = 1 + 0.5w(n), \quad w(n) = [1, 1, 1, 1, -1, -1, -1, -1, \ldots]$$

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:

$$\Delta f = \frac{f_1}{Q} = \frac{60}{80} = 0.75 \text{ Hz} \quad \Rightarrow \quad \Delta\omega = \frac{2\pi\Delta f}{f_1} = 0.0025\pi \text{ radians/sample}$$

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 $w = [w_0, w_1, \ldots, w_{10}]$ by the sample processing algorithm:
15.11. NOTCH AND COMB FILTERS FOR PERIODIC SIGNALS

for each input sample \( x \) do:
\[
\begin{align*}
& w_0 = 0.9615 \, w_{10} + 0.9807 \, x \\
& y = w_0 - w_{10} \\
& \text{delay} (10, w)
\end{align*}
\]

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 \( 0 \leq t \leq 1 \) sec and \( 1 \leq t \leq 2 \) sec, have been split into two side-by-side graphs.

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 3, 5 \( f_1 \), and so on, are still in the output.

Actually, for this example, the highest harmonic is 5 \( f_1 \) 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:

\[
H_1(z) = 0.99609 \frac{1 - 1.61803z^{-1} + z^{-2}}{1 - 1.61170z^{-1} + 0.99218z^{-2}}
\]

\[
H_3(z) = 0.99609 \frac{1 + 0.61803z^{-1} + z^{-2}}{1 + 0.61562z^{-1} + 0.99218z^{-2}} \quad \text{(notch at } 3f_1)\]

\[
H_5(z) = 0.99609 \frac{1 + z^{-1}}{1 + 0.99218z^{-1}} \quad \text{(notch at } 5f_1)\]

\[
H_6(z) = 0.99609 \frac{1 - z^{-1}}{1 - 0.99218z^{-1}} \quad \text{(notch at } 0)\]

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:

\[
H_0(z) = b \frac{1 - z^{-1}}{1 - az^{-1}}, \quad H_5(z) = b \frac{1 + z^{-1}}{1 + az^{-1}}
\]

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 \( a = 2b - 1 \).

\[\Box\]
15.11. NOTCH AND COMB FILTERS FOR PERIODIC SIGNALS

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

\[
z \rightarrow z^D
\]

(15.11.7)

that is,

\[
H(z) = b \frac{1 - z^{-1}}{1 - az^{-1}} \quad \rightarrow \quad H(z^D) = b \frac{1 - z^{-D}}{1 - az^{-D}}
\]

(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:

\[
\omega \rightarrow \omega D,
\]

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 \( 0 \leq \omega \leq 2\pi \), the new spectrum \( H(\omega D) \) will have period \( 0 \leq \omega D \leq 2\pi \), which becomes the scaled period \( 0 \leq \omega \leq 2\pi/D \) fitting exactly \( D \) times into the new Nyquist interval \( 0 \leq \omega \leq 2\pi \).

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:

\[
H(z) = \frac{1 - a}{1 - az^{-D}}
\]

(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 \( \omega_k = (2k + 1)\pi/D \):

\[
H_{\text{comb}}(z) = b \frac{1 + z^{-D}}{1 - az^{-D}}, \quad b = \frac{1 - a}{2}
\]

(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:

\[
\beta = \tan \left( \frac{D\Delta\omega}{4} \right), \quad a = \frac{1 - \beta}{1 + \beta}, \quad b = \frac{\beta}{1 + \beta}
\]

(15.11.11)

where, as in Eq. (15.11.4), the width is constrained to be in the interval: \( 0 \leq \Delta\omega \leq \pi/D \). Like Eq. (15.11.6), the magnitude response squared of (15.11.10) can be expressed simply in the form:

\[
|H_{\text{comb}}(\omega)|^2 = \frac{\beta^2}{\tan^2(\omega D/2) + \beta^2}
\]

(15.11.12)

The comb and notch filters of Eqs. (15.11.10) and (15.11.3) are complementary in the sense of Eq. (15.11.9); 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:

\[
|H_{\text{comb}}(\omega)|^2 + |H_{\text{notch}}(\omega)|^2 = 1
\]

(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°. Indeed, it is left as an exercise to show that:

\[
H_{\text{comb}}(\omega) = j H_{\text{notch}}(\omega) \tan(\omega D/2)/\beta
\]

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 \( \Delta\omega \):

\[
\Delta\omega = 0.1\pi \quad \Rightarrow \quad \beta = 1 \quad a = 0 \quad b = 0.5
\]

\[
\Delta\omega = 0.05\pi \quad \Rightarrow \quad \beta = 0.4142 \quad a = 0.4142 \quad b = 0.2929
\]

\[
\Delta\omega = 0.0125\pi \quad \Rightarrow \quad \beta = 0.0985 \quad a = 0.8207 \quad b = 0.0897
\]

corresponding to the three transfer functions:

\(^*\) Here, the lowpass analog filter \( \beta/(s + \beta) \) is transformed by \( s = (1 - z^{-D})/(1 + z^{-D}) \).
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:

\[
H_{\text{comb}}(z) = 0.0305 \frac{1 + z^{-50}}{1 - 0.9391 z^{-50}}
\]

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 \( \omega_k = (2k + 1)\pi / 50 \).

The canonical form uses a 51-dimensional state vector \( \mathbf{w} = [w_0, w_1, \ldots, w_{50}] \) to implement \( D - 1 \) zeros between the original impulse response samples:

\[
\mathbf{h} = [h_0, h_1, h_2, \ldots] \rightarrow \mathbf{h} = [h_0, 0, 0, \ldots, 0, h_1, 0, 0, \ldots, 0, h_2, \ldots] \quad (15.11.14)
\]

and, therefore, the quantity \( \sum h_k^2 \) 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:

\[
x(n) = x(n) + v(n), \quad n = 0, 1, \ldots, 1999,
\]

where \( x(n) \) is a periodic triangular wave of period \( D = 50 \), linearly alternating between \( \pm 1 \) 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 replicate x(n) are plotted in Fig. 15.11.10. The magnitude squared responses, \( |H(\omega)|^2 \), are plotted in Fig. 15.11.8.

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.
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):

$$T_F = NT_H$$  \hspace{1cm} (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:

$$f_H = \frac{1}{T_H}, \quad f_F = \frac{1}{T_F} \Rightarrow f_H = N f_F$$  \hspace{1cm} (15.12.2)

The frequencies $f_H$ and $f_F$ are related to the horizontal and vertical velocities of the scanning spot by

$$f_H = \frac{v_x}{a}, \quad f_F = \frac{v_y}{b}$$  \hspace{1cm} (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.

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:

$$g(x,y) = \sum_{k,m} c_{km} e^{2\pi i k x/a} e^{2\pi i m y/b}$$
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:

\[
V(t) = g(v_x t, v_y t) = \sum_{k,m} c_{km} e^{2\pi j k v_x t} e^{2\pi j m v_y t / b}
\]

Using Eq. (15.12.3), we can rewrite:

\[
V(t) = \sum_{k,m} c_{km} e^{2\pi j f_{km} t}
\]  

(15.12.4)

where

\[
f_{km} = k f_H + m f_F = (kN + m) f_H = (k + \frac{m}{N} f_H)
\]  

(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 \( k f_H \) and a fine structure at the harmonics \( m f_F \).

In the NTSC\(^2\) 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, \( f_{field} = 2f_F \). The field rate is approximately 60 Hz in the U.S., and the frame rate approximately 30 Hz. The exact values are [204]:

\[
f_H = \frac{4.5 \text{ MHz}}{286} = 15.73426 \text{ kHz}, \quad f_F = \frac{f_H}{525} = 29.97 \text{ Hz}
\]  

(15.12.6)

where, by convention, the values are derived from the sound carrier frequency of 4.5 MHz. The corresponding time delays are \( T_H = 63.55 \mu\text{sec} \) and \( T_F = 33.37 \mu\text{sec} \).

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.

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 \( \phi = 33^\circ \).

To simplify the algebra, we work with the following complex-valued version of the above, with the understanding that we must take real parts:

\[
V(t) = Y(t) + e^{j2\pi f_{sc} t} C(t) \equiv Y(t) + Z(t)
\]  

(15.12.7)

where \( C(t) = \{I(t) - jQ(t)\} e^{j\phi} \).

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:

\[
f_{sc} + f_{km} = f_{sc} + kf_H + m f_F
\]  

(15.12.8)

By choosing \( f_{sc} \) to be a half-multiple of the line frequency \( f_F \), the chrominance peaks will fall exactly halfway between the luminance peaks, as shown in Fig. 15.12.2. We can take, for example,

\[
f_{sc} = \left( d_H + \frac{1}{2} f_H \right) = \left( d_H + \frac{1}{2}(2d_H + 1)f_H \right)
\]  

(15.12.9)

Therefore, the chrominance macro-structure peaks are centered at half-multiples of \( f_H \):

\[
f_{sc} + f_{km} = (d_H + k) f_H + m f_F
\]  

(15.12.10)

Moreover, because \( f_H = N f_F \) with \( N \) odd, the subcarrier frequency \( f_{sc} \) will also be equal to a half-multiple of the frame frequency \( f_F \):

\[
f_{sc} = (d_H + \frac{1}{2}) f_H = (d_H + \frac{1}{2}) N f_F
\]

Setting

\[
d_f + \frac{1}{2} = (d_H + \frac{1}{2}) N \quad \Rightarrow \quad d_f = N d_H + \frac{N - 1}{2}
\]

we find,

\[
f_{sc} = (d_f + \frac{1}{2}) f_F = \frac{1}{2}(2d_f + 1)f_F
\]  

(15.12.11)
15.12. LINE AND FRAME COMBS FOR DIGITAL TV

It follows that the chrominance micro-structure peaks will be centered at half-multiples of \( f_f \) (about the \( k_f H \) macro-structure peaks), falling half-way between the micro-structure peaks of the luminance signal, as shown in Fig. 15.12.2:

\[
\omega_{k_F} = \frac{2\pi m}{f_s} + \frac{\pi}{f_s} \quad (15.12.15)
\]

In the NTSC system, we have the choices:

\[
d_H = 227, \quad D_F = Nd_H + N - 1 = 119437 \quad (15.12.13)
\]

which give the subcarrier frequency:

\[
f_{sc} = 227.5f_H = 119437.5f_F = 3.579545 \text{ MHz}
\]

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.

![Interleaved luminance and chrominance spectra.](image)

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 \( \pm 0.5 \) MHz [204], which translates to about \( \pm 32f_H \) harmonics about \( f_{sc} \); indeed, \( 227.5 \times 0.5/3.38 \approx 32 \).

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 \( f_{sc} = 0.5 \) 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 (or four times the line frequency), that is,

\[
f_s = 4f_{sc}
\]

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 \( f_f \):

\[
f_s = D_H f_H = D_F f_F
\]

where

\[
D_H = 2(2d_H + 1), \quad D_F = ND_H = 2(2d_F + 1)
\]

For the NTSC system, we have from Eq. (15.12.13)

\[
D_H = 910, \quad D_F = 477750
\]

and

\[
f_s = 910f_H = 477750f_F = 14.31818 \text{ MHz}
\]

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 \( \text{radians per sample} \), the subcarrier frequency \( f_{sc} \) becomes:

\[
\omega_{sc} = \frac{2\pi f_{sc}}{f_s} = \frac{\pi}{2}
\]

Similarly, the frame and line frequencies are in these units:

\[
\omega_f = \frac{2\pi f_f}{f_s} = \frac{2\pi}{D_F}, \quad \omega_H = \frac{2\pi f_H}{f_s} = \frac{2\pi}{D_H} = N\omega_F
\]

Using Eq. (15.12.15), the luminance video frequencies \( f_{km} \) become:

\[
\omega_{km} = \frac{2\pi f_{km}}{f_s} = \frac{2\pi k}{D_H} + \frac{2\pi m}{D_F} = k\omega_H + m\omega_F
\]

The shifted chrominance frequencies Eq. (15.12.8) can be expressed as half-multiples of either the line or the frame digital frequencies:
15.12. LINE AND FRAME COMBS FOR DIGITAL TV

\[ \omega_{sc} + \omega_{km} = \frac{(2d_H + 2k + 1) \pi}{D_H} + \frac{2\pi m}{D_F} \]
\[ = \frac{2\pi k}{D_H} + (2d_F + 2m + 1) \frac{\pi}{D_F} \]
\[ = (2kN + 2d_F + 2m + 1) \frac{\pi}{D_F} \]  

(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):

\[ H_{\text{comb}}(z) = \frac{1}{2} (1 + z^{-D}) \]  

(15.12.20)

with a complementary notch filter \( H_{\text{notch}}(z) = 1 - H_{\text{comb}}(z) \):

\[ H_{\text{notch}}(z) = \frac{1}{2} (1 - z^{-D}) \]  

(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.

Another simple popular type of comb filter uses two \( D \)-fold delays and is obtained by squaring Eq. (15.12.20):

\[ H_{\text{comb}}(z) = \frac{1}{4} (1 + z^{-D})^2 = \frac{1}{4} (1 + 2z^{-D} + z^{-2D}) \]  

(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 \( \Delta \omega \) is \( \pi / D \). The separation of \( C \) and \( Y \) through the notch filter \( H_{\text{notch}}(z) \) now has nulls at all the chrominance frequencies, and \( H_{\text{comb}}(z) \) 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.

Another simple popular type of comb filter uses two \( D \)-fold delays and is obtained by squaring Eq. (15.12.20):

\[ H_{\text{comb}}(z) = \frac{1}{4} (1 + z^{-D})^2 = \frac{1}{4} (1 + 2z^{-D} + z^{-2D}) \]  

(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:

\[ H_{\text{notch}}(z) = -\frac{1}{4} (1 - z^{-D})^2 = \frac{1}{4} (-1 + 2z^{-D} - z^{-2D}) \]  

(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:

\[ z^D H_{\text{comb}}(z) = \frac{1}{2} + \frac{1}{4} (z^D + z^{-D}) \]
\[ z^D H_{\text{notch}}(z) = \frac{1}{2} - \frac{1}{4} (z^D + z^{-D}) \]  

(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

\[ V_n = Y_n + Z_n = Y_n + e^{j\omega x_n} \tilde{C}_n \]  

(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:

\[ V_n = Y_n + Z_n = Y_n + e^{j\omega x_n} \tilde{C}_n \]  

(15.12.25)
The row index \( i \) takes on the values \( i = 0, 1, \ldots, N-1 \), 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.](image)

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):

\[
n \pm D_H = iD_H + j \pm D_H = (i \pm 1)D_H + j \rightarrow (i \pm 1, j)
\]

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:

\[
\begin{align*}
\n & = \frac{1}{2} (V_n + V_{n-D}) \\
\Delta V_n & = \frac{1}{2} (V_n - V_{n-D})
\end{align*}
\]

(15.12.27)

If \( D = D_H \), we may think of \( V_n \) 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:

\[
\begin{align*}
\n & = \frac{1}{2} (V_{i,j} + V_{i-1,j}) \\
\Delta V_{i,j} & = \frac{1}{2} (V_{i,j} - V_{i-1,j})
\end{align*}
\]

In a similar fashion, the outputs of the 2-H filters (15.12.24) can be expressed in terms of the video time index \( n \):

\[
\begin{align*}
\n & = \frac{1}{2} (V_n + V_{n-D}) \\
\Delta V_n & = \frac{1}{2} (V_n - V_{n-D})
\end{align*}
\]

(15.12.28)

The outputs of the luminance and chrominance combs can be expressed then in the form:

\[
\begin{align*}
\n & = \frac{1}{2} (Y_n + Y_{n-D}) + e^{i\omega scn} \frac{1}{2} (C_n - C_{n-D}) \\
\Delta V_n & = \frac{1}{2} (Y_n - Y_{n-D}) + e^{i\omega scn} \frac{1}{2} (C_n + C_{n-D})
\end{align*}
\]

which can be written in terms of the corresponding sum and difference signals:

\[
\begin{align*}
\n & = Y_n + e^{i\omega scn} \Delta C_n \\
\Delta V_n & = \Delta Y_n + e^{i\omega scn} \Delta C_n
\end{align*}
\]

(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

\[
\n = \frac{1}{2} (V_n + V_{n-D}) + \frac{1}{4} (V_{n+D} + V_{n-D}) \\
\Delta V_n & = \frac{1}{2} V_n - \frac{1}{4} (V_{n+D} + V_{n-D})
\]

or, in terms of the pixel locations, showing the weighted averaging of the current line with the lines above and below it:

\[
\begin{align*}
\n & = \frac{1}{2} (V_{i,j} + V_{i+1,j} + V_{i-1,j}) \\
\Delta V_{i,j} & = \frac{1}{2} (V_{i,j} + V_{i+1,j} - V_{i-1,j})
\end{align*}
\]

Using Eq. (15.12.25), we have for the delayed signal \( V_{n-D} \):

\[
V_{n-D} = Y_{n-D} + Z_{n-D} = Y_{n-D} + e^{i\omega scn} (n-D) C_{n-D}
\]

The property that makes possible the comb filtering separation of the luminance and chrominance is that the subcarrier signal \( e^{i\omega scn} \) changes sign from line to line and from frame to frame. This follows from the (intentional) choice of \( DH \) and \( DF \) 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:

\[
\omega sc = \frac{\pi}{2} 2(2d + 1) = 2\pi d + \pi
\]

which corresponds to a 180° phase shift. Indeed,

\[
e^{i\omega sc} = e^{2\pi id + \pi} = e^{i\pi} = -1
\]

It follows that:

\[
V_{n-D} = Y_{n-D} - e^{i\omega scn} C_{n-D}
\]

(15.12.28)
content), we may set $\Delta C_n = 0$. Similarly, ignoring the micro-structure of $Y$, we may set

$$\Delta Y_n = 0.$$  \hfill (15.12.29)

Then, (15.12.29) simplifies as follows:

$$\nabla_n = \bar{Y}_n$$

$$\Delta V_n = e^{j\omega_{sc}n}\bar{C}_n$$  \hfill (15.12.30)

Thus, the comb outputs are effectively the desired luminance and chrominance components. The chrominance part $e^{j\omega_{sc}n}\bar{C}_n$ is then sent into a subcarrier demodulator and $\bar{C}_n$ 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 $\Delta Y_n = 0$ is more severe than $\Delta C_n = 0$, 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 $\bar{Y}_n$ averaged across two lines, instead of $Y_n$ itself.

It follows from the above that a better approximation may be obtained by setting $\Delta C_n$ to zero, but not $\Delta Y_n$. The filtering equations (15.12.29) become in this case:

$$\nabla_n = \bar{Y}_n$$

$$\Delta V_n = \Delta Y_n + e^{j\omega_{sc}n}\bar{C}_n$$  \hfill (15.12.31)

The averaged signal $\bar{Y}_n$ can be expressed in terms of the desired one $Y_n$ and the missing vertical detail signal $\Delta Y_n$, as follows:

$$\bar{Y}_n = \frac{1}{2}(Y_n + Y_{n-D}) = Y_n - \frac{1}{2}(Y_n - Y_{n-D}) = Y_n - \Delta Y_n$$

Therefore, we may write Eq. (15.12.31) as:

$$\nabla_n = Y_n - \Delta Y_n$$

$$\Delta V_n = Y_n + e^{j\omega_{sc}n}\bar{C}_n$$  \hfill (15.12.32)

This result suggests a method of restoring the lost vertical detail [193,205,207]. Because the vertical detail signal $\Delta Y_n$ is common in both outputs, it can be extracted from the second output $\Delta V_n$ by lowpass filtering and then reinserted into the first to recover $Y_n$. This works because most of the energy in $\Delta Y_n$ is between 0–1 MHz, whereas the $C$-term in $\Delta V_n$ is centered around 3.58 MHz. Moreover, the term $\Delta Y_n$ can be removed from $\Delta V_n$ 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 $\Delta V_n$ 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 $z = e^{j\omega_{sc}} = e^{j\pi/2} = \pm j$.

$$H_{LP}(z) = \frac{1}{4}(1 + z^{-2})^2$$

$$H_{LP}(z) = \frac{1}{16}(1 + z^{-2})^2(-1 + 6z^{-2} - z^{-4})$$

$$H_{LP}(z) = \frac{1}{32}(1 + z^{-2})^2(-3 + 14z^{-2} - 3z^{-4})$$

$$H_{LP}(z) = \frac{1}{64}(1 + z^{-2})^4(1 - 4z^{-2} + 5z^{-4} + 5z^{-8} - 4z^{-10} + z^{-12})$$

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 $H_{LP}(z) + H_{BP}(z) = z^{-nL}$, where $M$ is the inherent delay introduced by the filters (i.e., half the filter order). Thus, we define:

$$H_{BP}(z) = z^{-M} - H_{LP}(z)$$  \hfill (15.12.34)

For the above four examples, we have $M = 2, 4, 10$. Using Eqs. (15.12.33) and (15.12.34), we find:

$$H_{BP}(z) = -\frac{1}{4}(1 - z^{-2})^2$$

$$H_{BP}(z) = \frac{1}{16}(1 - z^{-2})^4$$

$$H_{BP}(z) = \frac{1}{32}(1 - z^{-2})^2(3 - 2z^{-2} + 3z^{-4})$$

$$H_{BP}(z) = -\frac{1}{64}(1 - z^{-2})^4(1 + 4z^{-2} + 5z^{-4} + 5z^{-8} + 4z^{-10} + z^{-12})$$
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 \(0 \leq f \leq 4.2\) 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 \(\Delta V_n\), whereas the passband of the highpass/bandpass filter coincides with the chrominance band.

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 \(a = 8\).

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 \(m_x, m_y\):

\[
\hat{\sigma}_x^2 = \frac{1}{L} \sum_{n=0}^{L-1} (x(n) - m_x)^2, \quad \hat{\sigma}_y^2 = \frac{1}{L} \sum_{n=0}^{L-1} (y(n) - m_y)^2, \quad \hat{R} = \frac{\hat{\sigma}_y^2}{\hat{\sigma}_x^2}
\]

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 \(H_{LP}(z) = b/(1 - az^{-1})\), where \(0 < a < 1\), to extract the high-frequency signal \(x(n)\), where \(v(n)\) is zero-mean white noise of variance \(\sigma_v^2\).

How should you choose \(b\) so that the part \(x(-1)^{th}\) 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 \(\omega_0\), 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 \(n_{\text{eff}}\), and plot its magnitude response squared \(|H(\omega)|^2\) over 400 equally spaced frequencies over \(0 \leq \omega < \pi\).

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 \(\text{gr}an\), 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 \(v(n)\) is \(\text{gran}(0, 1, \&\text{seed})\) 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:

\[ f(t) = [u(t - 0.15) - u(t - 0.30)] - 0.75[u(t - 0.30) - u(t - 0.45)] \]
\[ s(t) = f(t) + f(t - 0.5) + f(t - 1) \]
\[ x(t) = s(t) + 0.5 \cos(2\pi f_1 t) \]

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 \( 0 \leq n \leq 1499 \).

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 \( 0 \leq f \leq f_s/2 \). Filter the sampled signal \( x(n) \) through both filters, and plot the resulting output signals \( y(n) \) for \( 0 \leq n \leq 1499 \). 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 \( 0 \leq n < 1800 \):

\[ f(n) = [u(n - 150) - u(n - 300)] - 0.75[u(n - 300) - u(n - 450)] \]
\[ s(n) = f(n) + f(n - 600) + f(n - 1200) \]
\[ x(n) = s(n) + v(n) \]

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 \( 0 \leq n < 1800 \). For display purposes, split each graph into three separate graphs that cover the time periods \( 0 \leq n < 600, 600 \leq n < 1200, \) and \( 1200 \leq n < 1800 \).

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 \( 0 \leq f \leq f_s \) 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 \texttt{ecg.m} of Appendix C, as follows:

\[ s = \texttt{ecg}(500)'; \]
\[ s = [s; s; s]; \]
\[ s0 = \texttt{sgfilt}(0, 9, s); \]
\[ s = s0 / \max(s0); \]

one beat of length 500
three beats
5-point smoother
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 \texttt{ecg.m} of Appendix C, as follows:

\[ s = \texttt{ecg}(600)'; \]
\[ s = [s; s; s]; \]
\[ s0 = \texttt{sgfilt}(0, 9, s); \]
\[ s = s0 / \max(s0); \]

one beat of length 600
three beats
9-point smoother
normalized to unity maximum

15.14 Show that the following periodic comb filter has NRR:

\[ H(z) = \frac{1 - a z^{-D}}{1 - az^{-1}} \Rightarrow R = \frac{1 - a}{2} \]

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
16.1 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:

\[
h(n) = R^n \sin(\omega_0 n) u(n), \quad H(z) = \frac{R \sin \omega_0 z^{-1}}{1 - 2R \cos \omega_0 z^{-1} + R^2 z^{-2}}
\] (16.1.1)

For \( 0 < R < 1 \), it corresponds to an exponentially decaying sinusoid of frequency \( \omega_0 \). 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:

\[
\begin{align*}
\text{for } n = 0, 1, 2, \ldots \text{ do:} \\
w_0 &= (2R \cos \omega_0) w_1 - R^2 w_2 + \delta(n) \\
y &= (R \sin \omega_0) w_1 \\
w_2 &= w_1 \\
w_1 &= w_0
\end{align*}
\]

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.
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 \( \omega_H = 2\pi f_H / f_s \).

\[
1 - 2R \cos \omega_0 z^{-1} + R^2 z^{-2} = (1 - pz^{-1})(1 - p^* z^{-1}) \quad (16.1.3)
\]

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:

\[
H_1(z) = \frac{1 - az^{-1}}{1 - 2az^{-1} + (a^2 + b^2) z^{-2}} \quad (16.1.4)
\]

\[
H_2(z) = \frac{bz^{-1}}{1 - 2az^{-1} + (a^2 + b^2) z^{-2}}
\]

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:

\[
H_1(z) + jH_2(z) = \frac{1 - az^{-1} + jbz^{-1}}{1 - 2az^{-1} + (a^2 + b^2) z^{-2}} = \frac{1 - p^* z^{-1}}{(1 - p z^{-1})(1 - p^* z^{-1})} = \frac{1}{1 - p z^{-1}}
\]
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:

\[ h_1(n) = R^n \cos(\omega_0 n) u(n), \quad h_2(n) = R^n \sin(\omega_0 n) u(n) \]

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].

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 \( x(t) = \cos(2\pi f t) \), obtained by setting \( t = nT \), are:

\[ x(n) = \cos(2\pi f n T) = \cos(\omega n) \]

where \( \omega = 2\pi f T = 2\pi f / f_s \).

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 \( \omega \) be such that\(^1\)

\[ \omega D = 2\pi \quad \Rightarrow \quad \omega = \frac{2\pi}{D} \]  \hspace{1cm} (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:

\[ f = \frac{f_s}{D} \]  \hspace{1cm} (16.1.6)

or, equivalently, if the sampling rate is an integral multiple of the frequency:

\[ f_s = Df \]  \hspace{1cm} (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,

\[ nT + T_D = (n + D) T \]

\(^1\)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 $T$:

$$ T_D = DT_f $$  \hspace{1cm} (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 $b_i$, $i = 0, 1, \ldots, D - 1$, we have the periodic sequence:

$$ h = [b_0, b_1, \ldots, b_{D-1}, b_0, b_1, \ldots, b_{D-1}, \ldots] $$  \hspace{1cm} (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:

$$ H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_{D-1} z^{-(D-1)}}{1 - z^{-D}} $$  \hspace{1cm} (16.1.10)

$$ h(n) = [b_0, b_1, \ldots, b_{D-1}] $$

Fig. 16.1.6  Discrete-time periodic signal of period $D = 4$.

As a concrete example, consider the case $D = 4$ with transfer function:

$$ H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}}{1 - z^{-4}} $$  \hspace{1cm} (16.1.11)

Its causal impulse response can be obtained by expanding the denominator using the infinite geometric series:

$$ H(z) = (b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}) (1 + z^{-4} + z^{-8} + \ldots) $$

$$ = (b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}) \cdot 1 $$

$$ + (b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}) \cdot z^{-4} $$

$$ + (b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}) \cdot z^{-8} + \ldots $$

Picking out the coefficients of the powers of $z^{-1}$ gives the causal periodic impulse response sequence:

$$ h = [b_0, b_1, b_2, b_3, b_0, b_1, b_2, b_3, b_0, b_1, b_2, b_3, \ldots] $$  \hspace{1cm} (16.1.12)

Writing

$$ (1 - z^{-D}) H(z) = b_0 + b_1 z^{-1} + \ldots + b_{D-1} z^{-(D-1)} $$

and transforming it to the time domain gives the difference equation for $h(n)$:

$$ h(n) = h(n - D) + b_0 \delta(n) + b_1 \delta(n - 1) + \ldots + b_{D-1} \delta(n - D + 1) $$

For example, with $D = 4$:

$$ h(n) = h(n - 4) + b_0 \delta(n) + b_1 \delta(n - 1) + b_2 \delta(n - 2) + b_3 \delta(n - 3) $$  \hspace{1cm} (16.1.13)

which generates Eq. (16.1.12). Indeed, iterating Eq. (16.1.13) with causal initial conditions gives:

$$ h(0) = b_0, \quad h(1) = b_1, \quad h(2) = b_2, \quad h(3) = b_3 $$

$$ h(n) = h(n - 4), \quad \text{for } n \geq 4 $$

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 $D$ feedback delays, but only $D-1 = 3$ feed-forward ones. The corresponding sample processing algorithm will be as follows:

for $n = 0, 1, 2, \ldots$ do:

$$ v_0 = \delta(n) $$

$$ y = w_0 = w_4 + b_0 v_0 + b_1 v_1 + b_2 v_2 + b_3 v_3 $$

$$ \text{delay}(3, v) $$

$$ \text{delay}(4, w) $$

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 $\delta(n - 2), \delta(n - 3)$.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$v_0$</th>
<th>$v_1$</th>
<th>$v_2$</th>
<th>$v_3$</th>
<th>$w_0$</th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
<th>$w_4$</th>
<th>$y = w_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$b_0$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$b_0$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$b_1$</td>
<td>$b_0$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$b_1$</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$b_2$</td>
<td>$b_1$</td>
<td>$b_0$</td>
<td>0</td>
<td>0</td>
<td>$b_2$</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$b_3$</td>
<td>$b_2$</td>
<td>$b_1$</td>
<td>$b_0$</td>
<td>0</td>
<td>$b_3$</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$b_0$</td>
<td>$b_3$</td>
<td>$b_2$</td>
<td>$b_1$</td>
<td>$b_0$</td>
<td>$b_0$</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$b_1$</td>
<td>$b_0$</td>
<td>$b_3$</td>
<td>$b_2$</td>
<td>$b_1$</td>
<td>$b_1$</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$b_2$</td>
<td>$b_1$</td>
<td>$b_0$</td>
<td>$b_3$</td>
<td>$b_2$</td>
<td>$b_2$</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$b_3$</td>
<td>$b_2$</td>
<td>$b_1$</td>
<td>$b_0$</td>
<td>$b_3$</td>
<td>$b_3$</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$b_0$</td>
<td>$b_3$</td>
<td>$b_2$</td>
<td>$b_1$</td>
<td>$b_0$</td>
<td>$b_0$</td>
</tr>
</tbody>
</table>
16.1. DIGITAL WAVEFORM GENERATORS

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:

\[
w_i = b_{D-i}, \quad i = 1, 2, \ldots, D
\]  
(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 \( w_0 = w_{D-1} \), or \( w_0 = w_{D-4} \), 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 and a steady state part.

\[
\begin{align*}
\text{for } n = 0, 1, \ldots, D - 1 & \quad \text{do:} \\
& w_0 = b_n \\
& \text{delay}(D, w)
\end{align*}
\]
(16.1.16)

and a steady state part

\[
\text{repeat forever:}
\begin{align*}
& w_0 = w_{D-1} \\
& \text{delay}(D, w)
\end{align*}
\]
(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.

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:

\[
\begin{align*}
& \text{for } n = 0, 1, 2, \ldots \quad \text{do:} \\
& w_0 = w_{D-1} + \delta(n) \\
& y = b_0 w_0 + b_1 w_1 + \cdots + b_{D-1} w_{D-1} \\
& \text{delay}(D, w)
\end{align*}
\]

The contents of the \( w \)-register at successive sampling instants are shown below:

<table>
<thead>
<tr>
<th>( n )</th>
<th>( w_0 )</th>
<th>( w_1 )</th>
<th>( w_2 )</th>
<th>( w_3 )</th>
<th>( w_4 )</th>
<th>( y = b_0 w_0 + b_1 w_1 + b_2 w_2 + b_3 w_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( b_0 )</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( b_1 )</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>( b_2 )</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>( b_3 )</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>( b_0 )</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( b_1 )</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>( b_2 )</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>( b_3 )</td>
</tr>
</tbody>
</table>

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.

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:

\[
H(z) = \frac{1}{1 - z^{-1}} \cdot N(z)
\]
(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.
Every time one of these impulses hits the FIR filter $N(z)$, it generates the impulse response $b = [b_0, b_1, b_2, b_3]$, translated in time to match the time of that input impulse, as required by time invariance. Because the duration of $b$ 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.

$[1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, \ldots]$ 

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 $D$th 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 $w$ for the case $D = 4$, at successive time instants (grouped every $D + 1 = 5$ samples):

<table>
<thead>
<tr>
<th>$n$</th>
<th>$q$</th>
<th>$w_0$</th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
<th>$w_4$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$b_0$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$b_0$</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>$b_0$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$b_1$</td>
<td>$b_1$</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>$b_0$</td>
<td>0</td>
<td>0</td>
<td>$b_2$</td>
<td>$b_2$</td>
<td>$b_2$</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>$b_0$</td>
<td>0</td>
<td>$b_3$</td>
<td>$b_3$</td>
<td>$b_3$</td>
<td>$b_3$</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>$b_0$</td>
<td>$b_0$</td>
<td>$b_0$</td>
<td>$b_0$</td>
<td>$b_0$</td>
<td>$b_0$</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>$b_1$</td>
<td>$b_0$</td>
<td>$b_3$</td>
<td>$b_2$</td>
<td>$b_1$</td>
<td>$b_1$</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>$b_1$</td>
<td>$b_0$</td>
<td>$b_3$</td>
<td>$b_2$</td>
<td>$b_2$</td>
<td>$b_2$</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>$b_1$</td>
<td>$b_0$</td>
<td>$b_3$</td>
<td>$b_2$</td>
<td>$b_3$</td>
<td>$b_3$</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>$b_1$</td>
<td>$b_0$</td>
<td>$b_3$</td>
<td>$b_2$</td>
<td>$b_3$</td>
<td>$b_3$</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>$b_1$</td>
<td>$b_1$</td>
<td>$b_0$</td>
<td>$b_3$</td>
<td>$b_2$</td>
<td>$b_3$</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>$b_2$</td>
<td>$b_1$</td>
<td>$b_0$</td>
<td>$b_3$</td>
<td>$b_2$</td>
<td>$b_2$</td>
</tr>
<tr>
<td>11</td>
<td>4</td>
<td>$b_2$</td>
<td>$b_1$</td>
<td>$b_0$</td>
<td>$b_3$</td>
<td>$b_3$</td>
<td>$b_3$</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>$b_2$</td>
<td>$b_1$</td>
<td>$b_0$</td>
<td>$b_3$</td>
<td>$b_3$</td>
<td>$b_3$</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>$b_2$</td>
<td>$b_1$</td>
<td>$b_1$</td>
<td>$b_0$</td>
<td>$b_3$</td>
<td>$b_3$</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>$b_2$</td>
<td>$b_2$</td>
<td>$b_1$</td>
<td>$b_0$</td>
<td>$b_3$</td>
<td>$b_3$</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>$b_3$</td>
<td>$b_2$</td>
<td>$b_1$</td>
<td>$b_0$</td>
<td>$b_3$</td>
<td>$b_3$</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>$b_3$</td>
<td>$b_2$</td>
<td>$b_1$</td>
<td>$b_0$</td>
<td>$b_0$</td>
<td>$b_0$</td>
</tr>
<tr>
<td>17</td>
<td>3</td>
<td>$b_3$</td>
<td>$b_2$</td>
<td>$b_1$</td>
<td>$b_0$</td>
<td>$b_0$</td>
<td>$b_0$</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>$b_3$</td>
<td>$b_2$</td>
<td>$b_2$</td>
<td>$b_1$</td>
<td>$b_0$</td>
<td>$b_2$</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>$b_3$</td>
<td>$b_3$</td>
<td>$b_2$</td>
<td>$b_1$</td>
<td>$b_0$</td>
<td>$b_3$</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>$b_0$</td>
<td>$b_3$</td>
<td>$b_2$</td>
<td>$b_1$</td>
<td>$b_0$</td>
<td>$b_0$</td>
</tr>
<tr>
<td>21</td>
<td>4</td>
<td>$b_0$</td>
<td>$b_3$</td>
<td>$b_2$</td>
<td>$b_1$</td>
<td>$b_1$</td>
<td>$b_1$</td>
</tr>
<tr>
<td>22</td>
<td>3</td>
<td>$b_0$</td>
<td>$b_3$</td>
<td>$b_2$</td>
<td>$b_2$</td>
<td>$b_1$</td>
<td>$b_2$</td>
</tr>
<tr>
<td>23</td>
<td>2</td>
<td>$b_0$</td>
<td>$b_3$</td>
<td>$b_3$</td>
<td>$b_2$</td>
<td>$b_1$</td>
<td>$b_3$</td>
</tr>
<tr>
<td>24</td>
<td>1</td>
<td>$b_0$</td>
<td>$b_0$</td>
<td>$b_3$</td>
<td>$b_2$</td>
<td>$b_1$</td>
<td>$b_0$</td>
</tr>
</tbody>
</table>

where $q$ must be initialized by $q = 0$.

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:

$$w[q] = b_n$$

cdelay2 ($D$, $&q$)
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 \( w \) 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 \( w \) 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 \( b[i] \), \( i = 0, 1, \ldots, D - 1 \), has already been defined:

```c
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;

for (n=0; n<D; n++) {
    *p = b[n]; // initialization part
    printf("%lf\n", *p); // fill buffer with b[n]'s
    cdelay(D, w, &p); // current output
} // update circular delay line

for (n=0; n<D; n++) {
    w[0] = b[n]; // steadystate part
    printf("%lf\n", w[0]); // first state = last state
    delay(D, w); // current output
} // update linear delay line
```

For comparison, we also list the linear delay-line version of Eqs. (16.1.16) and (16.1.17):

```c
for (n=0; n<D; n++) {
    w[0] = b[n]; // initialization part
    printf("%lf\n", w[0]); // fill buffer with b[n]'s
    delay(D, w); // update linear delay line
} // current output
```

The peaks in the spectrum are due to the zeros of the denominator which vanishes at the harmonics, that is,

\[
\sin \left( \frac{\pi m f}{f_s} \right) = 0 \Rightarrow \frac{\pi m D}{f_s} = \pi m
\]

with \( m \) an integer. Solving for \( f \):

\[
f_m = m \frac{f_s}{D} = mf_1
\]

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 \( m = 0, 1, \ldots, D - 1 \), 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

\[
1 - z^{-D} = 1 - e^{-j\omega_D} = e^{-j\omega_D/2} (e^{j\omega_D/2} - e^{-j\omega_D/2}) = 2j e^{j\omega_D/2} \sin \left( \frac{\omega_D}{2} \right)
\]

we have

\[
|H(\omega)| = \frac{|N(\omega)|}{|1 - e^{-j\omega_D}|} = \frac{|N(\omega)|}{2 |\sin(\omega_D/2)|}
\]

or, replacing \( \omega = 2\pi f / f_s \) in terms of the physical frequency \( f \):

\[
|H(f)| = \frac{|N(f)|}{2 |\sin(\frac{\pi f D}{f_s})|}
\]

\[\text{Fig. 16.1.10 Comb-like frequency spectrum, for } D = 8.\]

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 \( z = e^{j\omega f} = e^{2\pi j f / f_s} \) in the generating transfer function (16.1.10). Using the trigonometric identity

\[
1 - z^{-D} = 1 - e^{-j\omega_D} = e^{-j\omega_D/2} (e^{j\omega_D/2} - e^{-j\omega_D/2}) = 2j e^{j\omega_D/2} \sin \left( \frac{\omega_D}{2} \right)
\]
16.1.3 Wavetable Generators

The linear and circular buffer implementations of the direct form generator both use \((D+1)\)-dimensional buffers \(w = [w_0, w_1, \ldots, w_D]\), whereas the periodic waveform has only \(D\) samples in one period: \(b = [b_0, b_1, \ldots, b_{D-1}]\). As we saw, this causes the buffer contents to recirculate periodically.

A simpler approach is to use a buffer of length \(D\), that is, \(w = [w_0, w_1, \ldots, w_{D-1}]\) 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 \(*p = w[q]\).

![Circular pointer cycles over wavetable.](image)

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 \(p = w\):

\[
\text{for } i = 0, 1, \ldots, D - 1 \text{ do:} \\
\hspace{1cm} *p = b_i \\
\hspace{1cm} \text{cdelay}(D - 1, w, &p)
\]

or, in terms of the offset index \(q\), initialized at \(q = 0\):

\[
\text{for } i = 0, 1, \ldots, D - 1 \text{ do:} \\
\hspace{1cm} w[q] = b_i \\
\hspace{1cm} \text{cdelay2}(D - 1, &q)
\]

Note that the only difference with Eqs. (16.1.19) and (16.1.21) is the dimension of the buffer \(w\), which requires the \text{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:

\[
\text{for } i = 0, 1, \ldots, D - 1 \text{ do:} \\
\hspace{1cm} w[i] = b[(D - i) \mod D]
\]

where the modulo operation is felt only at \(i = 0\), giving in this case \(w[0] = b[D \mod D] = b[0]\). After the wavetable is loaded with the waveform samples, the circular pointer can be made to cycle over the wavetable by successive calls to \text{cdelay}:

\[
\text{repeat forever:} \\
\hspace{1cm} \text{output } y = \ast p \\
\hspace{1cm} \text{cdelay}(D - 1, w, &p)
\]

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, \(w[i] = b[i], i = 0, 1, \ldots, D - 1\) 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 \text{gdelay2} discussed below.

The following program segment illustrates the initialization (16.1.25) and the steady-state operation (16.1.28):

```c
double *b, *w, *p;

b = (double *) calloc(D, sizeof(double));
for (i=0; i<D; i++) { 
    *p = b[i];
    cdelay(D-1, w, &p);
}

w = (double *) calloc(D, sizeof(double));
p = w;
initialization: for (i=0; i<D; i++) {
    *p = b[i];
    cdelay(D-1, w, &p);
}
for (n=0; n<Ntot; n++) { 
    *p = b[n];
    printf("%f\n", *p);
    cdelay2(D-1, w, &p);
}

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 \(0 \leq m \leq D - 1\), the starting pointer \(p\) and corresponding offset index \(q\) should be:

```
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:

\[
m = 0: \ [b_0, b_1, b_2, b_3, b_0, b_1, b_2, b_3, b_0, b_1, b_2, b_3, \ldots]
\]

\[
m = 1: \ [b_3, b_0, b_1, b_2, b_3, b_0, b_1, b_2, b_3, b_0, b_1, b_2, b_3, \ldots]
\]

\[
m = 2: \ [b_2, b_3, b_0, b_1, b_2, b_3, b_0, b_1, b_2, b_3, b_0, b_1, b_2, b_3, \ldots]
\]

\[
m = 3: \ [b_1, b_2, b_3, b_0, b_1, b_2, b_3, b_0, b_1, b_2, b_3, b_0, b_1, b_2, b_3, \ldots]
\]

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, genetically 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

\[
f = \frac{f_s}{D}
\]

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 \( d \leq D \) will increase the fundamental frequency to:

\[
f = \frac{f_s}{d}
\]

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:

\[
\text{repeat forever:}
\]

\[\text{output } y = w[q] \]

\[\text{cdelay2}(D - 1, q) \]

\[\text{cdelay2}(D - 1, q) \]

\[
\text{Fig. 16.1.12 Circular pointer is decremented by 2, generating a subsequence of period 4.}
\]

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:

\[
[b_0, b_2, b_4, b_6, b_0, b_2, b_4, b_6, b_0, b_2, b_4, b_6, \ldots]
\]
16.1. DIGITAL WAVEFORM GENERATORS

which repeats with period \( d = 4 \). It should be compared to the full wavetable sequence, which repeats every \( D = 8 \) samples:

\[
[b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7, \ldots ]
\]

Similarly, cycling every four wavetable samples will generate the periodic sub-sequence of period \( D/4 = 8/4 = 2 \) and frequency \( f = 4f_s/D \):

\[
[b_0, b_4, b_0, b_4, b_0, b_4, b_0, b_4, b_0, b_4, b_0, b_4, b_0, b_4, b_0, b_4, b_0, b_4, \ldots ]
\]

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 \( \text{cdelay2} \) at each iteration:

\[
\text{repeat forever:}
\begin{align*}
\text{output } y &= w[q] \\
\text{cdelay2}(D - 1, &q) \\
\text{cdelay2}(D - 1, &q) \\
\text{cdelay2}(D - 1, &q) \\
\text{cdelay2}(D - 1, &q)
\end{align*}
\]

(16.1.34)

Rather than making multiple calls to \( \text{cdelay2} \), we define a generalized version of this routine, \( \text{gdelay2.c} \), which allows for an arbitrary real-valued shift of the pointer index \( q \):

\[
\text{/* gdelay2.c - generalized circular delay with real-valued shift */}
\]

\[
\text{void gdelay2}(D, c, q)
\]

\[
\text{int } D; \text{double } c, *q;
\]

\[
\{ \\
\text{c-shift, } q\text{-offset index} \\
\text{\quad *q } \leftarrow c; \quad \text{decrement by } c \\
\text{if } (*q < 0) \\
\quad \text{*q } \leftarrow D+1; \\
\text{if } (*q > D) \\
\quad \text{*q } \leftarrow D+1;
\}
\]

There are two basic differences with \( \text{cdelay2} \). 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 \( \text{cdelay2} \) is a special case of \( \text{gdelay2} \) with \( c = 1.0 \). In terms of this routine, Eq. (16.1.33) or (16.1.34) will read as:

\[
\text{repeat forever:}
\begin{align*}
\text{output } y &= w[q] \\
\text{gdelay2}(D - 1, &c, &q)
\end{align*}
\]

(16.1.35)

with \( c = 2 \) or \( c = 4 \), respectively. The successive calls to \( \text{gdelay2} \) in Eq. (16.1.35) update the offset \( q \) according to the iteration:

\[
a_{n+1} = (a_n - c) \mod D
\]

(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 \( d \):

\[
c = \frac{D}{d}
\]

(16.1.37)

which gives the number of times the sub-period \( d \) fits into the full period \( D \), or,

\[
d = \frac{D}{c}
\]

(16.1.38)

which expresses \( d \) as a fraction of the full period \( D \), or,

\[
D = cd
\]

(16.1.39)

Combining Eqs. (16.1.32) and (16.1.38), gives for the frequency of the generated subsequence:

\[
f = \frac{f_s}{d} = \frac{f_s}{cD}
\]

(16.1.40)

or, in terms of the digital frequency in radians/sample:

\[
\omega = \frac{2\pi f_s}{D} = \frac{2\pi f}{D} = \frac{2\pi c}{D}
\]

(16.1.41)

Equivalently, given a desired frequency \( f \) and table length \( D \), we obtain the required value of the shift:

\[
c = D \frac{f}{f_s} = DF
\]

(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 \( \text{gdelay2} \).

To keep \( f \) within the symmetric Nyquist interval \(|f| \leq f_s/2 \), requires that \( c \) satisfy the condition: \(|cf_s/D| \leq f_s/2 \), or,

\[
|c| \leq \frac{D}{2} \Rightarrow -\frac{D}{2} \leq c \leq \frac{D}{2}
\]

(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 \( D/2 < c \leq D \) is wrapped modulo-\( D \) to the value \( c - D \), which lies in the negative part of the Nyquist
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 \( f = c f_s / D \) is accomplished by \textit{incrementing} the offset index \( q \) according to the iteration:

\[
q_{n+1} = (q_n + c) \% D \quad \text{(16.1.44)}
\]

Such “forward” versions can be implemented easily by the routine \texttt{gdelay2} by calling it in Eq. (16.1.35) with \( c \) replaced by \(-c\). The wrap-around tests in \texttt{gdelay2} always force \( q \) to lie in the range \( 0 \leq q < D \). 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:

\[
i = \lfloor q \rfloor \quad \text{(truncating down)}
\]
\[
j = \lfloor q + 1 \rfloor \% D \quad \text{(truncating up)}
\]
\[
k = \lfloor q + 0.5 \rfloor \% D \quad \text{(rounding)}
\quad \text{(16.1.45)}
\]

The modulo-\( D \) operation is necessary to keep the index within the circular buffer range \( \{0, 1, \ldots, D - 1\} \). The returned output will be in these cases:

\[
y = w[i] \quad \text{(truncating down)}
\]
\[
y = w[j] \quad \text{(truncating up)}
\]
\[
y = w[k] \quad \text{(rounding)}
\quad \text{(16.1.46)}
\]

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)
```

and similarly in the other cases.

Because \( q \) lies (circularly) between the integers \( i \) and \( j \), a more accurate output can be obtained by \textit{linearly interpolating} between the wavetable values \( w[i] \) and \( w[j] \), that is, returning the output:

\[
y = w[i] + (q - i) (w[j] - w[i])
\quad \text{(16.1.48)}
\]

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 \( i = j + 1 \), except when \( q \) falls in the last integer subdivision of the \( \{0, D\} \) interval, that is, when \( D - 1 \leq q < D \). 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].

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

\[
q = mc = mDF
\quad \text{(16.1.49)}
\]

Because \texttt{gdelay2} decrements \( q \), we can obtain Eq. (16.1.49) by starting with \( q = 0 \) and calling \texttt{gdelay2} once with the \textit{opposite argument}:

\[
gdelay2(D - 1, -mc, &q)
\quad \text{(16.1.50)}
\]

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:

\[
b = \{b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7\}
\]

are stored in (circular) reverse order in the 8-dimensional circular wavetable:

\[
w = \{w_0, w_1, w_2, w_3, w_4, w_5, w_6, w_7\}
\]

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):

\[
q_0 = 0
\]
\[
q_1 = q_0 - c = -8 - 8/3 = 16/3 = 5.33
\]
\[
q_2 = q_1 - c = 16/3 - 8/3 = 8/3 = 2.67
\]
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 $\omega$ as given by Eq. (16.1.41); here, $\omega = 2\pi/3$.

\begin{center}
\includegraphics[width=0.5\textwidth]{fig16.1.14.png}
\end{center}

**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 $w[5] = b_2$, and if we truncate up, $w[6] = b_2$. Because, $q_1$ points nearer to $w[5]$ than to $w[6]$, we will output $w[5] = b_2$, if we round to the nearest buffer location. If we interpolate linearly between $w[5]$ and $w[6]$, we will output the value:

$$y = w[5] + (q_1 - i_1) (w[6] - w[5]) = b_2 + \frac{1}{3} (b_2 - b_3) = \frac{1}{3} b_2 + \frac{2}{3} b_3$$

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 $w[3] = b_2$, 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 $n = 0, 1, 2, \ldots$, 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:

- $[b_0, b_3, b_6, b_0, b_3, b_6, \ldots]$ (truncate down)
- $[b_0, b_2, b_5, b_0, b_2, b_5, \ldots]$ (truncate up)
- $[b_0, b_3, b_5, b_0, b_3, b_5, \ldots]$ (round)

and if we interpolate:

$$[b_0, b_3, b_6, b_0, b_3, b_6, \ldots]$$

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 `delay2`, 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:

$$q_1 = q_0 - c = \frac{4}{3} - \frac{8}{3} = -\frac{4}{3} = \frac{8}{3} - \frac{20}{3} = \frac{2}{3}$$

$$q_2 = q_1 - c = \frac{20}{3} - \frac{8}{3} = 4$$

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 $\omega_0 = 2\pi mc/d = 2\pi (mc)/D$ 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 $\pi/3$.

\begin{center}
\includegraphics[width=0.5\textwidth]{fig16.1.15.png}
\end{center}

**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:

$$[q_0, q_1, q_2] = [1, 6, 4]$$

$$[w[1], w[6], w[4]] = [b_7, b_2, b_4]$$
Up-truncation gives:
\[ [q_0, q_1, q_2] = [2, 7, 4] \]
\[ [w[2], w[7], w[4]] = [b_6, b_1, b_4] \]

For the rounding case, we have
\[ [q_0, q_1, q_2] = [1, 7, 4] \]
\[ [w[1], w[7], w[4]] = [b_7, b_1, b_4] \]

For the linear interpolation case, we have the outputs:
\[ 1 < q_0 < 2 \Rightarrow y_0 = w[1] + (q_0 - 1)(w[2] - w[1]) = \frac{1}{3}b_6 + \frac{2}{3}b_7 \]
\[ 6 < q_1 < 7 \Rightarrow y_1 = w[6] + (q_1 - 6)(w[7] - w[6]) = \frac{2}{3}b_1 + \frac{1}{3}b_2 \]
\[ q_2 = 4 \Rightarrow y_2 = w[4] = b_4 \]

Thus, depending on the output method, the following period-3 delayed subsequences will be generated:
\[ [b_7, b_2, b_4, b_7, b_2, b_4, \ldots] \] (truncate down)
\[ [b_6, b_1, b_4, b_6, b_1, b_4, \ldots] \] (truncate up)
\[ [b_7, b_1, b_4, b_7, b_1, b_4, \ldots] \] (round)
\[ [y_0, y_1, y_2, y_0, y_1, y_2, \ldots] \] (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 = [q]
    output y = Aw[i]
gdelay2(D-1, DF, &q)
```

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):

```c
/* wavgen.c - wavetable generator (truncation method) */
void gdelay2();
double wavgen(D, w, A, F, q);
int D;
double *w, A, *q;
A = amplitude, F = frequency, q = offset index
{
    double y;
    int i;
    i = (int) (*q);
    y = A * w[i];
    gdelay2(D-1, D*F, q);
    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:

```c
/* wavgenr.c - wavetable generator (rounding method) */
void gdelay2();
double wavgenr(D, w, A, F, q);
int D;
double *w, A, *q;
A = amplitude, F = frequency, q = offset index
{
    double y;
    int k;
    k = (int) (*q + 0.5);
    y = A * w[k];
    gdelay2(D-1, D*F, q);
    return y;
}
```

```c
/* wavgeni.c - wavetable generator (interpolation method) */
void gdelay2();
double wavgeni(D, w, A, F, q);
int D;
double *w, A, *q;
A = amplitude, F = frequency, q = offset index
{
    double y;
    int i, j;
    i = (int) *q;
    interpolate between w[i], w[j]
```
16.1. DIGITAL WAVEFORM GENERATORS

In computer music, such routines are known as wavelet 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.

![Wavetable Generator](image)

**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:

$$y(n) = \text{wavgen}(D, w, A(n), F(n), &q)$$

(16.1.52)

for $n = 0, 1, 2, \ldots$, 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 $w$ 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.

![Waveforms](image)

**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 $w$ 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;
int D = 1000
w = (double *) calloc(D, sizeof(double));

for (i=0; i<D; i++)
{ w[(int)q1] = sine(D, i);}

gdelay2(D-1, -m*D*F2, &q4);
gdelay2(D-1, +m*D*F2, &q5);

for (n=0; n<Ntot; n++)
{ y1[n] = wavgen(D, w, A, F1, &q1);
y2[n] = wavgen(D, w, A, F2, &q2);
}

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);
}
use: \( y_3[n] = \text{wavgen}(D, w, A, F_3, &q_3); \)

use: \( F_3 = \frac{10.5}{D} \)

use: \( y_4[n] = \text{wavgen}(D, w, A, F_4, &q_4); \)

use: \( F_4 = F_2 \)

use: \( y_5[n] = \text{wavgen}(D, w, A, F_5, &q_5); \)

use: \( F_5 = \frac{2}{5} \)

which gives \( c_1 = DF_3 = 10.5 \), 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 \( 1000 \) samples and therefore \( F_2 \)-cycle contains \( 1 \) of a cycle.

Because they have frequency \( F_2 \) and wavetable shift \( c_2 = DF_2 = 5 \), their effective frequency is \( \frac{2}{5} \) and the corresponding shift is \( q_2 = -mc_2 = -125 \) (or, rather \( q_3 = 1000 - 125 = 855 \)). These initial \( q \)-values are obtained by the two calls to \( \text{gdelay2} \) preceding the generation loop, with arguments \( \pm mc_2 \).

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.

\[
\begin{align*}
A_{\text{env}} & \quad \text{wenv} \quad A(n) \quad \text{w} \quad y(n) \\
F_{\text{env}} & \quad \text{env} \quad F \quad x(n) \quad \text{env} \quad A(n) \quad \text{w} \quad y(n)
\end{align*}
\]

As an example, consider the generation of a sinusoidal note of frequency \( F = 0.01 \) cycles/sample:

\[
x(n) = \sin(2\pi Fn), \quad n = 0, 1, \ldots, N_{\text{tot}} - 1
\]

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:

```c
double *w, *wenv, q, qenv;
allocate wavetables
w = (double *) calloc(D, sizeof(double));
load wavetables
wenv = (double *) calloc(D, sizeof(double));
triangular envelope
double q, qenv = 0;
initialize offsets
for (i=0; i<D; i++) {
    w[q] = sine(D, i);
    wenv[qenv] = trapez(D, D/4, 0, i);
    gdelay2(D-1, 1.0, &q);
gdelay2(D-1, 1.0, &qenv);
}
use:
Ntot = 1000 or 2000
envelope frequency
for (n=0; n<Ntot; n++) {
    A[n] = wavgen(D, wenv, Aenv, Fenv, &qenv);
    y[n] = wavgen(D, w, A[n], F, &q);  
}  
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 $N_{tot}$, 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 $c_{env} = D F_{env} = D/N_{tot} = 1$ or 0.5 in the two cases.

Figure 16.1.20 shows the two cases $N_{tot} = 1000,2000$. Because $F = 0.01$ cycles/sample, there are $F N_{tot}$ cycles of the sinusoid in the duration of $N_{tot}$, that is, $F N_{tot} = 0.01 \times 1000 = 10$ cycles in the first case, and $F N_{tot} = 0.01 \times 2000 = 20$ cycles in the second. For visual reference, the graphs also plot the triangular envelope $A(n)$ and its negative, $-A(n)$.

Figure 16.1.21 shows another example, where the envelope signal was chosen to be varying sinusoidally about a constant value:

$$A(n) = 1 + 0.25 \cos(2\pi F_{env} n)$$

so that the generated waveform will be:

$$y(n) = A(n) x(n) = (1 + 0.25 \cos(2\pi F_{env} n)) \sin(2\pi F n)$$

The envelope frequency was chosen to be $F_{env} = 2/N_{tot}$ for the first graph and $F_{env} = 3.5/N_{tot}$ for the second. These choices correspond to 2 and 3.5 envelope cycles in $N_{tot}$ samples. With these values of $F_{env}$, 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:

```c
q = qenv = 0;
initialize offsets
for (i=0; i<D; i++) {
    w[q] = sine(D, i);
    wenv[qenv] = 1 + 0.25 * sine(D, i);
    gdelay2(D-1, 1.0, &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 $w_m$ will produce the frequency:

$$F(n) = F_C + A_m \sin(2\pi F_m n)$$

so that if the signal generator $w$ is a unit-amplitude sinusoid, then the modulated output will be:
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:

```c
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); */
signals: y4(n)
gdelay2(D-1, 1.0, &q);
    wm[qm] = sine(D, i);
signals: y1(n)
    /* wm[qm] = 2 * square(D/2, i) - 1; */
signals: y2(n)
gdelay2(D-1, 1.0, &qm);
    /* 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 w and wm were D = 1000 and the signal duration Ntot = 1000. The signal y1(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 FmNtot = 3 cycles in the Ntot samples. The frequency F(n) rises and falls between the limits \[ F_c - A_m \leq F(n) \leq F_c + A_m, \] or \[ 0.5F_c \leq F(n) \leq 1.5F_c. \] The quantity F(n)/Fc is also plotted in order to help visualize the effect of increasing and decreasing frequency.

The signal y2(n) is a sinusoid whose frequency is modulated by a square wave that switches between the values \[ F_c + A_m \text{ and } F_c - A_m, \] 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 y2(n) switches frequency more abruptly than y1(n).

The signal y3(n) is a sinusoid whose frequency is linearly swept between the values \[ F_c \leq F(n) \leq F_c + A_m, \] 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 \[ D_1 = D \text{ and } D_2 = 0. \] Its frequency was chosen to be 2 cycles in 1000 samples, or \[ F_m = 0.002. \]

Finally, the signal y4(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.

![Audio effects signal processor](image)

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, reverbération, 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 reverbération 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:

$$y(n) = x(n) + ax(n - D)$$  \hspace{1cm} \text{(echo filter)} \hspace{1cm} (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 $|a| \leq 1$. The transfer function and impulse response of this filter are:

$$H(z) = 1 + az^{-D}, \quad h(n) = \delta(n) + a\delta(n - D) \hspace{1cm} (16.2.2)$$

Its block diagram realization is shown in Fig. 16.2.2. The frequency response is obtained from Eq. (16.2.2) by setting $z = e^{j\omega}$:

$$H(\omega) = 1 + ae^{-j\omega D}, \quad |H(\omega)| = \sqrt{1 + 2a \cos(\omega D) + a^2} \hspace{1cm} (16.2.3)$$

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 $0 < a < 1$):

$$1 + az^{-D} = 0 \quad \Rightarrow \quad z_k = \rho e^{j(2\pi k + 1)/D}, \quad k = 0, 1, \ldots, D - 1 \hspace{1cm} (16.2.4)$$

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.

![FIR comb filter](image)
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 \( w = [w_0, w_1, \ldots, w_D] \), we have:

```plaintext
for each input sample \( x \) do:
  \( y = x + a w_D \)
  \( w_0 = x \)
  \( \text{delay}(D, w) \)
```

Note that the quantities \( w_D \) in the linear case and \( s_D = \text{tap}(D, w, p, D) \) in the circular one represent the \( D \)th 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 \( a > 0 \)):

\[
y(n) = x(n) - a x(n-D) \quad (16.2.6)
\]

The transfer function and frequency response are now

\[
H(z) = 1 - az^{-D}, \quad H(\omega) = 1 - ae^{-j\omega D} \quad (16.2.7)
\]

having peaks at \( \omega_k = (2k+1)\pi/D \) and dips at \( \omega_k = 2\pi k/D, \quad k = 0,1,\ldots,D-1 \). 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:

\[
y(n) = x(n)+ax(n-D)+a^2 x(n-2D)+a^3 x(n-3D) \quad (16.2.8)
\]

Using the finite geometric series, we can express the transfer function as

\[
H(z) = 1 + az^{-D} + a^2 z^{-2D} + a^3 z^{-3D} = \frac{1 - a^4 z^{-4D}}{1 - az^{-D}} \quad (16.2.9)
\]

or, equivalently, at

\[
z_k = pe^{2\pi jk/4D}, \quad k = 0,1,\ldots,4D-1, \quad \text{but } k \text{ not a multiple of } 4
\]

The filter has peaks at frequencies for which \( k \) is a multiple of 4, indeed, if \( k = 4m, \quad m = 0,1,\ldots,D-1 \), then

\[
\omega_k = \frac{2\pi k}{4D} = \frac{2\pi (4m)}{4D} = \frac{2\pi m}{D} \Rightarrow e^{j\omega_k} = 1
\]

and the filter's response takes on the maximum value \( H(\omega_k) = 1 + a + a^2 + a^3 \).

The dips occur at the 32nd roots of unity, except at the 8th roots of unity at which there are peaks.

![Fig. 16.2.4 Comb filter of Eq. (16.2.6), with dips at \( \omega_k = 2\pi k/D, k = 0,1,\ldots,D-1 \).](image)

![Fig. 16.2.5 Comb filter of Eq. (16.2.8), with peaks at \( \omega_k = 2\pi k/D, k = 0,1,\ldots,D-1 \).](image)

Adding up an infinite number of successive echoes imitates the reverberating nature of a room and gives rise to an IIR comb filter:

\[
y(n) = x(n)+ax(n-D)+a^2 x(n-2D)+\cdots \quad (16.2.10)
\]

which has impulse response:

\[
h(n) = \delta(n)+a\delta(n-D)+a^2\delta(n-2D)+\cdots \quad (16.2.11)
\]

and transfer function:

\[
H(z) = 1 + az^{-D} + a^2 z^{-2D} + \cdots
\]

which can be summed by the geometric series into the form:

\[
H(z) = \frac{1}{1 - az^{-D}} \quad (plain \text{ reverberator}) \quad (16.2.12)
\]

The I/O equation (16.2.10) can then be recast recursively as

\[
y(n) = ay(n-D) + x(n) \quad (16.2.13)
\]

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 \( n = 0, D, 2D, \ldots \). 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 $p_k = \rho^k e^{j\omega_k}$, $k = 0, 1, \ldots, D - 1$, where $\omega_k = 2\pi k/D$ and $\rho = a^{1/D}$. They are spaced equally around the circle of radius $\rho$, as shown in Fig. 16.2.7, for $D = 8$. At the pole frequencies $\omega_k$, 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 $\rho$ is.

The effective time constant for the filter response to decay below a certain level, $\tau = \frac{\ln \epsilon}{\ln a}$, can be obtained following the discussion of Section 6.3.2. At time $t = mD$, the impulse response has dropped to $\rho^n = \rho^m D = a^n$; therefore, the effective time constant $n_{eff} = m_{eff} D$ will be such that

$$\rho^n D = \epsilon$$

which can be solved for $m_{eff}$ and $n_{eff}$:

$$n_{eff} = m_{eff} D = \frac{\ln \epsilon}{\ln a} D = \frac{\ln \epsilon}{\ln \rho}$$

and in seconds:

$$\tau_{eff} = n_{eff} T = \frac{\ln \epsilon}{\ln a} T D$$

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 $T_D = D/f_s$, 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 100 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:

$$y(n) = x(n) + ax(n - d(n))$$

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 $0 \leq d(n) \leq D$ will be:

$$d(n) = D \frac{1 - \cos(2\pi F_d n)}{2}$$
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 \( \pi/d \).

Because the variable delay \( d \) can take non-integer values within its range \( 0 \leq d \leq D \), the realization of Eq. (16.2.17) requires the calculation of the output \( s(n-d) \) 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 \( d \).

```c
/* tapi.c - interpolated tap output of a delay line */

double tap();
double tapi(D, w, p, d);

int D;
{
  int i, j;
  double si, sj;
  i = (int) d;
  j = (i+1) % (D+1);
  si = tap(D, w, p, i);
  sj = tap(D, w, p, j);
  return si + (d - i) * (sj - si);
}
```

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 \( 0 \leq d \leq D \), the implementation of Eq. (16.2.17) requires the calculation of the output \( s(n-d) \) 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 \( d \).

```c
/* tapi.c - interpolated tap output of a delay line */

double tap();
double tapi(D, w, p, d);

int D;
{
  int i, j;
  double si, sj;
  i = (int) d;
  j = (i+1) % (D+1);
  si = tap(D, w, p, i);
  sj = tap(D, w, p, j);
  return si + (d - i) * (sj - si);
}
```

The input \( d \) must always be restricted to the range \( 0 \leq d \leq D \). Note that if \( d \) is one of the integers \( 0, 1,..., D \), 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 \( 0 \leq j \leq D \), 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 \( p = w + q \).

```c
/* tapi2.c - interpolated tap output of a delay line */

double tap2();
double tapi2(D, w, q, d);

int D, q;
{
  int i, j;
  double si, sj;
  i = (int) d;
  j = (i+1) % (D+1);
  si = tap2(D, w, q, i);
  sj = tap2(D, w, q, 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 \( N_{tot} = 200 \) samples, so that there are \( F N_{tot} = 10 \) cycles in the 200 samples. The flanged signal is computed by

\[
y(n) = \frac{1}{2} [x(n) + x(n - d(n))] \quad (16.2.19)
\]

with \( d(n) \) given by Eq. (16.2.18), \( D = 20 \), and \( F_d = 0.01 \) cycles/sample, so that there are \( F_d N_{tot} = 2 \) cycles in the 200 samples.

The following program segment implements the calculation of the term \( s(n) = x(n - d(n)) \) and \( y(n) \). A delay-line buffer of maximal dimension \( D + 1 = 21 \) was used:

```c
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 * F_d * n));
  x = cos(2 * pi * F_d * n);
  s = tapi(D, w, p, d);
  y = 0.5 * (x + s);
  *p = x;
  cdelay(D, w, &p);
  ...
}
```
Flanged sinusoidal signal.

Figure 16.2.9 shows the signals $x(n)$, $s(n) = x(n - d(n))$, $y(n)$, 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 $d(n)$. 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.

The small variations in the time delays and amplitudes can be simulated by varying them slowly and randomly [135,136]. A low-frequency random signal varying between $[-0.5, 0.5]$ can be generated by the linearly interpolated generator routine $\text{ranl}$ of Appendix A.2. Given a desired rate of variation $F_{\text{ran}}$ cycles/sample for $v(n)$, we obtain the period $D_{\text{ran}} = 1/F_{\text{ran}}$ of the generator $\text{ranl}$.

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 $\text{ranl}$ was $D_{\text{ran}} = 1/F_{\text{ran}} = 40$ samples. The same program segment applies here, but with the change:

$$d = D \times (0.5 + \text{ranl}(D_{\text{ran}}, u, q, &\text{iseed}));$$

where the routine parameters $u, q, \text{iseed}$ are described in Appendix A.2.

Figure 16.2.11 shows the signals $x(n)$, $s(n) = x(n - d(n))$, $y(n)$, as well as the quantity $d(n)/D$.

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.
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 $H(\omega)$ 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:

$$\Delta \omega = \frac{\omega_0}{Q} = \frac{0.35\pi}{3.5} = 0.10\pi \quad \text{and} \quad \Delta \omega = \frac{\omega_0}{Q} = \frac{0.35\pi}{35} = 0.01\pi$$

The filter coefficients are then computed from Eq. (16.2.23), giving the transfer functions in the two cases:

$$H(z) = \frac{0.8633 - 0.9080z^{-1} + z^{-2}}{1 - 0.7838z^{-1} + 0.7265z^{-2}}, \quad \text{(for } Q = 3.5)$$

$$H(z) = \frac{0.9845 - 0.9080z^{-1} + z^{-2}}{1 - 0.8939z^{-1} + 0.9691z^{-2}}, \quad \text{(for } Q = 35)$$

The magnitude squared and phase responses are shown in Fig. 16.2.14.

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 $\omega_{\text{sweep}}$, 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):

$$Q = \frac{\omega_0}{\Delta \omega} \quad \Rightarrow \quad \Delta \omega = \frac{\omega_0}{Q}$$ (16.2.24)
for each time instant $n$ and input sample $x$ do:
compute current notch $\omega_0 = \omega_1 + \omega_2 \sin(\omega_{\text{sweep}} n)$

$w_0 = bx + 2b \cos(\omega_0) w_1 - (2b - 1) w_2$

$y = w_0 - 2 \cos(\omega_0) w_1 + w_2$

$w_1 = w_0$

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 $180^\circ$ at the notch. If the output of the allpass filter is added to its input, the $180^\circ$ 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.

The plain reverb filter shown in Fig. 16.2.6 is too simple to produce a realistic reverbération 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:

$$H(z) = \frac{-a + z^{-D}}{1 - az^{-D}}$$  \hspace{1cm} (allpass reverberator) \hspace{1cm} (16.2.25)

It has I/O difference equation:
Its frequency and magnitude responses are obtained by setting $z = e^{j\omega}$:

$$H(\omega) = \frac{-a + e^{-j\omega D}}{1 - ae^{-j\omega D}} \Rightarrow |H(\omega)| = 1, \text{ for all } \omega \quad (16.2.27)$$

The magnitude response is constant in $\omega$ because the numerator and denominator of $H(\omega)$ have the same magnitude, as can be seen from the simple identity:

$$|\frac{-a + e^{-j\omega D}}{1 - ae^{-j\omega D}}| = \sqrt{1 - 2a \cos(\omega D) + a^2} = |\frac{1 - ae^{-j\omega D}}{1 - ae^{-j\omega D}}|$$

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:

$$H(z) = A + \frac{B}{1 - az^{-D}} \quad (16.2.28)$$

where $A = -1/a$ and $B = (1 - a^2)/a$. Expanding the $B$-term into its geometric series, gives

$$H(z) = (A + B) + B (az^{-D} + a^2 z^{-2D} + a^3 z^{-3D} + \cdots)$$

and taking inverse $z$-transforms leads to the impulse response:

$$h(n) = (A + B) \delta(n) + Ba \delta(n-D) + Ba^2 \delta(n-2D) + \cdots \quad (16.2.29)$$

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].

For each input sample $x$ do:

$$s_D = \text{tap}(D, w, p, D)$$
$$s_0 = x + as_D$$
$$y = -as_0 + s_D$$
$$*p = s_0$$
$$\text{delay}(D, w)$$

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:

```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);
int D;
{
    double y, sD;
    sD = tap(D, w, *p, D);
    y = x + a * sD;
    **p = y;
    cdelay(D, w, p);
    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);
p is passed by address
int D;
{
    double y, s0, sD;
    sD = tap(D, w, *p, D);
    y = x + a * sD;
    **p = y;
    cdelay(D, w, p);
    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:
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.

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 \(w_i\) and corresponding circular pointer \(p_i\):

```c
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, a1, x[n]);
y2[n] = allpass(D, w2, &p2, a2, x[n]);
}
```

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:

\[
\begin{align*}
D_1 &= 29, D_2 = 37, D_3 = 44, D_4 = 50, D_5 = 27, D_6 = 31 \\
a_1 &= a_2 = a_3 = a_4 = a_5 = a_6 = 0.75 \\
b_1 &= 1, b_2 = 0.9, b_3 = 0.8, b_4 = 0.7
\end{align*}
\]

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:

\[
H(z) = \frac{1}{1 - z^{-D}G(z)} \quad \text{(lowpass reverberator)}
\]  

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 \( h(n) \):
\[
h(n) = \delta(n) + g(n-D) + (g \ast g)(n-2D) + (g \ast g \ast g)(n-3D) + \cdots
\]
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 \( g \ast g \) 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.](image)

The feedback filter \( G(z) \) can be FIR or IIR. It is described, in general, by the following \( M \)th order transfer function, which also includes the FIR case:
\[
G(z) = \frac{N(z)}{D(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_M z^{-M}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_M z^{-M}} \quad (16.2.33)
\]

The filtering operation by \( G(z) \) can be implemented by the canonical realization routine. Assuming a \( (D+1) \)-dimensional circular buffer \( w \) for the delay \( D \), and an \( (M+1) \)-dimensional linear delay-line buffer \( v = [v_0, v_1, \ldots, v_M] \) for \( G(z) \), we can write the sample processing algorithm of Eq. (16.2.32), as follows:

```c
for each input sample x do:
    u = can(M, a, M, b, v, wD)
    y = x + u
    w0 = y
    delay(D, w)
```

where the input to the filter \( G(z) \) is the \( D \)th tap output \( w_0 \) 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 \( G(z) \).

```c
/* lowpass.c - lowpass reverberator with feedback filter G(z) */

void cdelay();

double lowpass(D, w, p, M, a, b, v, x)
{
    int D;
    double *w, *p;

    for (n=0; n<Ntot; n++)
        y[n] = lowpass(D, w, &p, M, a, b, v, x[n]);
    return y;
}
```

As a simple example, consider the following first-order IIR filter [148] with transfer function:
\[
G(z) = \frac{b_0 + b_1 z^{-1}}{1 + 0.3 + 0.15 z^{-1}} \quad (16.2.34)
\]

and weight vectors \( a = [1, a_1] = [1, -0.5] \) and \( b = [b_0, b_1] = [0.3, 0.15] \).

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:

```c
double *w, *p;

double v[2] = {0.0, 0.0};

double a[2] = {1.0, -0.5};

double b[2] = {0.3, 0.15};

double *w = (double *) calloc(D+1, sizeof(double));

for (n=0; n<Ntot; n++)
    y[n] = lowpass(D, w, &p, M, a, b, v, x[n]);
```

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,
\[
x = [1]
\]
\[
x = [1, 1, 1, 1, 1]
\]
\[
x = [0, 1, 2, 3, 4, 5, 4, 3, 2, 1, 0]
\]

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
smoother (being successively lowpass filtered) and longer, and eventually will overlap as they decay.

The plain and allpass reverberators have poles that are equidistant from the origin of the unit circle at radius $\rho = \frac{a_1}{D}$, and are equally spaced around the circle at the $D$ root-of-unity angles $\omega_k = \frac{2\pi k}{D}$, $k = 0, 1, \ldots, D - 1$. 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.

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.

The pole locations of Eq. (16.2.32) are obtained as the roots of the denominator, that is, they are the solutions of

$$z^D = G(z) = \frac{N(z)}{D(z)}$$

For our example, $D(z)$ has order one and therefore Eq. (16.2.36) will have $D+1$ poles, say, $p_i$, $i = 1, 2, \ldots, D + 1$. Writing $p_i$ in its polar form $p_i = \rho e^{j\omega_i}$, we have

$$\rho^D e^{j\omega_i D} = G(p_i) = |G(p_i)| e^{j\arg G(p_i)}$$

Defining the phase delay of the $i$th pole by

$$d_i = -\frac{\arg G(p_i)}{\omega_i}$$

we have
\[ p_i^D e^{j\omega_i D} = |G(p_i)| e^{-j\omega_i d_i}, \]

which can be separated into the two equations:

\[ p_i^D = |G(p_i)|, \quad e^{j\omega_i(D+d_i)} = 1 \]

and give

\[ p_i = |G(p_i)|^{1/D}, \quad \omega_i = \frac{2\pi k_i}{D + d_i} \tag{16.2.37} \]

for some integer \( k_i \).

Although these are coupled equations in the unknowns \( p_i, \omega_i \), we can see how the angles \( \omega_i \) will be distributed around the unit circle, near the \( D \)th roots of unity. Similarly, assuming \( p_i \) is near 1 and replacing \( G(p_i) \approx G(e^{j\omega_i}) = G(\omega_i) \), we have the approximation:

\[ p_i \approx |G(\omega_i)|^{1/D} \tag{16.2.38} \]

which by the lowpass nature of \( G(\omega) \) implies that \( p_i \) will be smaller for higher frequencies \( \omega_i \) 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 \( \epsilon \)-level time constants, in units of the delay time \( T_D = TD \):

\[ t_1 = \frac{\ln \epsilon}{D \ln p_i}, \quad T_D = \frac{\ln \epsilon}{\ln |G(\omega_i)|} T_D \tag{16.2.39} \]

It follows from Eq. (16.2.38) that the stability of the reverber filter \( H(z) \), that is, \( p_i < 1 \), will be guaranteed if the feedback filter is normalized such that \(|G(\omega)| < 1\), for all \( \omega \). Regardless of the above approximation, this condition implies stability by Nyquist’s stability criterion or Rouché’s theorem [39]. For our example 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 \( p^D \approx 0 \) and therefore, it corresponds to \( N(p) \approx 0 \). 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 \( \omega_i \) and magnitudes \( \rho_i \), as well as the approximate magnitudes given by Eq. (16.2.38), and the exact 60-dB (\( \epsilon = 10^{-3} \)) time constants \( t_i \).

The first \( D \) pole angles are approximately equal to the \( D \)th 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 \( N(z) \).

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 \( D = 30, \alpha = 0.75 \), 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 drumsounds [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 \( w = [w_0, w_1, \ldots, w_D] \) 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 \( f_s \), 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].

| \( p_i \) | \( \omega_i/\pi \) | \( \rho_i \) | \( |G(\omega_i)|^{1/D} \) | \( \tau_i/T_D \) |
|---|---|---|---|---|
| 0.9888 | 0 | 0.9888 | 0.9869 | 76.594 |
| 0.7282 ± 0.6026 | ±0.2201 | 0.9452 | 0.9412 | 15.314 |
| 0.1128 ± 0.8651 | ±0.4587 | 0.8724 | 0.8715 | 6.326 |
| -0.4866 ± 0.6303 | ±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 \( D = 8 \).
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, w)
```

As a second example, consider the following second-order FIR filter:

```
H(z) = b_0 + b_1 z^{-D_1} + b_2 z^{-D_1} z^{-D_2}
```

The replacements (16.2.41) lead to the following three multi-delay filters, which are progressively more complicated:

```
H(z) = b_0 + b_1 \begin{bmatrix} z^{-D_1} \\ 1 - a_1 z^{-D_1} \end{bmatrix} + b_2 \begin{bmatrix} z^{-D_1} \\ 1 - a_2 z^{-D_1} \end{bmatrix} \begin{bmatrix} z^{-D_2} \\ 1 - a_2 z^{-D_2} \end{bmatrix}
```

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_0 x + b_1 w_{1D} + b_2 w_{2D}
  u_2 = can(G_2, w_{2D})
  w_{20} = w_{1D} + u_2
delay(D_2, w_2)
  u_1 = can(G_1, w_{1D})
  w_{10} = x + u_1
delay(D_1, w_1)
```

### 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} \rightarrow z^{-D} = \frac{z^{-D}}{1 - az^{-D}} \rightarrow \frac{z^{-D}}{1 - z^{-D} G(z)}
```

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:

```
H(z) = b_0 + b_1 z^{-1} + b_2 z^{-2}
```

Expanding in powers of $z^{-D}$, we have

```
H(z) = z^{-D} + az^{-2D} + a^2 z^{-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
```
16.2. DIGITAL AUDIO EFFECTS

where the statement $u_t = \text{can}(G_2, w_{2D})$ 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:

$$b_0 = 1, \quad b_1 = 0.8, \quad b_2 = 0.6$$

and the two cases for the feedback filters:

$$G_1(z) = G_2(z) = \begin{cases} 0.75 & \text{plain} \\ 0.3 + 0.15z^{-1} & \frac{1}{1 - 0.5z^{-1}} & \text{lowpass} \end{cases}$$

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].

and replace each single delay $z^{-1}$ by a multiple delay $z^{-D}$, getting the transfer function:

$$H(z) = b_0 + \frac{b_1z^{-D_1} + b_2z^{-D_1-D_2}}{1 - a_1z^{-D_1} - a_2z^{-D_1-D_2}}$$  \hspace{1cm} (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.

As a third example, we can start with the simple second-order IIR filter:

$$H(z) = b_0 + \frac{b_1z^{-1} + b_2z^{-2}}{1 - a_1z^{-1} - a_2z^{-2}}$$

The sample processing algorithm can be implemented with a circular $(D_1+D_2)$-dimensional delay-line buffer $w$ and pointer $p$, as follows:

\textbf{for each input sample $x$ do:}
- $s_1 = \text{tap}(D_1 + D_2, w, p, D_1)$
- $s_2 = \text{tap}(D_1 + D_2, w, p, D_1 + D_2)$
- $y = b_0x + b_1s_1 + b_2s_2$
- $s_0 = x + a_1s_1 + a_2s_2$
- $p = s_0$
- $\text{delay}(D_1 + D_2, w, &p)$

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

$$b_0 = 1, \quad b_1 = 0.8, \quad b_2 = 0.6$$

and for the following two choices of feedback gains, one of which is stable and the other unstable:
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:

\[ y = y_0 \left( \frac{x}{x_0} \right)^\rho \Rightarrow 20 \log_{10} \left( \frac{y}{y_0} \right) = \rho \left( 20 \log_{10} \left( \frac{x}{x_0} \right) \right) \quad (16.3.1) \]

where \( x \) is here a constant input, \( x_0 \) a desired threshold, and \( \rho \) 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 this relationships in dB, so that a 1 dB change in the input causes \( \rho \) dB change in the output, that is, \( \rho \) 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.

![Impulse response of multi-tap delay line.](image)

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, \( 0 < \lambda < 1 \),

\[ c_n = \lambda c_{n-1} + (1 - \lambda) |x_n| \quad (16.3.2) \]

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

\[ \text{The inverse quantity, } R = 1/\rho, \text{ is commonly called the “compression ratio,” whereas } \rho \text{ is the “slope.”} \]

![Static input/output relationship of compressor or expander.](image)
16.3. DYNAMIC RANGE CONTROL

Thus, the gain $g_n$ and the final output signal $y_n$ are computed as follows:

$$g_n = F(c_n)$$
$$y_n = g_n x_n$$  \hspace{1cm} (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 $\lambda$ 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:

$$A_n = \begin{cases} A_1, & 0 \leq n < 200 \\ A_2, & 200 \leq n < 400 \\ A_3, & 400 \leq n < 600 \end{cases}$$

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.

$$g = F(c) = \begin{cases} \left( \frac{c}{c_0} \right)^{-\rho}, & \text{if } c \geq c_0 \\ 1, & \text{if } c \leq c_0 \end{cases}$$ \hspace{1cm} (16.3.6)

where $c_0$ is a desired threshold and $\rho < 1$. For an expander, we have $\rho > 1$ and:

$$g = F(c) = \begin{cases} 1, & \text{if } c \geq c_0 \\ \left( \frac{c}{c_0} \right)^{\rho-1}, & \text{if } c \leq c_0 \end{cases}$$ \hspace{1cm} (16.3.7)

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. 16.3.2 Compressor/expander dynamics processor.](image-url)

and unit-step responses of this filter are,

$$H(z) = \frac{1 - \lambda}{1 - \lambda z^{-1}}, \quad h_n = (1 - \lambda \lambda^n u(n), \quad (h * u)_n = (1 - \lambda^{n+1}) u(n) \quad (16.3.3)$$

Thus the filter output responds exponentially quickly to a level change, with an effective time constant,

$$\lambda^{n_{\text{eff}}} = \epsilon 
\Rightarrow \quad n_{\text{eff}} = \frac{\ln \epsilon}{\ln \lambda} \Rightarrow \quad \lambda = \epsilon^{1/n_{\text{eff}}} = \epsilon^{T_s/t_{\text{eff}}} \quad (16.3.4)$$

and $\epsilon$ 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 $\epsilon$, Eq. (16.3.4) for calculating the filter parameter $\lambda$ can be written in the approximate equivalent exponential forms,

$$\epsilon = [0.1, 0.01, 0.001, 0.05] \approx [e^{-2.3}, \ e^{-4.6}, \ e^{-6.9}, \ e^{-8.6}]$$

$$\lambda = [e^{-2.3T_s/t_{\text{eff}}}, \ e^{-4.6T_s/t_{\text{eff}}}, \ e^{-6.9T_s/t_{\text{eff}}}, \ e^{-8.6T_s/t_{\text{eff}}}] \quad (16.3.5)$$

The time constant, $t_{\text{eff}}$, 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, $t_{\text{eff}}$ is typically specified in milliseconds.

When $t_{\text{eff}} = 0$, 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 $|x_n|\sqrt{c_n}$ 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:

$$g = F(c) = \begin{cases} \left( \frac{c}{c_0} \right)^{-\rho}, & \text{if } c \geq c_0 \\ 1, & \text{if } c \leq c_0 \end{cases}$$ \hspace{1cm} (16.3.6)

where $c_0$ is a desired threshold and $\rho < 1$. For an expander, we have $\rho > 1$ and:

$$g = F(c) = \begin{cases} 1, & \text{if } c \geq c_0 \\ \left( \frac{c}{c_0} \right)^{\rho-1}, & \text{if } c \leq c_0 \end{cases}$$ \hspace{1cm} (16.3.7)
16.3. Dynamic Range Control

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 $\lambda$-parameters, say, $\lambda_a$, $\lambda_r$.
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 $x_n$, 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 $\lambda_a$ to release mode $\lambda_r$ depending on the calculated level,

$$c_n = \begin{cases} 
\lambda_a c_{n-1} + (1 - \lambda_a) |x_n|, & \text{if } |x_n| \geq c_{n-1} \\
\lambda_r c_{n-1} + (1 - \lambda_r) |x_n|, & \text{if } |x_n| < c_{n-1} 
\end{cases}$$

(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 be taken to be either an FIR averager or an EMA filter of the form,

$$G_n = \frac{1}{L} \left[ g_n + g_{n-1} + \cdots + g_{n-L+1} \right] \quad \text{(FIR)}$$

(16.3.11)

$$G_n = \lambda G_{n-1} + (1 - \lambda) g_n \quad \text{(EMA)}$$

The averager and EMA filters behave approximately equivalently if the parameters, $L, \lambda$, are chosen such that:†

$$L \approx \frac{1 + \lambda}{1 - \lambda} \quad \Rightarrow \quad \lambda \approx \frac{L - 1}{L + 1}$$

(16.3.12)

In some implementations [164,169], the EMA smoother is chosen to have different attack and release time constants as follows,

$$G_n = \begin{cases} 
\lambda_a G_{n-1} + (1 - \lambda_a) g_n, & \text{if } g_n \geq g_{n-1} \\
\lambda_r G_{n-1} + (1 - \lambda_r) g_n, & \text{if } g_n < g_{n-1} 
\end{cases}$$

(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)
16.3 DYNAMIC RANGE CONTROL

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 $x(t) = A \cos(2\pi ft)$. Show that its mean-square average over one period, and its absolute average are given by:

$$\langle x^2(t) \rangle = \frac{1}{2} A^2, \quad |x(t)| = \frac{2}{\pi} A$$  \hspace{1cm} (16.3.14)

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 $\lambda_a, \lambda_r$, 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 $\lambda$ based on the attack value of $\lambda_a$.

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,

<table>
<thead>
<tr>
<th>Frequency (kHz)</th>
<th>Amplitude</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1$</td>
<td>$A_1$</td>
<td>$0 \leq t &lt; 25$ msec</td>
</tr>
<tr>
<td>$f_2$</td>
<td>$A_2$</td>
<td>$25 \leq t &lt; 50$ msec</td>
</tr>
<tr>
<td>$f_3$</td>
<td>$A_3$</td>
<td>$50 \leq t &lt; 75$ msec</td>
</tr>
</tbody>
</table>

Calculate the mean-absolute values of the three levels. Plot $x(t)$ versus $t$. 

16.3. DYNAMIC RANGE CONTROL

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. \( t \) 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 \( G(t) \). Plot the corresponding output \( y(t) \). Note: 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 \( \rho, c_0 \) 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 \( \rho, c_0 \) that you used.

f. Next, choose appropriate values for \( \rho, c_0 \) and an FIR or EMA smoother, such that the dynamics processor would act as an expander that attenuates the \( f_1 \) 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 \( \rho, c_0 \) 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]),

\[ [x, f] = \text{audioread}('speech.wav'); \quad \text{\% speech signal} \]
\[ [x, f] = \text{audioread}('music.wav'); \quad \text{\% music signal} \]

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).

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 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).

\[
\begin{align*}
    x(t) &= x_s(t) + x_m(t), \quad 'speech+music.wav' \\
    y_m(t) &= G(t)x_m(t), \quad 'ducked music.wav' \\
    y(t) &= x_s(t) + y_m(t), \quad 'speech+ducked.wav'
\end{align*}
\]

16.3.5 Example Graphs

16.4 Problems

16.1 It is desired to generate the periodic sequence \( h = [0, 1, 2, 0, 1, 2, 0, 1, 2, \ldots] \) of period three. Determine the filter \( H(z) \) whose impulse response is \( h \).
16.4 PROBLEMS

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.

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 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

\[ H(z) = \frac{1 + 2z^{-1} + 3z^{-2} - 4z^{-3} - 5z^{-4}}{1 - z^{-5}} \]

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:

\[ b = [b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7] \]
16.4. PROBLEMS

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:

\[ y(n) = x(n) + x(n - 8), \quad y(n) = x(n) + x(n - 8) + x(n - 16) \]
\[ y(n) = x(n) - x(n - 8), \quad y(n) = x(n) - x(n - 8) + x(n - 16) \]

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:

\[ y(n) = \frac{1}{3} \left[ x(n) + a_1(n)x(n - d_1(n)) + a_2(n)x(n - d_2(n)) \right] \]

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.
Repeat the flanging experiment using the recursive flanging processor:

\[ y(n) = ay(n - d(n)) + x(n) \]

where \( a = 0.8 \). State the processing algorithm in this case, using a circular buffer for the feedback delay line and the routine \text{tap1} to interpolate between buffer entries.

16.11 Computer Experiment: Reverberation Examples. Using the circular-buffer reverberator routines \text{plain}, \text{allpass}, \text{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 \( 0 \leq n \leq 500 \) 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 \( p_i, A_i, i = 1, 2, \ldots, D + 1 \) be the poles and residues of the \( H(z) \), that is,

\[ H(z) = \frac{1}{1 - z^{-D}G(z)} = \sum_{i=1}^{D+1} \frac{A_i}{1 - p_i z^{-1}} \]

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 \( \omega \) and duration \( L \) is applied to the input:

\[ x(n) = e^{j\omega n} (u(n) - u(n - L)) \]

Show that the output signal will be given by:
16. PROBLEMS

16. DIGITAL AUDIO EFFECTS

Fig. 16.3.18 Control signal for ducked music, and gains.

\[ y(n) = H(\omega) e^{j\omega n} (u(n) - u(n-L)) + \sum_{i=1}^{D+1} B_i p_i^n (u(n) - e^{j\omega L} p_i^n u(n-L)) \]

where \( B_i = p_i A_i / (p_i - e^{j\omega}), i = 1, 2, \ldots, D + 1 \). 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:

\[ x(n) = \cos(\omega n) (u(n) - u(n-150)) \]

with \( \omega = 0.2\pi \) and \( \omega = \pi \). Generate similar graphs also for the following frequencies:

\[ \omega = 0.4\pi, 0.6\pi, 0.8\pi, \text{ and } 0.99\pi. \]

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 \( 0 \leq n \leq 299 \).

16.16 Computer Experiment: Karplus-Strong 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, \( w[i] = \text{ran}(\text{seed}) - 0.5 \), for \( i = 0, 1, \ldots, D \). Then, run the sample processing algorithm (16.2.34) with zero input \( x(n) = 0 \), for \( 0 \leq n \leq 499 \). 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 \( f_i = f_i/D \).

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:

\[ H(z) = \frac{z^{-D}}{1 - az^{-D}}, \quad \text{and} \quad H(z) = \frac{z^{-D}}{1 - az^{-D}G(z)} \]

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 \texttt{plaindel1.c} and \texttt{lpdel1.c}. How do they differ from the routines \texttt{plain} and \texttt{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 \texttt{plaindel1} and \texttt{lpdel1} that you wrote in the previous problem.

Note, that you will need to use two circular buffers \( \{w_1, w_2\} \) 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 \( 0 \leq n \leq 2000 \), for the two inputs:

\[ x(n) = \delta(n), \quad x(n) = u(n) - u(n-100) \]

16.20 Computer Experiment: Multi-Tap Delay Effects. In the electronic music community, a multitap 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 \( x(n) \), 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 \( w \) 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 \( 0 \leq n \leq 1000 \), for the two inputs:

\[ x(n) = \delta(n), \quad x(n) = u(n) - u(n-200) \]

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 \( z^{D_1+D_2} = a_1 z^{D_1} + a_2 \).]

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:

\[ Y_L(z) = H_{LL}(z)X_L(z) + H_{LR}(z)X_R(z) \]

\[ Y_R(z) = H_{RL}(z)X_L(z) + H_{RR}(z)X_R(z) \]
16.4. PROBLEMS

847

16. DIGITAL AUDIO EFFECTS

848

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?

16.23 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 \( n = 0, 1, \ldots, 299 \), for the case when there is only a left input pulse of duration 5, that is,

\[
x_L(n) = u(n) - u(n - 5), \quad x_R(n) = 0
\]

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 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 round-off-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:

$$|H(\Omega)|^2 = \frac{G^2 + G_0^2 \varepsilon^2 F_N^2(w)}{1 + \varepsilon^2 F_N^2(w)}$$

where

$$w = \frac{\Omega}{\Omega_B}$$

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 $\Omega_B$ is an effective bandwidth frequency that corresponds to a desired bandwidth level $G_B$ (such as the 3-dB level). The parameter $\varepsilon$ is calculated from the condition that $F_N(w) = 1$ at $\Omega = \Omega_B$:

$$\frac{G^2 + G_0^2 \varepsilon^2}{1 + \varepsilon^2} = G_B^2 \Rightarrow \varepsilon = \frac{G^2 - G_B^2}{\sqrt{G_B^2 - G_0^2}}$$

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 $\Delta f$ 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 $\omega = 2\pi f/s$ and $f$ is the physical frequency in Hz. The requirement that the bandedge frequencies $\omega_1, \omega_2$ 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 \( \omega_0 \) and \( \Omega_B \) in terms of \( \omega_1 \) and \( \omega_2 \) :

\[
\Omega_B = \tan \left( \frac{\Delta\omega}{2} \right), \quad \tan^2 \left( \frac{\omega_0}{2} \right) = \tan \left( \frac{\omega_1}{2} \right) \tan \left( \frac{\omega_2}{2} \right)
\]

(17.2.5)

where \( \Delta\omega = \omega_2 - \omega_1 \). Equivalently, we have:

\[
\cos \omega_0 = \frac{\sin (\omega_1 + \omega_2)}{\sin \omega_1 + \sin \omega_2}
\]

(17.2.6)

Conversely, Eqs. (17.2.4) may be solved for \( \omega_1 \) and \( \omega_2 \) in terms of \( \omega_0 \) and \( \Delta\omega \) :

\[
\omega_1 = \frac{c_0 + j\sqrt{\Omega_B^2 + s_0^2}}{1 + j\Omega_B}, \quad \omega_2 = \frac{c_0 + j\sqrt{\Omega_B^2 + s_0^2}}{1 - j\Omega_B}
\]

(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:

\[
\cos \omega_1 = \frac{c_0 + \Omega_B \sqrt{\Omega_B^2 + s_0^2}}{\Omega_B^2 + 1}, \quad \cos \omega_2 = \frac{c_0 - \Omega_B \sqrt{\Omega_B^2 + s_0^2}}{\Omega_B^2 + 1}
\]

(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 \( G_0 \)) of the digital lowpass and highpass shelving equalizers:

\[
\omega_0 = 0, \quad \omega_1 = 0, \quad \omega_2 = \Delta\omega, \quad \text{(LP shelf)}
\]

(17.2.9)

\[
\omega_0 = \pi, \quad \omega_1 = \pi - \Delta\omega, \quad \omega_2 = \pi, \quad \text{(HP shelf)}
\]

(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:

\[
|H_a(\Omega)|^2 = \frac{G_{\text{low}}^2 + G_B^2 F_N^2(w)}{1 + \varepsilon^2 F_N^2(w)}
\]

(17.2.10)

where \( N \) is the analog filter order, \( \varepsilon \) is a constant, and \( F_N(w) \) is a function of the normalized frequency \( w = \Omega/\Omega_B \) given by:

\[
F_N(w) = \begin{cases} 
  w^N, & \text{Butterworth} \\
  C_N(w), & \text{Chebyshev, type-1} \\
  \frac{1}{C_N(w^{-1})}, & \text{Chebyshev, type-2} \\
  \cd(NuK_1,k), & \text{Elliptic}
\end{cases}
\]

(17.2.11)

where \( C_N(x) \) is the order-\( N \) Chebyshev polynomial, that is, \( C_N(x) = \cos(N \cos^{-1} x) \), and \( \cd(x,k) \) 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 \( \varepsilon \) 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 \( \varepsilon \) may be determined. Setting \( \Omega = \Omega_B \) in Eq. (17.2.10), we obtain:

\[
|H_a(\Omega_B)|^2 = \frac{G_{\text{low}}^2 + G_B^2 e^2}{1 + e^2} = G_b^2 \Leftrightarrow \varepsilon = \frac{G_B^2 - G_b^2}{\sqrt{G_b^2 - G_B^2}}
\]

(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:

\[
H_a(s) = H_0 \left[ \prod_{i=1}^{L} \frac{1 - s/z_i}{1 - s/p_i} \right] \left[ \prod_{i=1}^{L} \frac{1 - s/z_i^*}{1 - s/p_i^*} \right]
\]

(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:

\[
H_0 = \begin{cases} 
  G, & \text{Butterworth and Chebyshev-2} \\
  G^r G_B^{-r}, & \text{Chebyshev-1 and Elliptic}
\end{cases}
\]

(17.2.14)

The zeros \( z_0, z_1 \) and poles \( p_0, p_1 \) 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:

\[
H_a(s) = H_0 \left[ \prod_{i=1}^{L} \frac{z_1 - s}{p_0 - s} \right] \left[ \prod_{i=1}^{L} \frac{(z_1 - s)(z_1^* - s)}{(p_1 - s)(p_1^* - s)} \right]
\]

(17.2.15)
where \( H_\infty \) is the gain at \( \Omega = \infty \) (and at \( \omega = 0 \) and \( \omega = \pi \)) given by:

\[
H_\infty = \begin{cases} G_0, & \text{Butterworth and Chebyshev-1} \\ G_0^2 G_1^2, & \text{Chebyshev-2} \\ G_0^2 G_1^2, & \text{Elliptic} \end{cases}
\]

(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 \([321,322]\):

\[
D_0 = A_{00} + A_{01} \\
D_1 = A_{10} + A_{11} + A_{12}
\]

(17.2.19)

For the \( z^{-1} \) to \( z^{-1} \) transformation, we obtain the second- and fourth-order coefficients:

\[
\begin{align*}
& b_{00} = b_{00} \\
& b_{01} = c_0 (b_{01} - b_{00}) \\
& b_{11} = c_0 (b_{11} - 2b_{01}) \\
& b_{12} = (b_{01} - b_{11} + b_{12}) c_0^2 - b_{11} \\
& a_{01} = c_0 (a_{01} - 1) \\
& a_{02} = -a_{01} \\
& a_{11} = c_0 (a_{11} - 2) \\
& a_{12} = (1 - a_{11} + a_{12}) c_0^2 - a_{11} \\
& a_{13} = c_0 (a_{13} - 2a_{12}) \\
& a_{14} = a_{12}
\end{align*}
\]

(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 finding the roots of the numerator and denominator of Eq. (17.2.10), that is, solving:

for the zeros: \( G^2 + G_0^2 \epsilon^2 F_N(w) = 0 \)

(17.3.1)

or, equivalently,

\[
F_N(w) = \pm j \frac{G}{G_0 \epsilon}
\]

(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:

\[
z_0 = -\frac{\beta}{\beta_0}, \quad z_i = \frac{\beta}{\beta_0} (-s_i + j c_i), \quad p_0 = -\beta, \quad p_i = \beta (-s_i + j c_i)
\]

(17.3.3)
for \( l = 1, 2, \ldots, L \), where we introduced the parameters:

\[
g = G^{1/N}, \quad g_0 = G_0^{1/N}, \quad \beta = \varepsilon^{-1/N} \Omega_B = \varepsilon^{-1/N} \tan \left( \frac{\Delta u}{2} \right) \tag{17.3.4}
\]

\[
s_i = \sin \phi_i, \quad c_i = \cos \phi_i, \quad \phi_i = \frac{(2i - 1)\pi}{2N}, \quad i = 1, 2, \ldots, L \tag{17.3.5}
\]

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 \textit{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:

\[
z_0 = -\Omega_B \sin u, \quad z_i = j\Omega_B \cos(\phi_i - ju) = \Omega_B(-s_i \sin u + j c_i \cosh u)
\]

\[
p_0 = -\Omega_B \sin v, \quad p_i = j\Omega_B \cos(\phi_i - jv) = \Omega_B(-s_i \sin v + j c_i \cosh v)
\]

where \( i = 1, 2, \ldots, L \), and \( s_i, c_i, \phi_i \) are the same as in Eq. (17.3.5). The quantities \( u, v \) are given by:

\[
e^u = g_0^{-1} \beta, \quad \beta = \left( G_0^x + G_B \sqrt{1 + \varepsilon^2} \right)^{1/N}
\]

\[
e^v = \alpha = \left( \varepsilon^1 + \sqrt{1 + \varepsilon^2} \right)^{1/N}
\]

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, \( H_0 = G_0 = g_0^{1/N} \), over the \( N \) first-order sections.

For the \textit{type-2 Chebyshev case}, we have, \( F_N(w) = C_1(1/w) \), which leads to zeros and poles that are essentially the inverses of those of Eq. (17.3.6). For \( i = 1, 2, \ldots, L \), we have:

\[
z_0^* = -\Omega_B^{-1} \sin u, \quad z_i^* = j\Omega_B^{-1} \cos(\phi_i - ju) = \Omega_B^{-1}(-s_i \sin u + j c_i \cosh u)
\]

\[
p_0^* = -\Omega_B^{-1} \sin v, \quad p_i^* = j\Omega_B^{-1} \cos(\phi_i - jv) = \Omega_B^{-1}(-s_i \sin v + j c_i \cosh v)
\]

where \( s_i, c_i, \phi_i \) are the same as in Eq. (17.3.5), and the quantities \( u, v \) are defined by:

\[
e^u = g^{-1} \beta, \quad \beta = \left( G_0 \varepsilon + G_B \sqrt{1 + \varepsilon^2} \right)^{1/N}
\]

\[
e^v = \alpha = \left( \varepsilon + \sqrt{1 + \varepsilon^2} \right)^{1/N}
\]

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 \textit{elliptic case}, we have, \( F_N(w) = \text{cd}(uK_1, k) \), with, \( w = \text{cd}(uK, k) \). 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 \( i = 1, 2, \ldots, L \):

\[
z_i = j\Omega_B \text{cd}(u_i - ju_0)K_1, k), \quad p_i = j\Omega_B \text{cd}(u_i - jv_0)K, k)
\]

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:

\[
\text{sn}(ju_0NK_1, k_1) = \frac{jG}{G_0\varepsilon}, \quad \text{sn}(jv_0NK_1, k_1) = \frac{1}{\varepsilon}
\]

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 \( i = L + 1 \)):

\[
z_0 = j\Omega_B \cd((1 - ju_0)K, k) = j\Omega_B \text{sn}(ju_0K, k)
\]

\[
p_0 = j\Omega_B \cd((1 - jv_0)K, k) = j\Omega_B \text{sn}(jv_0K, k)
\]

where we used the identity \( 318 \): \( \text{cd}(K - x, k) = \text{sn}(x, k) \). The evaluation of the elliptic functions \( \text{cd} \) and \( \text{sn} \) 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,

\[
w_i = \frac{z_i}{\Omega_B} = \text{cd}(u_i - ju_0)K, k)
\]

we may verify the root condition (17.3.2):

\[
F_N(w_i) = \text{cd}((u_i - ju_0)NK_1, k_1) = \text{cd}((2i - 1)K - ju_0NK_1, k_1)
\]

\[
= (-1)^i \text{sn}(ju_0NK_1, k_1) = \pm j \frac{G}{G_0\varepsilon}
\]

where we used the property \( 318 \),

\[
\text{cd}((2i - 1)K + x, k_1) = (-1)^i \text{sn}(x, k_1)
\]

which is valid for integer \( i \). Similarly, for odd \( N \) we have

\[
\text{cd}((1 - ju_0)NK_1, k_1) = \text{sn}(ju_0NK_1, k_1) = \frac{jG}{G_0\varepsilon}
\]

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 \( G = 0 \), 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:

\[
z_i = j\Omega_B (k\zeta_i)^{-1}, \quad \zeta_i = \text{cd}(u_iK, k)
\]

The same conclusion can also be drawn by noting that when \( G = 0 \) the solution of Eq. (17.3.11) is, \( ju_0NK_1 = jK_1 \), that is, it corresponds to a pole of the \( \text{sn}(x, k_1) \) function. But because of the degree equation, we also have, \( ju_0K = jK_1 \), which is a pole of the \( \text{sn}(x, k) \) function. Therefore, \( z_0 = \infty \) and the zero factor, \( (1 - s/z_0) \), of \( H_2(s) \) 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,

\[
\text{cd}(x - jK', k) = \frac{1}{k} \text{cd}(x, k)
\]
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 \( G \rightarrow 0 \).

Using the Taylor series expansion \( \sin(x,k) = x \), 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:

\[
 ju_0 N K_1 = 0 \quad \Rightarrow \quad u_0 = \frac{G}{G_0 e^N K_1}, \quad z_0 = j\Omega_B \sin(ju_0 K, k) = -\Omega_B u_0 K = -\Omega_B G K / G_0 e^N K_1
\]

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 \( G \rightarrow 0 \):

\[
 H_0 (1 - s/z_0) = G + \frac{G_0 e^N K_1}{\Omega_B} - \frac{s}{\Omega_B} G_0 e^N K_1 / K
\]

Thus, in the odd-\( N \) case, the first-order numerator factor of Eq. (17.2.13) takes the following forms:

\[
 H_0 (1 - s/z_0) = \begin{cases} G (1 - s/z_0), & \text{if } G_0 \neq 0, G \neq 0 \\ G, & \text{if } G_0 = 0, G \neq 0 \\ (s/\Omega_B) (G_0 e^N K_1 / K), & \text{if } G_0 \neq 0, G = 0 \end{cases}
\]

For the even-\( N \) case, we have \( H_0 = G_B \), per Eq. (17.2.14). Similarly, the conjugate zeros \( z_i, i = 1, 2, \ldots, L \), are given as follows, for both even and odd \( N \):

\[
 z_i = \begin{cases} j\Omega_B \cos((ui - ju_0) K, k), & \text{if } G_0 \neq 0, G \neq 0 \\ j\Omega_B (K_0) i, & \text{if } G_0 = 0, G \neq 0 \\ j\Omega_B \xi_i, & \text{if } G_0 \neq 0, G = 0 \end{cases}
\]

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:

\[
 H_a(s) = \left[ \frac{gB + g_0 s}{\beta + s} \right]^L \prod_{i=1}^{L} \left[ \frac{g^2 \beta^2 + 2g_0 g_i \beta s + g_i^2 s^2}{\beta^2 + 2g_0 g_i \beta s + s^2} \right]
\]

(17.4.1)

where we defined the parameters:

\[
g = G^{1/N}, \quad g_0 = G_0^{1/N}, \quad \beta = \varepsilon^{-1/N} \Omega_B = \varepsilon^{-1/N} \tan \left( \frac{\Delta \omega}{2} \right)
\]

(17.4.2)

The parameter \( \varepsilon \) is given by Eq. (17.2.12) and \( \Omega_B \) by Eq. (17.2.5). Using the coefficient transformations given above, we find the coefficients of the digital lowpass shelving filter (17.2.18):

\[
 D_0 = \beta + 1 \quad D_i = \beta^2 + 2s_i \beta + 1
\]

\[
 b_0 = (gB + g_0) / D_0 \quad b_0 = (g^2 \beta^2 + 2g_0 g_i \beta s + g_i^2) / D_i
\]

(17.4.4)

\[
 a_0 = (\beta - 1) / D_0 \quad a_1 = (\beta^2 - 1) / D_i
\]

(17.4.5)

The coefficients of the second and fourth-order sections of the bandpass equalizer (17.2.18) are:

\[
 D_0 = \beta + 1 \quad D_i = \beta^2 + 2s_i \beta + 1
\]

\[
 b_0 = (g_0 + g_B) / D_0 \quad b_0 = (g^2 \beta^2 + 2g_0 g_i \beta s + g_i^2) / D_i
\]

\[
 a_0 = -(gB + g_0) / D_0 \quad a_i = -4c_i (g_0 + g_0 g_i / \beta s / D_i)
\]

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 biquadric 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:

\[
H_a(s) = \left[ \frac{b\Omega_B + g_0 s}{a\Omega_B + s} \right]^T \prod_{i=1}^{L} \left[ \frac{(b^2 + g_0^2 c_i^2) \Omega_B^2 + 2g bs_i\Omega Bs + g_0^2 s^2}{(a^2 + c_i^2)\Omega_B^2 + 2as_i\Omega Bs + s^2} \right] \tag{17.4.6}
\]

where we defined \(g_0 = G_0^{1/N}\) and:

\[
b = g_0 \sinh u = \frac{1}{2} (\beta - g_0^2 \beta^{-1}), \quad a = \sinh v = \frac{1}{2} (\alpha - \alpha^{-1}) \tag{17.4.7}
\]

\[
e^u = g_0^{-1} \beta, \quad \beta = (G\varepsilon^{-1} + G_B\sqrt{1 + \varepsilon^2})^{1/N}, \quad e^v = \alpha = (\varepsilon^{-1} + \sqrt{1 + \varepsilon^2})^{1/N} \tag{17.4.8}
\]

\[
s_i = \sin \phi_i, \quad c_i = \cos \phi_i, \quad \phi_i = \frac{(2i-1)\pi}{2N}, \quad i = 1, 2, \ldots, L \tag{17.4.9}
\]

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:

\[
\begin{align*}
D_0 &= a\Omega_B + 1 \\
\hat{b}_{00} &= (b\Omega_B + g_0)/D_0 \\
\hat{b}_{01} &= (b^2 + g_0^2 c_1^2)\Omega_B^2 + 2g bs_1\Omega Bs + g_0^2 s^2)/D_1 \\
\hat{a}_{01} &= (a\Omega_B - 1)/D_0 \\
\end{align*}
\tag{17.4.10}
\]

and using Eq. (17.2.20), we obtain the bandpass equalizer coefficients:

\[
\begin{align*}
D_0 &= a\Omega_B + 1 \\
\hat{b}_{00} &= (g_0 + b\Omega_B)/D_0 \\
\hat{b}_{01} &= (b^2 + g_0^2 c_1^2)\Omega_B^2 + 2g bs_1\Omega Bs + g_0^2 s^2)/D_1 \\
\hat{a}_{01} &= -2a\Omega_B/D_0 \\
\hat{a}_{02} &= (1 - a\Omega_B)/D_0 \\
\end{align*}
\tag{17.4.11}
\]

**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:

\[
H_a(s) = \left[ \frac{g\Omega_B + bs}{\Omega_B + as} \right]^T \prod_{i=1}^{L} \left[ \frac{g^2\Omega_B^2 + 2gs_1\Omega Bs + (b^2 + g^2 c_i^2)s^2}{\Omega_B^2 + 2as_1\Omega Bs + (a^2 + c_i^2)s^2} \right] \tag{17.4.12}
\]

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 \(s_i, c_i, \phi_i\) given by Eq. (17.4.9), and the quantities \(u, v\) defined by:

\[
e^u = g^{-1} \beta, \quad \beta = (G\varepsilon^{-1} + G_B\sqrt{1 + \varepsilon^2})^{1/N}, \quad e^v = \alpha = (\varepsilon^{-1} + \sqrt{1 + \varepsilon^2})^{1/N} \tag{17.4.14}
\]

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:

\[
\begin{align*}
D_0 &= \Omega_B + a \\
\hat{b}_{00} &= (g\Omega_B + b)/D_0 \\
\hat{b}_{01} &= (g^2\Omega_B^2 + 2gbs\Omega_B + b^2 + g^2 c_1^2)/D_1 \\
\hat{a}_{01} &= \Omega_B - a/D_0 \\
\hat{a}_{12} &= (\Omega_B^2 - a^2 - c_1^2)/D_1 \\
\end{align*}
\tag{17.4.15}
\]

and the coefficients of the bandpass equalizer:

\[
\begin{align*}
D_0 &= \Omega_B + a \\
\hat{b}_{00} &= (g\Omega_B + b)/D_0 \\
\hat{b}_{01} &= (g^2\Omega_B^2 + 2gbs\Omega_B + b^2 + g^2 c_1^2)/D_1 \\
\hat{b}_{02} &= -2bc_0/D_0 \\
\hat{b}_{03} &= -4c_0(b^2 + g^2 + g^2 c_0^2)/D_1 \\
\hat{a}_{03} &= (a - \Omega_B)/D_0 \\
\hat{a}_{02} &= (a - \Omega_B)/D_0 \\
\end{align*}
\tag{17.4.16}
\]

**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 \( \text{cd} \) (instead of usual \( \text{sn} \)) 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:

\[
\frac{N}{K} = \frac{K_1}{K_1'} \tag{17.4.17}
\]

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 \( F_N(w) = \text{cd}(NuK_1,k_1) \) is a rational function of \( w = \text{cd}(uK,k) \) given as follows (and normalized such that \( F_N(1) = 1 \)):

\[
F_N(w) = \left[ w^r \right] L \prod_{i=1}^L \left( \frac{w^2 - \zeta_i^2}{1 - w^2 k^2 \zeta_i^2} \right) \left( \frac{1 - k^2 \zeta_i^2}{1 - \zeta_i^2} \right) \tag{17.4.18}
\]

where \( N = 2L + r \), and \( \zeta_i \) and \( k \zeta_i \) \(-1\) are the zeros and poles of \( F_N(w) \), where:

\[
\zeta_i = \text{cd}(uK,k), \quad u_i = \frac{2i - 1}{N}, \quad i = 1, 2, \ldots, L \tag{17.4.19}
\]

Because the elliptic designs are equiripple in both the passband and stopband, the specifications of the equalizer must be modified by adding a gain \( G_s \), which 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 \( G_s \) 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 \( w_1 = \text{cd}(uK,k) = 1/k \) is satisfied with \( u = jK'/K \), that is, \( \text{cd}(uK,k) = \text{cd}(jK',k) = 1/k \), which is a standard property of the cd elliptic function [313,317]. Then, the same property and the degree equation (17.4.17) imply that:

\[
F_N(w_1) = \text{cd}(NuK_1,k_1) = \text{cd}(jNK'/K,k_1) = 1/k_1 \tag{17.4.20}
\]

Using Eq. (17.4.20), the requirement that the gain be equal to \( G_s \) at \( w = w_1 = 1/k \) gives the condition:

\[
|H_a(\Omega_s)|^2 = \frac{G_s^2 + G_0^2 \varepsilon_s^2/k_1^2}{1 + \varepsilon_s^2/k_1^2} = G_s^2 \quad \Leftrightarrow \quad k_1 = \frac{\varepsilon_s}{G_s}, \quad \varepsilon_s = \sqrt{\frac{G_s^2 - G_0^2}{G_s^2}} \tag{17.4.21}
\]

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 \( N, G_s, \Omega_s \), 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 \( \Omega_s \).

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 \( k_1 \):

\[
k_1 = k^N \prod_{i=1}^L \text{sn}^4(u_i K, k) \tag{17.4.23}
\]

where we used the property: \((1 - \zeta_i^2)/(1 - k^2 \zeta_i^2) = \text{sn}^2(u_i K, k) \). Noting the invariance [312] of the degree equation under the substitutions \( k \rightarrow k' \) and \( k_1 \rightarrow k_1' \), we also obtain the exact solution for \( k \) in terms of \( N, k_1 \), expressed via the complementary moduli \( k', k_1' \):

\[
k' = (k_1')^N \prod_{i=1}^L \text{sn}^4(u_i K_1', k_1') \tag{17.4.24}
\]

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.](image-url)
17. ORDER DETERMINATION

modulus \( k \) can be determined from the series expansion [317]:

\[
k = 4\sqrt{\frac{1}{2} \left( \sum_{m=0}^{\infty} q^m (m+1) \right)^2}
\]

(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 \( H_d(s) \) is constructed by Eq. (17.2.13), where the poles \( p_0, p_1 \) are given by Eqs. (17.3.10)–(17.3.12) and the zeros \( z_0, z_1 \) 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 \( H_d(s) \) 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, \( \Delta \omega_s = 2\pi \Delta f_s/\Omega_s \), at the stopband level \( G_s \) can be derived by inverting the relationship \( \Omega_s = \tan^{-1}(\Delta \omega_s/2) \). The left and right stopband edge frequencies can be calculated from Eq. (17.2.8) with \( \Omega_s \) replacing \( \Omega_B \):

\[
\cos \omega_{x1} = \frac{c_0 + \Omega_s \sqrt{c_0^2 + s_0^2}}{c_0^2 + 1}, \quad \cos \omega_{x2} = \frac{c_0 - \Omega_s \sqrt{c_0^2 + s_0^2}}{c_0^2 + 1}
\]

(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, \( w = \text{cd}(uK, k) = \cos(u\pi/2) \), and the function \( F_N(w) = \text{cd}(NuK_1, k_1) = \cos(Nu\pi/2) \) becomes equal to the \( N \)th order Chebyshev polynomial \( C_N \), and the \( C_N \) the \( \text{cd}(uK, k) = \cos(u\pi/2) \) becomes 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 \( \Delta f_s \) at \( G_s \). Conversely, if \( \Delta f, \Delta f_s \) 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:

\[
N = K_1'/K
\]

(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 \( \varepsilon \) decreases or \( \varepsilon_s \) 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 \( \Omega_s \) will get smaller, resulting in a narrower bandwidth \( \Delta f_s \).

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 \( \Delta f_s \) 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:

\[
k_1 = k^N \Rightarrow N = \frac{\ln k_1}{\ln k}
\]

(17.5.2)

For the type-1 Chebyshev case, we have \( C_N(1/k) = 1/k_1 \), or,

\[
\cosh(N \cosh^{-1}(1/k)) = 1/k_1 \Rightarrow N = \frac{\cosh^{-1}(1/k_1)}{\cosh^{-1}(1/k)}
\]

(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 \( \Delta f_s \) than \( \Delta f \). This implies that \( k = \Omega_B/\Omega_s > 1 \) and also \( k_1 = \varepsilon_s/\varepsilon > 1 \). The degree equation in this case is \( C_N(k) = k_1 \), or,

\[
\cosh(N \cosh^{-1} k) = k_1 \Rightarrow N = \frac{\cosh^{-1} k_1}{\cosh^{-1} k}
\]

(17.5.4)

The inequality \( \varepsilon_s < \varepsilon \) for the type-2 case can be seen from the identity:

\[
\varepsilon_s^2 - \varepsilon^2 = (G_B^2 - G_s^2)(G_s^2 - G_0^2) = (G_s^2 - G_0^2)(G_B^2 - G_0^2)
\]

(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):

\[
G > G_B > G_s > G_0 \quad \text{(boost)}
\]

(17.6.1)

\[
G < G_B < G_s < G_0 \quad \text{(cut)}
\]

(17.6.2)

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, \( G_B^2 = G_0^2/2 \), 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]:

\[
G_B^2 = G_0^2, \quad G_B^2 = \frac{1}{2}(G_0^2 + G_s^2)
\]

(17.6.2)

The corresponding values of \( \varepsilon \) defined by Eq. (17.2.12) are in these cases:

\[
\varepsilon = \sqrt{\frac{G}{G_0}}, \quad \varepsilon = 1
\]

(17.6.3)
A more general definition is the weighted arithmetic mean:

\[
G^2_B = \frac{G^2 + \alpha^2 G^2_0}{1 + \alpha^2} \Rightarrow \varepsilon = \alpha
\] (17.6.4)

where \(\alpha\) 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 \(\omega_0\), bandwidth \(\Delta\omega\), 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:

\[
G > G_B > G > G_0 \quad \text{(boost)}
\]

\[
G^{-1} < G_B^{-1} < G^{-1} < G_0^{-1} \quad \text{(cut)}
\] (17.6.5)

From the definition (17.2.12), it follows that:

\[
\varepsilon_{\text{cut}} = \sqrt{\frac{G^2 - G_B^2}{G^2 - G_0^2}} = \frac{G_0}{G} \sqrt{\frac{G^2 - G_B^2}{G^2 - G_0^2}} = \frac{G_0}{G} \varepsilon_{\text{boost}}
\] (17.6.6)

This implies that the root conditions (17.3.2) for the zeros and poles will exchange roles, and therefore, \(z_{\text{cut}} = p_{\text{cut}}\) and \(p_{\text{cut}} = z_{\text{cut}}\). The corresponding analog, and hence, the digital transfer functions will become inverses of each other:

\[
H_{\text{cut}}(z) = \frac{1}{H_{\text{boost}}(z)}
\] (17.6.7)

The elliptic modulus \(k_1\) remains invariant under this change because the quantity \(\varepsilon\) also changes in the same way as in Eq. (17.6.6), and therefore, the ratio \(\varepsilon_{\text{cut}}/\varepsilon_{\text{cut}}\) 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 \(\Omega_2 = \Omega_0\), that is, through Eq. (17.2.5), we may set \(\Omega_2 = 2\omega_2/\omega_0\) and \(\Omega_1 = 2^{-\frac{b}{2}}\omega_0\), where \(B\) plays the role of an equivalent octave bandwidth. Using some trigonometric identities, we obtain the following expression for \(\Delta\omega\) in terms of \(B\):

\[
\Omega_B = \tan \left(\frac{\Delta\omega}{2}\right) = \sin \omega_0 \sinh \left(\frac{\ln 2}{2} B\right)
\] (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 \(\omega_1\), we obtain the following “bandwidth equation” relating \(B, b,\) and \(\Omega_0 = \tan(\omega_0/2)\) [331]:

\[
2^b = \frac{\arctan(2^{b/2}\Omega_0)}{\arctan(2^{-b/2}\Omega_0)}
\] (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]:

\[
B = \frac{\omega_0}{\sin \omega_0} b
\] (17.6.10)

This approximation works very well for low frequencies \(\omega_0\), as well as for high \(\omega_0\) and narrow \(b\). For any values of \(\omega_0\) 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 \(2^b = \Omega_0/\tan(2^{-b} \arctan(2^{b/2}\Omega_0))\), we obtain the following convergent iteration, initialized at \(B_0 = B\) given by (17.6.10):

\[
2^{b_{n+1/2}} = \frac{\Omega_0}{\tan(2^{-b} \arctan(2^{b_n/2}\Omega_0))}, \quad n = 0, 1, 2, \ldots
\] (17.6.11)

At large values of \(\omega_0\) 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 \(\omega_0\), but then the approximation (17.6.10) is good and there is no need for the iteration. The calculated physical bandwidth at the \(n\)th 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 \(\Delta B_n = B_n - B\):

\[
\Delta B_n = \text{const} \cdot (-a)^n, \quad a = \left(2^b + \Omega_0^2 \right) \arctan(2^{-b/2}\Omega_0) / (1 + 2^b \Omega_0^2) \arctan(2^{-b/2}\Omega_0)
\] (17.6.12)

where the quantity \(a\) can be shown to be less than unity for all values of \(B\) and \(\Omega_0\), and a decreasing function of \(B\) (\(a \approx 1\) at low \(\omega_0\), which explains the slow convergence in that case.) Thus, \(\Delta B_n\) decreases exponentially, and so does the error \(|b_n - b|\), since it follows \(|\Delta B_n|\).

Once \(B\) has been calculated from \(b\) and \(\omega_0\), it may be used in (17.6.8) to obtain the linear bandwidth \(\Delta\omega\) and, from it, the actual bandedge frequencies \(\omega_1, \omega_2\) through Eqs. (17.2.8), or from \(\omega_2, \omega_1\) = 2 \arctan(2^{b/2}\Omega_0). The calculated bandedge frequencies will always lie within the Nyquist interval, for all values of \(\omega_0\) and \(B\). However, it must be emphasized that, although \(\omega_1, \omega_2\) are b-octaves apart, they will not necessarily lie symmetrically at \(\pm b/2\) octaves about \(\omega_0\), and may result in a very asymmetric band, especially at large \(\omega_0\).

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 \(G_0\) and \(G_1\). 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 \(\Delta\omega_0\) at an arbitrary intermediate level \(G_B\), such that \(G_0 < G < G_B < G_B < G\). For a given order \(N\), the required bandwidth \(\Delta\omega\) and the design parameter \(\Omega_B\) can be determined from \(\Delta\omega_0\) as follows:

\[
\Omega_B = \tan \left(\frac{\Delta\omega_0}{2}\right) = \frac{1}{w_B} \tan \left(\frac{\Delta\omega_0}{2}\right)
\] (17.6.13)
where the normalized frequency \( w_b \) is the solution of the equation:

\[
F_N (w_b) = \frac{\epsilon_b}{\epsilon}, \quad \epsilon_b = \frac{G^2 - G_b^2}{G_b^2 - G_0^2}
\]  

(17.6.14)

Eq. (17.6.14) was obtained from the equivalent magnitude condition:

\[
\frac{G^2 + G_b^2 \epsilon^2 F_b^2 (w_b)}{1 + \epsilon^2 F_b^2 (w_b)} = G_b^2
\]  

(17.6.15)

The solution of (17.6.14) is straightforward. For example, for the type-1 Chebyshev case, one may solve \( \cosh (N \omega_b) = \epsilon_b / \epsilon \) for \( u \) and then calculate \( w_b = \cosh (u) \), where \( k_1 \) is fixed from the levels \( G_b \) and \( G_r \), 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 \( G_b, G_r \) are given, then, the quantities \( \epsilon_b \) 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 \( G_r \) must also be fixed independently. The following trial-and-error approach works well: for each successive filter order \( N = 1, 2, \ldots \), calculate \( k \) from \( N, k_1 \) using the degree equation, then solve \( \cd (NuK_1, k_1) = \epsilon_b / \epsilon \) for \( u \), calculate the error \( e = \cd (uK, k) - w_b \), and keep the first \( N \) for which \( e \) becomes negative (it always starts from positive values provided that \( w_b < \epsilon_b / \epsilon \), 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 \( \Delta \omega_b \), 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 \( z^{-1} \) is replaced by the lowpass to bandpass transformation of Eq. (17.2.17), that is,

\[
s = \frac{1 - z^{-1}}{1 + z^{-1}} = \frac{1 - 2c_0 z^{-1} + z^{-2}}{1 - z^{-2}} \quad \Rightarrow \quad z^{-1} = \frac{z^{-1} (c_0 - z^{-1})}{1 - c_0 z^{-1}}
\]

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 properties of the normalized lattice.

The transposed (of the direct-form II) realization of the second-order sections of Eq. (17.2.18), after each delay \( z^{-1} \) has been replaced by Fig. 17.7.1, is shown in Fig. 17.7.2. The figure represents the transfer function:

\[
\frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}, \quad z^{-1} = \frac{(c_0 - z^{-1}) z^{-1}}{1 - c_0 z^{-1}}
\]  

(17.7.1)

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]:

\[
\gamma_1 = \frac{\hat{a}_1}{1 + \hat{a}_2}, \quad \gamma_2 = \hat{a}_2, \quad \tau_1 = \sqrt{1 - \gamma_1^2}, \quad \tau_2 = \sqrt{1 - \gamma_2^2}
\]  

(17.7.2)
is shown in Fig. 17.7.4, where the indicated state vectors $s$ and $w$ are two-dimensional. The corresponding time-domain description is:

$$y(n) = Cs(n) + Dx(n)$$

$$s(n + 1) = c_0[A s(n) + B x(n)] - s_0 w(n) \quad (17.7.7)$$

$$w(n + 1) = s_0[A s(n) + B x(n)] + c_0 w(n)$$

The ladder coefficients $d_0, d_1, d_2$ are the solutions of the triangular system:

$$\begin{bmatrix}
1 & y_1 & \hat{a}_2 \\
0 & 1 & \hat{a}_1 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\hat{d}_0 T_1 \hat{d}_2 \\
\hat{d}_1 T_2 \\
\hat{d}_2
\end{bmatrix}
= \begin{bmatrix}
\hat{b}_0 \\
\hat{b}_1 \\
\hat{b}_2
\end{bmatrix} \quad (17.7.3)$$

The first-order factor of Eq. (17.2.18) is obtained by setting $\hat{a}_2 = \hat{b}_2 = 0$, or equivalently, $y_2 = 0, d_2 = 0$, and $T_2 = 1$, which amounts to deleting the $y_2$ 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:

$$y_1 = \frac{\beta - 1}{\beta + 1}, \quad d_0 = \frac{\sqrt{2}}{2} \left[ g \left( g + g_0 \right) \right], \quad d_1 = \frac{\beta g - g_0}{\beta + 1} \quad (17.7.4)$$

Eq. (17.7.4) can also be used to implement the conventional biquadratic equalizer in its lattice form. For the $i$th second-order Butterworth factor of (17.2.18), we have:

$$y_1 = \frac{\beta^2 - 1}{\beta^2 + 1}, \quad y_2 = \frac{\beta^2 - 2 s_i \beta + 1}{\beta^2 + 2 s_i \beta + 1} \quad (17.7.5)$$

$$d_0 = \frac{\sqrt{2}}{2 \beta} \left[ g \left( g + g_0 \right) \left( 1 - \beta^2 \right) \left( g - g_0 \right) + 2 s_i \beta \left( g + g_0 \right) \right]$$

$$d_1 = \frac{\sqrt{2}}{2 \beta} \left[ g \beta \left( 1 + \beta s_i \right) - g_0 \left( \beta + s_i \right) \right]$$

$$d_2 = \frac{g^2 \beta^2 - 2 g g_0 s_i \beta + g_0^2}{\beta^2 + 2 s_i \beta + 1} \quad (17.7.6)$$

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)
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 \( \omega_0 \) 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 \( \beta \) 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:
\[
\begin{bmatrix}
1 & \hat{d}_1 & 1 \\
\hat{d}_1 & 1 & 1
\end{bmatrix}
\begin{bmatrix}
\hat{d}_0 \\
\hat{d}_1
\end{bmatrix} =
\begin{bmatrix}
\hat{b}_0 \\
\hat{b}_1
\end{bmatrix} \tag{17.8.1}
\]

Eq. (17.8.1) is equivalent to expanding the corresponding first-order transfer function in the form:
\[
\frac{B(\hat{z})}{A(\hat{z})} = \frac{\hat{b}_0 + \hat{b}_1 \hat{z}^{-1}}{1 + \hat{d}_1 \hat{z}^{-1}} = d_0 + d_1 A^R(\hat{z}) \quad \frac{A(\hat{z})}{A(\hat{z})} \tag{17.8.2}
\]

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:
\[
y_1 = \frac{\beta - 1}{\beta + 1}, \quad d_0 = \frac{g + g_0}{2}, \quad d_1 = \frac{g - g_0}{2} \tag{17.8.3}
\]

Similarly, for the type-1 Chebyshev case, we have:
\[
y_1 = \frac{a \Omega_B - 1}{a \Omega_B + 1}, \quad d_0 = \frac{1}{2} \left( \frac{b}{a} + g_0 \right), \quad d_1 = \frac{1}{2} \left( \frac{b}{a} - g_0 \right) \tag{17.8.4}
\]

Thus, the coefficients \( d_0, d_1 \) depend only on the gain, and the coefficients \( y_1, \tau_1 \), 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:
\[
\begin{bmatrix}
1 & \hat{d}_2 & 1 \\
\hat{d}_1 & \hat{d}_1 & 2 \\
\hat{d}_1 & 1 & 1
\end{bmatrix}
\begin{bmatrix}
\hat{d}_0 \\
\hat{d}_1 \\
\hat{d}_0 \tau_1 \tau_2
\end{bmatrix} =
\begin{bmatrix}
\hat{b}_0 \\
\hat{b}_1 \\
\hat{b}_2
\end{bmatrix} \tag{17.8.5}
\]

This is equivalent to expanding the second-order transfer function (17.7.1) in the form:
\[
\frac{B(\hat{z})}{A(\hat{z})} = d_0 + d_1 A^R(\hat{z}) + d_2 \tau_1 \tau_2 (1 + \hat{z}^{-1})^2 \quad A(\hat{z}) \tag{17.8.6}
\]

For the \( i \)th second-order Butterworth factors of Eq. (17.4.4), we have:
\[
d_0 = \frac{1}{2} g_0 (g_0 + g), \quad d_1 = \frac{1}{2} g_0 (g_0 - g), \quad d_2 = (g^2 - g_0^2) \sqrt{\frac{\beta (\beta^2 + 1)}{32 \gamma \tau}} \tag{17.8.7}
\]

with the reflection coefficients given by Eq. (17.7.5). For the type-1 Chebyshev case, we have:
\[
y_1 = \frac{(a^2 + c_1^2) \Omega_B^2 - 1}{(a^2 + c_1^2) \Omega_B^2 + 1}, \quad y_2 = \frac{(a^2 + c_1^2) \Omega_B^2 - 2 a s_1 \Omega_B + 1}{(a^2 + c_1^2) \Omega_B^2 + 2 a s_1 \Omega_B + 1}
\]

Replacing the unit delays \( \hat{z}^{-1} \) by Fig. 17.7.1, and splitting the multiplier \( \tau_2 \) into two factors, one depending on \( g \) and the other on \( \Omega_B \), 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 \( \Gamma_B \). Thus, the transposed, normalized-lattice, and state-space realizations are more flexible.
### 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:

$$
\begin{align*}
  f_1 &= 0 \text{ kHz}, \quad \Delta f_1 = 1 \text{ kHz}, \quad G_1 = 9 \text{ dB} \\
  f_2 &= 4 \text{ kHz}, \quad \Delta f_2 = 2 \text{ kHz}, \quad G_2 = 12 \text{ dB} \\
  f_3 &= 9 \text{ kHz}, \quad \Delta f_3 = 2 \text{ kHz}, \quad G_3 = -6 \text{ dB} \\
  f_4 &= 20 \text{ kHz}, \quad \Delta f_4 = 4 \text{ kHz}, \quad G_4 = 6 \text{ dB}
\end{align*}
$$

(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:

$$
G_{B1} = 6 \text{ dB}, \quad G_{B2} = 9 \text{ dB}, \quad G_{B3} = -3 \text{ dB}, \quad G_{B4} = 3 \text{ dB} \quad \text{(17.9.2)}
$$

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.

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,

$$
G_{B1} = 8.99 \text{ dB}, \quad G_{B2} = 11.99 \text{ dB}, \quad G_{B3} = -5.99 \text{ dB}, \quad G_{B4} = 5.99 \text{ dB} \quad \text{(17.9.3)}
$$

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 $G_0$:

$$
G_{B1} = 0.01 \text{ dB}, \quad G_{B2} = 0.01 \text{ dB}, \quad G_{B3} = -0.01 \text{ dB}, \quad G_{B4} = 0.01 \text{ dB} \quad \text{(17.9.4)}
$$

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 $G_s$:

$$
G_{s1} = 8.99 \text{ dB}, \quad G_{s2} = 11.99 \text{ dB}, \quad G_{s3} = -5.99 \text{ dB}, \quad G_{s4} = 5.99 \text{ dB} \\
G_{s1} = 0.01 \text{ dB}, \quad G_{s2} = 0.01 \text{ dB}, \quad G_{s3} = -0.01 \text{ dB}, \quad G_{s4} = 0.01 \text{ dB} \quad \text{(17.9.5)}
$$

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:

$$
G_{b1} = 6 \text{ dB}, \quad G_{b2} = 9 \text{ dB}, \quad G_{b3} = -3 \text{ dB}, \quad G_{b4} = 3 \text{ dB} \quad \text{(17.9.6)}
$$
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

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 kHz. The input was a 3000-sample long unit-amplitude sinusoid of frequency of 400 Hz. For the first 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 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.

Fig. 17.9.6 Response of equalizer with time-varying center frequency and bandwidth.

Fig. 17.9.7 Sinusoidal response of equalizer with time-varying gain.
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]:

\[
\begin{align*}
\sigma &= \frac{-\hat{a}_1}{2}, \quad \omega = \sqrt{\hat{a}_2 - \frac{\hat{a}_1^2}{4}}, \quad p = \sigma + j\omega \\
a_1 &= \hat{b}_1 - \hat{b}_0\hat{a}_1, \quad a_2 = \hat{b}_2 - \hat{b}_0\hat{a}_2 \\
\alpha_r &= \frac{a_1}{2}, \quad \alpha_i = -\frac{a_1\sigma + a_2}{2}\omega, \quad \alpha = \alpha_r + j\alpha_i \\
P &= \frac{|\alpha|}{1 - |p|^2}, \quad Q = \text{Im}\left(\frac{\alpha}{1 - p^2}\right), \quad k = \frac{P + Q}{\sqrt{P - Q}} \\
B_1 &= \sqrt{\frac{|\alpha| - \alpha_i}{P - Q}}, \quad B_2 = -\text{sign}(\alpha_r)\sqrt{\frac{|\alpha| + \alpha_i}{P + Q}} \\
C_1 &= \frac{\alpha_r}{B_1}, \quad C_2 = \frac{\alpha_r}{B_2}
\end{align*}
\]

which define the \(ABCD\) state-space parameters:

\[
A = \begin{bmatrix} \sigma & \omega k \\ -\omega/k & \sigma \end{bmatrix}, \quad B = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}, \quad C = [C_1, C_2], \quad D = \hat{b}_0
\] (17.10.2)

In the special case when \(\alpha_r = 0\) and \(\alpha_i > 0\), we have:

\[
B_1 = 0, \quad B_2 = -\frac{2|\alpha|}{\sqrt{P + Q}}, \quad C_1 = \sqrt{2|\alpha|(P - Q)}, \quad C_2 = 0
\] (17.10.3)

and in the case, \(\alpha_r = 0\) and \(\alpha_i < 0\):

\[
B_1 = \frac{2|\alpha|}{\sqrt{P - Q}}, \quad B_2 = 0, \quad C_1 = 0, \quad C_2 = -\sqrt{2|\alpha|(P + Q)}
\] (17.10.4)

The condition that the poles be conjugate pairs is equivalent to the reality of the quantity \(\omega\). 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:

\[
A = \begin{bmatrix} -\hat{a}_1 & 0 \\ 0 & \hat{b}_1 \end{bmatrix}, \quad B = \begin{bmatrix} B_1 \\ 0 \end{bmatrix}, \quad C = [q_1/B_1, 0], \quad D = \hat{b}_0
\] (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:

\[
s \rightarrow s + \frac{\omega_0^2}{s} \tag{17.11.1}
\]

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:

\[
\omega_{2,1} = \pm \frac{\Delta\omega}{2} + \left(\frac{\omega_0^2 + \Delta\omega^2}{4}\right)^{1/2}
\]

They satisfy the relationships,

\[
\omega_0^2 = \omega_1\omega_2, \quad \Delta\omega = \omega_2 - \omega_1
\]

In octaves, we have,

\[
\omega_1 = 2^{\sqrt{\hat{b}/2}}\omega_0, \quad \omega_2 = 2^{\sqrt{\hat{b}/2}}\omega_0, \quad \Delta\omega = 2\omega_0\sinh(\hat{b}/2)
\]

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, \(s \rightarrow 1/s\).

### 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
- `hpeq0,1,2`: 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 bandwith 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. 17.7.2
- `dfzfilt`: 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)
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
hpeqexl_a
resp_a

18.1 Introduction

The DTFT is defined, in general, for an infinitely-long signal,

\[ X(\omega) = \sum_{n=-\infty}^{\infty} x(n) e^{-j\omega n}, \quad \omega = \frac{2\pi f}{f_s} \]

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 shown below,
18. SHORT-TIME FOURIER TRANSFORM

884

18.2 SHORT-TIME FOURIER TRANSFORM

\[ X(\omega, mR) = \sum_{n=0}^{N-1} x(mR + n)w(n)e^{-j\omega n} \]

where \( R \leq N \) and the \( N \) time samples within the \( m \)th segment being transformed are,

\[ x_m(n) = x(mR + n)w(n), \quad n = 0, 1, \ldots, N - 1 \]

The discrete STFT is obtained by replacing the above DTFTs by \( N \)-point DFTs, that is, evaluating them at the \( N \) DFT frequencies,

\[ \omega_k = \frac{2\pi k}{N}, \quad k = 0, 1, \ldots, N - 1 \]

Thus, setting, \( X_{k,m} = X(\omega_k, mR) \), we define the STFT,

\[ X_{k,m} = \sum_{n=0}^{N-1} x(mR + n)w(n)e^{-j\omega_k n} = \sum_{n=0}^{N-1} x_m(n)e^{-j\omega_k n} \quad \text{(STFT)} \quad (18.2.1) \]

\[ k = 0, 1, \ldots, N - 1, \quad m = 0, 1, \ldots, M \]

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, \( x(n), n = 0, 1, \ldots, L_x - 1 \), 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 \( N \),

\[ M = \text{floor} \left( \frac{L_x}{R} \right) \Rightarrow x_{\text{ext}}(n) = \begin{cases} x(n), & 0 \leq n \leq L_x - 1 \\ 0, & L_x \leq n \leq L_{\text{ext}} - 1 \end{cases} \quad (18.2.2) \]

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, \( X_{\text{frames}} \), whose \( m \)th column represents the \( m \)th time segment \( x_m(n) \),

\[ X_{\text{frames}} = [x_0, x_1, \ldots, x_m, \ldots, x_M], \quad x_m = \begin{bmatrix} x_m(0) \\ \vdots \\ x_m(N-1) \end{bmatrix} \]

\[ X = \text{DFT}(X_{\text{frames}}) = [\text{DFT}(x_0), \text{DFT}(x_1), \ldots, \text{DFT}(x_M)] \]

In MATLAB, all the DFTs can be computed with a single FFT call, acting column-wise on the columns of \( X_{\text{frames}} \),

\[ X = \text{fft}(X_{\text{frames}}) = [\text{fft}(x_0), \text{fft}(x_1), \ldots, \text{fft}(x_M)] \]

Assembling the overlapping frames into the frame matrix, \( X_{\text{frames}} \), can be done conveniently with the help of the MATLAB function, buffer. But prior to calling the \text{fft} function, each column of \( X_{\text{frames}} \) 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 \textit{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,

\[ f_k = \frac{k f_s}{N}, \quad k = 0, 1, \ldots, N - 1 \]
\[ t_m = m R T_s = \frac{m R}{f_s}, \quad m = 0, 1, \ldots, M \quad (18.3.1) \]

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 segment with instantaneous frequency, \( \dot{f}(t) = f_1 + \alpha t \), then consisting of a sum of two sinusoids of frequencies \( f_1, f_3 \) over the second segment, and finally varying again like a chirp over the third segment,

\[ x(t) = \begin{cases} \cos(2\pi f_1 t + \pi \alpha t^2), & 0 \leq t < T_0 \\ \cos(2\pi f_2 t) + \cos(2\pi f_3 t), & T_0 < t < 2T_0 \\ \cos(2\pi f_1 t + \pi \alpha t^2), & 2T_0 < t < 3T_0 \end{cases} \]

with parameter values and sampling rate,

\[ T_0 = 1000 \text{ sec} \]
\[ f_1 = 1 \text{ Hz}, \quad f_2 = 2 \text{ Hz}, \quad f_3 = 3 \text{ Hz}, \quad \alpha = \frac{f_1}{T_0} \frac{\text{Hz}}{\text{sec}} \]
\[ f_s = 10 \text{ Hz}, \quad T_s = \frac{1}{f_s} = 0.1 \text{ sec} \]
The spectrogram was 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:

```matlab
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('{\it t / T}_0'); ylabel('{\it f} (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('{\it t / T}_0'); ylabel('{\it f} (Hz)'); zlabel('dB');xaxis(0,3, 0:3); yaxis(0,5, 0:5); zaxis(-200,0, -200:100:0);title('spectrogram - 3D surface plot');
```

![Spectrogram example.](image)

### 18.4 Inverse STFT and OLA Reconstruction

The inverse STFT (ISTFT) can be obtained by performing the inverse DFT, reconstructing the \(m\)th segment,

\[
x(mR_n + n)w(n) = x_m(n) = \frac{1}{N} \sum_{k=0}^{N-1} X_{k,m} e^{j\omega_k n}
\]

\(0 \leq n \leq N - 1, \quad m = 0, 1, \ldots, M\)  \hspace{1cm} (18.4.1)

However, solving for \(x(mR_n + n)\) 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,

\[
y(n) = \sum_{m=-\infty}^{\infty} x_m(n - mR) \quad \text{(ISTFT, OLA reconstruction)} \hspace{1cm} (18.4.2)
\]

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,

\[
y(n) = x(n) \hat{w}(n)
\]

where \(\hat{w}(n)\) is the overlapped-added version of the window,

\[
\hat{w}(n) = \sum_{m=-\infty}^{\infty} w(n - mR) \hspace{1cm} (18.4.3)
\]

Thus, even if \(\hat{w}(n)\) is not constant in \(n\), we can still solve for \(x(n)\) by,

\[
y(n) = x(n) \hat{w}(n) = \sum_{m=-\infty}^{\infty} x_m(n - mR) \Rightarrow x(n) = \frac{\sum_{m=-\infty}^{\infty} x_m(n - mR)}{\sum_{m=-\infty}^{\infty} w(n - mR)} \hspace{1cm} (18.4.4)
\]

Since \(\hat{w}(n)\) is periodic in \(n\) with period of \(R\) samples, it can be expanded in its \(R\)-point discrete Fourier series,

\[
\hat{w}(n) = \sum_{m=-\infty}^{\infty} w(n - mR) = \frac{1}{R} \sum_{r=0}^{R-1} W(\omega_r) e^{j\omega_r n}, \quad \omega_r = \frac{2\pi r}{R} \hspace{1cm} (18.4.5)
\]

\[
W(\omega_r) = \sum_{n=0}^{N-1} w(n) e^{-j\omega_r n} = \text{DFT of } w(n) \text{ evaluated at } \omega = \omega_r
\]
Thus, the condition for the COLA property is that,
\[ W(\omega_r) = 0, \quad r = 1, 2, \ldots, R - 1 \] (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],
- **Hanning:**
  \[ w(n) = 0.5 - 0.5 \cos\left(\frac{2\pi n}{N}\right), \quad n = 0, 1, \ldots, N - 1 \]
- **Bartlett:**
  \[ w(n) = 1 - \frac{|2n - N + 1|}{N - 1}, \quad n = 0, 1, \ldots, N - 1 \]
and the modified Hanning [387],
\[ w(n) = 0.5 - 0.5 \cos\left(\frac{2\pi n}{N}\right), \quad n = 0, 1, \ldots, N - 1 \]

### 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, \( L_s = MR_s + N \),
\[ y(n) = 0, \quad n = 0, 1, \ldots, L_s - 1 \]

b. The STFT \( X_{k,m} \) of the input \( x(n) \) is computed relative to the analysis hop size \( R_a \),
\[ X_{k,m} = \sum_{n=0}^{N-1} x(mR_a + n) w(n) e^{-j\omega_km} \] (18.5.1)
\[ 0 \leq k \leq N - 1, \quad 0 \leq m \leq M \]
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 \( w(n) \),
\[ y_m(n) = w(n) \cdot \frac{1}{N} \sum_{k=0}^{N-1} Y_{k,m} e^{j\omega_km}, \quad 0 \leq n \leq N - 1 \] (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 \( y(n) \),
\[ y(n) = \sum_{m=-\infty}^{\infty} y_m(n - mR_s) \] (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, \( X_{\text{buff}} = [x_0, x_1, \ldots, x_M] \), i.e., \( X_{\text{buff}}(n, m) = x_m(n) \).
18.6. STFT COMPUTATION

Then, the \( N \)-point FFT of that matrix will generate, after windowing, the FFTs of all the columns, resulting in the STFT matrix \( X \),

\[
\begin{align*}
X_{\text{buff}} &= \text{buffer}(x, N, N - R_a, \text{‘nodelay’}) \\
W &= \text{repmat}(w, 1, M + 1) = \text{replicated window} \\
X &= \text{fft}(W \ast X_{\text{buff}}, N)
\end{align*}
\]  

(18.6.1)

where \( w \) is the \( N \)-dimensional column vector of the window, \( w_n, n = 0, 1, \ldots, N - 1, \)

\[
w = \begin{bmatrix}
w_0 \\
w_1 \\
\vdots \\
w_{N-1}
\end{bmatrix}
\]

and \( W \) is its replication into an \( N \times (M + 1) \) matrix so that it can be multiplied point-wise by \( X_{\text{buff}} \). The replication operation was implemented with the built-in function, \text{repmat}, but it can also be implemented as follows, assuming \( W \) is a column vector,

\[
W = w(:, \text{ones}(1, M + 1)) = \begin{bmatrix}
w_0 & w_0 & \cdots & w_0 \\
w_1 & w_1 & \cdots & w_1 \\
\vdots & \vdots & \vdots & \vdots \\
w_{N-1} & w_{N-1} & \cdots & w_{N-1}
\end{bmatrix}
\]

\( M+1 \) columns

The \( N - R_a \) argument of the \text{buffer} function indicates the amount of overlap of successive segments, while the \text{‘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,

\[
\begin{align*}
Y_{\text{buff}} &= \text{ifft}(Y, N) \\
y_m(n) &= Y_{\text{buff}}(n, m) = \text{nth element of mth column}
\end{align*}
\]  

(ISTFT)  

(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,

\[
y(mR_s + n) = y(mR_s + n) + y_m(n)w(n)
\]

(OLA reconstruction)  

(18.6.3)

where the \( n \)-loop can be vectorized and we must initialize \( y(n) \) to zero, that is, \( y(n) = 0 \), for \( n = 0, 1, \ldots, L_s - 1 \), where, \( L_s = MR_s + N \).

The ISP function, \text{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.

\[
\begin{align*}
X_{\text{buff}} &= \text{buffer}(x, N, N - R_a, \text{‘nodelay’}) \\
W &= \text{repmat}(w, 1, M + 1) = \text{replicated window} \\
X &= \text{fft}(W \ast X_{\text{buff}}, N) = \text{input STFT} \\
Y &= f(X) = \text{desired modification, output STFT} \\
Y_{\text{buff}} &= \text{ifft}(Y, N) = \text{output time frames} \\
y &= \text{ola}(W \ast Y_{\text{buff}}, R_s) = \text{OLA reconstruction with windowing}
\end{align*}
\]  

(18.6.4)

where the operation \text{‘\ast’}, denotes element-wise multiplication.

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 \( L_a \),
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, \( X_{k,m} \rightarrow Y_{k,m} \), 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,

\[
X_{k,m} = |X_{k,m}| e^{j\Phi_{k,m}} \\
Y_{k,m} = |Y_{k,m}| e^{j\Psi_{k,m}}
\]

where the magnitudes are preserved,

\[
|Y_{k,m}| = |X_{k,m}|
\]

and the output phases \( \Psi_{k,m} \) are computed recursively from the input phases \( \Phi_{k,m} \), as follows,

\[
\begin{align*}
\omega_k &= \frac{2\pi k}{N} \\
\Psi_{k,0} &= \Phi_{k,0} \\
\text{for } m = 1, 2, \ldots, M, \\
\Delta \omega_{k,m} &= \frac{1}{R_a} \left[ \Phi_{k,m} - \Phi_{k,m-1} - R_a \omega_k \right] \mod 2\pi \\
\omega_{k,m} &= \omega_k + \Delta \omega_{k,m} \\
\Psi_{k,m} &= \Psi_{k,m-1} + R_s \omega_{k,m}
\end{align*}
\]

where the \( k \)-loop may be vectorized, and the notation, \([x]_{\mod 2\pi} \), stands for the phase unwrapping of the angle \( x \) modulo \( 2\pi \), that is, adding to, or subtracting from, \( x \) enough multiples of \( 2\pi \) until it lies in the standardized angle interval, \(-\pi \leq x \leq \pi\). It can be implemented easily by the vectorized anonymous MATLAB function, \( \text{mod2pi}(x) \),

\[
\text{mod2pi} = @(x) \text{mod}(x+\pi, 2\pi) - \pi; \quad \% \text{usage: } y = \text{mod2pi}(x)
\]

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,

\[
\begin{align*}
\frac{r}{R_s} &= \frac{R_a}{R_s} \\
\frac{1}{r} &= \frac{R_s}{R_a}
\end{align*}
\]

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, \( R_a = \text{round}(r R_s) \).

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:

\[
\Omega(t) = \frac{d\Phi(t)}{dt}
\]

Considering the signal values at two nearby time instants, \( t \) and \( t + \Delta t \), we may expand the phase to first-order in \( \Delta t \) and approximate the phase and the instantaneous frequency as,

\[
\begin{align*}
\Phi(t + \Delta t) &= \Phi(t) + \dot{\Phi}(t) \Delta t = \Phi(t) + \Omega(t) \Delta t \\
\Omega(t + \Delta t) &= \Omega(t)
\end{align*}
\]

The phase vocoder is based on the implicit assumption that the signal is a sum of such sinusoidal terms with varying amplitudes and phases,

\[
x(t) = \sum_i A_i(t) e^{j\Phi_i(t)} , \quad \Omega_i(t) = \dot{\Phi}_i(t)
\]
18.9. Phase Vocoder Model

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 \( \omega_k \) are the DFT frequencies,

\[
\omega_k = \frac{2\pi k}{N}, \quad k = 0, 1, \ldots, N - 1
\]

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 \( m \)th frame are as follows, for the analysis and synthesis cases,

\[
t_i^{\text{in}} = (mR_a + n)T = t_i^a + nT \quad \text{(analysis)} \\
t_i^{\text{out}} = (mR_s + n)T = t_i^s + nT \quad \text{(synthesis)}
\]

where, \( t_i^a = mR_aT \), and, \( t_i^s = mR_sT \), are the beginning times of the \( m \)th segments. It follows that the sampled signal for the analysis frames, \( x(mR_a + n) \), would be according to the sinusoidal model of Eq. (18.9.2),

\[
x(t_i^{\text{in}} + nT) = \sum_i A_i(t_i^a + nT) e^{j\phi_i(t_i^a + nT)}
\]

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 \( m \)th frame,

\[
A_i(t_i^{\text{in}} + nT) \approx A_i(t_i^a) , \quad 0 \leq n \leq N - 1
\]

\[
\phi_i(t_i^{\text{in}} + nT) \approx \phi_i(t_i^a + nT)\Omega_i(t_i^a)
\]

Defining the digital instantaneous frequencies in [rads/sample],

\[
\omega_i = \Omega_i(t_i^a) T
\]

then, the phase approximations can be written as,

\[
\phi_i(t_i^{\text{in}} + nT) \approx \phi_i(t_i^a) + n\omega_i(t_i^a)
\]

so that the signal can be approximated as follows within the \( m \)th frame,

\[
x(t_i^{\text{in}} + nT) \approx \sum_i A_i(t_i^a) e^{j\phi_i(t_i^a) + nj\omega_i(t_i^a)}
\] (18.9.3)

Inserting Eq. (18.9.3) into the analysis STFT, we have,

\[
X_{k,m} = \sum_{n=0}^{N-1} x(mR_a + n) w(n) e^{-j\omega_k n} \\
\]

\[
= \sum_{n=0}^{N-1} \sum_i A_i(t_i^a) e^{j\phi_i(t_i^a) + nj\omega_i(t_i^a)} w(n) e^{-j\omega_k n} \\
= \sum_i A_i(t_i^a) e^{j\phi_i(t_i^a)} \sum_{n=0}^{N-1} e^{jn\omega_i(t_i^a)} e^{-j\omega_k n} w(n)
\]

where the summation over \( n \) is recognized to be the frequency-shifted DTFT of the window \( w(n) \),

\[
\hat{W}(\omega) = \sum_{n=0}^{N-1} e^{-j\omega n}w(n) = \text{DTFT of } w(n)
\]

Thus, the phase approximations can be written as,

\[
\phi_i(t_i^{\text{in}} + nT) \approx \phi_i(t_i^a) + nj\omega_i(t_i^a)
\]

so that the signal can be approximated as follows within the \( m \)th frame,

\[
x(t_i^{\text{in}} + nT) \approx \sum_i A_i(t_i^a) e^{j\phi_i(t_i^a) + nj\omega_i(t_i^a)}
\] (18.9.3)

Inserting Eq. (18.9.3) into the analysis STFT, we have,

\[
X_{k,m} = \sum_{n=0}^{N-1} x(mR_a + n) w(n) e^{-j\omega_k n} \\
\]

\[
= \sum_{n=0}^{N-1} \sum_i A_i(t_i^a) e^{j\phi_i(t_i^a) + nj\omega_i(t_i^a)} w(n) e^{-j\omega_k n} \\
= \sum_i A_i(t_i^a) e^{j\phi_i(t_i^a)} \sum_{n=0}^{N-1} e^{jn\omega_i(t_i^a)} e^{-j\omega_k n} w(n)
\]

where the summation over \( n \) is recognized to be the frequency-shifted DTFT of the window \( w(n) \),

\[
\hat{W}(\omega) = \sum_{n=0}^{N-1} e^{-j\omega n}w(n) = \text{DTFT of } w(n)
\]

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 \( \omega \), 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,

\[
X_{k,m} = \sum_i A_i(t_i^a) e^{j\phi_i(t_i^a)} W(\omega_i(t_i^a) - \omega_k) e^{i[\omega_i(t_i^a) - \omega_k](N-1)/2}
\] (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 \( \omega_i(t_i^a) \) falls within the \( k \)th DFT bin, that is, \( \omega_i(t_i^a) = \omega_k \), will effectively contribute to the above sum. Therefore, we can keep approximately only the \( i = k \) term,

\[
X_{k,m} \approx A_k(t_i^a) e^{j\phi_k(t_i^a)} W(\omega_k(t_i^a) - \omega_k) e^{i[\omega_k(t_i^a) - \omega_k](N-1)/2}
\] (18.9.5)

In this expression, the frequency, \( \omega_k(t_i^a) \), is not equal to \( \omega_k \) but it is nearby, i.e., we can introduce a deviation from \( \omega_k \) to be determined,

\[
\omega_{k,m} = \omega_k(t_i^a) = \omega_k + \Delta \omega_{k,m}
\] (18.9.6)

In order to obtain the phase of the STFT, that is, in polar form,
18.9. PHASE VOCODER MODEL

\[ X_{k,m} = |X_{k,m}| e^{i\Phi_{k,m}} \]

we observe that Eq. (18.9.5) is already separated into real factors and phases, so that up to multiples of 2\(\pi\) we must have,

\[ \Phi_{k,m} = \Phi_k(t_m^\alpha) + [\omega_k(t_m^\alpha) - \omega_k] (N - 1)/2 \]

\[ \Phi_{k,m} = \Phi_k(t_m^\alpha) + R_a \omega_k(t_m^\alpha) \]

Our objective, eventually, is to determine the instantaneous frequencies, \(\omega_k(t_m^\alpha)\), 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, \(t_m^\alpha = (m - 1)R_aT\), to time, \(t_m^\alpha = mR_aT\). But since, we have,

\[ t_m^\alpha = t_{m-1}^\alpha + R_aT \]

we may use the approximate expansion of Eq. (18.9.1) to write,

\[ \Phi_k(t_m^\alpha) = \Phi_k(t_{m-1}^\alpha + R_aT) \approx \Phi_k(t_{m-1}^\alpha) + R_a T \Omega_k(t_m^\alpha), \text{ or,} \]

\[ \Phi_k(t_m^\alpha) = \Phi_k(t_{m-1}^\alpha) + R_a \omega_k(t_m^\alpha) \]

and also from Eq. (18.9.1), we have approximately, \(\omega_k(t_{m-1}^\alpha) = \omega_k(t_m^\alpha)\), so that,

\[ \Phi_{k,m} = \Phi_k(t_m^\alpha) + [\omega_k(t_m^\alpha) - \omega_k] (N - 1)/2 \]

\[ \Phi_{k,m} = \Phi_k(t_m^\alpha) + R_a \omega_k(t_m^\alpha) \]

\[ \Phi_{k,m} = \Phi_{k,m-1} + R_a \omega_k(t_m^\alpha) \]

\[ \Phi_{k,m} = \Phi_{k,m-1} + R_a (\omega_k + \Delta \omega_{k,m}) \]

thus, up to multiples of 2\(\pi\),

\[ \Phi_{k,m} = \Phi_{k,m-1} + R_a (\omega_k + \Delta \omega_{k,m}) \]

\[ R_a \Delta \omega_{k,m} = \Phi_{k,m} - \Phi_{k,m-1} + R_a \omega_k \]

In a similar fashion, we can show that the synthesis STFTs,

\[ Y_{k,m} = |Y_{k,m}| e^{i\Psi_{k,m}} \]

will satisfy similar recursions which will ensure that the instantaneous frequencies are preserved:

\[ \Psi_{k,m} = \Psi_{k,m-1} + R_s (\omega_k + \Delta \omega_{k,m}) \]

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 2\(\pi\). Thus, phase-unwrapping it modulo-2\(\pi\) to fall within the standard interval \([-\pi, \pi]\), we obtain,

\[ R_a \Delta \omega_{k,m} = \left[ \Phi_{k,m} - \Phi_{k,m-1} - R_a \omega_k \right]_{\text{mod}2\pi} \]

thus,

\[ \Delta \omega_{k,m} = \frac{1}{R_a} \left[ \Phi_{k,m} - \Phi_{k,m-1} - R_a \omega_k \right]_{\text{mod}2\pi} \]

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, \(f \rightarrow rf\), 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, \(f \rightarrow f + f_0\).

If we have an audio signal \(x(n)\) recorded at a sampling rate \(f_s\) and we play it back at 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 \(f_{s}' = f_s/r\), then it would sound like the original, but if we play it back at \(f_s\), then \(r\) times faster, then it would have a pitch \(r\) times higher but it will also have \(r\) times longer 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 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.

In a similar fashion, we can show that the phase modification algorithm.

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 \(r < 1\).
18.11. **COMPUTER EXPERIMENTS**

**Phase Vocoder Design**

Write a MATLAB function, `phvoc`, to implement a phase vocoder, with syntax,

```matlab
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 sub-functions, `stft`, `phmap`, `istft`,

```matlab
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`,

\[
w(n) = 0.5 - 0.5 \cos \left( \frac{2\pi n}{N-1} \right), \quad n = 0,1,\ldots,N-1
\]

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.,

```matlab
n = 1:N;
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.,

```matlab
n = 1:N;
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
```

**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:

```matlab
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,

\[ [x,FS] = \text{audioread('flute2.wav')}; x = x(:,1).'; \]

and send it through your \text{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,

\[ \text{audiowrite('pitchmod.wav', [x, zeros(1,FS), y], FS)}; \]

to hear the pitch-shifted result, run the command,

\[ \text{soundsc([x, zeros(1,FS), y], FS)}; \]

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, \( 0 \leq f \leq 1800 \) 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?

\[ \text{Note: MATLAB's FFT function will truncate the signal to length 4096 before computing the spectrum, but that is good enough here to demonstrate the pitch-shifting property.} \]

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 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 included 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,

\[ \text{load x4.mat}; \]

Plot this signal versus time in the range \( 0 \leq t \leq 4 \) msec using a stem plot.

Then, resample this signal at the rate \( f_s/r \) with \( r = 2 \), and plot it again versus \( 0 \leq t \leq 4 \). 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 \( f_s/r \) and plot it again versus \( 0 \leq t \leq 4 \). This is the “chipmunked” version of the original.

Finally, pass the latter signal into your \text{phvoc} function with a time-stretching factor \( r \), or speed-up factor \( 1/r \), and plot the result again versus \( 0 \leq t \leq 4 \). 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.

\[ \text{COLA Property} \]

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,

\[ \tilde{w}(n) = \sum_{m=0}^{M} w(n - mR) \quad (18.11.1) \]

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 \( 0 \leq n \leq 800 \), 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.

\[ \text{Do-It-Yourself Spectrogram} \]

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.

\[ R = 256; \quad \% \text{hop size} \]
\[ N = 2048; \quad \% \text{FFT length} \]
\[ [x,fs] = \text{audioread('flute2.wav')}; x = x(:,1).'; \]
\[ X = \text{stft}(x,R,N).'; \quad \% X assumed to have size Nx(M+1) \]
\[ M = \text{size}(X,2) - 1; \]
\[ k = 0:N/2; \quad \% \text{positive half of Nyquist interval} \]
\[ f = k*fs/N; \quad \% 0 < f <= fs/2 \]
\[ t = (0:M)*R/fs; \quad \% M+1 time frames, hop interval R*Ts = R/fs \]
\[ Xmag = \text{abs}(X(k+1,:)); \quad \% \text{extract positive-frequency half} \]
\[ Xmag = Xmag/\text{max}(\text{max}(Xmag)); \quad \% \text{normalize to unity maximum} \]
18.11. COMPUTER EXPERIMENTS

\[ S = 20 \times \log_{10}(X_{\text{mag}}); \quad \% \text{dB} \]

\[
\text{surf}(t,f,S,'edgecolor','\text{none}'); \quad \% \text{spectrogram, } t \text{ horizontal, } f \text{ vertical}
\]

\[
\text{colormap(jet); colorbar;}
\]

\[
\text{axis tight; view}(0,90); \quad \text{ylabel}('\text{\{t\}} (\text{sec})'); \quad \text{ylim}([0,1800]); \quad \% \text{show range } 0 \leq f \leq 1800 \text{ Hz}
\]

**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

resampled at $f_s / r$, replayed at $f_s$

phase vocoder, time-stretched by factor $r = 2$
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, $x$, assumed here to be a column, and produces a length-$N$ real-valued vector of DCT values, $C$, while the inverse DCT reverses the process, recovering the original signal vector,

$$
\begin{align*}
    x &= \begin{bmatrix} x_0 & x_1 & \cdots & x_{N-1} \end{bmatrix} \\
    C &= \begin{bmatrix} C_0 & C_1 & \cdots & C_{N-1} \end{bmatrix} \\
    x &= \begin{bmatrix} x_0 & x_1 & \cdots & x_{N-1} \end{bmatrix}
\end{align*}
$$

(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.](image)

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.
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,

\[ C_k = \sum_{n=0}^{N-1} x_n \cos \left( \frac{\pi k}{N} \left( n + \frac{1}{2} \right) \right), \quad k = 0, 1, \ldots, N - 1 \]  

(DCT) (19.2.1)

and its inverse,

\[ x_n = \frac{1}{N} C_0 + \frac{2}{N} \sum_{k=1}^{N-1} C_k \cos \left( \frac{\pi k}{N} \left( n + \frac{1}{2} \right) \right), \quad n = 0, 1, \ldots, N - 1 \]  

(IDCT) (19.2.2)

The DCT pair satisfies the following Parseval-like identity,

\[ \sum_{n=0}^{N-1} x_n^2 = \sum_{k=0}^{N-1} C_k^2 \]  

(Parseval) (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 \( \delta_k \) denotes the Kronecker delta,

\[ D_k = \frac{1}{s_k} C_k, \]  

where \( s_k = \sqrt{\frac{N}{2}} (\delta_k + 1) \), \( k = 0, 1, \ldots, N - 1 \) (19.2.4)

so that,

\[ s_0 = \sqrt{N}, \quad s_k = \sqrt{\frac{N}{2}}, \quad k = 1, 2, \ldots, N - 1 \]

\[ D_0 = \sqrt{\frac{1}{N}} C_0, \quad D_k = \sqrt{\frac{2}{N}} C_k, \quad k = 1, 2, \ldots, N - 1 \]

With this definition, Eqs. (19.2.1)–(19.2.3) read as follows,

\[ D_k = \frac{1}{s_k} \sum_{n=0}^{N-1} x_n \cos \left( \frac{\pi k}{N} \left( n + \frac{1}{2} \right) \right), \quad k = 0, 1, \ldots, N - 1 \]  

(DCT) (19.2.5)

\[ x_n = \sum_{k=0}^{N-1} D_k \frac{1}{s_k} \cos \left( \frac{\pi k}{N} \left( n + \frac{1}{2} \right) \right), \quad n = 0, 1, \ldots, N - 1 \]  

(IDCT) (19.2.6)

\[ \sum_{n=0}^{N-1} x_n^2 = \sum_{k=0}^{N-1} D_k^2 \]  

(Parseval) (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 matrix element,

\[ B_{kn} = \cos \left( \frac{\pi k}{N} \left( n + \frac{1}{2} \right) \right), \quad k = 0, 1, \ldots, N - 1 \]

\[ n = 0, 1, \ldots, N - 1 \]  

(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,

\[ C = B x \Rightarrow x = B^{-1} C \]

Define the diagonal matrix of the scale factors \( s_k \),

\[ S = \text{diag}([s_0, s_1, \ldots, s_{N-1}]) = \begin{bmatrix} s_0 & 0 & \cdots & 0 \\ 0 & s_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & s_{N-1} \end{bmatrix} \]

Then, it can be shown that the matrix \( B \) satisfies the following orthogonality-like property from which its inverse can be determined,

\[ BB^T S^2 = \text{diag}([s_0^2, s_1^2, \ldots, s_{N-1}^2]) \]

\[ B^{-1} = B^T S^{-2} = B^T \text{diag}([s_0^{-2}, s_1^{-2}, \ldots, s_{N-1}^{-2}]) \]  

(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, $D = S^{-1}C$,

\[
\begin{align*}
C &= Bx \\
x &= B^T S^{-2} C \\
D &= S^{-1} Bx = Ax \\
x &= B^T S^{-1} D = A^T D
\end{align*}
\]  

(19.2.10)

where we defined the rescaled DCT matrix,

\[
A = S^{-1}B
\]  

(19.2.11)

which is orthogonal, that is,

\[
AA^T = I_N \Rightarrow A^{-1} = A^T
\]  

(19.2.12)

Indeed, we have from Eq. (19.2.9),

\[
AA^T = S^{-1}BB^T S^{-1} = S^{-1}S^2S^{-1} = I_N
\]

The normalized DCT matrix $A$ can be computed for any $N$ by the following one-line anonymous MATLAB function:

\[
A = @(N) \sqrt{2/N} \cdot [\text{ones}(1,N)/\sqrt{2}; \cos(\pi*(1:N-1)'*(0:N-1)+1/2)/N]
\]

Although the above matrix formulation is very efficient in MATLAB for moderate sizes (i.e., $N \leq 1024$), there is an even faster implementation based on computing the $N$-point DCT vector $C$ from a related $2N$-point FFT. The computational steps are summarized as follows.\(^1\) Define the extended signal and DCT column vectors of length-$2N$ obtained by appending to $x$ its reversed version, and similarly, appending its negative reversed version to the DCT,

\[
y = [x_0, x_1, \ldots, x_{N-1}, x_{N-1}, \ldots, x_0]^T
\]  

(19.2.13)

The extended signal $C_{ext} = [C_0, C_1, \ldots, C_{2N-2}, 0, -C_{N-1}, -C_{N-2}, \ldots, -C_1]^T$ then, it can be shown that the $2N$-point DFT of $y$ is related to the vector $C_{ext}$ by,

\[
Y_k = 2e^{j\pi k/2N} C_{ext}^k, \quad k = 0, 1, \ldots, 2N-1, \quad \text{or,}
\]  

(19.2.14)

\[
C_{ext}^k = \frac{1}{2} e^{-j\pi k/2N} Y_k, \quad k = 0, 1, \ldots, 2N-1
\]

where we have the $2N$-point DFT pair,

\[
Y_k = \sum_{n=0}^{2N-1} y_n e^{-j2\pi nk/2N}, \quad k = 0, 1, \ldots, 2N-1
\]

\[
y_n = \frac{1}{2N} \sum_{k=0}^{2N-1} Y_k e^{j2\pi nk/N}, \quad n = 0, 1, \ldots, 2N-1
\]

---

\(^1\)See, for example, Problem 10.26 in Chap. 10.
Because of the rounding process, the actual realized compression factor, \( r_{\text{actual}} \), may be slightly different from \( r \). The sorting and construction of the new DCT coefficient matrix could be done as follows:

\[
M = \text{size}(D,2); \quad \text{\% no. of frames}
\]
\[
N_r = \text{round}(r \times N); \quad \text{\% no. of kept coefficients per frame}
\]
\[
[\cdot,\, \text{Ir}] = \text{sort}(abs(D), 'descend'); \quad \text{\% sort column-wise, descending order}
\]
\[
\text{Ir} = \text{Ir}(1:N_r,:); \quad \text{\% Ir = Nr x M matrix of sorting indices}
\]
\[
C = \text{zeros(size}(D)); \quad \text{\% kept DCT coefficients}
\]
\[
\text{for } m = 1:M \quad \text{\% rebuild DCT from the kept coefficients}
\]
\[
\text{Dr}((:,m)) = D(\text{Ir}(:,m),m); \quad \text{\% Dr = Nr x M matrix, sorted coefficients}
\]
\[
C(\text{Ir}(:,m),m) = \text{Dr}((:,m)); \quad \text{\% C = new DCT coefficients}
\]
\]

In a storage/retrieval or transmitting/receiving system, one would store or transmit both the index and coefficient matrices, \( \text{Ir}, D_r \), 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,

\[
\begin{array}{c|c|c|c|c|c|c}
\% & \% & \% & \% & \% & \% & \% \\
\text{Ir} & \text{Dr} & \text{Ir} & \text{Dr} & \text{Ir} & \text{Dr} & \text{Ir} & \text{Dr} \\
1 & 1 & 1 & 1 & 1.2623 & 4.9797 & 2.6349 & 6.5811 \\
2 & 2 & 2 & 2 & 0.9433 & -0.7086 & 1.2451 & -1.2616 \\
3 & 3 & 3 & 3 & 0.1282 & -0.3166 & 0.4891 & -1.4116 \\
4 & 5 & 5 & 5 & 0.0959 & 0 & 0 & 0
\end{array}
\]

\[
D_{\text{thr}} = r_{\text{thr}} \cdot |D|_{\text{max}} \quad (19.3.2)
\]

This method can be implemented by the example code:

\[
\begin{align*}
D_{\text{thr}} &= r_{\text{thr}} \times \text{max}(|\text{abs}(D)|); \\
I &= \text{find}(|\text{abs}(D)| < D_{\text{thr}}); \\
C &= D; \\
C(I) &= 0; \\
r_{\text{actual}} &\equiv 1 - \frac{\text{length}(I)}{(N \times M)};
\end{align*}
\]

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, \( r_{\text{actual}} = 0.4 \)).

\[
\begin{array}{c|c|c|c|c|c|c}
\% & \% & \% & \% & \% & \% & \% \\
\text{method-1} & C & \text{method-2} & C & \text{method-1} & C & \text{method-2} \\
2 & -0.9433 & -0.7086 & 1.2451 & -1.2616 & -0.9433 & -0.7086 & 1.2451 & -1.2616 \\
3 & 0.1282 & -0.3166 & 0.4891 & -1.4116 & 0.1282 & -0.3166 & 0.4891 & -1.4116 \\
4 & -0.0959 & 0 & 0 & 0 & -0.0959 & 0 & 0 & 0
\end{array}
\]

The method 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 \texttt{dct2} function [393] compresses an image with an effective 40% compression ratio:

\[
\begin{align*}
X &= \text{imread('cameraman.tif')}; \quad \text{\% read image, 256x256 matrix} \\
D &= \text{dct2}(X); \quad \text{\% compute its 2D-DCT} \\
D_{\text{max}} &= \text{max}(|\text{abs}(D)|); \quad \text{\% Dmax = 30393.4687} \\
D_{\text{thr}} &= D_{\text{max}} \times 0.4; \quad \text{\% select a threshold} \\
r_{\text{thr}} &= D_{\text{thr}} / D_{\text{max}}; \quad \text{\% with threshold factor,} \\
C &= D; \\
C(\text{abs}(D) < D_{\text{thr}}) &= 0; \quad \text{\% compressed DCT}
\end{align*}
\]
ra = length(find(C))/prod(size(C)) % actual compression ratio,
% ra = 26617/65536 = 0.4061
Y = idct2(C);
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.

\[
D_k = \frac{2}{N-1} \sum_{n=0}^{N-1} x_n \cos \left( \frac{\pi}{N} \left( k + \frac{1}{2} \right) \left( n + \frac{1}{2} \right) \right), \quad k = 0, 1, \ldots, N-1
\]

\[
y_n = \frac{1}{N} \sum_{k=0}^{N-1} D_k \cos \left( \frac{\pi}{N} \left( k + \frac{1}{2} \right) \left( n + \frac{1}{2} \right) \right), \quad n = 0, 1, \ldots, 2N-1
\]

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,

\[
x = \begin{bmatrix} a & b \end{bmatrix} \xrightarrow{\text{MDCT}} D \xrightarrow{\text{IMDCT}} y = \begin{bmatrix} \frac{1}{2} (a - a_R) \\ \frac{1}{2} (b + b_R) \end{bmatrix}
\]

where \(a_R, b_R\) denote the reversed vectors, e.g., for \(N = 4\),

\[
a = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \Rightarrow a_R = \begin{bmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} = J a
\]

where, defining the \(N \times N\) reversing matrix \(J\) having ones along its anti-diagonal line, one may think of \(a_R\) as the result of the matrix operation, \(a_R = J a\). Thus, introducing also the \(N \times N\) identity matrix \(I\), one may write Eq. (19.4.2) as,

\[
x = \begin{bmatrix} a \\ b \end{bmatrix} \xrightarrow{\text{MDCT}} D \xrightarrow{\text{IMDCT}} y = \begin{bmatrix} \frac{1}{2} (I - J) a \\ \frac{1}{2} (I + J) b \end{bmatrix} = \begin{bmatrix} \frac{1}{2} (I - J) \\ 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}
\]

Pictorially, we have for one block,

\[
x = \begin{bmatrix} a \\ b \end{bmatrix} \xrightarrow{\text{MDCT}} D \xrightarrow{\text{IMDCT}} y = \begin{bmatrix} (a-a_R)/2 \\ (b+b_R)/2 \end{bmatrix}
\]

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,

The 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.
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,
\[
\frac{1}{2} (b + b_R) + \frac{1}{2} (b - b_R) = b
\]
(19.4.4)
This is precisely the time-domain aliasing cancellation (TDAC) property.

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,
\[
F_{kn} = \cos \left( \frac{\pi}{N} \left( k + \frac{1}{2} \right) \left( n + \frac{1}{2} + \frac{1}{2}N \right) \right), \quad k = 0, 1, \ldots, N - 1 \quad n = 0, 1, \ldots, 2N - 1
\]
(19.4.5)
Then, the MDCT and IMDCT transformations (19.4.1) can be expressed in matrix form,
\[
\begin{align*}
D &= Fx \\
y &= \frac{1}{N} F^T D
\end{align*}
\]
(19.4.6)
We may split \(F\) into its two \(N \times N\) submatrices, \(F = [A, B]\), defined by their matrix elements,
\[
A_{kn} = F_{kn} = \cos \left( \frac{\pi}{N} \left( k + \frac{1}{2} \right) \left( n + \frac{1}{2} + \frac{1}{2}N \right) \right), \quad k = 0, 1, \ldots, N - 1
\]
\[
B_{kn} = F_{k,n+N} = \cos \left( \frac{\pi}{N} \left( k + \frac{1}{2} \right) \left( n + \frac{1}{2} + \frac{1}{2}N \right) \right), \quad n = 0, 1, \ldots, N - 1
\]
(19.4.7)
Then, it can be shown that the submatrices \(A, B\) satisfy the relationships,
\[
\begin{align*}
\frac{1}{N} A^T A &= \frac{1}{2} \left[ (I - J) \right] \\
\frac{1}{N} B^T B &= \frac{1}{2} \left[ (I + J) \right] \\
A^T B &= B^T A = 0
\end{align*}
\]
(19.4.8)
And, these imply Eq. (19.4.3) because,
\[
\frac{1}{N} F^T F = \frac{1}{N} \begin{bmatrix} A^T & B^T \end{bmatrix} \begin{bmatrix} A & B \end{bmatrix} = \frac{1}{N} \begin{bmatrix} A^T A & A^T B \\ B^T A & B^T B \end{bmatrix} = \begin{bmatrix} (I - J) & 0 \\ 0 & \frac{1}{2} (I + J) \end{bmatrix}
\]

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.

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,
\[
F_{kn} = \cos \left( \frac{\pi}{N} \left( k + \frac{1}{2} \right) \left( n + \frac{1}{2} + \frac{1}{2}N \right) \right), \quad k = 0, 1, \ldots, N - 1 \quad n = 0, 1, \ldots, 2N - 1
\]
(19.4.5)
Then, the MDCT and IMDCT transformations (19.4.1) can be expressed in matrix form,
\[
\begin{align*}
D &= Fx \\
y &= \frac{1}{N} F^T D
\end{align*}
\]
(19.4.6)
19.5. PRINCEII-BRADLEY WINDOWS

\[ W = \begin{bmatrix} w_0 & 0 & 0 & 0 \\ 0 & w_1 & 0 & 0 \\ 0 & 0 & w_2 & 0 \\ 0 & 0 & 0 & w_3 \end{bmatrix} \]

\[ W_R = \begin{bmatrix} w_3 & 0 & 0 & 0 \\ 0 & w_2 & 0 & 0 \\ 0 & 0 & w_1 & 0 \\ 0 & 0 & 0 & w_0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

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,

\[ x = \begin{bmatrix} a \\ b \end{bmatrix} \rightarrow \hat{W}x = \begin{bmatrix} Wa \\ W_R b \end{bmatrix} \]

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,

\[ \hat{W}x \rightarrow W_{\text{MDCT/IMDCT}} \rightarrow \frac{1}{2}(W^2a - WW_Ra_R) \rightarrow \frac{1}{2}(W_Rb + W_Rb_R) = y \]

where we made use of the properties, \( W_R = JWJ \) and \( J^2 = I \), which imply,

\[ (Wa)_R = JWa = JWJJa = W_R a_R \]

\[ (W_R b)_R = JW_R b = J^2WJb = Wb_R \]

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,

\[ \frac{1}{2}(W^2b + WW_Rb_R) + \frac{1}{2}(W^2b - WW_Rb_R) = \frac{1}{2}(W^2 + W_R^2)b \]

where \( b_R \) 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),

\[ \frac{1}{2}(W^2 + W_R^2) = I \] (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],

\[ \hat{w}_n = \sqrt{2} \sin \left( \frac{\pi}{2N} \left( n + \frac{1}{2} \right) \right), \quad n = 0, 1, \ldots, 2N - 1 \]

\[ \hat{w}_n = \sqrt{2} \sin \left( \frac{\pi}{2N} \left( n + \frac{1}{2} \right) \right) \quad \text{(vorbis)} \] (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, \( f_k, k = 0, 1, \ldots, N \), 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 \( f_k \),

\[ \hat{w}_n = \left[ \frac{2}{N} \sum_{k=0}^{n} f_k \right]^{1/2}, \quad n = 0, 1, \ldots, N - 1, \quad \text{where} \quad S = \sum_{k=0}^{N} f_k \] (19.5.5)

then, the second length-N half is taken to be the reversed version of \( \hat{w}_n \), obtained by replacing \( n \) by \( N - 1 - n \) in Eq. (19.5.5),

\[ \hat{w}_{N-n} = \left[ \frac{2}{N} \sum_{m=0}^{N-n} f_m \right]^{1/2} = \left[ \frac{2}{N} \sum_{k=n+1}^{N} f_{N-k} \right]^{1/2}, \quad n = 0, 1, \ldots, N - 1 \] (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,

\[ \hat{w}_{N-n} + \hat{w}_{n+N} = \left[ \frac{2}{N} \sum_{k=0}^{N} f_k \right]^{1/2}, \quad n = 0, 1, \ldots, N - 1 \]

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 \( \beta \),

\[ f_k = I_0 \left( \beta \sqrt{1 - \left( \frac{k - N/2}{N/2} \right)^2} \right), \quad k = 0, 1, \ldots, N \] (19.5.7)

Similarly, a Hamming window would have,

\[ f_k = 0.54 - 0.46 \cos \left( \frac{2\pi k}{N} \right), \quad k = 0, 1, \ldots, N \] (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 \( \alpha \),

\[ \sum_{n=0}^{N-1} \cos \left( \frac{\alpha}{N} \left( n + \frac{1}{2} \right) \right) = \frac{\sin \alpha}{2 \sin \left( \frac{\alpha}{2N} \right)} \] (19.6.1)

\(^\dagger\) the usual \( I_0(\beta) \) denominator is not necessary since \( f_k \) gets rescaled by \( S \).

\(^\ddagger\) A Kaiser window with \( \beta = 5.2 \) is good approximation to Hamming window.
Hints: Note the following,

\[ \sum_{n=0}^{N-1} z^n = \frac{z^N - 1}{z - 1} \]

\[ \cos \theta = \Re \{ e^{i\theta} \} \]

\[ e^{i\theta} - 1 = 2j e^{i\theta/2} \sin(\theta/2) \]

\[ \sin(\alpha) = 2 \sin(\alpha/2) \cos(\alpha/2) \]

Moreover, show the special limiting values, for \( \alpha = 2\pi N m \), with arbitrary integer \( m \),

\[ \sum_{n=0}^{N-1} \cos \left( \frac{\alpha}{N} n + \frac{\alpha}{2} \right) \bigg|_{a=2\pi N m} = \frac{\sin \alpha}{2 \sin \left( \frac{\alpha}{2N} \right)} \bigg|_{a=2\pi N m} = N(-1)^m \quad (19.6.2) \]

Then, show the following identity for real \( \alpha, \beta \),

\[ \sum_{n=0}^{N-1} \cos \left( \frac{\alpha}{N} n + \frac{\alpha}{2} \right) \cos \left( \frac{\beta}{N} n + \frac{\beta}{2} \right) = \frac{\sin(\alpha + \beta)}{4 \sin \left( \frac{\alpha + \beta}{2N} \right)} + \frac{\sin(\alpha - \beta)}{4 \sin \left( \frac{\alpha - \beta}{2N} \right)} \quad (19.6.3) \]

2. To prove the matrix equation (19.2.9) for the DCT matrix \( B \), work with the \( km \) matrix element \( (B^T B)_{km} \), and using the identities of the previous question show that,

\[ (B^T B)_{km} = \sum_{n=0}^{N-1} B_{kn} B_{mn} = \frac{N}{2} \left[ \delta(k + m) + \delta(k - m) \right], \quad k, m = 0, 1, \ldots, N - 1 \quad (19.6.4) \]

Then, explain why this is equivalent to,

\[ (B^T B)_{km} = s_k^T \delta(k - m), \quad k, m = 0, 1, \ldots, N - 1 \quad (19.6.5) \]

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,

\[ \frac{1}{N} (A^T A)_{nm} = \frac{1}{2} \left[ \delta(n - m) - \delta(n + m - N + 1) \right] \]

\[ \frac{1}{N} (B^T B)_{nm} = \frac{1}{2} \left[ \delta(n - m) + \delta(n + m - N + 1) \right] \quad (19.6.6) \]

\[ (A^T B)_{nm} = 0 \]

for \( n = 0, 1, \ldots, N - 1 \), and, \( m = 0, 1, \ldots, N - 1 \). Explain why the matrix elements of the \( N \times N \) reversing matrix \( J \) are,

\[ J_{nm} = \delta(n + m - N + 1), \quad n, m = 0, 1, \ldots, N - 1 \]

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,

\[ D = \text{dctf}(X); \quad \% \text{forward DCT} \]

\[ X = \text{idctf}(D); \quad \% \text{inverse DCT} \]

where the DCT is performed on each column of the \( N \times M \) input matrix of signal \( X \). Internally, your functions must contain a single FFT/IFFT call that handles all the frames at once.

Create a test M-file that tests your functions on a 1024 \( \times \) 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 \( X = A^T D \).

**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,

\[ [y, ra] = \text{dctcompr}(x, N, r, \text{method}); \]

where \( x \) = 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,
19.6. DERIVATIONS AND COMPUTER EXPERIMENTS

\[
X = \text{buffer}(x, N) \\
D = \text{dct}(X) \\
C = f(D, r) = \text{method 1 or 2} \\
Y = \text{idct}(C) \\
y = \text{concatenate } Y \text{ columns}
\]

(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,

\[
x_n = \sum_{k=0}^{N-1} D_k F_k(n), \quad n = 0,1,\ldots,N-1
\]

(19.6.8)

where the basis functions \( F_k(n) \) are defined as follows, for \( k = 0,1,\ldots,N-1, \)

\[
F_k(n) = \frac{1}{s_k} \cos \left( \frac{\pi k}{N} \left( n + \frac{1}{2} \right) \right), \quad n = 0,1,\ldots,N-1
\]

(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, \texttt{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:

\[
\% w = \text{pbwin}(N, \text{type}, \beta); \\
\%
\% N = window half-length \\
\% type = 0,1,2,3, for rectangular, sine, vorbis, KBD \\
\% beta = Kaiser shape parameter when type=3 \\
\%
\% w = length-2N window, column vector
\]

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,

\[
\begin{array}{ccc}
\text{sine} & \text{vorbis} & \text{KBD} \\
0.2759 & 0.0845 & 0.1836 \\
0.7857 & 0.6591 & 0.7356 \\
1.1759 & 1.2512 & 1.2078 \\
1.3870 & 1.4117 & 1.4023 \\
1.3870 & 1.4117 & 1.4023 \\
1.1759 & 1.2512 & 1.2078 \\
0.7857 & 0.6591 & 0.7356 \\
0.2759 & 0.0845 & 0.1836
\end{array}
\]

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, \texttt{mdct} and \texttt{imdct}, that implement the forward and inverse MDCT transformations defined in Eq. (19.4.1), with usage:

\[
D = \text{mdct}(X); \quad \% \text{forward MDCT} \\
Y = \text{imdct}(D); \quad \% \text{inverse MDCT}
\]

\%
\% 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, \texttt{mdctcompr}, that uses your functions \texttt{mdct}, \texttt{imdct}, \texttt{pbwin}, and implements the MDCT/TDAC compression scheme depicted in Fig. 19.1.2. It must have usage,
19.6. DERIVATIONS AND COMPUTER EXPERIMENTS

\[ y, ra \] = 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,

- \( X = \text{buffer}(x, 2N, N, 'nodelay') \) — create 50% overlapping frames
- \( w = \text{pbwin}(N, \text{win}, \beta) \) — length-2N window (column)
- \( W = \text{repmat}(w, 1, M) \) — replicate window, \( M \) = number of frames
- \( D = \text{mdct}(W .* X) \) — MDCT of windowed frames
- \( C = f(D, r_{th}) \) — compress MDCT using threshold method
- \( Y = W .* \text{imdct}(C) \) — inverse MDCT followed by windowing
- \( y = \text{ola}(Y, N) \) — overlap/add frames

\[ y = y(1 : \text{length}(x)) \] — make \( x, y \) 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., \( |D| < r_{th} \cdot |D|_{\text{max}} \).

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 \( y \) from \( Y \),

\[
\begin{align*}
N &= \text{size}(Y,1)/2; & \text{Y has size } 2N \times M \\
M &= \text{size}(Y,2); & \text{number of frames} \\
L &= N \times M + N; & \text{length of } y, \text{ before made equal to length}(x) \\
y &= \text{zeros}(L,1); & \text{pre-allocate } y \\
n &= (1:2N); & \text{work with column vectors} \\
\text{for } m = 0:M-1: \\
y(m*\text{N} + n) &= y(m*\text{N} + n) + Y(:,m+1); \\
\end{align*}
\]

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 \( r_{th} \) 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,

\[
\begin{align*}
t &= 0:0.01:0.99; & \text{100 time instants in the interval } [0,1) \\
x &= \sin(10^t \cdot t^2) + 2^t; & \text{signal samples}
\end{align*}
\]

Using the parameters \( N = 20, r_{th} = 0.005 \), and a KBD window with shape parameter \( \beta = 15 \), calculate the compressed output signal \( y \) and the achieved compression ratio \( r_a \), and plot both \( x \) and \( y \) 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, \( r_{th} = 0.05 \), 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 \texttt{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 \( x \) 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 column-wise the input \( x \), as well as the windowed/IMDCTs, and overlap-add all columns to obtain the final reconstructed output, \( y \), displayed in the last column, which should agree exactly with \( x \) except for the first 4 and last 4 outputs.
19.6. DERIVATIONS AND COMPUTER EXPERIMENTS

Fig. 19.6.1 DCT basis functions, N = 24.
19.6. Derivations and Computer Experiments

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 $L^2(R)$ of square integrable functions on the real line $R$:

$$
\cdots \subset V_{-2} \subset V_{-1} \subset V_0 \subset V_1 \subset V_2 \subset \cdots \subset L^2(R)
$$

(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

---

1 adapted from the author’s book on Applied Optimum Signal Processing [45]
20.1. MULTIRESOLUTION ANALYSIS

of the $V_j$ subspaces is the entire $L^2 \mathbb{R}$ space, and their intersection, the zero function:

$$\lim_{j \to \infty} V_j = \bigcup_{j=\infty}^{\infty} V_j = L^2 \mathbb{R}, \quad \lim_{j \to -\infty} V_j = \bigcap_{j=-\infty}^{\infty} V_j = \{0\} \quad (20.1.2)$$

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:

$$\phi_{jn}(t) = 2^j \phi(2^j t - n) \quad (20.1.3)$$

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 $L^2 \mathbb{R}$ inner product:

$$(\phi_{jn}, \phi_{jm}) = \int_{-\infty}^{\infty} \phi_{jn}(t) \phi_{jm}(t) dt = \delta_{nm} \quad (20.1.4)$$

The factor $2^{j/2}$ in Eq. (20.1.3) serves to preserve the unit norm of $\phi_{jn}$ 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 $0 \leq t \leq 1$ 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 $\phi(t), \psi(t)$.](image)

Fig. 20.1.2 shows the scaled versions of the scaling and wavelet functions (for the $D_2$ case). Each successive copy $\phi(t), \phi(2t), \phi(2^2 t), \phi(2^3 t)$, 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.

\[\text{In this chapter, all time signals are assumed to be real-valued.}\]

The projection of an arbitrary signal $f(t) \in L^2 \mathbb{R}$ onto the subspace $V_j$ is defined by the following expansion in the $\phi_{jn}$ basis:

$$f_j(t) = \sum_n c_{jn} \phi_{jn}(t) = \sum_n c_{jn} 2^{j/2} \phi(2^j t - n) \quad (20.1.5)$$

with coefficients following from the orthonormality of $\phi_{jn}(t)$:

$$c_{jn} = (f_j, \phi_{jn}) = \int_{-\infty}^{\infty} f(t) \phi_{jn}(t) dt = \int_{-\infty}^{\infty} f(t) 2^{j/2} \phi(2^j t - n) dt \quad (20.1.6)$$

The functions $\phi_{jn}(t)$ are orthonormal for each fixed $j$, and form a basis of $V_j$. Because $V_i \subset V_j$ for $i \leq j$, 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 $L^2 \mathbb{R}$ is actually spanned by the scaled and translated versions of $\psi$, that is, $\psi_{im}(t) = 2^{j/2} \psi(2^j t - n)$ for $i \geq j$, which are orthogonal to $\phi_{jn}(t)$, and are also mutually orthonormal,

$$(\phi_{jn}, \psi_{im}) = 0, \quad i \geq j, \quad (\psi_{in}, \psi_{i'm}) = \delta_{ii'} \delta_{nn'} \quad (20.1.7)$$

Thus, we have the direct sum $L^2 \mathbb{R} = V_j \oplus V_j^\perp$, resulting in the decomposition of $f(t)$ into two orthogonal parts:

$$f(t) = f_j(t) + w_j(t), \quad f_j(t) \in V_j, \quad w_j(t) \in V_j^\perp, \quad f_j(t) \perp w_j(t) \quad (20.1.8)$$

The component $w_j(t)$ is referred to as the “detail,” and incorporates the details of $f(t)$ at all the higher resolution levels $i \geq j$, or finer time scales $2^{-i} \leq 2^{-j}$. It admits the $\psi$-basis expansion:

$$w_j(t) = \sum_{i,j} \sum_n d_{im} \psi_{in}(t) = \sum_{i,j} \sum_n d_{im} 2^{i/2} \psi(2^i t - n) \quad (20.1.9)$$

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 $i \leq j$, the signal $f_j(t)$ incorporates information about $f(t)$ from all coarser resolutions (cf. Eq. (20.2.8)).

The factor $2^{j/2}$ in Eq. (20.1.3) serves to preserve the unit norm of $\phi_{jn}$ 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 $0 \leq t \leq 1$ 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. 20.1.2 Daubechies $D_2$ functions $\phi(t), \psi(t)$ and their compressed versions.](image)
with detail coefficients \( d_{in} = \langle w_j, \psi_{in} \rangle = \langle f, \psi_{in} \rangle \). In summary, one form of the multiresolution decomposition is,

\[
    f(t) = f_j(t) + w_j(t) = \sum_n c_{jn} \phi_{jn}(t) + \sum_{i-j} \sum_n d_{in} \psi_{in}(t)
\]

(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 \( \psi_{in}(t) \):

\[
    f(t) = \sum_{i=-\infty}^\infty \sum_n d_{in} \psi_{in}(t), \quad d_{in} = \int_{-\infty}^{\infty} f(t) \psi_{in}(t) dt
\]

(20.1.11)

Yet another, and most practical, version of the multiresolution decomposition is obtained by noting that \( V_\infty = L_2R \). 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 \( i \geq j \), and therefore,

\[
    f(t) = f_j(t) = \sum_n c_{jn} \phi_{jn}(t)
\]

(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)):

\[
    f(t) = \sum_n c_{jn} \phi_{jn}(t) = \sum_n c_{jn} \phi_{jn}(t) + \sum_{i-j} \sum_n d_{in} \psi_{in}(t) = f_j(t) + w_j(t)
\]

(20.1.13)

The mapping of the expansion coefficients from level \( J \) to levels \( j \) through \( J-1 \),

\[
    c_{jn} \rightarrow \{c_{jn} ; \quad d_{in}, j \leq i \leq J - 1 \}
\]

(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 \( 2^J \phi(2^J t - n) \) tends to a Dirac delta function for large \( J \) (see [413] for a proof), so that,

\[
    2^J \phi(2^J t - n) \approx 2^J \delta(2^J t - n) = \delta(t - n2^{-J})
\]

(20.1.15)

Therefore, \( 2^{J/2} \phi(2^{J/2} t - n) \approx 2^{-J/2} \delta(t - n2^{-J}) \), and Eq. (20.1.6) gives,

\[
    c_{jn} \approx \int_{-\infty}^{\infty} f(t) 2^{-J/2} \delta(t - n2^{-J}) \phi_0 dt = 2^{-J/2} \int f(n2^{-J}) dt
\]

(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 \( 0 \leq t \leq 1 \),

\[
    f(t) = \begin{cases} 
        \sin(4\pi t), & 0 \leq t < 0.25 \\
        \sin(2\pi t), & 0.25 \leq t < 0.75 \\
        \sin(4\pi t), & 0.75 \leq t < 1 
    \end{cases}
\]

(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.

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:

\[
    J = 8; \quad N = 2^J; \\
    t1 = (0:N/4-1)'/N; \quad t2 = (N/4:N/4-1)'/N; \quad t3 = (3*N/4:N-1)'/N; \\
    t = [t1; t2; t3]; \\
    y = [\sin(4*\pi*t1); \sin(2*\pi*t2); \sin(4*\pi*t3)]; \quad \% define signal
\]

\[
    h = \text{daub}(1); \quad \% use \text{h = daub}(3) \text{ for Fig. 20.1.4} \\
    j = 5; \quad Y = \text{dwtdec}(y,h,j); \quad \% DWT decomposition to level \( j \) \\
    fJ = Y(:,1); \quad wJ = \text{sum}(Y(:,2:end),2); \quad \% approximation \text{f}_J(t) \text{ and detail w}_J(t)
\]
20. 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

$$\phi(t) = \sum_n h_n 2^{1/2}\phi(2t-n)$$

which is known as the dilation or refinement equation. The coefficients $h_n$ are given by:

$$h_n = (\phi, \phi_{1n}) = 2^{1/2}\int_{-\infty}^{\infty} \phi(t) \phi(2t-n) dt$$

Moreover, the wavelet function $\psi(t)$ and its translates $\psi_{jn} = \psi(t-n)$ 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:

$$V_0 \oplus W_0 = V_1$$

(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 $\phi_{jn}$ basis, as in Eq. (20.2.1),

$$\psi(t) = \sum_n g_n 2^{1/2}\phi(2t-n)$$

(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 $\phi$-basis, $\psi_{jn}(t) = 2^j/2\psi(2^jt - n)$. The dilation equations can also be written with respect to the $\phi_{jn}$, $\psi_{jn}$ 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,

$$\phi_{jk}(t) = \sum_m h_m \phi_{j+1,m+2k}(t) = \sum_n h_{n-2k} \phi_{j+1,n}(t)$$

$$\psi_{jk}(t) = \sum_m g_m \phi_{j+1,m+2k}(t) = \sum_n g_{n-2k} \psi_{j+1,n}(t)$$

(20.2.5)

Using the orthogonality property $(\phi_{j+1,n}, \phi_{j+1,m}^*) = \delta_{nm}$, we have the inner products,

$$h_{n-2k} = (\phi_{jk}, \phi_{j+1,n})$$

$$g_{n-2k} = (\psi_{jk}, \psi_{j+1,n})$$

(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 $\phi_{jk}$ and $\psi_{jk}$, 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,

$$\phi_{j+1,n}(t) = \sum_k h_{n-2k} \phi_{jk}(t) + \sum_k g_{n-2k} \psi_{jk}(t)$$

(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 $V_j$,

$$V_j = V_{j-1} \oplus W_{j-1} = V_{j-2} \oplus W_{j-2} \oplus W_{j-1} = \cdots = \bigoplus_{i=-\infty}^{j-1} W_i$$

(20.2.8)
20.2. DILATION EQUATIONS

which states that $V_j$ incorporates the details of all coarser resolutions. Similarly, increasing $j$ and using $V_n = L^2 R$, we obtain the subspace interpretation of Eq. (20.1.10),

$$
V_{j+1} = V_j @ W_j \\
V_{j+2} = V_{j+1} @ W_{j+1} = V_j @ W_j @ W_{j+1} \\
V_{j+3} = V_{j+2} @ W_{j+2} = V_j @ W_j @ W_{j+1} @ W_{j+2}
$$

\[ (20.2.9) \]

... 

$L^2 R = V_j @ (W_j @ W_{j+1} @ W_{j+2} @ \cdots) = V_j @ V_j^\perp$

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),

$$
V_j = V_j @ (W_j @ W_{j+1} \cdots @ W_{J-1}), \quad j < J
$$

\[ (20.2.10) \]

The coefficients $h_n$ define a lowpass filter $H(z) = \sum h_n z^{-n}$ 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, which 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:

\[
\text{h = daub(K,type); \quad \% Daubechies scaling filters - daublets, symmlets, coiflets}
\]

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)` returns $h = [0.7071 0.7071]$, $\phi(t)$ for the Haar.
- `h = daub(1,2)` returns $h = [1.2220 0.2220 -0.2220 0.1111]$, $\phi(t)$ for the Daublet $D_2$.
- `h = daub(1,3)` returns $h = [1.4142 0.4142 -0.4142 -0.4142]$, $\phi(t)$ for the Daublet $D_3$.

Daublets (minimum phase) have length $= 2K$ and $K$ vanishing moments for $\psi(t)$. Symmlets (least asymmetric) have length $= 2K$ and $K$ vanishing moments for $\psi(t)$. Coiflets have length $= 6K$ and $2K$ vanishing moments for $\psi(t)$, and $2K-1$ for $\phi(t)$.

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)` returns $h = [0.7071 0.7071]$, $\phi(t)$ for the Haar.
- `h = daub(2)` returns $h = [0.8814 0.8814 -0.8814 -0.8814]$, $\phi(t)$ for the Daublet $D_2$.
- `h = daub(3)` returns $h = [0.7330 0.7330 -0.7330 -0.7330]$, $\phi(t)$ for the Daublet $D_3$.

The filters $h_n$ can be taken to be causal, i.e., $h_n, 0 \leq n \leq M$, 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 $-2K \leq n \leq 4K - 1$). 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, $g_n = (-1)^n h^{-n}_n$, where $h^n_n = h_{M-n}$, $n = 0, \ldots, M$ is the reversed version of $h_n$.

In the $z$-domain, we have

\[ H(z) = z^{-M} H(z^{-1}) \]

The Nyquist gain of $G(z)$, is equal to

\[ H(1) = 2 \]

In the frequency domain, this reads:

\[ G(\omega) = e^{-jM(\omega+\pi)} H(\omega + \pi) \quad \Rightarrow \quad g_n = (-1)^n h_{M-n}, \quad 0 \leq n \leq M \]  

\[ (20.2.11) \]

The function `cmf` implements this definition:

\[
g = cmf(h); \quad \% \text{conjugate mirror filter}
\]

For example, if $h = [h_0, h_1, h_2, h_3]$, then $g = [h_3, -h_2, h_1, -h_0]$, e.g., we have for the daublet $D_2$:

\[
\begin{align*}
\text{h = daub(2) \Rightarrow g = cmf(h) =} & \quad [0.4830 \quad 0.8365 \quad 0.2241 \quad -0.1294] \\
\text{g = cmf(h) =} & \quad [-0.1294 \quad -0.2241 \quad 0.8365 \quad -0.4830]
\end{align*}
\]

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):

\[ H(0) = \sum_{n=0}^M h_n = \sqrt{2} \]  

\[ G(\pi) = \sum_{n=0}^M (-1)^n g_n = \sum_{n=0}^M h_{M-n} = H(0) = \sqrt{2} \]  

\[ (20.2.13) \]

\[ (20.2.14) \]

The dilation equations can be expressed in the frequency domain as follows:

\[ \Phi(\omega) = 2^{-1/2} H(2^{-1} \omega) \Phi(2^{-1} \omega) \]

\[ \Psi(\omega) = 2^{-1/2} G(2^{-1} \omega) \Phi(2^{-1} \omega) \]

\[ (20.2.14) \]

where $\Phi(\omega), \Psi(\omega)$ are the Fourier transforms:

\[
\Phi(\omega) = \int_{-\infty}^{\infty} \phi(t) e^{-j\omega t} dt, \quad \Psi(\omega) = \int_{-\infty}^{\infty} \psi(t) e^{-j\omega t} dt
\]

\[ (20.2.15) \]

In fact, setting $\omega = 0$ in the first of (20.2.14) and assuming that $\Phi(0) \neq 0$, we immediately obtain the gain conditions (20.2.13). The iteration of Eqs. (20.2.14) leads to the
20.2. DILATION EQUATIONS

![Image of dilation equations]

lowpass and highpass filters

Fig. 20.2.1 Haar, Daubechies $D_2$, and Symmlet $S_6$ scaling and wavelet filters.

infinite product expressions:

\[
\Phi(\omega) = \Phi(0) \prod_{j=1}^{\infty} 2^{-1/2} H(2^{-j}\omega) \]

\[
\Psi(\omega) = \Phi(0) \prod_{j=2}^{\infty} 2^{-1/2} G(2^{-j}\omega) \]

(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 $j \leq J$ 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:

```matlab
epsilon = 1e-5; Jmax = 30;
f = linspace(0,10,513); w = pi*f;
h = daub(2)/sqrt(2); g = cmf(h);
% use h=daub(6,2)/sqrt(2) for right graph
Ph0 = abs(freqz(h,1,w/2));
Ps0 = abs(freqz(g,1,w/2/2));

for J = 2:Jmax,
 Phi = Ph0.*abs(freqz(h,1,w/2^J));
 Psi = Ps0.*abs(freqz(g,1,w/2^J));
 if norm(Phi-Ph0) < norm(Ph0) * epsilon, J, break; end
 Ph0 = Phi;
 Ps0 = 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 $\omega = 2\pi m$, 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,

\[
\phi^{(r+1)}(t) = \sum_n h_n 2^{1/2} \phi^{(r)}(2t - n)
\]

\[
\psi^{(r+1)}(t) = \sum_n g_n 2^{1/2} \psi^{(r)}(2t - n)
\]

(20.2.17)

for $r = 0,1,2,\ldots$, 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 $\phi$. 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 $0 \leq n \leq M$, then the dilation equations imply that $\phi(t)$ and $\psi(t)$ will have compact support over $0 \leq t \leq M$. Thus, we may evaluate the $r$th iterate $\phi^{(r)}(t)$ at the equally-spaced points, $t = 2^{-r}n$, for $0 \leq n \leq M2^r$, spanning the support interval. To this end, we define the discrete-time signals of the sampled $\phi^{(r)}(t)$:

\[
f^{(r)}(n) = 2^{-r/2} \phi^{(r)}(2^{-r}n)
\]

(20.2.18)

where $2^{-r/2}$ is a convenient normalization factor. It follows then from Eq. (20.2.17) that

\[
2^{-(r+1)/2} \phi^{(r+1)}(2^{-r+1}n) = \sum_m h_m 2^{-r/2} \phi^{(r)}(2^{-r}n - m), \quad \text{or,}
\]

\[
f^{(r+1)}(n) = \sum_m h_m f^{(r)}(n - 2^r m) = \sum_k h^{[r]}(k) f^{(r)}(n - k)
\]

(20.2.19)

where we defined the à trous filter corresponding to the interpolation factor $2^r$ by

\[
h^{[r]}(k) = \sum_m h_m \delta(k - 2^r m)
\]

(20.2.20)

\[\text{†a trous* means “with holes” in French.}\]
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 $r$th iterate with the $r$th a trous filter. The recursion can be iterated for $r = 0, 1, 2, \ldots, J$, for sufficiently large $J$ (typically, $J = 10$ works well.) The MATLAB function casc implements this algorithm:

```matlab
[phi, psi, t] = casc(h, J, phi0); % cascade algorithm
```

where $t$ is the vector of final evaluation points $t = 2^{-J}n$, $0 \leq n \leq M2^J$. 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:

```matlab
h = daub(2); J = 10; phi0 = 1;
[phi, psi, t] = casc(h, J, phi0);
figure; plot(t, phi, t, psi, '-');
end
```

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:

```matlab
phi0 = 1;
for r = 0:J-1,
    phi = conv(phi, upr(h, r));
end
```

where the function upr constructs the a 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 $h = [h_0, h_1, h_2, h_3]$, then for $r = 2$, the a trous filter will be,

$$h_r = \text{upsample}(h, 2^r) = [h_0, 0, 0, 0, h_1, 0, 0, 0, h_2, 0, 0, 0, h_3, 0, 0, 0]$$

### 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 $(\phi_j, \phi_k) = (\psi_j, \psi_k) = \delta_k$ and $(\phi_j, \psi_k) = 0$, therefore $h_n, g_n$ must satisfy the orthogonality properties:

\[
\begin{align*}
\sum_n h_n h_{n-2k} &= \delta_k \\
\sum_n g_n g_{n-2k} &= \delta_k \\
\sum_n h_n g_{n-2k} &= 0
\end{align*}
\]

But $(\phi_j, \phi_k) = (\psi_j, \psi_k) = \delta_k$ and $(\phi_j, \psi_k) = 0$, therefore $h_n, g_n$ must satisfy the orthogonality properties:

\[
\begin{align*}
\sum_n h_n h_{n-2k} &= \delta_k \\
\sum_n g_n g_{n-2k} &= \delta_k \\
\sum_n h_n g_{n-2k} &= 0
\end{align*}
\]

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 $H(\omega), G(\omega)$:

\[
\begin{align*}
\sum_n h_n g_{n-k} &\iff H(\omega)G^*(\omega) \\
(-1)^k \sum_n h_n g_{n-k} &\iff H(\omega + \pi)G^*(\omega + \pi)
\end{align*}
\]

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:

\[
\begin{align*}
[1 + (-1)^k] \sum_n h_n g_{n-k} &= 2 \delta_k \\
[1 + (-1)^k] \sum_n g_n g_{n-k} &= 2 \delta_k \\
[1 + (-1)^k] \sum_n h_n g_{n-k} &= 0
\end{align*}
\]

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):

\[
\begin{align*}
|H(\omega)|^2 + |H(\omega + \pi)|^2 &= 2 \\
|G(\omega)|^2 + |G(\omega + \pi)|^2 &= 2 \\
H(\omega)G^*(\omega) + H(\omega + \pi)G^*(\omega + \pi) &= 0
\end{align*}
\]

The conjugate mirror filter choice (20.2.11) for $G(\omega)$ automatically satisfies the third of Eqs. (20.3.4). Indeed, using the $2\pi$-periodicity of $H(\omega)$, we have,

$$G^*(\omega) = e^{iM(\omega+\pi)}H(\omega + \pi)$$

$$G^*(\omega + \pi) = e^{iM(\omega+2\pi)}H(\omega + 2\pi) = e^{iM\pi}H(\omega)$$

so that,

$$H(\omega)G^*(\omega) + H(\omega + \pi)G^*(\omega + \pi) = e^{iM\pi}H(\omega)H(\omega + \pi)[e^{iM\pi} + 1] = 0$$
where $\omega = -1$, 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:

$$
\frac{1}{2} [H^*(\omega)H(\omega) + G^*(\omega)G(\omega)] = 1
$$

(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:

$$
\sum_{n=\text{even}} h_n = \sum_{n=\text{odd}} h_n = \frac{1}{\sqrt{2}}
$$

(20.3.6)

The correlation constraints and the DC gain condition,

$$
\sum_n h_n h_{n-2k} = \delta_k, \quad \sum_n h_n = \sqrt{2},
$$

(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:

$$
\sum_{n=0}^{N-1} (-1)^n n h_n = 0 \iff \left[ \frac{dH(\omega)}{d\omega} \right]_{\omega = \pi} = 0, \quad i = 0, 1, \ldots, K-1
$$

(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 $h = [h_0, h_1]$ must satisfy:

$$
h_0^2 + h_1^2 = 1, \quad h_0 + h_1 = \sqrt{2}
$$

(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 $h = [h_0, h_1, h_2, h_3]$ must satisfy,

$$
h_0^2 + h_1^2 + h_2^2 + h_3^2 = 1
$$

$$
h_0 + h_2 = \frac{1}{\sqrt{2}}, \quad h_1 + h_3 = \frac{1}{\sqrt{2}}
$$

(20.3.10)

$$
- h_1 + 2h_2 - 3h_3 = 0
$$

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:

$$
\left( h_0 + h_2 - \frac{1}{\sqrt{2}} \right)^2 + \left( h_1 + h_3 - \frac{1}{\sqrt{2}} \right)^2
$$

$$
= 1 + (h_0^2 + h_1^2 + h_2^2 + h_3^2) - \sqrt{2}(h_0 + h_1 + h_2 + h_3) + 2(h_0h_2 + h_1h_3)
$$

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:

$$
4h_0^2 - \sqrt{2}h_0 - \frac{1}{4} = 0 \quad \Rightarrow \quad h_0 = \frac{1 \pm \sqrt{3}}{4}
$$

The “+” choice leads to the following minimum-phase filter (the “−” choice leads to the reverse of that, which has maximum phase):

$$
h = [h_0, h_1, h_2, h_3] = \frac{1}{4\sqrt{2}} \left[ 1 + \sqrt{3}, \quad 3 + \sqrt{3}, \quad 3 - \sqrt{3}, \quad 1 - \sqrt{3} \right]
$$

(20.3.11)

$$
= [0.4830, \quad 0.8365, \quad 0.2241, \quad -0.1294]
$$

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 $h = [h_0, h_1, h_2, h_3, h_4, h_5]$:

$$
h_0^2 + h_1^2 + h_2^2 + h_3^2 + h_4^2 + h_5^2 = 1, \quad h_0h_4 + h_1h_5 = 0
$$

$$
h_0 + h_2 + h_4 = \frac{1}{\sqrt{2}}, \quad h_1 + h_3 + h_5 = \frac{1}{\sqrt{2}}
$$

(20.3.12)

$$
- h_1 + 2h_2 - 3h_3 + 4h_4 - 5h_5 = 0
$$

$$
- h_1 + 2^2h_2 - 3^2h_3 + 4^2h_4 - 5^2h_5 = 0
$$

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:

$$
h_2 = -4h_0 + 2h_1 + \sqrt{3}/8
$$

(20.3.13)

$$
h_3 = -2h_0 + 3\sqrt{3}/8
$$

$$
h_4 = 3h_0 - 2h_1 + 3\sqrt{3}/8
$$

$$
h_5 = 2h_0 - h_1 + \sqrt{3}/8
$$

(20.3.14)

Inserting these into the first two of Eqs. (20.3.12), we obtain the quadratic system:

$$
34h_0^2 - (32h_1 - \sqrt{3}/4)h_0 + 10h_1^2 - 5\sqrt{2}h_1/4 - 3/8 = 0
$$

$$
h_1^2 - \sqrt{2}h_1/8 - 3h_0 - 3\sqrt{2}h_0/8 = 0
$$

(20.3.14)
Solving the second for \( h_1 \) in terms of \( h_0 \), we find:

\[
\begin{align*}
  h_1 &= \frac{1}{16} \left[ \sqrt{2} + \sqrt{768} h_0^2 + 96 \sqrt{2} h_0 + 2 \right] \\
  &\quad \text{(20.3.15)}
\end{align*}
\]

and inserting this into the first of (20.3.14), we obtain:

\[
64 h_0^2 + 2 \sqrt{2} h_0 - 2 h_0 \sqrt{768} h_0^2 + 96 \sqrt{2} h_0 + 2 - \frac{3}{8} = 0
\]

or, the equivalent quartic equation:

\[
1024 h_0^2 - 128 \sqrt{2} h_0^3 - 48 h_0^4 - \frac{3}{2} \sqrt{768} h_0 + \frac{9}{64} = 0 \quad \text{(20.3.16)}
\]

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

\[
\begin{align*}
  h_0 &= \frac{\sqrt{2}}{32} \left[ 1 + \sqrt{10} + \sqrt{5 + 2 \sqrt{10}} \right] \\
  &\quad \text{(20.3.17)}
\end{align*}
\]

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:

\[
H(z) = h_0 + h_1 z^{-1} + h_2 z^{-2} + h_3 z^{-3} + h_4 z^{-4} + h_5 z^{-5} = h_0 (1 + z^{-1})^3 (1 - z_1 z^{-1}) = h_0 (1 - z_1 z^{-1})
\]

where \( z_1 \) is the following zero lying inside the unit circle:

\[
z_1 = \frac{\sqrt{10} - 1 + \sqrt{2 \times \sqrt{10} - 5}}{1 + \sqrt{10} + \sqrt{5 + 2 \sqrt{10}}} \Rightarrow |z_1| = \frac{\sqrt{5}}{1 + \sqrt{10} + \sqrt{5 + 2 \sqrt{10}}} = 0.3254 \quad \text{(20.3.18)}
\]

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,

\[
\left. \frac{d^i G(\omega)}{d\omega^i} \right|_{\omega=0} = 0, \quad i = 0, 1, \ldots, K - 1 \quad \text{(20.3.19)}
\]

In turn, these are equivalent to the \( K \) vanishing moment constraints for the wavelet function \( \psi(t) \), that is,

\[
\int_{-\infty}^{\infty} t^i \psi(t) \, dt = 0, \quad i = 0, 1, \ldots, K - 1 \quad \text{(20.3.20)}
\]

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 \( \omega \) and setting \( \omega = 0 \).

Because the \( \Delta_k \) 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 \( \phi \) and \( \psi \) bases can be stated in terms of the Fourier transforms \( \Phi(\omega) \) and \( \Psi(\omega) \) as identities in \( \omega \):

\[
\begin{align*}
  \sum_{m=-\infty}^{\infty} |\Phi(\omega + 2\pi m)|^2 &= \sum_{m=-\infty}^{\infty} |\Psi(\omega + 2\pi m)|^2 = 1 \\
  \sum_{m=-\infty}^{\infty} \Phi(\omega + 2\pi m)\Psi^*(\omega + 2\pi m) = 0
\end{align*}
\]

These follow by applying Parseval's identity to the cross-correlation inner products of the \( \phi \) and \( \psi \) bases. For example, we have,

\[
\delta_k = \langle \phi_{p_0}, \phi_{q_k} \rangle = \langle \phi_{p_0}, \phi_{q_k} \rangle = \int_{-\infty}^{\infty} \phi(t) \phi(t - k) \, dt = \frac{1}{2\pi} \int_{-\infty}^{\infty} |\Phi(\omega)|^2 e^{i\omega k} \, d\omega
\]

where the last expression was obtained by noting that because \( k \) is an integer, the exponential \( e^{i\omega k} \) is periodic in \( \omega \) with period \( 2\pi \), which allowed us to fold the infinite integration range into the \([-\pi, \pi]\) range. But this result is simply the inverse DFT of the first of Eqs. (20.3.21). The other results are shown in a similar fashion using the inner products \( \langle \psi_{p_0}, \psi_{q_k} \rangle = \delta_k \) and \( \langle \phi_{p_0}, \psi_{q_k} \rangle = 0 \).

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 \( H(\pi) = 0 \). 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:

\[
\Phi(0) = \int_{-\infty}^{\infty} \phi(t) \, dt = 1 \quad \text{(20.3.22)}
\]

### 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:

\[
\begin{align*}
  f(t) &= \sum_{n} c_{J_n} \phi_{J_n} + \sum_{k} c_{J_k} \phi_{J_k} \left[ \sum_{j=J_{k-1}}^{J-1} \sum_{j=J_{k}+1}^{J-1} d_{jk} \psi_{jk} \right] \\
  &= \sum_{n} c_{J_n} \phi_{J_n} + \sum_{k} c_{J_k} \phi_{J_k} \left[ \sum_{j=J_{k}+1}^{J-1} \sum_{j=J_{k}+1}^{J-1} d_{jk} \psi_{jk} \right]
\end{align*}
\]

(20.4.1)

with the various terms corresponding to the direct-sum decomposition:

\[
V_J = V_{J_0} \oplus (W_{J_0} \oplus W_{J_1} \oplus \cdots \oplus W_{J_{J-1}})
\]

(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
discrete wavelet transforms

\[
y_{\text{down}}(n) = x(2n)
\]

\[
y_{\text{up}}(n) = \sum_k x(k) \delta(n - 2k) = \begin{cases} 
  x(k), & \text{if } n = 2k \\
  0, & \text{otherwise}
\end{cases}
\] (20.4.6)

and pictorially,

\[
x(n) \xrightarrow{f} \frac{1}{2} f(n/2) \xrightarrow{f_{\text{down}}(n)} \quad \frac{1}{2} f(n/2) \xrightarrow{f_{\text{up}}(n)} \quad y(n)
\]

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,

\[
c_j = \sum_n \bar{h}_{2k-n} c_{j+1,n} = (\bar{h} * c_{j+1}) (2k)
\]

\[
d_j = \sum_n \bar{g}_{2k-n} d_{j+1,n} = (\bar{g} * c_{j+1}) (2k)
\]

(20.4.7)

Similarly, the synthesis algorithm (20.4.5) is equivalent to upsampling the signals \(c_{j,k}\) and \(d_{j,k}\) by two and then filtering them through \(h_n, g_n\),

\[
c_{j+1,n} = \sum_k h_{n-2k} c_{j,n} + \sum_k g_{n-2k} d_{j,n} = \sum_m h_{n-m} c_{j,m}^{\text{up}} + \sum_m g_{n-m} d_{j,m}^{\text{up}}
\] (20.4.8)

where \(c_{j,m}^{\text{up}} = \sum_k c_{j,k} \delta(m - 2k)\). Symbolically,

\[
c_{j+1} = h * c_j^{\text{up}} + g * d_j^{\text{up}}
\] (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 \(\tilde{H}(\omega) = H^*(\omega)\) and \(\tilde{G}(\omega) = G^*(\omega)\). Therefore, at the first analysis stage, the input signal \(c_1\) 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, \(c_3, d_3\), 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.
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,

\[
X(f) = \sum_n x(n) e^{-j \omega n} = \sum_n x(n) e^{-j 2\pi f n / f_s}
\]  

(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 \( f_{s\downarrow} = f_s / L \), the downsampled signal \( x_{\downarrow}(n) = x(nL) \) will have the following replicated spectrum at multiples of \( f_{s\downarrow} \):

\[
Y_{\downarrow}(f) = \frac{1}{L} \sum_{m=0}^{L-1} X(f - mf_{s\downarrow})
\]  

(20.4.11)

where according to (20.4.10),

\[
Y_{\downarrow}(f) = \sum_n y_{\downarrow}(n) e^{-j 2\pi f n / f_{s\downarrow}} = \sum_n x(nL) e^{-j 2\pi f n / f_{s\downarrow}}
\]  

(20.4.12)

In particular, for downsampling by \( L = 2 \), we have \( f_{s\downarrow} = f_s / 2 \) and

\[
Y_{\downarrow}(f) = \frac{1}{2} [X(f) + X(f - f_{s\downarrow})] = \sum_n x(2n) e^{-j 2\pi f n / f_s}
\]  

(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.

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 \( f_{s\downarrow} = f_s / 2 \), thereby filling the gaps created by the ideal filter, and resulting in the spectrum \( C_4(f) \) 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,

\[
\bar{H}(f,f_s / 2) = \sum_n h_n e^{-j 2\pi f n / f_{s\downarrow}} = \sum_n h_n e^{-j 2\pi f n / f_s} = \bar{H}(2f,f_s)
\]  

(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 \( f_{s\downarrow} = f_s / 4 \) resulting in the spectrum \( C_5(f) \) 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:

\[
\bar{H}(f,f_s / 4) = \bar{H}(2f,f_s / 2) = \bar{H}(4f,f_s)
\]  

(20.4.15)

which is the four-times compressed version of that rate \( f_s / 16 \), 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 downsampling 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 spectra $C(f)$, namely, by first filtering by the combined filter,

$$\tilde{H}(f, f_s) \tilde{H}(2f, f_s) \tilde{H}(4f, f_s) = \tilde{H}(f, f_s) \tilde{H}(f, f_s/2) \tilde{H}(f, f_s/4)$$

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]:

$$f_s \quad \frac{1}{2} f_s \quad \frac{1}{4} \quad f_s \quad f_s/2 \quad f_s/4 \quad \equiv \quad f_s \quad f_s/2 \quad f_s/4 \quad f_s/8$$

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.

$$c_j, d_j \rightarrow [c_{j-1}, d_{j-1}], \quad (\text{level } J-1)$$
$$\rightarrow [c_{j-2}, d_{j-2}, d_{j-1}], \quad (\text{level } J-2)$$
$$\rightarrow [c_{j-3}, d_{j-3}, d_{j-2}, d_{j-1}], \quad (\text{level } J-3)$$
$$\vdots$$
$$\rightarrow [c_{j-0}, d_{j-0}, d_{j-0+1}, \ldots, d_{j-1}], \quad (\text{level } J)$$

Starting with the coefficients at any level, the inverse discrete wavelet transform (IDWT) applies the synthesis algorithm to reconstruct the original signal $x$.

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 $N$:

$$\hat{x}(n) = \sum_{p=-\infty}^{\infty} x(n + pN)$$

For non-ideal filters $H(f), G(f)$, 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 perfect 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:

$$c_j = (h \ast c)_{\text{down}}$$
$$d_j = (g \ast c)_{\text{down}}$$
$$c_j = h \ast c_{j-1} + g \ast d_{j-1} \quad j = J + 1, J_0 + 2, \ldots, J$$

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 $x = [x_0, x_1, \ldots, x_{N-1}]^T$, where $N = 2^J$, the DWTs at successive stages are defined as the following sets of coefficients:

$$x = c_j \rightarrow [c_{j-1}, d_{j-1}], \quad (\text{level } J-1)$$
$$\rightarrow [c_{j-2}, d_{j-2}, d_{j-1}], \quad (\text{level } J-2)$$
$$\rightarrow [c_{j-3}, d_{j-3}, d_{j-2}, d_{j-1}], \quad (\text{level } J-3)$$

$$\vdots$$
$$\rightarrow [c_{j-0}, d_{j-0}, d_{j-0+1}, \ldots, d_{j-1}], \quad (\text{level } J)$$

The signal $\hat{x}(n)$ is periodic with period $N$, and therefore, we only need to know it over one period, $0 \leq n \leq N - 1$. It is characterized by the property that it has the same $N$-point DFT as the signal $x(n)$, that is,

$$X(\omega_k) = \sum_{n=-\infty}^{\infty} x(n) e^{-j\omega_k n} = \sum_{n=0}^{N-1} \hat{x}(n) e^{-j\omega_k n}$$

where $\omega_k$ are the DFT frequencies $\omega_k = 2\pi k/N$, $k = 0, 1, \ldots, N-1$. The signal $\hat{x}(n)$ 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 $N$:

$$Y = \text{modwrap}(X, N); \quad \% \text{mod-N reduction of a matrix}$$

For example, we have for the signal $x = [1, 2, 3, 4, 5, 6, 7, 8]$ and $N = 3$,

$$x = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 4 & 5 & 6 & 1 & 2 & 3 & 4 & 5 \end{bmatrix}$$

$$\hat{x} = \begin{bmatrix} 12 & 15 & 6 \end{bmatrix} = \text{modwrap}(x, 3)$$
Circular convolution is defined as the modulo-$N$ reduction of ordinary linear convolution, that is,

$$y = h \ast x \Rightarrow y_{\text{circ}} = \tilde{y} = h \ast \tilde{x} \quad (20.5.6)$$

or more explicitly,

$$y(n) = \sum_m h(m)x(n-m) \Rightarrow \tilde{y}(n) = \sum_p y(n+pN)$$

Its MATLAB implementation is straightforward with the help of the function modwrap, for example,

```matlab
y = modwrap(conv(h,x), N);
```

This code has been incorporated into the function `circonv`, with usage:

```matlab
y = circonv(h,x,N); % mod-N circular convolution
```

For example, we have the outputs for $N = 8$:

- $h = [1, 2, 3, 2, 1]$;
- $x = [1, 2, 3, 4, 5, 6, 7, 8]$;
- $H = \text{fft}(h,8)$; $X = \text{fft}(x,8)$; % calculate 8-point DFTs
- $Y = H \ast X$; % point-wise multiplication of the DFTs
- $ytildex = \text{ifft}(Y,8)$; % inverse DFT generates $\tilde{y} = [55, 48, 33, 26, 27, 36, 45, 54]$ % time-domain calculation
- $y_{\text{circ}} = \text{circonv}(h,x,8)$; % time-domain calculation

The convolutional operation $y = h \ast x$ can be represented in matrix form:

$$y = Hx$$

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 $\tilde{h}_n = h_{-n}$ is given by the transposed matrix

$$\tilde{H} = H^T$$

because $\tilde{H}_{nm} = \tilde{h}_{n-m} = h_{m-n} = H_{mn}$. Thus, in matrix notation, the typical convolutional and down- and up-sampling operations being performed at the analysis and synthesis stages have the forms:

$$y = (H^T x)_{\text{down}}, \quad y = Hx_{\text{up}} \quad (20.5.10)$$

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 $x$. Denoting $\tilde{H} = \text{modwrap}(H,N)$, then the circular version of (20.5.10) would read:

$$\tilde{y} = (\tilde{H} \ast x)_{\text{down}}, \quad \tilde{y} = \tilde{H}x_{\text{up}} \quad (20.5.11)$$
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 \( \tilde{H}_{\text{down}} \), before they are wrapped modulo-\( N \), are \( (\tilde{H}_{\text{down}})_{nk} = h_{n-2k} \).

To clarify these remarks, we look at some examples. Consider a length-6 filter, such as \( \tilde{H} = \text{convmtx}(h(:), 6) \), where \( h \) is the function \( H \). The wrapped/downsampled convolution matrix \( \tilde{H}_{\text{down}} \) obtained by keeping \( N/2 \) columns, \( \tilde{H}_{\text{down}} = \text{modwrap}(\tilde{H}, 6) \), before they are wrapped modulo-\( N \), typically large, the convolution matrix \( \tilde{H}_{\text{down}} \) can be calculated very simply by its “downsampled” version \( \tilde{H}_{\text{down}} \) obtained by keeping \( N/2 \) columns, \( \tilde{H}_{\text{down}} = \text{modwrap}(\tilde{H}, 6) \).

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 \( \text{convmtx} \) and the function \( \text{modwrap} \):

\[
H = \text{convmtx}(h(:), 6); \quad H_{\text{down}} = \text{modwrap}(H, 6); \quad \tilde{H}_{\text{down}} = \tilde{H}_{\text{down}}\mathbf{x}
\]

Because \( h \) is fairly short and \( N \) typically large, the convolution matrix \( H \) can be defined as sparse. This can be accomplished by replacing \( \text{convmtx} \) by the function \( \text{convmat} \), which we encountered before in Sec. 23.10. Similar convolution matrices \( G_{\text{down}} \) can be constructed for the conjugate mirror filter \( g_n \). The function \( \text{dwtmat} \) constructs both matrices for any scaling filter \( h \) and signal length \( N \) using \( \text{convmat} \):

\[
[H,G] = \text{dwtmat}(h,N); \quad \text{dwtmat}(h,N)
\]

The output matrices \( H,G \) are defined as sparse and have dimension \( N \times (N/2) \). They represent the matrices \( \tilde{H}_{\text{down}}, G_{\text{down}} \).

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

\[
[H_j,G_j] = \text{dwtmat}(h,N_j), \quad j_0 + 1 \leq j \leq J
\]

Then, the analysis and synthesis algorithms are as follows, initialized with \( c_0 = \mathbf{x} \),

\[
\begin{align*}
(DWT) \quad & c_{j-1} = H_j^T c_j \quad j = J, J-1, \ldots, J_0 + 1 \\
(IDWT) \quad & c_j = H_j c_{j-1} + G_j d_{j-1} \quad j = J_0 + 1, J_0 + 2, \ldots, J
\end{align*}
\]

The column vector \( c_j \) has dimension \( N_j = 2^j \), while the vectors \( c_{j-1}, d_{j-1} \) 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:

\[
\mathbf{w} = \begin{bmatrix} c_{J_0} \\ d_{J_0} \\ \vdots \\ d_{J-1} \end{bmatrix}
\]

Its total dimension is \( N = 2^J \), as can be verified easily,

\[
2^{J_0} + 2^{J_0} + (2^{J_0+1} + \cdots + 2^{J-1}) = 2^J
\]
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:

$$
\begin{bmatrix}
  \mathbf{c}_{j-1} \\
  \mathbf{d}_{j-1}
\end{bmatrix} = \begin{bmatrix}
  H_j^T \\
  G_j^T
\end{bmatrix} \mathbf{c}_j \iff \mathbf{c}_j = H_j \mathbf{c}_{j-1} + G_j \mathbf{d}_{j-1} = [H_j, G_j] \begin{bmatrix}
  \mathbf{c}_{j-1} \\
  \mathbf{d}_{j-1}
\end{bmatrix}
$$

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:

$$
H_j^T H_j = G_j^T G_j = I_{N_j/2}, \quad H_j^T G_j = 0, \quad H_j G_j^T + G_j H_j^T = I_{N_j}
$$

(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:

$$
R_k = \sum_n h_n g_{n-2k} \Rightarrow \begin{cases}
  R_0 = h_0^2 + h_1^2 + h_2^2 + h_3^2 + h_4^2 + h_5^2 \\
  R_1 = h_1 h_2 + h_2 h_3 + h_3 h_4 + h_4 h_5 \\
  R_2 = h_2 h_3 + h_3 h_4 + h_4 h_5 + h_5 h_0
\end{cases}
$$

(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:

$$
H_3 = \begin{bmatrix}
  h_0 & h_4 & h_2 \\
  h_1 & h_5 & h_3 \\
  h_2 & h_0 & h_4 \\
  h_3 & h_1 & h_5 \\
  h_4 & h_2 & h_0 \\
  h_5 & h_3 & h_1
\end{bmatrix}, \quad H_2 = \begin{bmatrix}
  h_0 + h_4 & h_2 \\
  h_1 + h_5 & h_3 \\
  h_2 & h_0 + h_4 \\
  h_3 & h_1 + h_5 \\
  h_4 & h_2 & h_0 \\
  h_5 & h_3 & h_1
\end{bmatrix}
$$

(20.5.19)

with similar definitions for $G_j, j = 3, 2, 1$. By explicit multiplication, we can verify:

$$
H_3 H_3^T = G_3^T G_3
$$

Similarly, we have,

$$
H_2 H_2^T = G_2^T G_2
$$

$$
H_1 H_1^T = G_1^T G_1
$$

(20.5.20)

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 $H_j$, 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:

$$
H_{j-1} = \text{dwtwrap}(H_j(:, 1 : N_{j-1}/2)) \quad j = J, J-1, \ldots, J_0 + 1
$$

(20.5.20)

and similarly for $G_{j-1}$. This simple operation has been incorporated into the function dwtwrap, with usage:
20.5. DISCRETE WAVELET TRANSFORM

\[ H_{\text{lower}} = \text{dwtwrap}(H); \]

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, \( 2^j \times 2^j \), it is convenient to use a cell array to store them in MATLAB. The function \( \text{dwtcell} \) constructs and stores them in sparse form:

\[ F = \text{dwtcell}(H,N); \]

with the conventions \( H_j = F[1,j] \) and \( G_j = F[2,j] \), for \( J_0 + 1 \leq j \leq J \), where \( N \) is the highest dimension. The function \( \text{fwtm} \) implements the analysis algorithm in (20.5.15).

Its inputs are the signal vector \( x \), the cell array \( F \), and the lowest desired level \( J_0 \),

\[ w = \text{fwtm}(x,F,J_0); \]

The output \( w \) 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 \( \text{fwtm} \) is extremely fast, even faster than the convolution-based function \( \text{fwt} \) discussed below. The function \( \text{ifwtm} \) implements the inverse DWT synthesis algorithm in (20.5.15),

\[ x = \text{ifwtm}(w,F,J_0); \]

An example is the following MATLAB code using the \( D_1 \) scaling filter:

\[
\begin{align*}
x &= [1 2 3 4 5 6 7 8]; \\
h &= \text{daub}(3); \\
F &= \text{dwtcell}(h); \\
w &= \text{fwtm}(x,F,0); \\
x &= \text{ifwtm}(w,F,0); \\
w &= \text{fwtm}(x,F,1); \\
w &= \text{fwtm}(x,F,2); \\
w &= \text{fwtm}(x,F,3);
\end{align*}
\]

These outputs can be understood by looking at the individual matrix operations. Defining, \( c_1 = x = [1, 2, 3, 4, 5, 6, 7, 8] \), and the level-3 matrices \( H_3, G_3 \), obtained from the call, \( [H_3, G_3] = \text{dwtmat}(h,8) \),

\[
H_3 = \begin{bmatrix}
0.3327 & 0 & -0.0854 & 0.4599 \\
0.8069 & 0 & 0.0352 & -0.1350 \\
0.4599 & 0.3327 & 0 & -0.0854 \\
-0.1350 & 0.8069 & 0 & 0.0352 \\
-0.0854 & 0.4599 & 0.3327 & 0 \\
0.0352 & -0.1350 & 0.8069 & 0 \\
0 & -0.0854 & 0.4599 & 0.3327 \\
0 & 0.0352 & -0.1350 & 0.8069
\end{bmatrix}, 
G_3 = \begin{bmatrix}
0.0352 & 0 & 0.0869 & -0.1350 \\
0.0869 & 0 & 0.0352 & 0 & -0.1350 & 0.0352 & 0 & 0.8069 \\
-0.1350 & 0.8069 & 0 & 0.0352 & 0.0869 & -0.1350 & 0.0352 & 0 \\
0.0854 & 0 & 0.0352 & 0 & -0.1350 & 0.0352 & 0 & 0 \\
0 & 0.0869 & 0.4599 & 0.3327 & 0 & -0.1350 & 0.0352 & 0 \\
0 & 0 & 0.0352 & -0.1350 & 0.8069 & 0 & 0.0854 & 0
\end{bmatrix}
\]

we calculate the level-2 coefficient vectors \( c_2, d_2 \), and the level-2 DWT,

\[
c_2 = H_2^T c_1 = \begin{bmatrix}
2.5702 \\
5.3986 \\
8.6288 \\
8.8583
\end{bmatrix}, 
\quad d_2 = G_2^T c_1 = \begin{bmatrix}
0 \\
0 \\
-3.7938 \\
0.9653
\end{bmatrix}, 
\quad w = \begin{bmatrix}
c_2 \\
d_2
\end{bmatrix}
\]

which agrees with the above MATLAB output of \( \text{fwtm}(x,F,2) \). Then, from the matrices \( H_2, G_2 \), obtained from \([H_2, G_2] = \text{dwtmat}(h,4) \), or, from \( H_2 = \text{dwtwrap}(H_3) \),

\[
H_2 = \begin{bmatrix}
0.2472 & 0.4599 \\
0.8421 & -0.1350 \\
-0.2472 & -0.4599 \\
-0.1350 & 0.8421
\end{bmatrix}, 
G_2 = \begin{bmatrix}
0.8421 & -0.1350 \\
-0.2472 & -0.4599 \\
-0.1350 & 0.8421 \\
-0.4599 & 0.2472
\end{bmatrix}
\]

we calculate the level-1 coefficient vectors \( c_1, d_1 \), and the level-1 DWT,

\[
c_1 = H_1^T c_2 = \begin{bmatrix}
7.9539 \\
10.0461
\end{bmatrix}, 
\quad d_1 = G_1^T c_2 = \begin{bmatrix}
-4.4090 \\
2.2467
\end{bmatrix}, 
\quad w = \begin{bmatrix}
c_1 \\
d_1
\end{bmatrix}
\]

which agrees with the above MATLAB output of \( \text{fwtm}(x,F,1) \). Finally, from the matrices \( H_1, G_1 \), obtained from \([H_1, G_1] = \text{dwtmat}(h,2) \), or, from \( H_1 = \text{dwtwrap}(H_2) \),

\[
H_1 = \begin{bmatrix}
0.7071 \\
0.7071 \\
-0.7071
\end{bmatrix}, 
G_1 = \begin{bmatrix}
0.7071 \\
0.7071
\end{bmatrix}
\]

we find the level-0 coefficient vectors \( c_0, d_0 \), and the level-0 DWT,

\[
c_0 = H_1^T c_1 = 12.7279, 
\quad d_0 = G_1^T c_1 = -1.4794, 
\quad w = \begin{bmatrix}
c_0 \\
d_0 \\
c_1 \\
d_1
\end{bmatrix}
\]

Orthogonal DWT Transformation

The mapping of a length-\( N \) signal vector \( x \) to the length-\( N \) vector \( w \) of wavelet coefficients given in Eq. (20.5.16) is equivalent to an orthogonal matrix transformation, say, \( w = W^T x \), with inverse \( x = W w \), such that \( W^T W = WW^T = I_N \). 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 \( J_0 = 2, 1, 0 \),

\[
W = [H_3, G_3] \\
W = [H_3 H_2, G_3] = [H_3 H_2, H_3 G_2, G_3] \\
W = [H_3 H_2 H_1, G_3] = [H_3 H_2 H_1, H_3 H_2 G_1, H_3 G_2, G_3]
\]

We verify the reconstruction of \( x \) from \( w \) starting at \( J_0 = 0 \),

\[
[H_3 H_2 H_1, H_3 H_2 G_1, H_3 G_2, G_3] \begin{bmatrix}
c_0 \\
d_0 \\
c_1 \\
d_1
\end{bmatrix} = \begin{bmatrix}
H_3 H_2 (H_1 c_0 + G_1 d_0) + H_3 G_2 d_1 + G_3 d_2 \\
H_3 (H_2 c_1 + G_2 d_1) + G_3 d_2 = \\
H_3 c_2 + G_3 d_2 = c_3 = x
\end{bmatrix}
\]

\]
The construction of $W$ can be carried out with the following very simple recursive algorithm, stated in MATLAB notation,

$$W = I_N, \quad (N = 2^J)$$

for $j = J, J-1, \ldots, J_0 + 1$,  

$$W(:, 1 : 2^j) = W(:, 1 : 2^j) [H_j, G_j]$$  \hspace{1cm} (20.5.21)

The algorithm updates the first $2^j$ columns of $W$ at each level $j$. The MATLAB function 

\[
W = \text{fwtmat}(h, N, J_0);
\]

\[
\text{w} = \text{fwtmat}(h, 8, 0);
\]

\[
\text{w} = \text{w}' * \text{x}; \quad \text{gives w} = \text{[12.7279, -1.4794, -4.4090, 2.2467, 0, 0, -3.7938, 0.9653]'}
\]

which generates the same DWT as the example above:

$$x = [1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8]' \ ; \ h = \text{daub}(3) \ ; \ W = \text{fwtmat}(h, 8, 0); \ \text{w} = \text{w}' * \text{x}; \quad \text{gives w} = \text{[12.7279, -1.4794, -4.4090, 2.2467, 0, 0, -3.7938, 0.9653]'}
$$

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:

\[
h = \text{daub}(3); \ N = 64; \ W = \text{fwtmat}(h, N, 0); \ \text{spy}(W); \ \text{percent nonzero} = 100 * \text{nnz}(W) / N^2
\]

The percentages of nonzero entries were 30.5% for $N = 64$, and 6.7% for $N = 512$.

![Sparsity patterns of DWT matrices.](image)

**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 $h_\mu, g_\mu$ of the analysis algorithm by their reversed versions, which are delayed by the filter order $M$ to make them causal, that is, $h_\mu^R = h_{\mu-M} = h_{M-\mu}$, or in the $z$-domain

$$H^R(z) = z^{-M} H(z) = z^{-M} H(z^1).$$

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 $H(z)$ is equivalent to advancing the input and then filtering by $H^R(z)$. In the $z$-domain,

$$Y(z) = H(z) X(z) = z^M H^R(z) X(z) = H^R(z) [z^M X(z)]$$

With these changes, Eq. (20.5.9) now reads,

\[
c_j, -1 = (\text{circonv}(h^R, c_j))_{\text{down}} \quad j = J, J-1, \ldots, J_0 + 1
\]

\[
d_{j+1} = (\text{circonv}(g^R, c_{j+1}))_{\text{down}}
\]

\[
\text{d}_j = \text{circonv}(h,c_{j-1}) + \text{circonv}(g,c_{j+1}) \quad j = J_0 + 1, J_0 + 2, \ldots, J
\]

The concrete MATLAB implementation for computing the forward DWT is:

\[
g = \text{caef}(h); \quad \% \text{conjugate mirror of } h
\]

\[
hR = \text{flip}(h); \quad \% \text{reversed } h
\]

\[
gR = \text{flip}(g); \quad \% \text{filter order}
\]

\[
c = x(:, 1); \quad \% \text{initial smooth, } x \text{ has length } 2^J
\]

\[
w = [\;]; \quad \% \text{DWT coefficient vector}
\]

\[
\text{for } j=J:J-1:J_0+1,
\]

\[
c = \text{advance}(c, M); \quad \% \text{loop from finest down to coarsest level}
\]

\[
d = \text{dn2}(\text{circonv}(gR, c, 2^j)); \quad \% \text{length}(c) = 2^j
\]

\[
c = \text{dn2}(\text{circonv}(hR, c, 2^j));
\]

\[
w = [d; \; w]; \quad \% \text{prepend detail } d \text{ to previous details}
\]

\[
w = [c; \; w]; \quad \% \text{prepend last smooth}
\]

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 $w$. Similarly, the inverse DWT can be calculated by the loop:

\[
w = w(:, 1); \quad \% \text{work columnwise}
\]

\[
c = \text{wcoeff}(w, J_0); \quad \% \text{coarsest smooth at level } J_0
\]

\[
\text{for } j=J_0+1:J,
\]

\[
c = \text{circonv}(h, up2(c), 2^j) + \text{circonv}(g, up2(d), 2^j); \quad \% \text{output } c \text{ is } 2^j \text{-dimensional}
\]

\[
x = c; \quad \% \text{reconstructed } x
\]
Here, the function \texttt{wcoeff}(w, J_0, j-1) extracts the subvector \(d_{j-1}\) from the wavelet transform vector \(w\), and the function \texttt{up2} upsamples by a factor of two.

The MATLAB functions \texttt{fwt} and \texttt{iwt} incorporate the above code segments to realize the convolutional forms of the DWT and IDWT:

\[
\begin{align*}
  w & = \texttt{fwt}(x, h, J_0) ; & \quad & \text{\small fast wavelet transform} \smallcr
  x & = \texttt{iwt}(w, h, J_0) ; & \quad & \text{\small inverse fast wavelet transform}
\end{align*}
\]

Some examples are,

\[
\begin{align*}
  x & = [1 \, 2 \, 3 \, 4 \, 5 \, 6 \, 7 \, 8] ; \quad & \quad & h = \texttt{daub}(3) ; \smallcr
  w & = \texttt{fwt}(x, h, 0) ; & \quad & \% w = [12.7279, -1.4794, -4.4090, 2.2467, 0, 0, -3.7938, 0.9653] \smallcr
  x & = \texttt{iwt}(w, h, 0) ; & \quad & \% \text{returns } x = [1, 2, 3, 4, 5, 6, 7, 8] \smallcr
  w & = \texttt{fwt}(x, h, 3) ; & \quad & \% w = [7.9539, 10.0461, -4.4090, 2.2467, 0, 0, -3.7938, 0.9653] \smallcr
  w & = \texttt{fwt}(x, h, 2) ; & \quad & \% w = [2.5702, 5.3986, 8.6288, 8.8583, 0, 0, -3.7938, 0.9653] \smallcr
\end{align*}
\]

A second optional output of \texttt{fwt} (and \texttt{fwtm}) is the \(N \times (J_0 + 1)\) matrix \(V\) whose columns are the sub-blocks of \(w\) according to their resolution,

\[
\begin{align*}
  w & = \begin{bmatrix}
  c_{j_0} \\
  d_{j_0} \\
  \vdots \\
  d_{j_0+1}
\end{bmatrix} \Rightarrow
  V = \begin{bmatrix}
  c_{j_0} & 0 & 0 & \cdots & 0 \\
  0 & d_{j_0} & 0 & \cdots & 0 \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  0 & 0 & 0 & \cdots & d_{j_0+1}
\end{bmatrix}
\end{align*}
\]

It is obtained by the calls,

\[
\begin{align*}
  [w, V] & = \texttt{fwt}(x, h, J_0) ; \smallcr
  [w, V] & = \texttt{fwtm}(x, h, J_0) ;
\end{align*}
\]

The computations in \texttt{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 \((x = [1, 2, 3, 4, 5, 6, 7, 8])\) with \(D_3\) and \(J_0 = 0\) from the packages:

\[
\begin{align*}
  w & = [12.7279, -1.4794, -4.4090, 2.2467, 0, 0, 0, -3.7938, 0.9653] = \texttt{fwt} - \texttt{ours} \smallcr
  w & = [12.7279, 1.4794, 4.4090, -2.2467, 3.7938, -0.9653, 0, 0, 0] = \texttt{Waveshell50, Ref. \cite{584}} \smallcr
  w & = [12.7279, -1.4794, -4.4090, 2.2467, 3.7938, 0.9653, 0, 0, 0] = \texttt{Waveshell, Ref. \cite{588}} \smallcr
  w & = [12.7279, 1.4794, -2.2467, 4.4090, -0.9653, 3.7938, 0, 0, 0] = \texttt{WMTSA, Ref. \cite{592}} \smallcr
  w & = [12.7279, -1.4794, 2.2467, -4.4090, 0.9653, 0, 0, 0, -3.7938] = \texttt{UniWave, Ref. \cite{593}} \smallcr
  w & = [12.7279, 1.4794, -4.4090, 2.2467, 0, 0, 0, 0, -3.7938, -0.9653] = \texttt{Getz, Ref. \cite{594}} \smallcr
  w & = [12.7279, -1.4794, -4.4090, 2.2467, 0, 0, 0, 0, -3.7938, 0.9653] = \texttt{Wavekit, Ref. \cite{595}}
\end{align*}
\]

\[20.6 \quad \text{Multiresolution Decomposition}\]

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 \texttt{dwtdec},

\[
f(t) = \sum_{j=0}^{J_0} c_{j_0} \psi_{j_0} (t) = \sum_{j=0}^{J_0} \sum_{n} c_{j_0 n} \psi_{j_0 n}(t) + \sum_{j=0}^{J_0} \sum_{n} d_{j n} \psi_{j n}(t) , \quad (20.6.1)
\]

can be given a vectorial interpretation. Let \(x\) 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, \(w = W^T x\), and inverse \(x = W w\).

Writing the DWT \(w\) in the partitioned form of Eq. (20.5.23), we may write \(x\) as the sum of multiresolution components, corresponding to the terms of (20.6.1), with each term representing the part of \(x\) arising from a particular level \(j\) with all the other levels having zero coefficients:

\[
x = W \begin{bmatrix}
  c_{j_0} \\
  d_{j_0} \\
  \vdots \\
  d_{j_0+1}
\end{bmatrix} = W \begin{bmatrix}
  c_{j_0} \\
  0 \\
  \vdots \\
  0
\end{bmatrix} + W \begin{bmatrix}
  0 \\
  d_{j_0} \\
  \vdots \\
  d_{j_0+1}
\end{bmatrix} + \cdots + W \begin{bmatrix}
  0 \\
  0 \\
  \vdots \\
  0
\end{bmatrix}
\]

\[
= x_{j_0} + (x_{j_0} + x_{j_0+1} + \cdots + x_{j-1})
\]

\[
x_j = x_{j_0} + x_{j+1} \quad (20.6.2)
\]

The terms \(x_{j_0}, x_{j+1}\) represent the two parts of \(x\) lying in the subspaces \(V_{j_0}\) and \(V_{j+1}\). The individual terms of \(x_j = x_{j_0} + x_{j+1} + \cdots + x_{j-1}\) contain all the details for levels \(j_0 \leq j \leq J - 1\). The various components are mutually orthogonal, as follows from the property \(W W^T = I\), and the non-overlapping of the sub-blocks of \(w\),

\[
x_j, x_j = 0, \quad j_0 \leq j \leq J - 1
\]

\[
x_j, x_j = 0, \quad j_0 \leq i, j \leq J - 1, \quad i \neq j \quad (20.6.3)
\]

For the “diagonal” terms, we obtain the norms, again following from \(W W^T = I\),

\[
\|x_j\|^2 = \|c_{j_0}\|^2, \quad \|x_j\|^2 = \|d_{j_0}\|^2, \quad j_0 \leq j \leq J - 1
\]

where \(\|x\|^2 = x^T x\), which lead to the sum,

\[
\|x\|^2 = \|x_{j_0}\|^2 + \sum_{j=j_0}^{J-1} \|x_j\|^2 + \sum_{j=j_0}^{J-1} \|d_{j_0}\|^2 = \|w\|^2 \quad (20.6.5)
\]

The \(N \times (J - J_0 + 1)\) matrix \(X = [x_{j_0}, x_{j_0+1} \cdots x_{j-1}]\) incorporates the individual orthogonal columns and is produced as the output of the MATLAB function \texttt{dwtdec},
20.7. WAVELET DENOISING

\[ X = \text{dwtdec}(x,h,J0); \]

% 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 \( \text{dwtdec} \), we have for \( x = [1, 2, 3, 4, 5, 6, 7, 8]^\top \) and \( D_3 \),

\[
\begin{bmatrix}
1 & 12.7279 \\
2 & -1.4794 \\
3 & -4.4090 \\
4 & 2.2467 \\
5 & 0 \\
6 & 0 \\
7 & -3.7938 \\
8 & 0.9653 \\
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
w1 & 12.7279 \\
w2 & -1.4794 \\
w3 & -4.4090 \\
w4 & 2.2467 \\
w5 & 0 \\
w6 & 0 \\
w7 & -3.7938 \\
w8 & 0.9653 \\
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
X1 & 4.5000 & 0.5631 & -0.8716 & -3.1915 \\
X2 & 4.5000 & 0.0337 & -3.3518 & 0.8181 \\
X3 & 4.5000 & -0.3251 & -1.9538 & 0.7789 \\
X4 & 4.5000 & -0.8188 & 0.6399 & -0.3211 \\
X5 & 4.5000 & -0.5631 & 1.1967 & -0.1336 \\
X6 & 4.5000 & -0.0337 & 1.8578 & -0.3241 \\
X7 & 4.5000 & 0.3251 & 1.6287 & 0.5462 \\
X8 & 4.5000 & 0.8188 & 0.8541 & 1.8271 \\
\end{bmatrix}
\]

generated with the MATLAB code and test,

\[
h = \text{daub}(3); x = [1 2 3 4 5 6 7 8]'; X = \text{dwtdec}(x,h,0); [w,V] = \text{fwt}(x,h,0); W = \text{fwtmat}(h,8,0); \text{norm}(X-W*V)
\]

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 \textit{wdenoise}:

\[
F = \text{inline}('1./(1 + abs(x)).^4');
\]

% bumps function

\[
N = 2048; t = (0:N-1)'/N; x = \text{zeros(size}(t)) ;
\]

% normalize time to \( 0 \leq t \leq 1 \)

\[
t0 = [10 13 15 23 25 40 44 65 76 78 81]/100;
\]

% signal parameters

\[
a = [40,50,30,40,50,42,21,43,31,12] * 1.0523 ;
\]

w = [5,5,6,10,30,10,10,5,8,5]/1000;

\[
\text{for } i=1:\text{length}(a),
\text{x} = \text{x} + a(i) * F(t-t0(i))/w(i));\text{end}
\]

% construct noise-free signal

\[
\text{seed} = 2009; \text{randn('state',seed); } v = \text{randn(size}(t));
\]

% generate noise

\[
y = x + v ;
\]

% noisy signal with SNR, \( \sigma_x/\sigma_v = 7 \)

\[
h = \text{daub}(8,2); J0=5; \text{type}=1; \]

% use Symmlet-8 and soft thresholding

\[
\text{xd} = \text{wdenoise}(y,h,J0,\text{type});
\]

% wavelet denoising

\[
\text{figure; plot}(t,y,'-''); \text{figure; plot}(t,x,'-'); \text{figure; plot}(t,xd,'r-');
\]

% top row

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:

\[
\begin{aligned}
X & \xrightarrow{\text{DWT}} W & \xrightarrow{\text{thresh}} W_{\text{thr}} & \xrightarrow{\text{IDWT}} X_{\text{thr}}
\end{aligned}
\]

Given a wavelet coefficient \( d \), we denote the thresholding operation by \( d_{\text{thr}} = f(d, \lambda) \).
where $\lambda$ 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:

\[
\begin{align*}
\hat{d}_{\text{thr}} &= f(d, \lambda) = d u(|d| - \lambda) & \text{(hard)} \\
\hat{d}_{\text{thr}} &= f(d, \lambda) = \text{sign}(d) (|d| - \lambda) u(|d| - \lambda) & \text{(soft)}
\end{align*}
\]

(20.7.1)

or, equivalently,

\[
\begin{align*}
\hat{d}_{\text{thr}}^{\text{hard}} &= \begin{cases} 
  d, & |d| \geq \lambda \\
  0, & |d| < \lambda
\end{cases}, & \hat{d}_{\text{thr}}^{\text{soft}} &= \begin{cases} 
  d - \text{sign}(d) \lambda, & |d| \geq \lambda \\
  0, & |d| < \lambda
\end{cases}
\end{align*}
\]

(20.7.2)

The simplest possibility is to use the same threshold for all levels. Donoho & Johnstone [569] suggest the following “universal” threshold,

\[
\lambda = \sigma \sqrt{2 \log N} \quad \text{(universal threshold)}
\]

(20.7.3)

where $\sigma^2$ is the variance of the additive noise in the data. Since $\sigma$ is not known, it can be estimated from the wavelet detail coefficients $d_{j-1}$ at level $J-1$, which for a smooth desired signal are presumably dominated mostly by the noise component. The vector $\hat{d} = d_{J-1}$ has length $N/2 = 2^J - 1$ and one may estimate $\sigma$ by using either the standard deviation of $\hat{d}$, or its mean-absolute-deviation (MAD), that is,

\[
\hat{\sigma} = \text{std}(\hat{d}), \quad \hat{\sigma} = \frac{\text{mad}(\hat{d})}{0.6745} = \frac{\text{median}(|\hat{d} - \text{median}(\hat{d})|)}{0.6745}
\]

(20.7.4)

where the factor 0.6745 arises from the implicit assumption that $\hat{d}$ is a vector of zero-mean independent normally-distributed components (for a zero-mean, unit-variance, gaussian random variable $x$, one has the relationship, median$(|x|) = 0.6745$). Donoho & Johnstone’s [569] so-called VisuShrink method uses the universal threshold with the MAD estimate of $\sigma$ 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 $y$ and $w = W^T y$ be the noisy data vector and its DWT, and let $\hat{x}$ and $\hat{w} = W^T \hat{y}$ be the sought estimate and its DWT of the desired signal component $x$ in the noisy signal model $y = x + v$. An estimation criterion similar to the smoothing spline and reproducing kernel criteria that we considered earlier is the following performance index,

\[
\mathcal{J} = \|y - \hat{x}\|^2 + P(\hat{x}) = \min
\]

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 $\|y - \hat{x}\|^2 = \|w - \hat{w}\|^2$. Therefore, with a redefinition of $P$, we may replace the above criterion with one that is formulated in the wavelet domain:

\[
\mathcal{J} = \|w - \hat{w}\|^2 + P(\hat{w}) = \|c_{J_0} - \hat{c}_{J_0}\|^2 + \sum_{j=J_0}^{J-1} \|d_j - \hat{d}_j\|^2 + P(d_{J_0}, \ldots, d_{J-1}) = \min
\]

(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:

\[
\mathcal{J} = \|c_{J_0} - \hat{c}_{J_0}\|^2 + \sum_{j=J_0}^{J-1} \|d_j - \hat{d}_j\|^2 + 2 \lambda \sum_{j=J_0}^{J-1} \sum_{n=0}^{N_j-1} |\hat{d}_{jn}| = \min
\]

(20.7.6)

where $N_j = 2^j$ is the dimension of the vector $d_j$. The minimization with respect to $c_{J_0}$ gives $\hat{c}_{J_0} = c_{J_0}$. Since the $d_{J_0}$ terms are decoupled, their minimization can be carried on a typical such term, that is, with the simple scalar criterion:

\[
\mathcal{J} = |d - \hat{d}|^2 + 2 \lambda |\hat{d}| = \min
\]

(20.7.7)

whose solution is the soft-thresholding rule,

\[
\hat{d} = \begin{cases} 
  d - \text{sign}(d) \lambda, & |d| \geq \lambda \\
  0, & |d| < \lambda
\end{cases}
\]

(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 $H(\omega)\tilde{H}(2\omega)\tilde{H}(4\omega)$, where $\omega = 2\pi f_s / f_s$, with $f_s$ being the sampling rate at the finest scale. This frequency property is preserved whether the output is undecimated, as in
20.8. UNDECIMATED WAVELET TRANSFORM

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.](image)

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:

\[
\frac{1}{2} \left[ H(\omega)H(\omega) + G(\omega)G(\omega) \right] = 1
\]

(20.8.1)

where \( H(\omega) = H^*(\omega) \) denotes the frequency response of the time-reversed filter \( \tilde{h}_n \).

This relationship admits a block diagram realization as shown in Fig. 20.8.2.

![Fig. 20.8.2 Analysis and synthesis of single stage.](image)

Because it is an identity in \( \omega \), 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.](image)

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,

\[
h^{[r]}(k) = \sum_n h(n) \delta(k - 2^n) \quad \Leftrightarrow \quad H^{[r]}(\omega) = H(2^r\omega)
\]

(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:

\[
a_{j-1} = \tilde{h}^{[J-j]} * a_j \quad J \geq j \geq J_0 + 1, \quad \text{(analysis)}
\]

\[
b_{j-1} = \tilde{g}^{[J-j]} * b_j \quad J_0 + 1 \leq j \leq J, \quad \text{(synthesis)}
\]

(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 \( a_{J_0}(n) = x(n) \). Different à trous filters are used in each stage, unlike the DWT that uses the same filters \( h, g \). 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 \( x \). The à trous filters \( h^{[r]}, g^{[r]} \) can be represented by \( N \times N \) matrices \( H_r, G_r \), which are the ordinary convolution matrices of \( h^{[r]}, g^{[r]} \) reduced modulo-\( N \) column-wise. Similarly, the time-reversed filters \( \tilde{h}^{[r]}, \tilde{g}^{[r]} \) will be represented by the transposed matrices \( H_r^T, G_r^T \). The construction of these matrices is straightforward, for example,

\[
h = \text{h (:); } g = \text{cnnf(h);}
\]

\[
h_r = \text{upr(h, r); } g_r = \text{upr(g, r);}
\]

\[
h_{r} = \text{convmtx(hr, N); } g_{r} = \text{convmtx(gr, N);}
\]

\[
h_{r} = \text{modwrap(Hr, N); } g_{r} = \text{modwrap(Gr, N);}
\]

These steps have been incorporated into the function `uwtmat`, except the function `convmat` is used in place of `convmtx` to make the matrices sparse:
The matrices $H_r, G_r$ satisfy the matrix version of Eq. (20.8.1):

$$\frac{1}{2} \begin{bmatrix} H_r H_r^T + G_r G_r^T \end{bmatrix} = I_N$$  \hspace{1cm} (20.8.4)

The concrete matrix realization of the UWT can be stated then as follows:

$$\begin{bmatrix} a_{j-1} = H_{j-j} a_j \\ b_{j-1} = G_{j-j} b_j \end{bmatrix} \quad J \geq j \geq J_0 + 1, \quad \text{(analysis)}$$  \hspace{1cm} (20.8.5)

$$\begin{bmatrix} a_j = \frac{1}{2} \left[ H_{j-j} a_{j-1} + G_{j-j} b_{j-1} \right] \end{bmatrix} \quad J_0 + 1 \leq j \leq J, \quad \text{(synthesis)}$$

where all the vectors are $N$-dimensional, initialized at $a_f = x$. 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:

$$U = [a_{J_0}, b_{J_0}, b_{J_0+1}, \ldots, b_{J-1}] \quad \text{(UWT)}$$  \hspace{1cm} (20.8.6)

The functions $\text{uwtm}$ and $\text{iuwtm}$ implement the algorithms in Eq. (20.8.5):

- $U = \text{uwtm}(x,h,J_0);$ \quad $\%$ UWT in matrix form
- $x = \text{iuwtm}(U,h);$ \quad $\%$ inverse UWT in matrix form

An example is as follows:

```matlab
h = daub(3); x = [1 2 3 4 5 6 7 8]';
J0=0; U = uwtm(x,h,J0); \% or set $J_0 = 1$ and $J_0 = 2$
xinv = iuwtm(U,h); norm(x-xinv)
```

The following MATLAB code shows a possible implementation of the analysis part:

```matlab
M=length(h)-1;
g=cmf(h); hr=flip(h); gr=flip(g);
a = x;
for r=0:J-J0-1,
a = advance(a, 2^r*M);
hr = upr(hr,r);
gr = upr(gr,r);
a = circonv(gr,r,a);
b = circonv(hr,r,a);
end
U = [b,a];
```

The functions $\text{uwt}$ and $\text{iuwt}$ 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 $\text{fwtm}$. The functions $\text{uwt}$ and $\text{iuwt}$ are much faster versions that produce the same results and are implemented using circular convolutions:

- $U = \text{uwt}(x,h,J0);$ \quad $\%$ UWT in convolutional form
- $x = \text{iuwt}(U,h);$ \quad $\%$ inverse UWT

The functions $\text{uwtm}$ and $\text{iuwtm}$ 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 $\text{fwtm}$. The functions $\text{uwt}$ and $\text{iuwt}$ are much faster versions that produce the same results and are implemented using circular convolutions:
20.8. UNDECIMATED WAVELET TRANSFORM

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 all stages of the analysis and synthesis filter banks in Fig. 20.8.3 operate
filters rather than the time-reversed ones. Although this algorithm works, it is wasteful
because the à trous filters have length \(2^r(M+1)\) consisting mostly of zeros and
only \((M+1)\) nonzero coefficients, where \(M\) is the filter order of \(h\). 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:

\[
y(n) = \sum_{k} h^r (k) x(n-k) = \sum_{m} h_{m} x(n-2^r m)
\]

\[
\hat{y}(n)_{\text{mod-}N} = \sum_{p} y(n+pN) = \sum_{p,m} h_{m} x(n+pN-2^r m)
\]

We assume that \(N = 2^L\) and that the à trous factor is such that \(r \leq J\), 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 \(0 \leq k \leq 2^r - 1,\) we may define the
\(k\)-th sub-block input and output signals:

\[
x_k(i) = x(2^r i + k), \quad y_k(i) = y(2^r i + k), \quad 0 \leq k \leq 2^r - 1
\]

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:

\[
y_k(i) = y(2^r i + k) = \sum_{m} h_m x(2^r i + k - 2^r m) = \sum_{m} h_m x_k(i - m)
\]

\[
\hat{y}(2^r i + k)_{\text{mod-}N} = \sum_{p,m} h_{m} x(2^r i + k + 2^r pL - 2^r m) = \sum_{p,m} h_{m} x_k(i + pL - m)
\]

These operations have been incorporated into the MATLAB function convat,

\[
y = \text{convat}(h,x,r);
\]

which is equivalent to the mod-\(N\) operation, where \(N\) is the length of \(x:\)

\[
y = \text{circonv}(\text{upr}(h,r), x, N);
\]

The essential part of the function uwt is then,

\[
\text{uwt}(x) = \left[\begin{array}{c}
\text{upr}(h_{\text{adj}}, x, J) @ \text{upr}(h_{\text{adj}}, x, J-1) @ \ldots @ \text{upr}(h_{\text{adj}}, x, 1) @ x
\end{array}\right]
\]

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 \(J_0 = 0:\)

\[
x = \begin{bmatrix}
1 \\
2 \\
3 \\
4 \\
5 \\
6 \\
7 \\
8
\end{bmatrix}, \quad \Rightarrow \quad w = \begin{bmatrix}
12.7279 \\
1.4794 \\
-4.4090 \\
2.4677 \\
3.2453 \\
-4.5243 \\
5.4022 \\
-4.7674
\end{bmatrix}, \quad \Rightarrow \quad U = \begin{bmatrix}
12.7279 \\
1.4794 \\
-4.4090 \\
2.4677 \\
3.2453 \\
-4.5243 \\
5.4022 \\
-4.7674
\end{bmatrix}
\]

\[
x = \begin{bmatrix}
6 \\
7 \\
8 \\
9 \\
10 \\
11 \\
12 \\
13
\end{bmatrix}, \quad \Rightarrow \quad w = \begin{bmatrix}
12.7279 \\
1.4794 \\
-4.4090 \\
2.4677 \\
3.2453 \\
-4.5243 \\
5.4022 \\
-4.7674
\end{bmatrix}, \quad \Rightarrow \quad U = \begin{bmatrix}
12.7279 \\
1.4794 \\
-4.4090 \\
2.4677 \\
3.2453 \\
-4.5243 \\
5.4022 \\
-4.7674
\end{bmatrix}
\]

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 \(U = [a_{j_0}, b_{j_0}, b_{j_0+1}, \ldots, b_{j-1}],\) and
one output, the signal \(x.\) The UWT multiresolution decomposition resolves \(x\) into com-
ponents arising from the individual inputs when the other inputs are zero:

\[
U = [a_{j_0}, b_{j_0}, b_{j_0+1}, \ldots, b_{j-1}] \overset{\text{UWT}}{\Rightarrow} x
\]

\[
= [a_{j_0}, 0, 0, \ldots, 0] \overset{\text{UWT}}{\Rightarrow} [x_{j_0}
\]

\[
+ [0, b_{j_0}, 0, \ldots, 0] \overset{\text{UWT}}{\Rightarrow} [x_{j_0}]
\]

\[
+ [0, 0, b_{j_0+1}, 0, \ldots, 0] \overset{\text{UWT}}{\Rightarrow} [x_{j_0+1}
\]

\[
\ldots
\]

\[
+ [0, 0, 0, \ldots, b_{j-1}] \overset{\text{UWT}}{\Rightarrow} [x_{j-1}]
\]

so that we have the sum,

\[
x = x_{j_0} + x_{j_0} + x_{j_0+1} + \ldots + x_{j-1}
\]  

(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:

\[ X = \text{uwtdec}(x,h,J0); \]

where \( X \) consists of the columns, \( X = [X_{J0}, \bar{x}_{J0}, \bar{x}_{J0+1}, \ldots, \bar{x}_{J-1}] \).

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 is a subset of the dataset that we used repeatedly in Ch.9 of [45].

![UWT decomposition of monthly housing data, using \( S_8 \) with \( J_0 = 8 \) and \( J_0 = 5 \).](image)

The upper left graph shows the smooth component arising from the UWT coefficients \( a_{J0} \). The remaining graphs arise from the detail coefficients. \( b_j, J_0 \leq j \leq J - 1 \). 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:

```matlab
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
```

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 \( b_j \) of \( U \) get thresholded by a possibly level-dependent threshold and the inverse UWT is constructed. The procedure is depicted below:

\[ x_{\text{UWT}} \rightarrow U = [a_{J0}, b_{J0}, \ldots, b_{J-1}] \rightarrow \text{thresh} \rightarrow U_{\text{thr}} = [a_{J0}, b_{\text{thr}}_{J0}, \ldots, b_{\text{thr}}_{J-1}] \rightarrow \text{IUWT} \rightarrow x_{\text{thr}} \]

The MATLAB function `wduwt` implements this denoising procedure using the universal threshold (20.7.3) and soft or hard thresholding:

\[ y = \text{wduwt}(x,h,J0,\text{type}); \quad \text{wavelet denoising with UWT} \]

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:

```matlab
F = inline('1./(1 + abs(x)).^4'); % bumps function
N = 2048; t = (0:N-1)'/N; x = zeros(size(t)); % normalize time to 0 \leq t \leq 1
v0 = [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),
    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;
```

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.
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

------

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:

```matlab
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, \( J_{max} \leq J-1 \), 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,
20.10. PROBLEMS

including a plot of the UWT wavelet coefficients. The MATLAB code used to generate the four graphs was as follows:

```matlab
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
Xwtd = 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(Xwtd,'xd'); figure; plotdec(V,'wd'); % upper graphs
figure; plotdec(Xuwt,'xd'); figure; plotdec(U,'wd'); % lower graphs
```

Show that $z_1$ must 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 transfer function has the following form with $K = 3$ zeros at Nyquist:

$$H(z) = h_0 (1 + z^{-1})^3 (1 - (a + jb) z^{-1}) (1 - (a - jb) z^{-1})$$

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

$$b = \sqrt{\frac{a + a^2 - 3a^3}{3 - a}}$$

and that $a$ is obtained as the following solution of the quartic equation:

$$12a^4 - 72a^3 + 152a^2 - 132a + 27 = 0 \Rightarrow a = \frac{3}{2} - \frac{1}{6} \sqrt{15 + 12\sqrt{10}}$$

Verify that the zero $z_1 = a + jb$ coincides with that in Eq. (20.3.18).

20.3 Prove the downsampling replication property (20.4.11) by working backwards, that is, start from the Fourier transform expression and show that

$$\frac{1}{L} \sum_{m=0}^{L-1} X(f - m \frac{fs}{L}) = \sum_k s(k) x(k) e^{-2\pi jfk/L} = \sum_k x(k) e^{-2\pi jfk/L} = Y_{down}(f)$$

where $s(k)$ is the periodic “sampling function” with the following representations:

$$s(k) = \frac{1}{L} \sum_{m=0}^{L-1} e^{-2\pi jkm/L} = \frac{1 - e^{-2\pi jk}}{L (1 - e^{-2\pi jk/L})} = \sum_n \delta(k - nL)$$

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:

$$s(k) = [\ldots, 1, 0, 0, \ldots, 0, 1, 0, 0, \ldots, 0, 1, 0, 0, \ldots, 0, \ldots] = \sum_n \delta(k - nL)$$

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:

$$d_{\text{thr}} = f(d, \lambda, a) = \frac{|d|^a}{|d|^a + \lambda^a}, \quad a > 0, \lambda > 0$$

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):

$$H(z) = h_0 (1 + z^{-1})^2 (1 - z_1 z^{-1})$$

### 20. DISCRETE WAVELET TRANSFORMS

showing that $z_1$ is 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).
21. DISCRETIZATION METHODS

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 \texttt{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 \texttt{c2dm} and \texttt{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 \texttt{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, \texttt{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 are characterized by a \textit{transfer function}, \( h(t) \), such that the input/output equation is given by continuous-time convolution,

\[
\int_{-\infty}^{\infty} h(t-t')x(t-t')dt', \quad x(t) \rightarrow \mathcal{H} \rightarrow y(t) \quad (21.1.1)
\]

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),

\[
y(t) = \int_{0}^{\infty} h(t-t')x(t')dt', \quad t \geq 0 \quad (21.1.2)
\]

Essentially all practical continuous-time LTI systems are a subset of the above, and are described by \textit{linear constant-coefficient differential equations} (LCCDEs) derived from a physical description of the system. This implies that the impulse response \( h(t) \), \( x(t) \) also satisfy an LCCDE, but with an impulsive input, \( \delta(t) \).

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 \( H(s) \),

\[
\ddot{y}(t) + a_1 \dot{y}(t) + a_2 y(t) = b_0 \dot{x}(t) + b_1 x(t) + b_2 x(t) \quad (21.1.3)
\]

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,

\[
\dot{y}(t) + a_1 y(t) = b_0 \dot{x}(t) + b_1 x(t) \quad (21.1.4)
\]

where

\[
H(s) = \frac{Y(s)}{X(s)} = \frac{b_0s + b_1}{s + a_1}
\]

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, \( y(0^-), \overset{.}{y}(0^-) \), specified at \( t = 0^- \). Alternatively, as is done in the
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,

\[
\begin{align*}
1. & \quad y(0^+) = y(0^-) + b_0 x(0^-) \\
2. & \quad y'(0^+) = y'(0^-) + b_0 x'(0^-) + (b_1 - a_1 b_0) x(0^-) \\
3. & \quad y''(0^+) = y''(0^-) + b_0 x''(0^-) + (b_1 - a_1 b_0) x'(0^-) + (b_2 - a_2 b_0 - a_1 (b_1 - a_1 b_0)) x(0^-)
\end{align*}
\]

(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,

\[
y'(t) = \begin{cases} 
\dot{y}_{zi}(t) & \text{zero-input IC} \ (0^-) \neq 0 \\
\dot{y}_{zs}(t) & \text{zero-state IC} \ (0^-) = 0 \\
0 & \text{homogeneous IC} \ (0^-) \neq 0 \\
g(t) \ast x(t) & \text{forced response IC} \ (0^-) = 0 \\
\end{cases} \quad y(t) = \begin{cases} 
\dot{y}_{zi}(t) & \text{homogeneous IC} \ (0^-) \neq 0 \\
\dot{y}_{zs}(t) & \text{forced response IC} \ (0^-) = 0 \\
\end{cases}
\]

(21.2.2)

The mapping between the two sets of conditions at \( t = 0^- \) to those at \( t = 0^+ \) is as follows, up to order 3,

\[
\begin{align*}
1. & \quad y(0^+) = y(0^-) + b_0 x(0^-) \\
2. & \quad y'(0^+) = y'(0^-) + b_0 x'(0^-) + (b_1 - a_1 b_0) x(0^-) \\
3. & \quad y''(0^+) = y''(0^-) + b_0 x''(0^-) + (b_1 - a_1 b_0) x'(0^-) + (b_2 - a_2 b_0 - a_1 (b_1 - a_1 b_0)) x(0^-)
\end{align*}
\]

(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,
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,

\[
\begin{align*}
\dot{y}_z(0^-) &= 0, \quad y_z(0^-) = b_0 x(0^+) \\
\dot{y}_z(0^-) &= 0, \quad y_z(0^-) = b_0 \dot{x}(0^+) + (b_1 - a_1 b_0) x(0^+) \\
\ddot{y}_z(0^-) &= 0, \quad \ddot{y}_z(0^-) = b_0 \ddot{x}(0^+) + (b_1 - a_1 b_0) \dot{x}(0^+) + (b_2 - a_2 b_0 - a_1 (b_1 - a_1 b_0)) x(0^+).
\end{align*}
\]  
\[(21.2.7)\]

It follows that the zero-state component \( y_z(t) \) can be determined by three methods:

(i) convolution, \( y_z(t) = h(t) \ast x(t) \)

(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.\(^\dagger\) Consider a 1st order, or a 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,

\[
H(s) = \frac{b_0 s + b_1}{s + a_1}, \quad H(s) = \frac{b_0 s^2 + b_1 s + b_2}{s^2 + a_1 s + a_2} = \frac{b_0 s^2 + b_1 s + b_2}{(s - p_1)(s - p_2)}.
\]

Then, the forced response will have the following forms for exponential, sinusoidal, step, ramp, or quadratic input signals,

\[
\begin{align*}
x(t) &\quad y_f(t) &\quad \text{input type} \\
e^{-pt} &\quad B e^{pt} &\quad \text{exponential, } p \neq p_i, \quad B = H(p) \\
e^{pt} &\quad B_1 e^{pt} &\quad \text{exponential, } p = p_i, \quad B_1 \equiv \lim_{s \to p_i} H(s) \\
H(\Omega t) e^{j\Omega t} &\quad H(\Omega_0) e^{j\Omega_0 t} &\quad \text{sinusoidal, } H(\Omega_0) = H(s) \big|_{s=\Omega_0} \\
\cos(\Omega_0 t) &\quad \text{Re}[H(\Omega_0) e^{j\Omega_0 t}] &\quad \text{sinusoidal} \\
\sin(\Omega_0 t) &\quad \text{Im}[H(\Omega_0) e^{j\Omega_0 t}] &\quad \text{sinusoidal} \\
a_0 &\quad B_0 &\quad \text{constant, } \quad B_0 = A_0 H(0) \\
A_0 + A_1 t &\quad B_0 + B_1 t &\quad \text{linear} \\
A_0 + A_1 t + A_2 t^2 &\quad B_0 + B_1 t + B_2 t^2 &\quad \text{quadratic}
\end{align*}
\]

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,

\[
y_f(t) = B_0 + B_1 t + B_2 t^2 \\
x(t) = A_0 + A_1 t + A_2 t^2
\]


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 in the left-hand \( s \)-plane,

\[
\begin{align*}
(1\text{st order}): &\quad H(s) = \frac{b_0 s + b_1}{s + a_1} = \frac{b_0 s + b_1}{s - p_1} \\
(2\text{nd order}): &\quad H(s) = \frac{b_0 s^2 + b_1 s + b_2}{s^2 + a_1 s + a_2} = \frac{b_0 s^2 + b_1 s + b_2}{(s - p_1)(s - p_2)}
\end{align*}
\]  
\[(21.4.1)\]

and corresponding differential equations,

\[
\begin{align*}
(1\text{st order}): &\quad y(t) + a_1 y(t) = b_0 x(t) + b_1 x(t) \\
(2\text{nd order}): &\quad y(t) + a_1 y(t) + a_2 y(t) = b_0 x(t) + b_1 x(t) + b_2 x(t)
\end{align*}
\]  
\[(21.4.2)\]

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, \( y(0^-), y'(0^-) \) 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 \( t = 0^- \),

\[
\begin{align*}
(1\text{st order}): &\quad [s Y(s) - y(0^-)] + a_1 Y(s) = b_0 X(s) + b_1 X(s) \\
(2\text{nd order}): &\quad [s^2 Y(s) - s y(0^-) - y'(0^-)] + a_1 [s Y(s) - y(0^-)] + a_2 Y(s) = b_0 s^2 X(s) + b_1 s X(s) + b_2 X(s)
\end{align*}
\]

and solve for the Laplace transform of the output \( Y(s) \),

\[
\begin{align*}
(1\text{st order}): &\quad Y(s) = \frac{y(0^-)}{s + a_1} + \left(\frac{b_0 s + b_1}{s + a_1}\right) X(s) \\
(2\text{nd order}): &\quad Y(s) = \left(\frac{a_2 s + b_1 s + b_2}{s^2 + a_1 s + a_2}\right) X(s) + \left(\frac{b_0 s^2 + b_1 s + b_2}{s^2 + a_1 s + a_2}\right) X(s)
\end{align*}
\]  
\[(21.4.3)\]

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
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,

\[
H(s) = \frac{b_0 s^2 + b_1 s + b_2}{(s - p_1)(s - p_2)}.
\]

Then, the zero-state output will have Laplace transform,

\[
Y(s) = H(s)X(s) = \frac{b_0 s^2 + b_1 s + b_2}{s - p_1}(s - p_2)(s - j\Omega_0)
\]

which can be expanded in partial fractions in the form,

\[
Y(s) = \frac{b_0 s^2 + b_1 s + b_2}{s - p_1}(s - p_2)(s - j\Omega_0) = \frac{H(\Omega_0)}{s - j\Omega_0} + \frac{R_1}{s - p_1} + \frac{R_2}{s - p_2}
\]

with, \( H(\Omega_0) = H(s) \big|_{s=j\Omega_0} \) and inverted into the time domain,

\[
y(t) = H(\Omega_0) e^{j\Omega_0 t} + R_1 e^{p_1 t} + R_2 e^{p_2 t}, \quad t \geq 0
\]

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 \texttt{residue} applied on \( Y(s) \), or by the use of the symbolic toolbox functions \texttt{laplace} and \texttt{partfrac}.

If the input were real-valued, that is, either, \( \cos(\Omega_0 t) u(t) \), or, \( \sin(\Omega_0 t) u(t) \), 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

Solution

a. Taking Laplace transforms of both sides of Eq. (21.6.1) with no initial conditions, we have,

\[ s^2Y(s) + 3sY(s) + 2Y(s) = sX(s) \Rightarrow \]

\[ H(s) = \frac{Y(s)}{X(s)} = \frac{s}{s^2 + 3s + 2} \]

For the PFE, we have,

\[ H(s) = \frac{s}{s^2 + 3s + 2} = \frac{s}{(s + 2)(s + 1)} = A \frac{s}{s + 2} + B \frac{s}{s + 1} \]

where

\[ A = \frac{s}{s + 1} \bigg|_{s=2} = \frac{-2}{-2 + 1} = 2, \quad B = \frac{s}{s + 2} \bigg|_{s=1} = \frac{-1}{-1 + 2} = -2 \]

so that

\[ H(s) = \frac{s}{s^2 + 3s + 2} = \frac{s}{(s + 2)(s + 1)} = \frac{2}{s + 2} - \frac{1}{s + 1} \]

(21.6.2)

Using the `residue` function we find,

```matlab
syms s y0 dy0 Y
H = s/(s^2+3*s+2); Y = partfrac(H); % H = 2 / ( s+2 ) - 1 / ( s+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,

b. Inverting the PFE in Eq. (21.6.2), we find,

\[ h(t) = 2e^{-2t}u(t) - e^{-t}u(t) \]

where we used the basic transform pair,

\[ e^{-at}u(t) \leftrightarrow \frac{1}{s + a} \]

Using the symbolic toolbox, we obtain the same,

\[ h(t) = 2e^{-2t}u(t) - e^{-t}u(t) \]

\[ y(t) = c_1 e^{-t} + c_2 e^{-2t} \]

and at \( t = 0^- \), impose the conditions,

\[ y(0^-) = c_1 + c_2 = y_0 \quad \Rightarrow \quad c_1 = y_0 + 2y_0 \]

\[ y(0^-) = -c_1 - 2c_2 = y_0 \quad \Rightarrow \quad c_2 = -y_0 - y_0 \]

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 \( y_0, dy_0 \) stand for the constants \( y_0, y_0 \),

```matlab
y0 = 0; dy0 = 0; Y = solve(s^2*y + s*y0 - dy0 + 3*(s^2*y - y0) + 2*y0 = 0, Y) Y = partfrac(Y,s) % Y = (dy0 + 2*y0)/(s + 1) - (dy0 + y0)/(s + 2)
```

Alternatively, we can use the `dsolve` function,

```matlab
y0 = dy0; yz1 = dsolve(’D2y + 3*Dy + 2*y = 0’, ’y(0)=y0’, ’Dy(0)=dy0’)"
d. The convolutional expression for the zero-state output is,

\[ y_{zs}(t) = \int_{-\infty}^{\infty} h(t')x(t-t')dt' \]  \hspace{1cm} (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 \( t - t' \geq 0 \). Similarly, because \( h(t') \) is causal, its argument must be \( t' \geq 0 \). Combining the two inequalities, we have,

\[ t - t' \geq 0 \Rightarrow t \geq 0 \]
\[ t' \geq 0 \Rightarrow 0 \leq t' \leq t \]

Thus, \( y_{zs}(t) \) must also be causal, and for \( t \geq 0 \), the integral in (21.6.5) simplifies into,

\[ y_{zs}(t) = \int_{0}^{t} h(t')x(t-t')dt' = \int_{0}^{t} (2e^{-2t'} - e^{-t'}) 10e^{-3(t-t')}dt' \]
\[ = 10e^{-3t} \int_{0}^{t} (2e^{-2t'} - e^{-t'}) e^{3t'}dt' = 10e^{-3t} \int_{0}^{t} (2e^{t'} - e^{2t'})dt' \]
\[ = 10e^{-3t} \left[ 2(e^t - 1) - \frac{1}{2}(e^{2t} - 1) \right] = -5e^{-t} + 20e^{-2t} - 15e^{-3t} \]

thus,

\[ y_{zs}(t) = \left[ -5e^{-t} + 20e^{-2t} - 15e^{-3t} \right] u(t) \] \hspace{1cm} (21.6.6)

The integration can also be performed with the int function of the symbolic toolbox,

\[
\begin{align*}
s & \text{sym} 
\end{align*}
\]

\[
\begin{align*}
x & = 10e^{-3*(t-tau)}; \\
h & = 2*exp(-2*tau) - exp(-tau); \\
yzs & = int(h*x, tau, 0, t) & \% yzs = 20*exp(-2*t) - 5*exp(-t) - 15*exp(-3*t)
\end{align*}
\]

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,

\[ Y(s) = H(s)X(s) = \frac{s}{s^2 + 3s + 2} \times \frac{10}{s + 3} = \frac{10s}{(s + 1)(s + 2)(s + 3)} \]

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,

\[
\begin{align*}
A & = (s + 1)Y(s) \bigg|_{s=-1} = \frac{10s}{(s + 2)(s + 3)} \bigg|_{s=-1} = \frac{10(-1)}{(-1 + 1)(-1 + 3)} = -5 \\
B & = (s + 2)Y(s) \bigg|_{s=-2} = \frac{10s}{(s + 1)(s + 3)} \bigg|_{s=-2} = \frac{10(-2)}{(-2 + 1)(-2 + 3)} = 20 \\
C & = (s + 3)Y(s) \bigg|_{s=-3} = \frac{10s}{(s + 1)(s + 2)} \bigg|_{s=-3} = \frac{10(-3)}{(-3 + 1)(-3 + 2)} = -15
\end{align*}
\]

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 \texttt{residue}, where the outputs \( r_1, r_2, r_3 \) correspond to \( C, B, A, \) respectively,

\[
\begin{align*}
[r, p] = \text{residue}([10, 0], \text{conv}([1, 3, 2], [1, 3]));
\end{align*}
\]

\[
\begin{align*}
\% r &= \% -15.0000 \\
\% 20.0000 \%
\text{\%} -5.0000 \\
\% -3.0000 \\
\% -2.0000 \\
\% -1.0000
\end{align*}
\]

The indicated convolution operation, \( \text{conv}([1, 3, 2], [1, 3]) \), 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,

\[
\begin{align*}
s & \text{sym} 
\end{align*}
\]

\[
\begin{align*}
x & = 10/(s+3); \\
y & = H*x; \\
y & = \% Y = 10*s/(s+3)^3(s+2+3*s+2)) \\
y & = \% \text{partfrac}(Y) \% Y = 10/(s+3) - 5/(s+1) - 15/(s+3) \\
yzs & = \% \text{ilaplace}(Y) \% yzs = 20*exp(-2*t) - 5*exp(-t) - 15*exp(-3*t) \\
\end{align*}
\]

For \( t \geq 0 \), we obtain from the above solution,

\[
\begin{align*}
y_{zs}(t) = -5e^{-t} + 20e^{-2t} - 15e^{-3t} & \Rightarrow y_{zs}(0^+) = -5 + 20 - 15 = 0 \\
y_{zs}(t) = 5e^{-t} - 40e^{-2t} + 45e^{-3t} & \Rightarrow y_{zs}(0^+) = 5 - 40 + 45 = 10
\end{align*}
\]

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 \texttt{ilaplace} requires the \( t = 0^- \) conditions, whereas the solution using \texttt{dsolve}, requires the \( t = 0^+ \) conditions.

In the present case, since we just found \( y_{zs}(0^+) = 0 \), \( y_{zs}(0^+) = 10 \), we can apply the \texttt{dsolve} function, noting that \( x(t) = -3 \cdot 10e^{-3t} \) for \( t \geq 0^+ \),

\[
\begin{align*}
s & \text{sym} 
\end{align*}
\]

\[
\begin{align*}
yzs & = \% \text{dsolve}('2y\cdot3^*\text{DY}\cdot2^*y = 10\cdot(-3)^*\text{exp}(-3*t')', 'y(0)=0', '\text{DY}(0)=10')
\end{align*}
\]
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, \( y(0^-) = y_0, \quad y'(0^-) = y'_0, \)

\[
y(t) = (y_0 + 2y_0 - 5)e^{-t} + (20 - y_0 - y_0)e^{-2t} - 15e^{-3t}, \quad t \geq 0^+ \tag{21.6.7}
\]

and for the particular values, \( y(0^-) = 0, \quad y'(0^-) = -5, \)

\[
y(t) = -10e^{-t} + 25e^{-2t} - 15e^{-3t}, \quad t \geq 0^+ \tag{21.6.8}
\]

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",

\[
y(t) = (y_0 + 2y_0 - 5)e^{-t} + (20 - y_0 - y_0)e^{-2t} - 15e^{-3t} \]

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 \( x(t) = Ae^{-st}, \) and are

\[
Y(s) = \text{partfrac}(Y,s) \quad \text{where} \quad Y(s) = \frac{dy_0+2y_0-5}{s+1} + \frac{20-dy_0-y_0}{s+2} - \frac{15}{s+3}
\]

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,

\[
s y y_0 y Y \quad X = 10/(s+3); 
Y = \text{solve}(s^2*Y-s*y0-dy0 + 3*(s*Y-y0) + 2*Y == s*X, Y) 
Y = \text{partfrac}(Y,s) \quad \text{where} \quad Y(s) = \frac{dy_0+2y_0-5}{s+1} + \frac{20-dy_0-y_0}{s+2} - \frac{15}{s+3}
\]

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,

\[
y(0^+) = y(0^-) + b_0x(0^+) 
y(0^+) = y(0^-) + b_0x(0^+) + (b_1 - b_0a_1)x(0^-) \tag{21.6.10}
\]

For our particular system, we have, \([b_0, b_1, b_2] = [0, 1, 0], \) so that Eqs. (21.6.10) become,

\[
y(0^+) = y(0^-) \tag{21.6.11}
y(0^+) = y(0^-) + x(0^-) \tag{21.6.11}
\]

Thus, for the input \( x(t) = 10e^{-3t}u(t), \) and initial conditions \( y_0, y'_0 \) at \( t = 0^- \), we have,

\[
y(0^+) = y(0^-) \tag{21.6.12}
y(0^+) = y_0 + 10
\]

These are the conditions that must be used in applying the classical method, or the \texttt{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, \( y_{\text{forced}}(t) = \)
21.6. CONTINUOUS-TIME EXAMPLE

For the homogeneous solution we form a linear combination of the characteristic modes $e^{t}, e^{-2t}$. Thus,

\[ y(t) = c_1 e^{t} + c_2 e^{-2t} - 15 e^{-3t} \]

\[ y'(t) = c_1 e^{t} - 2c_2 e^{-2t} + 45 e^{-3t} \] (classical method)

for $t \geq 0$. Imposing the $t = 0^+$ conditions (21.6.12), we have,

\[ y(0^+) = c_1 + c_2 - 15 = y_0 \quad \Rightarrow \quad c_1 = y_0 + 2y_0 - 5 \]

\[ y'(0^+) = -c_1 - 2c_2 + 45 = y_0 + 10 \quad \Rightarrow \quad c_2 = 20 - y_0 - y_0 \]

Thus, we obtain the same solution as that in Eq. (21.6.7), for $t \geq 0$,

\[ y(t) = c_1 e^{t} + c_2 e^{-2t} - 15 e^{-3t} = (y_0 + 2y_0 - 5)e^{-t} + (20 - 2y_0)e^{-2t} - 15 e^{-3t} \]

Finally, the same solution can be obtained with the `dsolve` function applied with the initial conditions at $t = 0^+$ of Eq. (21.6.12).

```matlab
syms t y0 dy0 y(t)
dy = diff(y(t));
x = 10*exp(-3*t);
y = dsolve(diff(dy) + 2*dy^2 = 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,

```matlab
M = 2001; % desired range of t's
x = 10*exp(-3*t);
H = s/(s^2+3*s+2); % transfer function object - class(H) is tf
S = ss(H); % S is state-space model of H
y = lsim(S,x,0); % run model S with initial state si=0
plot(t,y,'-'); % 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, $H_a(s)$,

\[ y(t) + A_1 y(t) + A_2 y(t) = B_0 x(t) + B_1 x(t) + B_2 x(t) \]

\[ H_a(s) = \frac{B_0 s^2 + B_1 s + B_2}{s^2 + A_1 s + A_2} \] (CT system) \hspace{1cm} (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, $H_d(z)$,

\[ y_n + a_1 y_{n-1} + a_2 y_{n-2} = b_0 x_n + b_1 x_{n-1} + b_2 x_{n-2} \]

\[ H_d(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \] (DT system) \hspace{1cm} (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, $n = 0, 1, 2, \ldots$,
\[ y_n = y(n_T) = y(nT) \]
\[ x_n = x(n_T) = x(nT) \]

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,

\[
\begin{align*}
\text{for } n = 0, 1, 2, \ldots, \\
y_n &= -a_1 y_{n-1} - a_2 y_{n-2} + b_0 x_n + b_1 x_{n-1} + b_2 x_{n-2}
\end{align*}
\] (21.7.3)

where the two previously computed outputs, \(y_{n-2}, y_{n-1}\), are used to compute the current one, \(y_n\). For example, we have explicitly for \(n = 0, 1, 2, 3, \ldots\),

\[
\begin{align*}
y_0 &= -a_1 y_{-1} - a_2 y_{-2} + b_0 x_0 + b_1 x_{-1} + b_2 x_{-2} \\
y_1 &= -a_1 y_0 - a_2 y_{-1} + b_0 x_1 + b_1 x_{0} + b_2 x_{-1} \\
y_2 &= -a_1 y_1 - a_2 y_0 + b_0 x_2 + b_1 x_{1} + b_2 x_{0} \\
y_3 &= -a_1 y_2 - a_2 y_1 + b_0 x_3 + b_1 x_{2} + b_2 x_{1}, \text{ and so on,}
\end{align*}
\]

The second one may be justified by the following approximation of the derivative,

\[
\dot{y}(0^-) \approx \frac{y(-T) - y(-2T)}{T} \approx \frac{y_{-1} - y_{-2}}{T} \Rightarrow y_{-2} \approx y_{-1} - T \dot{y}(0^-)
\]

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,

\[
\dot{y}(0^-) \approx \frac{y_{-1} - 2y_{-2} + y_{-3}}{T^2} = \frac{y(0^-) - 2y(-T) + y(-2T)}{T^2} + \frac{y(-3)}{T}
\]

\[
\Rightarrow y_{-3} = T^2 \ddot{y}(0^-) - 2T \dot{y}(0^-) + y(-3)
\]

Similarly, for a first-order system we have the differential and corresponding difference equations,

\[
\begin{align*}
y(t) + A_1 y(t) &= H_a(s) x(t) \\
y_n + a_1 y_{n-1} &= b_0 x_n + b_1 x_{n-1}
\end{align*}
\] (21.7.6)

and iterated in the following form,

\[
\begin{align*}
y_{-1} &\approx y(0^-) \\
y_{-2} &\approx y(0^-) - T \dot{y}(0^-) \\
\text{for } n = 0, 1, 2, \ldots, \\
y_n &= -a_1 y_{n-1} + b_0 x_n + b_1 x_{n-1}
\end{align*}
\] (21.7.7)

where now only the starting value \(y_{-1}\) is needed, and we may choose it as in Eq. (21.7.4),

\[
y_{-1} \approx y(0^-)
\]

### 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,

\[
H_d(z) = H_a(z) \bigg|_{s = f(z)}
\] (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, \(s = f(z)\),

\[
s = f(z) = \frac{1 - z^{-1}}{p + q z^{-1}} \Rightarrow H_d(z) = H_a(s) \bigg|_{s = \frac{1 - z^{-1}}{p + q z^{-1}}} = H_a \left( \frac{1 - z^{-1}}{p + q z^{-1}} \right)
\] (21.8.2)

where the parameters \(p, q\) are defined as follows in the three cases, in terms of the discretization time step \(T\),

- **forward Euler**: \(p = 0, \quad q = T \quad \Rightarrow s = \frac{1}{T} (z - 1)\)
- **backward Euler**: \(p = T, \quad q = 0 \quad \Rightarrow s = \frac{1}{T} (1 - z^{-1})\)
- **trapezoidal/bilinear/Tustin**: \(p = q = \frac{1}{2} T \quad \Rightarrow s = \frac{2}{T} \frac{1 - z^{-1}}{1 + z^{-1}}\)

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,

\[
\begin{align*}
B_0 s^2 + B_1 s + B_2 &= \frac{B_0}{s^2 + A_1 s + A_2} \bigg|_{s = \frac{1 - z^{-1}}{p + q z^{-1}}} = \frac{B_0 \left( \frac{1 - z^{-1}}{p + q z^{-1}} \right)^2 + B_1 \left( \frac{1 - z^{-1}}{p + q z^{-1}} \right) + B_2}{1 + A_1 z^{-1} + A_2 z^{-2}} \\
&= b_0 + b_1 z^{-1} + b_2 z^{-2} \\
&\quad \frac{1 + a_1 z^{-1} + a_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}
\end{align*}
\]

The algebra can be carried out quickly with MATLAB’s symbolic toolbox,
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,

\[ H_d(z) = (1 - z^{-1}) \cdot Z \left[ \frac{H_a(s)}{s} \right] \] (21.9.1)

where \( Z \left[ G(s) \right] \) 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,

\[ H_a(s) = \frac{B_0s + B_1}{s + p_1} \Rightarrow \frac{H_a(s)}{s} = \frac{B_0s + B_1}{s(s + p_1)} = \frac{R_0}{s} + \frac{R_1}{s + p_1} \] (21.9.2)

with residues,

\[ R_0 = \frac{B_1}{p_1}, \quad R_1 = \frac{B_0 - B_1p_1}{p_1} \] (21.9.3)

Similarly, for a second-order transfer function with distinct poles \( p_1 \neq p_2 \), we obtain,

\[ H_a(s) = \frac{B_0s^2 + B_1s + B_2}{s(s + p_1)(s + p_2)} \Rightarrow \frac{H_a(s)}{s} = \frac{B_0s^2 + B_1s + B_2}{s(s + p_1)(s + p_2)} = \frac{R_0}{s} + \frac{R_1}{s + p_1} + \frac{R_2}{s + p_2} \] (21.9.4)

where the residues are given by,

\[ R_0 = \frac{B_2}{p_1p_2}, \quad R_1 = \frac{B_0p_2^2 - B_1p_1 + B_2}{p_1(p_1 - p_2)}, \quad R_2 = \frac{B_0p_1^2 - B_1p_2 + B_2}{p_2(p_2 - p_1)} \] (21.9.5)

Step 2: Next, replace single- and double-pole terms as follows in terms of \( z \) (applicable also when \( p_1 = 0 \)),

\[ \frac{1}{s + p_1} = \frac{1}{1 - e^{-p_1T}z^{-1}} \] (21.9.6)

\[ \frac{1}{(s + p_1)^2} = \frac{Tz^{-1}}{(1 - e^{-p_1T}z^{-1})^2} \] (21.9.7)

Step 3: After making these replacements, multiply by an overall factor of \((1 - z^{-1})^2\) to obtain the final DT transfer function. Thus, for the first-order case, we have,

\[ H_d(z) = (1 - z^{-1}) \left[ \frac{R_0}{1 - e^{-p_1T}z^{-1}} + \frac{R_1}{1 - e^{-p_1T}z^{-1}} \right] \] (21.9.8)

where, with \( R_0, R_1 \) given by Eq. (21.9.3),

\[ b_0 = R_0 + R_1, \quad b_1 = -(R_1 + e^{-p_1T}R_0), \quad a_1 = -e^{-p_1T} \] (21.9.9)

Similarly, the second-order case with distinct poles gives,

\[ H_d(z) = (1 - z^{-1}) \left[ \frac{R_0}{1 - e^{-p_1T}z^{-1}} + \frac{R_1}{1 - e^{-p_1T}z^{-1}} + \frac{R_2}{1 - e^{-p_1T}z^{-1}} \right] \] (21.9.10)

where, with \( R_0, R_1, R_2 \) given by Eq. (21.9.5),

\[ b_0 = R_0 + R_1 + R_2 = B_0 \]

\[ b_1 = -R_1 - R_2 - R_1e^{-p_1T} - R_2e^{-p_1T} - R_0(e^{-p_1T} + e^{-p_1T}) \]

\[ b_2 = R_1e^{-p_1T} + R_2e^{-p_1T} + R_0e^{-p_1T}e^{-p_1T} \]

\[ a_1 = -e^{-p_1T} - e^{-p_1T}, \quad a_2 = e^{-p_1T}e^{-p_1T} \]
Lastly, for the case of a double-pole, we have,

\[
H_d(z) = (1 - z^{-1}) \left[ \frac{R_0}{1 - z^{-1}} + \frac{R_1}{1 - e^{-p_1 T} z^{-1}} + \frac{R_2 Tz^{-1}}{(1 - e^{-p_1 T} z^{-1})^2} \right]
\]

\[
= \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}
\]

(21.9.13)

where, with \(R_0, R_1, R_2\) given by Eq. (21.9.7),

\[
b_0 = R_0 + R_1 = 0
\]

\[
b_1 = (R_2 T - 2R_0 - R_1)e^{-p_1 T} - R_1
\]

\[
b_2 = (R_1 - R_2 T + R_0 e^{-p_1 T}) e^{-p_1 T}
\]

\[
a_1 = -2e^{-p_1 T}, \quad a_2 = e^{-2p_1 T}
\]

(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,

\[
H_d(z) = \frac{(1 - z^{-1})^2}{Tz^{-1}} \cdot Z \left[ \frac{H_d(s)}{s^2} \right]
\]

(first-order hold) \hspace{1cm} (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 high-fidelity audio signal sampled at a rate of 40 kHz (40,000 samples/sec), the time interval between samples is \(T = 1/40000 \text{sec} = 25 \text{ μsec}\). A modern DSP has an instruction time of about 1 nsec for performing a typical multiplication or addition. Therefore, during the interval of \(T = 25 \text{ μsec} = 25,000 \text{ nsec}\), 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,

\[
H_d(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}
\]

(21.10.1)

The direct 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 \(y(n)\),

\[
y(n) = -a_1 y(n - 1) - a_2 y(n - 2) + b_0 x(n) + b_1 x(n - 1) + b_2 x(n - 2)
\]

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,

\[
v_1(n) = x(n - 1)
\]

\[
v_2(n) = x(n - 2) = v_1(n - 1)
\]

\[
w_1(n) = y(n - 1)
\]

\[
w_2(n) = y(n - 2) = w_1(n - 1)
\]

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 \(n\),

\[
y(n) = -a_1 w_1(n) - a_2 w_2(n) + b_0 x(n) + b_1 v_1(n) + b_2 v_2(n)
\]

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,

\[
v_1(n + 1) = x(n)
\]

\[
v_2(n + 1) = x(n - 1) = v_1(n)
\]

\[
w_1(n + 1) = y(n)
\]

\[
w_2(n + 1) = y(n - 1) = w_1(n)
\]

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, \(w_1(n), w_2(n)\),

![Sample-by-Sample Processing Diagram](image-url)
21. DISCRETIZATION METHODS

21.10 SAMPLE-BY-SAMPLE PROCESSING

initialize \(w_1, w_2, v_1, v_2\)
for each input sample \(x\), do,

\[
\begin{align*}
  y &= -a_1 w_1 - a_2 w_2 + b_0 x + b_1 v_1 + b_2 v_2 \\
  w_2 &= w_1 \\
  w_1 &= y \\
  v_2 &= v_1 \\
  v_1 &= x
\end{align*}
\]

(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.

\[
\begin{align*}
  w_0(n) &= -a_1 w_1(n) - a_2 w_2(n) + x(n) \\
  y(n) &= b_0 w_0(n) + b_1 w_1(n) + b_2 w_2(n) \\
  w_2(n+1) &= w_1(n) \\
  w_1(n+1) &= w_0(n)
\end{align*}
\]

(canonical realization) (21.10.4)

The transposed realization (also known as the observer-canonical form) is shown below together with its sample processing algorithm.

\[
\begin{align*}
  v_1(n) &= b_1 x - a_1 y + v_2(n) \\
  v_2(n+1) &= b_2 x - a_2 y(n)
\end{align*}
\]

(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,

\[
\begin{align*}
  y(n) &= b_0 x(n) + v_1(n) \\
  v_1(n+1) &= b_1 x(n) - a_1 y(n) + v_2(n) \\
  v_2(n+1) &= b_2 x(n) - a_2 y(n)
\end{align*}
\]

(transposed realization) (21.10.6)

Every realization has a transposed version obtained by the following transposition rules:

- replace adders by nodes
- replace nodes by adders
21.10. SAMPLE-BY-SAMPLE PROCESSING

- 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. 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,

\[
\begin{align*}
\mathbf{s}(n+1) &= A\mathbf{s}(n) + Bx(n) \\
y(n) &= C\mathbf{s}(n) + Dx(n)
\end{align*}
\]  
(ABCD state-space realization) (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 \( s \), then,
for each input sample \( x \), do,
\[
\begin{align*}
\mathbf{y} &= C\mathbf{s} + Dx(n), \text{ output} \\
\mathbf{s} &= A\mathbf{s} + Bx(n), \text{ next state}
\end{align*}
\]
(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,

\[
\begin{align*}
\mathbf{s}(n) &= \begin{bmatrix} w_1(n) \\ w_2(n) \end{bmatrix} \text{ canonical, } \\
\mathbf{s}(n) &= \begin{bmatrix} v_1(n) \\ v_2(n) \end{bmatrix} \text{ transposed (21.10.9)}
\end{align*}
\]

The corresponding \( A, B, C, D \) matrices have dimensions, \( 2 \times 2, 2 \times 1, 1 \times 2, \) and \( 1 \times 1 \), respectively, and are given as follows in the two cases,

\[
\begin{align*}
\text{(canonical): } A &= \begin{bmatrix} -a_1 & -a_2 \\ 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad C = [c_1, c_2], \quad D = b_0 \\
\text{(transposed): } A &= \begin{bmatrix} -a_1 & 1 \\ -a_2 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}, \quad C = [1, 0], \quad D = b_0
\end{align*}
\]  
(21.10.10)

where we defined the parameters,

\[
\begin{align*}
c_1 &= b_1 - b_0 a_1 \\
c_2 &= b_2 - b_0 a_2
\end{align*}
\]

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,

\[
\begin{align*}
y(n) &= b_0 x(n) + v_1(n) \\
v_1(n+1) &= b_1 x(n) - a_1 y(n) + v_2(n) = b_1 x(n) - a_1 [b_0 x(n) + v_1(n)] + v_2(n) \\
v_2(n+1) &= b_2 x(n) - a_2 y(n) = b_2 x(n) - a_2 [b_0 x(n) + v_1(n)]
\end{align*}
\]

or,

\[
\begin{align*}
v_1(n+1) &= -a_1 v_1(n) + v_2(n) + (b_1 - b_0 a_1) x(n) = -a_1 v_1(n) + v_2(n) + c_1 x(n) \\
v_2(n+1) &= -a_2 v_1(n) + (b_2 - b_0 a_2) x(n) = -a_2 v_1(n) + c_2 x(n)
\end{align*}
\]

\[
y(n) = v_1(n) + b_0 x(n)
\]

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,

\[
\begin{align*}
w_1(n+1) &= w_0(n) = -a_1 w_1(n) - a_2 w_2(n) + x(n) \\
w_2(n+1) &= w_1(n)
\end{align*}
\]

\[
y(n) = b_0 w_0(n) + b_1 w_1(n) + b_2 w_2(n)
\]

\[
= b_0 [-a_1 w_1(n) - a_2 w_2(n) + x(n)] + b_1 w_1(n) + b_2 w_2(n)
\]

\[
= (b_1 - b_0 a_1) w_1(n) + (b_2 - b_0 a_2) w_2(n) + b_0 x(n)
\]

\[
c_1 w_1(n) + c_2 w_2(n) + b_0 x(n)
\]

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,

\[
\begin{align*}
A & \rightarrow A^T \\
B & \rightarrow C^T \\
C & \rightarrow B^T \\
D & \rightarrow D
\end{align*}
\tag{21.10.11}
\]

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,

\[
\begin{align*}
zs(z) &= Azs(z) + Bx(z) \\
y(z) &= Czs(z) + Dx(z)
\end{align*}
\]

For example, for our 2nd order example mentioned above,

\[
\begin{align*}
A & \rightarrow \begin{bmatrix} b_0 & b_1 \\ 1 & a_1 \end{bmatrix} \\
B & \rightarrow \begin{bmatrix} 0 & 0 \\ 1 & a_2 \end{bmatrix} \\
C & \rightarrow \begin{bmatrix} 1 & 0 \\ -1 & z \end{bmatrix} \\
D & \rightarrow D
\end{align*}
\tag{21.10.11}
\]

\[
\begin{align*}
H_d(z) &= \frac{Y(z)}{X(z)} = C(zI - A)^{-1}B + D
\end{align*}
\tag{21.10.12}
\]

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,

\[
h_d(n) = CA^nB u(n - 1) + D\delta(n)
\tag{21.10.13}
\]

We demonstrate Eq. (21.10.12) explicitly for the canonical realization with parameters given by Eq. (21.10.10),

\[
H_d(z) = \frac{Y(z)}{X(z)} = \frac{1}{z^2 + a_1z + a_2}
\]

\[
\begin{align*}
\det(zI - A) &= z^2 + a_1z + a_2 \\
(zI - A)^{-1} &= \frac{1}{\det(zI - A)} \begin{bmatrix} z & a_2 \\ 1 & z + a_1 \end{bmatrix} = \frac{1}{z^2 + a_1z + a_2} \begin{bmatrix} z & a_2 \\ 1 & z + a_1 \end{bmatrix}
\end{align*}
\]

\[
C(zI - A)^{-1}B = \frac{C_1z + C_2}{z^2 + a_1z + a_2}
\]

\[
H_d(z) = C(zI - A)^{-1}B + D = \frac{C_1z + C_2}{z^2 + a_1z + a_2} + b_0 = \frac{b_1z + b_2}{z^2 + a_1z + a_2} + b_0
\]

MATLAB’s built-in function, \texttt{tf2ss}, maps a transfer function defined by numerator and denominator coefficients, \(\texttt{num, den}\), to an ABCD state space form that is by default the canonical realization,

\[
[A,B,C,D] = \texttt{tf2ss(num,den)}; \quad \% \ \text{canonical state-space form}
\]

For example, for our 2nd order \(H_d(z)\), it generates the parameters of the canonical form in Eq. (21.10.10),

\[
\begin{align*}
\texttt{num} &= \begin{bmatrix} b_0, b_1, b_2 \end{bmatrix}; \\
\texttt{den} &= \begin{bmatrix} 1, a_1, a_2 \end{bmatrix}; \\
[A,B,C,D] &= \texttt{tf2ss(num,den)}; \quad \% \ \text{canonical state-space form}
\end{align*}
\]

\[
\begin{align*}
y_{-1} &= y(0^-) \\
y_{-2} &= y(0^-) - Ty(0^-)
\end{align*}
\tag{21.11.1}
\]

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 \(x_n\),

\[
\begin{align*}
w_1(0) &= y_{-1} = y(0^-) \\
w_2(0) &= y_{-2} = y(0^-) - Ty(0^-) \\
v_1(0) &= x_{-1} = 0, \quad \text{causal input} \\
v_2(0) &= x_{-2} = 0
\end{align*}
\tag{21.11.2}
\]

In the other realizations, however, the internal states, \(w_1(n)\) or \(v_1(n)\), 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, \(y_{-1}, y_{-2}\), to the initial value, \(s(0)\), of the two-dimensional state vector \(s(n)\). 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., \(x_{-2} = x_{-1} = 0\),

\[
\begin{align*}
y_{-2} &= Cs(-2) + Dx_{-2} = Cs(-2) \\
s(-1) &= As(-2) + Bx_{-2} = As(-2) \\
y_{-1} &= Cs(-1) + Dx_{-1} = Cs(-1) = CAs(-2) \\
s(0) &= As(-1) + Bx_{-1} = As(-1) = A^2s(-2)
\end{align*}
\]
for an $M$th order case, $F$ is an $M \times M$ matrix defined as $F = [C; CA; CA^2; \ldots CA^{M-1}]$

†https://en.wikipedia.org/wiki/Observability

or, arranging, $y_{-2}, y_{-1}$, into a column,

$$
\begin{bmatrix}
y_{-2} \\
y_{-1}
\end{bmatrix} = 
\begin{bmatrix} C \\ CA\end{bmatrix} \text{s}(\text{-}2) \equiv F \text{s}(\text{-}2)
$$  (21.11.3)

$$
\text{s}(0) = A^2 \text{s}(\text{-}2)
$$

where we defined the so-called “observability” matrix, which is a $2 \times 2$ matrix in the 2nd order case,‡

$$
F = \begin{bmatrix} C \\ CA\end{bmatrix} \quad \text{(observability matrix)} \quad (21.11.4)
$$

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 is invertible, i.e., the inverse $F^{-1}$ exists. This allows Eq. (21.11.3) to be solved for $s(\text{-}2)$ which is then used to calculate $s(0)$,

$$
\text{s}(\text{-}2) = F^{-1} \begin{bmatrix} y_{-2} \\
y_{-1}\end{bmatrix} \Rightarrow \text{s}(0) = A^2 \text{s}(\text{-}2) = A^2 F^{-1} \begin{bmatrix} y_{-2} \\
y_{-1}\end{bmatrix}
$$

Thus, given the approximate initial output values (21.11.1), we calculate the initial state vector $s(0)$ by,

$$
\text{s}(0) = A^2 F^{-1} \begin{bmatrix} y_{-2} \\
y_{-1}\end{bmatrix} = A^2 \text{F}^{-1} \begin{bmatrix} y(0^-)-T\dot{y}(0^-) \\
y(0^-)\end{bmatrix}
$$  (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,

$$
F = \text{obsv}(A,C); \quad \% \text{based on a given ABCD state-space form}
$$

$$
F = \text{obsmat(num,den,type)}; \quad \% \text{type = 'c', 't', for canonical or transposed}
$$

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,

$$
\begin{align*}
C &= [c_1, c_2], \quad CA = [c_1, c_2] \begin{bmatrix} -a_1 & -a_2 \\
1 & 0 \end{bmatrix} = [-a_1 c_1 + c_2, -a_2 c_1] \\
F &= \begin{bmatrix} C \\ CA\end{bmatrix} = \begin{bmatrix} c_1 \\ -a_1 c_1 + c_2 \end{bmatrix} \begin{bmatrix} c_2 \\ -a_2 c_1 \end{bmatrix} \\
F^{-1} &= \frac{1}{a_1 c_1 c_2 - a_2 c_1^2} \begin{bmatrix} -a_2 c_1 & -c_2 \\
a_1 c_1 + c_2 & c_1 \end{bmatrix} \\
A^2 &= \begin{bmatrix} -a_1 & -a_2 \\
-1 & 0 \end{bmatrix} \begin{bmatrix} -a_1 & -a_2 \\
1 & 0 \end{bmatrix} = \begin{bmatrix} a_1^2 & -a_1 a_2 \\
a_1 & -a_2 \end{bmatrix} \\
A^2 F^{-1} &= \begin{bmatrix} a_1^2 & -a_1 a_2 \\
a_1 & -a_2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\
a_1 & 1 \end{bmatrix} \quad \% \text{or,}
\end{align*}
$$

The transposed case is simpler,

$$
\begin{align*}
C &= [1, 0], \quad CA = [1, 0] \begin{bmatrix} -a_1 & 1 \\
-a_2 & 0 \end{bmatrix} = [-a_1, 1] \\
F &= \begin{bmatrix} C \\ CA\end{bmatrix} = \begin{bmatrix} 1 \\ -a_1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \Rightarrow F^{-1} = \begin{bmatrix} 1 \\ a_1 \end{bmatrix} \\
A^2 &= \begin{bmatrix} -a_1 & 1 \\
-a_2 & 0 \end{bmatrix} \begin{bmatrix} -a_1 & 1 \\
-0 & 0 \end{bmatrix} = \begin{bmatrix} a_1^2 & -a_1 a_2 \\
a_1 & -a_2 \end{bmatrix} \\
A^2 F^{-1} &= \begin{bmatrix} a_1^2 & -a_1 a_2 \\
a_1 & -a_2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\
a_1 & 1 \end{bmatrix} \quad \% \text{or,}
\end{align*}
$$

(21.11.6)

(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 (b) of the example of Sec. 21.6.

Example

In the example of Sec. 21.6, we considered the following linear system,

$$
\dot{y}(t) + 3\dot{y}(t) + 2y(t) = x(t) \Rightarrow H_a(s) = \frac{s}{s^2 + 3s + 2} \quad (21.11.8)
$$
driven by the input, $x(t) = 10e^{-2t}u(t)$, and subject to the initial conditions at $t = 0^-$,

$$y(0^-) = 0, \quad y'(0^-) = -5$$

Using Laplace and `dsolve` methods, we found the following exact solutions for the zero-input, zero-state, and total responses, for $t \geq 0$,

$$y_z(t) = -5e^{-t} + 5e^{-2t}$$

$$y_z(t) = -5e^{-t} + 20e^{-2t} - 15e^{-3t} \quad (21.11.9)$$

$$y(t) = y_z(t) + y_z(t) = -10e^{-t} + 25e^{-2t} - 15e^{-3t}$$

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-order 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, $y_z(t), y_z(t), y(t)$, 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),

$$b_0 = \frac{p}{1 + 3p + 2p^2}, \quad b_1 = \frac{-q - p}{1 + 3p + 2p^2}, \quad b_2 = \frac{-2q - 3p}{1 + 3p + 2p^2} \quad (21.11.10)$$

$$a_1 = \frac{3(q - p) - 2 + 4pq}{1 + 3p + 2p^2}, \quad a_2 = \frac{1 - 3q + 2q^2}{1 + 3p + 2p^2}$$

and in particular, for the trapezoidal case with $p = q = T/2$,

$$b_0 = \frac{T}{T^2 + 3T + 2}, \quad b_1 = 0, \quad b_2 = \frac{-T}{T^2 + 3T + 2} \quad (21.11.11)$$

Similarly for the ZOH case, we find,

$$b_0 = 0, \quad b_1 = (e^{-T} - e^{-2T}), \quad b_2 = -(e^{-T} - e^{-2T})$$

$$a_1 = -(e^{-T} + e^{-2T}), \quad a_2 = e^{-3T}$$

(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.

```matlab
y0 = 0; doty0 = -5; \% initial conditions at t=0-
x = @(t) 10*exp(-3*t).*t(t=0); \% input signal
T = 0.01; \% sampling time interval
t = 0:T:6; \% sampled times, t=n*T
ysi = -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,ysi,'b-','t',yzs,'r-','t',ysi,'g--'); \% plot exact outputs
title('exact'); \% plot exact outputs
xlabel('t(t)'); ylabel('y(t)'); legend('total','zero-state','zero-input','location','ne')
xaxis(0.6,0:0; yaxis(-1.5,1.5, -1.5:0.5:1.5);
xst = x(t); \% input signal samples
s = tf('s'); \% transfer function variable, class tf
H = s/(s^2+3s+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 0 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]
ya = lsim(S,xst,si); \% approximate total output with non-zero ICs, si
yazs = lsim(S,xst); \% approximate zero-state output with zero ICs, si=0
```

```matlab
```
1011

% approximate zero-input output with non-zero ICs, si
figure; plot(t, ya, 'b-', t, yazs, 'r--', t, yazi, 'g--'); % plot LSIM outputs

% approximate zero-input output with non-zero ICs, si
figure; plot(t, ya, 'b-', t, yazs, 'r--', t, yazi, 'g--'); % plot LSIM outputs

% 21. DISCRETIZATION METHODS

% canonical-form -- zero-state response yczs(n)
for n=0:length(t)-1  % iterate canonical form
    yczs(n+1) = b0*x(n*T) + v1;  % initial values
    v1 = b1*x(n*T) - a1*yczs(n+1) + v2;  % initial values
    v2 = b2*x(n*T) - a2*yczs(n+1);
end

doty = filter(b,a,x(t),si); % output using FILTER with initial conditions
% canonical-form -- total response yt(n)
A = tf2ss(b,a);  % state matrix for canonical form
F = obsmat(b,a,'c');  % observability matrix for canonical
s1 = A'*inv(F)'*[y0-T*doty0; y0];  % initial state vector
w1 = s1(1); w2 = s1(2);  % initial values
for n=0:length(t)-1  % iterate canonical form
    w0 = -a1*w1 - a2*w2 + x(n*T);  % initial values
    y(n+1) = b0*w0 + b1*w1 + b2*w2;
    w2 = w1;
    w1 = w0;
end

dotyf = filter(b,a,x(t),si); % output using FILTER with 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
s1 = A'*inv(F)'*[y0-T*doty0; y0];  % initial state vector for transposed form
w1 = s1(1); w2 = s1(2);  % initial values
for n=0:length(t)-1  % iterate transposed form
    w0 = -a1*w1 - a2*w2 + x(n*T);  % initial values
    y(n+1) = b0*w0 + b1*w1 + b2*w2;
    w2 = w1;
    w1 = w0;
end

dotyf = filter(b,a,x(t),si); % output using FILTER with initial conditions

% canonical-form -- zero-input response yczi(n)
A = tf2ss(b,a);  % state matrix for canonical form
F = obsmat(b,a,'c');  % observability matrix for canonical
s1 = A'*inv(F)'*[y0-T*doty0; y0];  % initial state vector
w1 = s1(1); w2 = s1(2);  % initial values
for n=0:length(t)-1  % iterate canonical form
    w0 = -a1*w1 - a2*w2 + x(n*T);  % initial values
    yczi(n+1) = b0*w0 + b1*w1 + b2*w2;
    w2 = w1;
    w1 = w0;
end

dotyf = filter(b,a,x(t),si); % output using FILTER with 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
s1 = A'*inv(F)'*[y0-T*doty0; y0];  % initial state vector for transposed form
w1 = s1(1); w2 = s1(2);  % initial values
for n=0:length(t)-1  % iterate transposed form
    w0 = -a1*w1 - a2*w2 + x(n*T);  % initial values
    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

dotyf = filter(b,a,x(t),si); % output using FILTER with initial conditions
% transposed-form -- zero-state response ytzs(n)
for n=0:length(t)-1  % iterate transposed form
    ytzs(n+1) = b0*x(n*T) + v1;  % initial values
    v1 = b1*x(n*T) - a1*ytzs(n+1) + v2;  % initial values
    v2 = b2*x(n*T) - a2*ytzs(n+1);
end

dotyf = filter(b,a,x(t),si); % output using FILTER with initial conditions
21.11. Initialization Procedures

% 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

pyz = 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

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)
%----------------------------------

% l1 = y0; % initialize
% l2 = y0-T*doty0; % initialize
% v1 = 0; % initialize v,v2, where x(t) is causal
% v2 = 0; % for n=0:length(t)-1
% yd(n+1) = b1*l1 + b0*x(n*T) + b1*v1 + b2*v2; % difference equation
% w2 = l2; % time updates
% w1 = yd(n+1);
% v2 = w1;
% 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,

\[ y(t) = x(t) \quad \Rightarrow \quad y(t) = y(0^-) + \int_{0^-}^{t} x(t') dt' \quad t \geq 0^- \tag{21.12.1} \]

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.
21. DISCRETIZATION METHODS

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,

$$x^*(t) = \sum_{n} x(nT) \delta(t-nT) \text{ ideally-sampled signal}$$ (21.13.1)

Assuming a causal signal $x(t)$, the above summation can be restricted to $n \geq 0$. The Laplace transform of the ideally sampled signal $x^*(t)$ is referred to as the starred Laplace transform,

$$X^*(s) = \int_{0}^{\infty} x^*(t) e^{-st} dt = \int_{0}^{\infty} \sum_{n} x(nT) \delta(t-nT) e^{-st} dt = \sum_{n} x(nT) \int_{0}^{\infty} \delta(t-nT) e^{-st} dt$$

or,

$$X^*(s) = \sum_{n \geq 0} x(nT) e^{-nsT} \text{ starred Laplace transform}$$ (21.13.2)

With the replacement, $z = e^{sT}$, Eq. (21.13.2) is recognized as the $z$-transform of the sequence $x(nT)$,

$$X(z) = X^*(s) \bigg|_{z=e^{sT}} = \sum_{n \geq 0} x(nT) z^{-n} = z\text{-transform}$$ (21.13.3)

Often, the following abused notation is used for this $z$-transform, $X(z) = Z[X(s)]$, referred to as the $z$-transform of a Laplace transform, that is,

$$X(z) = Z[X(s)] = X^*(s) \bigg|_{z=e^{sT}} = \sum_{n \geq 0} x(nT) z^{-n}$$ (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,
21.13. IDEAL SAMPLING, STARRED LAPLACE TRANSFORM, Z-TRANSFORM

Sample \( x(t) \) at the sampling instants \( t_n = nT \) to obtain the sampled signal \( x(nT) \), and finally, perform a \( z \)-transform,

\[
X(s) \xrightarrow{\mathcal{L}^{-1}} x(t) \xrightarrow{\text{sample}} x(nT) \xrightarrow{Z} X(z) = \sum_{n=0}^{\infty} x(nT)z^{-n}
\]

(21.13.5)

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

\[
X(z) = Z \left[ \mathcal{L}^{-1} [X(s)] \right]_{\text{sampled}}
\]

(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 as a discrete-time system with transfer function, \( G_d(z) = Z[G_a(s)] \), and impulse response, \( g_d(n) = g_a(nT) \), as depicted below,

![Diagram of sampling and filtering process]

Indeed, we have the following relationships, assuming causal system and input,

\[
Y(s) = G_a(s)X^* (s)
\]

\[
y(t) = \int_0^\infty g_a(t - t')x^* (t') \, dt' = \sum_{m=0}^{\infty} g_a(t - mT)x(mT)
\]

\[
y(nT) = \sum_{m=0}^{\infty} g_a(nT - mT)x(mT) \implies g_d(n) = g_a(nT)
\]

(21.13.7)

\[
Y^* (s) = [G_a(s)X^*(s)]^* = G_d^* (s)X^*(s) \implies G_d(z) = Z[G_a(s)]
\]

\[
Y(z) = G_d(z)X(z)
\]

where the indicated factorization, \( [G_a(s)X^*(s)]^* = G_d^* (s)X^*(s) \), is valid because one of the factors is already starred.\(^1\) 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,

\[
e^x \approx 1 + x
\]

\[
e^x = \frac{1}{e^{-x}} \approx \frac{1}{1-x}
\]

\[
e^x = \frac{e^{x/2}}{e^{-x/2}} \approx \frac{1 + x/2}{1 - x/2}
\]

It follows that since \( T \) is small, we may make the same approximations,

\[
Z = e^{sT} \approx 1 + sT \quad \implies \quad s = \frac{1}{T}(z - 1) \quad \text{forward Euler}
\]

\[
Z = e^{sT} = \frac{1}{e^{-sT}} \approx \frac{1}{1-sT} \quad \implies \quad s = \frac{1}{T}(1 - z^{-1}) \quad \text{backward Euler}
\]

\[
Z = e^{sT} = \frac{e^{sT/2}}{e^{-sT/2}} \approx \frac{1 + sT/2}{1 - sT/2} \quad \implies \quad s = \frac{2}{T} - \frac{1}{1 + z^{-1}} \quad \text{trapezoidal}
\]

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),

\[
H_d(z) = (1 - z^{-1}) \cdot Z \left[ \frac{H_a(s)}{s} \right]
\]

(21.14.1)

where the notation, \( G_d(z) = Z[G(s)] \), was defined above, that is,

\[
G_d(z) = Z[G(s)] = G^* (s) \bigg|_{z = e^{sT}} = \sum_{n=0}^{\infty} g(nT)z^{-n}
\]

(21.14.2)

\[
G(s) \xrightarrow{\mathcal{L}^{-1}} g(t) \xrightarrow{\text{sample}} g(nT) \xrightarrow{Z} G_d(z) = \sum_{n=0}^{\infty} g(nT)z^{-n}
\]

(21.14.3)

\[
G_d(z) = Z \left[ \mathcal{L}^{-1}[G(s)] \right]_{\text{sampled}}
\]

(21.14.4)

Let us assume that the given (stable and causal) continuous-time system has a proper\(^1\) 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,

\[
H_a(s) = R_0 + \sum_{i=1}^{M} \frac{R_i}{s + p_i}
\]

(21.14.5)

with \( \text{Re}(p_i) > 0 \), so that the corresponding causal impulse response is,

\[
h_a(t) = R_0 \delta(t) + \sum_{i=1}^{M} R_i e^{-p_i t} u(t)
\]

(21.14.6)

\(^1\)We note the properties that, in general, \( G_1 G_2 \neq G_1^* G_2^* \), but, \( [G_1 G_2]^* = G_1^* G_2^* \)

\(^1\)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 $t \geq 0$,

$$y(t) = \sum_{i=1}^{M} R_i e^{-p_i t} \int_{t}^{t'} [R_0 \delta(t-t') + \sum_{i=1}^{M} R_i e^{-p_i(t-t')} u(t-t')] x(t') dt'$$

or,

$$y(t) = R_0 x(t) + \sum_{i=1}^{M} R_i e^{-p_i t} \int_{0}^{t} e^{p_i t'} x(t') dt' = R_0 x(t) + \sum_{i=1}^{M} R_i y_i(t) \quad (21.14.7)$$

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,

$$y_i(t_n) - e^{-p_i T} y_i(t_{n-1}) = e^{-p_i t_n} \int_{t_{n-1}}^{t_n} e^{p_i t'} x(t') dt' \quad (21.14.8)$$

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,

$$e^{-p_i t_n} \int_{t_{n-1}}^{t_n} e^{p_i t'} dt' = e^{-p_i t_n} \frac{e^{p_i t_n} - e^{p_i t_{n-1}}}{p_i} = 1 - e^{-p_i t_{n-1}} = \frac{1 - e^{-p_i T}}{p_i}$$

where we used $t_n - t_{n-1} = T$. We obtain then the zero-order hold approximation of the exact equation (21.14.8),

$$y_i(t_n) - e^{-p_i T} y_i(t_{n-1}) = \frac{1 - e^{-p_i T}}{p_i} x(t_{n-1}) \quad (21.14.9)$$

and taking $z$-transforms of both sides, we find,

$$Y_i(z) = e^{-p_i T} z^{-1} Y_i(z) = \frac{1 - e^{-p_i T}}{p_i} \left(1 - z^{-1}\right)$$

or,

$$Y_i(z) = \frac{1 - e^{-p_i T}}{p_i} z^{-1} X(z)$$

which can be written as an identity in $z$ in the form,

$$Y_i(z) = \frac{1 - e^{-p_i T}}{p_i} z^{-1} X(z) = \frac{1}{1 - e^{-p_i T} z^{-1}} \left[1 - \frac{1}{1 - e^{-p_i T} z^{-1}}\right] X(z) \quad (21.14.10)$$

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,

$$y(t_n) = R_0 x(t_n) + \sum_{i=1}^{M} R_i y_i(t_n)$$

$$Y(z) = R_0 X(z) + \sum_{i=1}^{M} R_i Y_i(z) = R_0 X(z) + \sum_{i=1}^{M} R_i \left[1 - \frac{1}{1 - e^{-p_i T} z^{-1}}\right] X(z)$$

$$H_d(z) = Y(z) \frac{1}{X(z)} = R_0 + \sum_{i=1}^{M} R_i \left[1 - \frac{1}{1 - e^{-p_i T} z^{-1}}\right]$$

$$H_d(z) = R_0 + \sum_{i=1}^{M} R_i \left[1 - \frac{1}{1 - e^{-p_i T} z^{-1}}\right] = \left(R_0 + \sum_{i=1}^{M} R_i \right) - \sum_{i=1}^{M} R_i \left[1 - \frac{1}{1 - e^{-p_i T} z^{-1}}\right]$$

with discrete-time impulse response, for $n \geq 0$,

$$h_d(n) = R_0 \delta[n] + \sum_{i=1}^{M} R_i \left[1 - e^{-p_i T} \right] e^{-p_i T(n-1)} u[n-1] \quad (21.14.11)$$

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, $G(s) = H_d(s)/s$, and perform its PFE, and follow the progression of steps shown in Eq. (21.14.3),

$$G(s) = \frac{H_d(s)}{s} = R_0 + \sum_{i=1}^{M} R_i \left[1 - \frac{1}{s + p_i}\right] = R_0 + \sum_{i=1}^{M} R_i \left[\frac{1}{s} - \frac{1}{s + p_i}\right]$$

$$g(t) = R_0 u(t) + \sum_{i=1}^{M} R_i \left[u(t) - e^{-p_i T} u(t)\right]$$

$$g(t_n) = R_0 u(t_n) + \sum_{i=1}^{M} R_i \left[u(t_n) - e^{-p_i T n} u(t_n)\right]$$

$$G_d(z) = \sum_{n=0}^{\infty} g(t_n) z^{-n} = \frac{R_0}{1 - z^{-1}} + \sum_{i=1}^{M} R_i \left[\frac{1}{1 - z^{-1}} - \frac{1}{1 - e^{-p_i T} z^{-1}}\right]$$

or,

$$G_d(z) = \left(R_0 + \sum_{i=1}^{M} R_i \right) \frac{1}{1 - z^{-1}} - \sum_{i=1}^{M} R_i \frac{1}{1 - e^{-p_i T} z^{-1}} \quad (21.14.13)$$
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),

\[
(1 - z^{-1}) G_d(z) = \left( R_0 + \sum_{i=1}^{M} \frac{R_i}{p_i} \right) - \sum_{i=1}^{M} R_i \left( 1 - z^{-1} \right) - \frac{1}{z^{-1} - 1} = H_d(z)
\]

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,

\[
G(s) = \frac{R_0}{s} + \sum_{i=1}^{M} \frac{R_i}{p_i} \left( \frac{1}{s} - \frac{1}{s + p_i} \right)
\]

\[
G_d(z) = \frac{R_0}{1 - z^{-1}} + \sum_{i=1}^{M} \frac{R_i}{p_i} \left( \frac{1}{1 - z^{-1}} - \frac{1}{z^{-1} - 1} \right)
\]

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,

\[
H_d(z) = (1 - z^{-1}) \cdot Z \left[ \frac{H_a(s)}{s} \right] = Z \left[ F(s) \right], \quad F(s) = \frac{1 - e^{-sT}}{s} H_a(s)
\]

which, in the time-domain, means that the discrete-time samples \( h_d(n) \) will be,

\[
h_d(n) = f(t_n^+) (21.14.14)
\]

After using the differentiation and delay properties of Laplace transforms, we find from the definition of \( F(s) \),

\[
sF(s) = H_a(s) - e^{-sT} H_a(s) \Rightarrow \hat{f}(t) = h_a(t) - h_a(t - T)
\]

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,

\[
\hat{f}(t) = \int_{t-T}^{t} h_a(t') dt'
\]

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,

\[
g(t) = \int_{0}^{t} h_a(t') dt' \Rightarrow f(t) = \int_{0}^{t} h_a(t') dt' - \int_{0}^{t-T} h_a(t') dt' = \int_{t-T}^{t} h_a(t') dt'
\]

Thus, evaluating Eq. (21.14.14) at \( t = nT \), or rather, at \( t_n^+ \), we have, for \( n \geq 0 \),

\[
h_d(n) = f(t_n^+) = \int_{t_n^+ - T}^{t_n^+} h_a(t') dt' \quad \text{(zero-order hold)} (21.14.16)
\]

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 \( t \geq 0 \),

\[
f(t) = \int_{t-T}^{t} h_a(t') dt' = \int_{t-T}^{t} R_0 \delta(t') + \sum_{i=1}^{M} R_i e^{-p_i t} u(t') dt'
\]

\[
= R_0 \int_{t-T}^{t} \delta(t') dt' + \sum_{i=1}^{M} \frac{R_i}{p_i} \left( e^{-p_i t} - e^{-p_i (t-T)} \right)
\]

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),

\[
h_d(n) = f(t_n^+) = R_0 \delta[n] + \sum_{i=1}^{M} \frac{R_i}{p_i} (1 - e^{-p_i (t-nT)}) u[n - 1] (21.14.17)
\]

**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,

\[
g_{zoh}(t) = u(t) - u(t - T) \Rightarrow g_{zoh}(s) = \frac{1 - e^{-sT}}{s} (21.14.18)
\]

so that the staircase output becomes, for a causal input,

\[
x_{zoh}(t) = \int_{0}^{t} x^*(t') g_{zoh}(t - t') dt' = \int_{0}^{t} \sum_{n=0}^{\infty} x(nT) \delta(t' - nT) \int_{0}^{t} \sum_{n=0}^{\infty} x(nT) \int_{t-nT}^{t-nT+T} u(t - T) dt'
\]

\[
= \sum_{n=0}^{\infty} x(nT) g_{zoh}(t - nT) = \sum_{n=0}^{\infty} x(nT) \left[ u(t - nT) - u(t - nT - T) \right] (21.14.19)
\]

The zero-order hold discretization formula (21.14.1) follows if the staircase-reconstructed signal \( x_{zoh}(t) \) is filtered further by the analog filter \( H_a(s) \), as shown below, followed by the sampling of the resulting analog output.
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,

\[ H_d(z) = (1 - z^{-1}) \cdot Z \left[ \frac{H_a(s)}{s} \right]_{\text{sampled}} \]  \hspace{1cm} (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,

\[ Z^{-1} \left[ \frac{H_d(z)}{1 - z^{-1}} \right] = L^{-1} \left[ \frac{H_a(s)}{s} \right]_{\text{sampled}} \]  \hspace{1cm} (step invariance) (21.15.2)

Since the Laplace transform of a unit-step \(u(t)\) is \(1/s\), it follows that the quantity \(H_d(s)/s\) 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,

\[ \frac{1 - e^{-sT}}{s} \approx 1 - (1 - sT) = T \]

which leads to the approximation of the zero-order hold,

\[ H_d(z) = Z \left[ (1 - e^{-sT}) \frac{H_a(s)}{s} \right] \approx Z \left[ T H_a(s) \right] \]

or, in the time domain,

\[ h_d(n) \approx T h_a(nT) \]  \hspace{1cm} (impulse invariance) (21.15.3)

The same result can also be obtained by approximating the integral in Eq. (21.14.16) as follows,

\[ h_d(n) = \int_{nT-T}^{nT} h_a(t') \, dt' \approx T h_a(nT-T) \]  \hspace{1cm} (21.15.4)

A slightly better approximation results if we use the trapezoidal approximation to the integral,

\[ h_d(n) = \int_{nT-T}^{nT} h_a(t') \, dt' \approx T \frac{h_a(nT) + h_a(nT-T)}{2} \]

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\(^1\) of the impulse invariance method, for \(n \geq 0\),

\[ h_d(n) \approx T h_a(nT) - \delta[n] \frac{T}{2} h_a(0^+) \]  \hspace{1cm} (impulse invariance) (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.

21.16 First-Order Hold

The first-order hold has a discrete-time transfer function similar to Eq. (21.14.1),

\[H_d(z) = \frac{(1 - z^{-1})^2}{Tz^{-1}} \cdot \frac{H_a(s)}{s} \quad \text{(first-order hold)} \tag{21.16.1}\]

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 \( t \geq 0 \),

\[H_a(s) = R_0 + \sum_{i=1}^{M} \frac{R_i}{s + p_i}, \quad \text{Re}(p_i) > 0 \tag{21.16.2}\]

\[y(t) = R_0x(t) + \sum_{i=1}^{M} R_iy_i(t)\]

with the partial output \( y_i(t) \) satisfying,

\[y_i(t_n) - e^{-p_iT}y_i(t_{n-1}) = \frac{1}{T_{p_i}} \left[ (e^{-p_iT} + p_iT - 1)x(t_n) + (1 - e^{-p_iT} - p_iTe^{-p_iT})x(t_{n-1}) \right] \tag{21.16.3}\]

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 \( x(t_n) \),

\[x(t') = x(t_{n-1}) + \frac{x(t_n) - x(t_{n-1})}{T}(t' - t_{n-1}), \quad t_{n-1} \leq t' \leq t_n\]

The integral (21.16.3) can then be done exactly, resulting in the following difference equation, and its \( z \)-transform,

\[y_i(t_n) - e^{-p_iT}y_i(t_{n-1}) = \frac{1}{T_{p_i}} \left[ (e^{-p_iT} + p_iT - 1)x(t_n) + (1 - e^{-p_iT} - p_iTe^{-p_iT})x(t_{n-1}) \right] \]

\[Y_i(z) = \frac{1}{T_{p_i}} \left[ (e^{-p_iT} + p_iT - 1) + (1 - e^{-p_iT} - p_iTe^{-p_iT})z^{-1} \right] X(z) \tag{21.16.4}\]

Thus, the overall output and discrete-time transfer function will be,

\[Y(z) = R_0X(z) + \sum_{i=1}^{M} R_iY_i(z)\]

\[Y(z) = R_0X(z) + \sum_{i=1}^{M} R_i \frac{(e^{-p_iT} + p_iT - 1) + (1 - e^{-p_iT} - p_iTe^{-p_iT})z^{-1}}{1 - e^{-p_iT}z^{-1}} X(z) \]

\[H_d(z) = \frac{Y(z)}{X(z)} = R_0 + \sum_{i=1}^{M} \frac{R_i}{T_{p_i}} \left[ (e^{-p_iT} + p_iT - 1) + (1 - e^{-p_iT} - p_iTe^{-p_iT})z^{-1} \right] \]

Next, we show the Eq. (21.16.5) is identical to (21.16.1) under the substitutions of Eq. (21.9.8),

\[\frac{1}{s + p_i} \Rightarrow \frac{1}{1 - e^{-p_iT}z^{-1}} \quad \text{and} \quad \frac{1}{s} \Rightarrow \frac{1 - z^{-1}}{1 - Tz^{-1}} \tag{21.16.5}\]

For the given \( H_a(s) \) in Eq. (21.16.2), we have,

\[H_a(s) = R_0 + \sum_{i=1}^{M} \frac{R_i}{s} \cdot Z \left[ \frac{1}{s^2} \right] = R_0 + \sum_{i=1}^{M} \frac{R_i}{s} \cdot Z \left[ \frac{1}{s^2} \right] + \frac{1}{s^2}\]

and making the substitutions (21.16.6), we find the corresponding \( z \)-transform,

\[Z \left[ \frac{H_a(s)}{s^2} \right] = R_0 + \sum_{i=1}^{M} \frac{R_i}{s^2} \cdot Z \left[ \frac{1}{s^2} \right] \]

Thus, according to Eq. (21.14.1), we must have,

\[H_d(z) = \frac{(1 - z^{-1})^2}{Tz^{-1}} \cdot Z \left[ \frac{H_a(s)}{s^2} \right] = \frac{1}{s} \cdot Z \left[ \frac{1}{s^3} \right] \]

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,

\[H_d(z) = \frac{(1 - z^{-1})^2}{Tz^{-1}} \cdot Z \left[ \frac{H_a(s)}{s} \right] = (1 - z^{-1}) \cdot Z \left[ \frac{1}{s} \right] = (1 - z^{-1}) \cdot \frac{Tz^{-1}}{(1 - z^{-1})^2} = \frac{T}{z - 1} \]

while for the first-order hold we have,

\[H_d(z) = \frac{(1 - z^{-1})^2}{Tz^{-1}} \cdot Z \left[ \frac{H_a(s)}{s^2} \right] = (1 - z^{-1})^2 \cdot Z \left[ \frac{1}{s^3} \right] \]

\[= (1 - z^{-1})^2 \cdot \frac{T^2z^{-1}(1 + z^{-1})}{2(1 - z^{-1})^3} = \frac{T}{2} \frac{1 + z^{-1}}{z - 1} \]

where we used the transformations,

\[\frac{1}{s^3} \xrightarrow{\text{sample}} \frac{1}{2T^2} u(t) \quad \frac{1}{2T^2} u(t) \xrightarrow{\text{sample}} \frac{1}{2} u(nT) \]

\[\frac{1}{s^3} \xrightarrow{\text{sample}} \frac{1}{2} T^2u(t) \quad \frac{1}{2} T^2u(t) \xrightarrow{\text{sample}} \frac{1}{2} (nT)^2u(nT) \]

\[\frac{1}{s^3} \xrightarrow{\text{sample}} \frac{T^2}{2} \frac{z^{-1}}{(1 - z^{-1})^2} \]

\[\frac{1}{s^3} \xrightarrow{\text{sample}} \frac{T^2z^{-1}(1 + z^{-1})}{2(1 - z^{-1})^3} \]
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,

\[
\lim_{s \to \infty} \left[ \frac{T z^{-1}}{(1 - z^{-1})^2} H_d(z) \right] = \mathcal{L}^{-1} \left[ \frac{H_a(s)}{s^2} \right]_{\text{sampled}}
\]

(ramp invariance) \ (21.17.1)

This follows by realizing that \( 1/s^2 \) is the Laplace transform of the ramp input, \( t u(t) \), and that \( T z^{-1}/(1 - z^{-1})^2 \) 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, \( y(t) \leftrightarrow Y(s) \), and initial conditions \( y(0^-), y'(0^-) \), we have the limits,

\[
y(0^+) = \lim_{s \to \infty} [sY(s)]
\]

\[
y'(0^+) = \lim_{s \to \infty} [sY(s) - y(0^-)]
\]

where the second one follows from the first by noting that \([sY(s) - y(0^-)]\) is the Laplace transform of \( y(t) \). The solution of the differential equation (21.1.3) for an arbitrary causal input \( x(t) \), subject to the initial conditions, \( y(0^-), y'(0^-) \), can be expressed in the \( s \)-domain in the following form,

\[
y(0^+) = \frac{s + a_1}{s^2 + a_1 s + a_2} y(0^-) + \frac{b_0 s^2 + b_1 s + b_2}{s^2 + a_1 s + a_2} X(s)
\]

\[
y'(0^+) = \frac{(s + a_1) y(0^-) + y'(0^-)}{s^2 + a_1 s + a_2} + \frac{b_0 s^2 + b_1 s + b_2}{s^2 + a_1 s + a_2} X(s)
\]

after a long-division step. Thus,

\[
y(s) = \frac{(s + a_1) y(0^-) + y'(0^-)}{s^2 + a_1 s + a_2} + b_0 X(s) + \frac{b_1 - a_1 b_0}{s^2 + a_1 s + a_2} X(s)
\]

\[
(21.18.1)
\]

multiplying by \( s \), we have,

\[
sY(s) = \frac{s + a_1}{s^2 + a_1 s + a_2} s y(0^-) + \frac{b_0 s^2 + b_1 s + b_2}{s^2 + a_1 s + a_2} s X(s)
\]

Taking the limit as \( s \to \infty \), we have,

\[
\lim_{s \to \infty} [sY(s)] = \lim_{s \to \infty} \left[ \frac{s + a_1}{s^2 + a_1 s + a_2} s y(0^-) + \frac{b_0 s^2 + b_1 s + b_2}{s^2 + a_1 s + a_2} s X(s) \right] = y(0^-) + b_0 x(0^+)
\]

(21.18.2)

where we assumed that the limit \( x(0^+) = \lim_{s \to -\infty} [sX(s)] \) 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 \( y'(0^-) \) part, it follows from Eq. (21.18.1) that,

\[
sY(s) = \frac{s^2 y(0^-) - s a_2 y'(0^-)}{s^2 + a_1 s + a_2} + b_0 s^2 X(s) + \frac{(b_1 - a_1 b_0) s + (b_2 - a_2 b_0)}{s^2 + a_1 s + a_2} X(s)
\]

Assuming that the limits, \( x(0^+) = \lim_{s \to -\infty} [sX(s)] \), and \( x'(0^+) = \lim_{s \to -\infty} [s^2 X(s)] \), both exist, it follows that the limit of the first term will be \( y(0^-) \), the limit of the second term, \( b_0 x(0^+) \), and the limit of the third term, \( (b_1 - a_1 b_0) x(0^+) \). Thus, we have,

\[
y'(0^+) = \lim_{s \to \infty} [sY(s) - y(0^-)] = y(0^-) + b_0 x(0^+) + (b_1 - a_1 b_0) x(0^+)
\]

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, \( \mathbf{v}(t) = [v_1(t), v_2(t), v_3(t)]^T \) that satisfies the matrix equations,

\[
\frac{d\mathbf{v}(t)}{dt} = A \mathbf{v}(t) + B x(t)
\]

\[
y(t) = C \mathbf{v}(t) + D x(t)
\]

(21.18.2)

where \( A, B, C, D \) are defined by

\[
A = \begin{bmatrix} -a_1 & -a_2 & -a_3 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}
\]

(21.18.3)

\[
C = [b_1 - b_0 a_1, b_2 - b_0 a_2, b_3 - b_0 a_3], \quad D = b_0
\]

in terms of the transfer function coefficients,

\[
H(s) = \frac{b_0 s^3 + b_1 s^2 + b_2 s + b_3}{s^3 + a_1 s^2 + a_2 s + a_3}
\]

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 \( t = 0 \),

\[
\mathbf{v}(0^-) = \mathbf{v}(0^+)
\]

(21.18.4)
Differentiating \( \dot{y}(t) \) twice, and substituting, \( \ddot{y} = A\dot{v} + B\dot{x} \), we find,

\[
\begin{align*}
\dot{y}(t) &= C\dot{v}(t) + Dx(t) \\
\ddot{y}(t) &= CA\dot{v}(t) + CB\dot{x}(t) + D\ddot{x}(t)
\end{align*}
\]

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,

\[
\begin{align*}
y(0^+) - y(0^-) &= Dx(0^+) \\
\dot{y}(0^+) - \dot{y}(0^-) &= CBx(0^+) + D\ddot{x}(0^-) \\
\ddot{y}(0^+) - \ddot{y}(0^-) &= CABx(0^+) + CB\dot{x}(0^-) + Dx(0^-)
\end{align*}
\] (21.18.5)

Using Eq. (21.18.3), we recognize that these are exactly equivalent to Eq. (21.2.1), indeed,

\[
CAB = [b_1 - b_0a_1, b_2 - b_0a_2, b_3 - b_0a_3]
\]
\[
\begin{bmatrix}
-a_1 & -a_2 & -a_3 \\
1 & 0 & 0 \\
0 & 1 & 0
\end{bmatrix}
\]
\[
= b_2 - b_0a_2 - a_1(b_1 - b_0a_1)
\]
\[
CB = [b_1 - b_0a_1, b_2 - b_0a_2, b_3 - b_0a_3]
\]
\[
\begin{bmatrix}
1 \\
0 \\
0
\end{bmatrix}
\]
\[
= b_1 - b_0a_1
\]
\[
D = b_0
\]

### 21.19 MATLAB function – c2d2

The supplied MATLAB function \texttt{c2d2} in the ISP2e toolbox is a simple alternative to the built-in function \texttt{c2d} for converting a first-order or second-order CT system to a DT system. Its inputs are the analog transfer function coefficients, \( B = [B_0, B_1, B_2] \), \( A = [a_1, A_1, A_2] \), the time-step \( T \), and the discretization method, and its outputs are the numerator and denominator coefficients of the discrete system, \( b = [b_0, b_1, b_2] \), and, \( a = [a_1, a_2] \), for example,

\[
\begin{align*}
B &= [2, 1, 1]; \\
A &= [1, 4, 3]; \\
T &= 0.01;
\end{align*}
\]

\[
[b,a] = \text{c2d2}(B,A,T,’tr’); \\ % trapezoidal (default) \\
[b,a] = \text{c2d2}(B,A,T,’fe’); \\ % forward Euler \\
[b,a] = \text{c2d2}(B,A,T,’be’); \\ % backward Euler \\
[b,a] = \text{c2d2}(B,A,T,’zoh’); \\ % zero-order hold
\]
22.1 FEEDBACK CONTROL SYSTEMS

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

\[
G_c(s) = k_p + \frac{k_i}{s} + k_d s
\]

(PID controller) (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.
- Increasing \( k_d \) 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,

\[
\text{lim } t \to \infty e(t) = \text{lim } s \to 0 [sE(s)] = \text{lim } s \to 0 [sH_{\text{err}}(s)R(s)] = \text{lim } s \to 0 \left[ \frac{sR(s)}{1 + G_c(s)G(s)} \right]
\] (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,

- step input: \( \text{lim } t \to \infty e(t) = \text{lim } s \to 0 \left[ \frac{sR(s)}{1 + G_c(s)G(s)} \right] = \text{lim } s \to 0 \left[ \frac{s}{s + G(s)(k_p + k_i + k_d s^2)} \right] \)

- ramp input: \( \text{lim } t \to \infty e(t) = \text{lim } s \to 0 \left[ \frac{sR(s)}{1 + G_c(s)G(s)} \right] = \text{lim } s \to 0 \left[ \frac{1}{s + G(s)(k_p + k_i + k_d s^2)} \right] \)

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
22.3. DIGITAL CONTROL SYSTEMS

Fig. 22.3.1 Typical digital control system.

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, $G_c(z) = G_c^*(s) \bigg|_{z=e^{T}}$. The corresponding z-domain transfer functions of the feedback system are obtained as follows:

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

$$Y(s) = G(s) \left[ G_{zh}(s) F^*(s) + F_{dist}^*(s) \right]$$

$$F(s) = G_c(s) E^*(s) = G_c(s) \left[ R^*(s) - Y^*(s) \right]$$

and taking the starred-Laplace transforms,\(^1\)

$$Y^*(s) = \left[ G(s) G_{zh}(s) \right]^* F^*(s) + G^* (s) F_{dist}^*(s)$$

$$F^*(s) = G_c^*(s) \left[ R^*(s) - Y^*(s) \right]$$

\(^1\) using the properties that, in general, $[G_1 G_2]^* \neq G_1^* G_2^*$, but, $[G_1 G_2]^* = G_1^* G_2^*$

Denoting, $G_d(z) = \left[ G(s) G_{zh}(s) \right]^*$, and, $G(z) = G^*(s)$, $Y(z) = Y^*(s)$, etc., we may rewrite the above as,

$$Y(z) = G_d(z) F(z) + G(z) F_{dist}(z)$$

$$F(z) = G_c(z) \left[ R(z) - Y(z) \right]$$

which leads to the feedback discrete-time transfer functions:

$$Y(z) = H_d(z) R(z) + H_{dist}(z) F_{dist}(z)$$

$$E(z) = H_{err}(z) R(z) - H_{dist}(z) F_{dist}(z)$$

(22.3.1)

with,

$$H_d(z) = \frac{G_c(z) G_d(z)}{1 + G_c(z) G_d(z)} = \text{closed-loop}$$

$$H_{err}(z) = \frac{G_c(z) G_d(z)}{1 + G_c(z) G_d(z)} = \text{error}$$

$$H_{dist}(z) = \frac{G(z)}{1 + G_c(z) G_d(z)} = \text{disturbance}$$

(22.3.2)

where the discrete-time transfer functions $G_d(z)$, $G(z)$ are defined by,

$$G_d(z) = Z \left[ G_{zh}(s) G(s) \right] = \left[ G_{zh}(s) G(s) \right]^* \bigg|_{z=e^{T}}$$

$$G(z) = Z \left[ G(s) \right] = G^*(s) \bigg|_{z=e^{T}}$$

(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.

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,

$$G_c(z) = k_p + \frac{k_i}{s} + k_d s = \text{analog PID controller}$$

(22.3.4)

$$G_c(z) = k_p + \frac{k_i T}{T} \left( \frac{1 + z^{-1}}{1 - z^{-1}} \right) + \frac{k_d}{T} (1 - z^{-1}) = \text{digital PID controller}$$
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, \( av(t) \), represents a drag force taken to be proportional to the velocity, and \( \alpha \) is a drag coefficient,

\[
\frac{dv(t)}{dt} = f(t) - \alpha v(t) \quad \Rightarrow \quad G(s) = \frac{V(s)}{F(s)} = \frac{1}{s + \alpha} \tag{22.4.1}
\]

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,

\[
G_c(s) = k_p + \frac{k_i}{s} + k_d s \tag{PID controller}
\]

we will use a PI controller defined by,

\[
G_c(s) = k_p + \frac{k_i}{s} \tag{PI controller} \tag{22.4.4}
\]

The figure below shows the closed-loop responses when the reference speed is set to 40 mph for a period 0 ≤ \( t \) ≤ 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,

\[
H(s) = \frac{G_c(s)G(s)}{1 + G_c(s)G(s)} = \frac{k_p s + k_i}{s^2 + (k_p + \alpha) s + k_i} \tag{22.4.5}
\]

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,

\[
Y(s) = H(s)R(s) = \frac{H(s)}{s} \Rightarrow y(t) = \text{ilaplace}(Y(s))
\]

For the two cases in the above figure, we find for, \( k_p = 2, k_i = 2 \),

\[
H(s) = \frac{2s + 2}{s^2 + 3s + 2} = \frac{2(s + 1)}{(s + 1)(s + 2)} = \frac{2}{s + 2}
\]

\[
Y(s) = \frac{H(s)}{s} = \frac{2}{s^2 + 2} = \frac{1}{s - 1} \Rightarrow y(t) = 1 - e^{-2t}, \quad t \geq 0
\]

while for, \( k_p = 2, k_i = 4.5 \), we have,

\[
H(s) = \frac{2s + 4.5}{s^2 + 3s + 4.5}
\]

\[
Y(s) = \frac{H(s)}{s} = \frac{1 - \frac{s + 1}{s^2 + 3s + 4.5}}{s} = 1 - \left[ \frac{1}{s} \left( 3 - j \right) e^{-1.5t} + \frac{1}{s} \left( 3 + j \right) e^{1.5t} \right] - \left[ \frac{1}{s} \left( 3 - j \right) e^{-1.5t} + \frac{1}{s} \left( 3 + j \right) e^{1.5t} \right]
\]

\[
y(t) = 1 - 2 \text{Re} \left[ \frac{1}{6} (3 - j) e^{-1.5t} e^{1.5jt} \right] = 1 - 1.5 e^{-1.5t} [3 \cos(1.5t) - \sin(1.5t)]
\]

The MATLAB code used to generate the above graphs was as follows,
22.4. EXAMPLES

where the system transfer function is,

\[
G(s) = \frac{1}{s(s + a)}
\]  

The control system is implemented as the feedback system shown below,

where the overall reference input \( r(t) \) represents the desired reference angle \( \theta_{\text{ref}}(t) \), 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., \( y(t) \rightarrow r(t) \), or for the tracking error signal, \( e(t) = r(t) - y(t) \rightarrow 0 \).

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:

\[
G_c(s) = k_p + \frac{k_i}{s} + k_ds
\]  

(22.4.8)

Following the discussion of Sec. 22.1, we can show that the overall transfer relationships from the two inputs \( R(s) \), \( F_{\text{dist}}(s) \) to the two outputs \( Y(s) \) and \( E(s) \) are as follows:

\[
Y(s) = H(s)R(s) + H_{\text{dist}}(s)F_{\text{dist}}(s)
\]

\[
E(s) = H_{\text{err}}(s)(R(s) - H_{\text{dist}}(s)F_{\text{dist}}(s))
\]  

(22.4.9)

where for the PID case in particular,

\[
H(s) = \frac{G_c(s)G(s)}{1 + G_c(s)G(s)} = \frac{k_ds^2 + k_ps + k_i}{s^3 + (a + k_d)s^2 + k_ps + k_i} = \text{closed-loop}
\]

\[
H_{\text{err}}(s) = \frac{1}{1 + G_c(s)G(s)} = \frac{s^2(s + a)}{s^3 + (a + k_d)s^2 + k_ps + k_i} = \text{error}
\]  

(22.4.10)

\[
H_{\text{dist}}(s) = \frac{G(s)}{1 + G_c(s)G(s)} = \frac{s}{s^3 + (a + k_d)s^2 + k_ps + k_i} = \text{disturbance}
\]

The steady-state tracking error due to a particular reference input \( r(t) \) is found to be,

\[
\lim_{t \to \infty} e(t) = \lim_{t \to \infty} sH_{\text{err}}(s)R(s) = \lim_{t \to \infty} \left[ \frac{s^3(s + a)R(s)}{s^3 + (a + k_d)s^2 + k_ps + k_i} \right]
\]  

(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,

\[
G_d(z) = Z \left[ \frac{G_{zh}(s)G(s)}{G(s)} \right] = \frac{(aT - e^{-aT} - 1)z^{-1} + (1 - e^{-aT} - aT e^{-aT})z^{-2}}{a^2(1 - z^{-1})(1 - a^2T z^{-1})} 
\]

(22.4.12)

\[
G(z) = Z\left[\frac{G(s)}{s}\right] = \frac{(1 - e^{-aT})z^{-1}}{a(1 - z^{-1})(1 - e^{-aT}z^{-1})} 
\]

(22.4.13)

and for \( G_c(z) \), we may use the digital PID approximation given by Eq. (22.3.4),

\[
G_c(z) = \frac{k_p + \frac{k_i}{T} \left(1 + z^{-1}\right) + \frac{k_d}{T} (1 - z^{-1})}{\text{discrete PID controller}} 
\]

(22.4.14)

**Computer Experiments**

With the above background information, please carry out the following experiments.

(a) Starting with the parameter values,

\[
a = 2, \quad k_p = 10, \quad k_i = 5, \quad k_d = 3 
\]

(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 \( H_{err}(s) \), and disturbance transfer function \( H_{dist}(s) \), using, for example, the MATLAB code:

```
\[ a = 2; \quad k_p = 10; \quad k_i = 5; \quad k_d = 3; \quad s = tf('s'); \quad G = 1/(s(s+a)); \quad G_c = k_p + k_i/s + k_d/s; \quad H = \text{minreal}(G^c/G/(1+G_c^c)); \quad H_{err} = \text{minreal}(1/(1+G_c^c)); \quad H_{dist} = \text{minreal}(G/(1+G_c^c)); \]
```

where the \text{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 \text{roots} or \text{pzmap}, e.g.,

\[
p = \text{roots}(\text{den}(\text{H})); \quad \% \ \text{H.den(1)} \ is \ the \ vector \ of \ denominator \ coefficients \ of \ H(s) \]

\[
p = \text{pzmap}(\text{H}); \]

Next, define \( t \) as a vector of 1001 equally-spaced time samples spanning the interval \( 0 \leq t \leq 20 \). Using the \text{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,

\[
u = 0(t) \quad \text{double}(t)=0; 
\]

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.

\[
r(t) = u(t) + u(t-t_0) \quad (\text{switches from } r = 1 \text{ to } r = 2 \text{ at } t = t_0) 
\]

\[
r(t) = 0.11u(t) \quad (\text{uniformly moving aircraft}) 
\]

\[
r(t) = \arctan(0.1)u(t) \quad (\text{uniformly moving with correct angle}) 
\]

(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 \( 10 \leq t \leq 14 \) 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 \( H(s) \), using the \text{lsim} command, as follows

\[
y = \text{lsim}(H,r,t); 
\]

On the same graph, plot both \( y(t) \) and \( r(t) \) with different line styles, 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, \( r(t) = 0.11u(t) \), 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 \( k_ds \) of the PID controller, the following modified variant is often used:

\[
G_c(s) = k_p + \frac{k_i}{s} + \frac{k_ds}{s + \tau} 
\]

(22.4.16)

where \( \tau \) is a very small quantity. In this case, show that the closed-loop transfer function becomes,

\[
H(s) = \frac{(k_d + \tau k_p)s^2 + (k_p + \tau k_i)s + k_i}{\tau s^4 + (\tau a + 1)s^3 + (a + k_d + \tau k_p)s^2 + (k_p + \tau k_i)s + k_i} 
\]

(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, $4 \leq t \leq 6$, 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),

\[
\begin{align*}
fdist &= 2*(u(t-4)-u(t-6)); \quad \% \text{wind gust} \\
seed &= 2016; \quad \text{rng(seed);} \quad \% \text{initialize random number generator} \\
fdist &= \text{randn(size(t));} \quad \% \text{zero-mean, unit-variance noise}
\end{align*}
\]

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:

\[
\begin{align*}
y_{dist} &= \text{lsim}(H_{dist},fdist,t); \\
y &= \text{lsim}(H,r,t); \\
y_{tot} &= y + y_{dist};
\end{align*}
\]

For each of the resulting eight cases (2 disturbances $\times$ 4 reference signals), plot the signals $y_{tot}(t)$ 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 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`:

\[
\begin{align*}
p &= \text{pzmap}(H); \quad \% \text{poles of } H \\
teff &= 1/\text{abs}(\text{max}(\text{real}(p))); \quad \% \text{effective time constant} \\
Ts &= \text{teff}/20; \quad \% \text{initial choice of } T_s
\end{align*}
\]

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:

\[
\begin{align*}
T &= Ts; \quad \% \text{to be changed later to } T = Ts/2, 2*Ts, 3*Ts \\
Gd &= \text{c2d}(G,T); \quad \% \text{ZOH discretization by default}
\end{align*}
\]

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:

\[
\begin{align*}
z &= \text{tf}(\cdot); \\
Gc &= kp + ki*T*(z+1)/(z-1)/2 + kd*(z-1)/z/T; \\
Hd &= \text{feedback}(Gc*Gd,1);
\end{align*}
\]

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 $0 \leq t \leq 20$, we must redefine:

\[
\text{tn} = 0:T:20;
\]

Using this new vector of $t$'s, construct the discrete-time reference inputs:

\[
\begin{align*}
r_n &= u(t_n)+2u(t_n-10) \\
r_n &= 0.1t_n \\
r_n &= \arctan(0.1t_n)
\end{align*}
\]

and compute the output $y_n$ of the discrete closed-loop system:

\[
\begin{align*}
y_n &= \text{lsim}(Hd,rn); \\
\% y_n &= \text{filter}(Hd.num{1}, Hd.den{1}, rn); \quad \% \text{alternative evaluation of } y_n
\end{align*}
\]

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,

\[
\begin{align*}
\text{figure; plot(tn,yn,'r-', t,y,'b-');}
\end{align*}
\]

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

- Step response, $k_p = 10$, $k_i = 5$, $k_d = 3$
- Tracking step changes
- Tracking error
- Ramp tracking with correct angle
- Ramp tracking, $k_i = 0$
- Tracking error, $k_i = 0$

Equations:

- Step response: $y(t) = y(t)$
- Tracking step changes: $y(t) = y(t)$
- Tracking error: $e(t) = y(t) - r(t)$
- Ramp tracking: $y(t) = r(t)$
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,

\[
ml^2 \ddot{\theta} = \tau = mL(g \sin \theta - \ddot{x} \cos \theta) \Rightarrow \ddot{\theta} = \frac{g}{L} \sin \theta - \frac{\ddot{x}}{L} \cos \theta \tag{22.4.18}
\]

For small deviations \( \theta \) 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,

\[
\dot{\theta} = \omega_0^2 \theta - \frac{\ddot{x}}{L}, \quad \omega_0 = \sqrt{\frac{g}{L}} \tag{22.4.19}
\]

where \( \omega_0 \) 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, \( e^{i\omega_0 t}, e^{-i\omega_0 t} \), 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,

\[
x(t) \xrightarrow{G(s)} \theta(t) \quad G(s) = \frac{s^2/L}{s^2 - \omega_0^2} \tag{22.4.20}
\]

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.
22. EXAMPLES

![Control System Diagram]

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 \( k_p, k_i \),

\[
G_c(s) = k_p L + \frac{k_i L}{s}
\]  

(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,

\[
H(s) = \frac{\Theta(s)}{X(s)} = \frac{G(s)}{1 + G_c(s) G(s)} = \frac{s^2/L}{(k_p - 1)s^2 + k_i s + \omega_0^2}
\]  

(22.4.22)

\[
H_c(s) = \frac{E(s)}{X(s)} = \frac{1}{1 + G_c(s) G(s)} = \frac{s^2 - \omega_0^2}{(k_p - 1)s^2 + k_i s + \omega_0^2}
\]

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 \( \alpha_1, \alpha_2 \) arbitrary positive numbers, one can determine the gains \( k_p, k_i \) by requiring the identity in \( s \),

\[
(k_p - 1)s^2 + k_i s + \omega_0^2 = (k_p - 1)(s + \alpha_1)(s + \alpha_2)
\]

which leads to the following solution for \( k_p, k_i \) in terms of the given \( \alpha_1, \alpha_2 \),

\[
k_p = 1 + \frac{\omega_0^2}{\alpha_1 \alpha_2}
\]

\[
k_i = \omega_0^2 \left( \frac{1}{\alpha_1} + \frac{1}{\alpha_2} \right)
\]  

(22.4.23)

With these choices for \( k_p, k_i \), the transfer functions (22.4.22) read,

\[
H(s) = \frac{\alpha_1 \alpha_2}{L\omega_0^2} \frac{s^2}{(s + \alpha_1)(s + \alpha_2)}
\]  

(22.4.24)

\[
H_c(s) = -\frac{\alpha_1 \alpha_2}{\omega_0^2} \frac{s^2 - \omega_0^2}{(s + \alpha_1)(s + \alpha_2)}
\]

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 \( t \geq 0 \),

\[
\theta(t) = \frac{x_0 \alpha_1 \alpha_2}{L \omega_0^2 (\alpha_1 - \alpha_2)} \left[ \alpha_1 e^{-\alpha_1 t} - \alpha_2 e^{-\alpha_2 t} \right]
\]

\[
e(t) = x_0 - \frac{x_0}{\omega_0^2 (\alpha_1 - \alpha_2)} \left[ \alpha_2 (\alpha_1^2 - \omega_0^2) e^{-\alpha_1 t} - \alpha_1 (\alpha_2^2 - \omega_0^2) e^{-\alpha_2 t} \right]
\]

(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 \( t \geq 0 \),

\[
\theta(t) = \frac{v_0 \alpha_1 \alpha_2}{L \omega_0^2 (\alpha_1 - \alpha_2)} \left[ e^{-\alpha_1 t} - e^{-\alpha_2 t} \right]
\]

\[
e(t) = v_0 t - \frac{v_0 (\alpha_1 + \alpha_2)}{\alpha_1 \alpha_2} + \frac{v_0 [\alpha_2^2 (\alpha_1^2 - \omega_0^2) - \alpha_1^2 (\alpha_2^2 - \omega_0^2)]}{\alpha_1 \alpha_2 \omega_0^2} e^{-\alpha_1 t} - \frac{v_0 [\alpha_2^2 (\alpha_1^2 - \omega_0^2) - \alpha_1^2 (\alpha_2^2 - \omega_0^2)]}{\alpha_1 \alpha_2 \omega_0^2} e^{-\alpha_2 t}
\]

(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 \( t \geq 0 \),

\[
\theta(t) = \theta_0 + \frac{\theta_0}{\alpha_1 - \alpha_2} \left[ \alpha_2 e^{-\alpha_1 t} - \alpha_1 e^{-\alpha_2 t} \right]
\]

\[
e(t) = \frac{1}{2} a_0 t^2 - v_1 t + x_1 = \frac{a_0}{\alpha_1 \alpha_2} \left[ \alpha_2^2 (\alpha_1^2 - \omega_0^2) - \alpha_1^2 (\alpha_2^2 - \omega_0^2) \right] e^{-\alpha_1 t} - \frac{a_0}{\alpha_1 \alpha_2} \left[ \alpha_2^2 (\alpha_1^2 - \omega_0^2) - \alpha_1^2 (\alpha_2^2 - \omega_0^2) \right] e^{-\alpha_2 t}
\]

\[
\theta_0 = \frac{a_0}{g}, \quad v_1 = \frac{a_0 (\alpha_1 + \alpha_2)}{\alpha_1 \alpha_2}, \quad x_1 = a_0 \left( \frac{1}{\alpha_1^2} + \frac{1}{\alpha_2^2} + \frac{1}{\alpha_1 \alpha_2} - \frac{1}{\omega_0^2} \right)
\]  

(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 \( \theta_0 \) 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, \( \theta_0 = a_0 / g \).
Examples

(d) With $\alpha_1 \neq \alpha_2$, suppose that the cart is moving back and forth sinusoidally with a frequency $\omega$, 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,

$$
x(t) = A \sin(\omega t) = A \text{Im}[e^{j\omega t}]
$$

$$
\theta(t) = \frac{A\alpha_1\alpha_2}{L\omega_0^2} \text{Im} \left[ R e^{j\omega t} + R_1 e^{-\alpha_1 t} + R_2 e^{-\alpha_2 t} \right]
$$

(22.4.28)

$$
R = \frac{(j\omega)^2}{(\alpha_1 + j\omega)(\alpha_2 + j\omega)}
$$

$$
R_1 = \frac{\alpha_1^2}{(\alpha_1 - \alpha_2)(\alpha_1 + j\omega)}
$$

$$
R_2 = \frac{\alpha_2^2}{(\alpha_2 - \alpha_1)(\alpha_2 + j\omega)}
$$

(e) Repeat parts (a–d) when $\alpha_1, \alpha_2$ 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, $0 \leq t \leq 10$, 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, \texttt{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 $180^\circ$ 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.

```matlab
w0 = 1; L = 1; a1 = 3; a2 = 2;
x0 = 1/2; v0 = 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:

$$\frac{dT(t)}{dt} = -k[T(t) - T_{\text{ext}}(t)] + R(t)$$  \hfill (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:

$$T_c(t) = \begin{cases} 
T_H, & \text{if } \text{mod}(t, 24) < 12 \\
T_L, & \text{if } \text{mod}(t, 24) \geq 12
\end{cases}$$  \hfill (22.4.30)

where the modulo operation, 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 $0 \leq \text{mod}(t, 24) < 24$.

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:

$$R(t) = \begin{cases} 
R_0, & \text{if } T(t) < T_c(t) \\
0, & \text{if } T(t) \geq T_c(t)
\end{cases}$$  \hfill (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, $t_n = n\Delta t$, $n = 1, 2, 3, \ldots$, where $\Delta t$ 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:

$$\frac{T(t_{n+1}) - T(t_n)}{\Delta t} = -k[T(t_n) - T_{\text{ext}}(t_n)] + R(t_n)$$

Using the simplified notation $T(n)$ to denote $T(t_n)$, \footnote{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).} and similarly for $R(t_n)$ and $T_c(t_n)$, this difference equation can be rearranged into:

$$R(t_n) = \frac{T(t_n) - T_{\text{ext}}(t_n)}{n} - kT(t_n)$$

\footnote{$n$ is a MATLAB index, and $T(n)$, a MATLAB array.}
\[ T(n + 1) = T(n) - k \Delta t \left[ T(n) - T_{\text{ext}}(n) \right] + \Delta t R(n), \quad n \geq 1 \]  

(22.4.32)

where

\[
R(n) = \begin{cases} 
R_0, & \text{if } T(n) < T_c(n) \\
0, & \text{if } T(n) \geq T_c(n) 
\end{cases}
\]

(22.4.33)

with

\[
T_c(n) = \begin{cases} 
T_H, & \text{if } \text{mod}(t_n, 24) < 12 \\
T_L, & \text{if } \text{mod}(t_n, 24) \geq 12 
\end{cases}
\]

(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:

\[
T_{\text{ext}}(n) = A - B \cos \left( \frac{2\pi t_n}{24} \right)
\]

(22.4.35)

Consider the following realistic numerical values:

\[
A = 40 \, ^\circ F, \quad B = 10 \, ^\circ F \\
k = 0.35 \, \text{hr}^{-1}, \quad R_0 = 20 \, ^\circ F/\text{hr} \\
T_H = 70 \, ^\circ F, \quad T_L = 60 \, ^\circ F, \quad T_0 = 35 \, ^\circ F
\]

Define the time vector \( t_n \) to span a 48-hr period and sampled every 3 seconds:

\[
T_{\text{max}} = 48; \quad Dt = 3/3600; \quad \text{units of hours} \\
t_n = Dt:Dt:T_{\text{max}}; \quad \text{units of hours}
\]

\[ T_{\text{e}} = A - B \cos \left( \frac{2\pi t_n}{24} \right) \]

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:

\[ T_{\text{ext}}(n) = A - B \cos \left( \frac{2\pi t_n}{24} \right) \]

(22.4.35)

Consider the following realistic numerical values:

\[
A = 40 \, ^\circ F, \quad B = 10 \, ^\circ F \\
k = 0.35 \, \text{hr}^{-1}, \quad R_0 = 20 \, ^\circ F/\text{hr} \\
T_H = 70 \, ^\circ F, \quad T_L = 60 \, ^\circ F, \quad T_0 = 35 \, ^\circ F
\]

Define the time vector \( t_n \) to span a 48-hr period and sampled every 3 seconds:

\[
T_{\text{max}} = 48; \quad Dt = 3/3600; \quad \text{units of hours} \\
t_n = Dt:Dt:T_{\text{max}}; \quad \text{units of hours}
\]

The essential MATLAB code for this problem is listed below:

```matlab
A = 40; B = 10; 
T0 = 35; T_H = 70; T_L = 60; 
R0 = 20; 

T_{\text{max}} = 48; \quad \text{hours} \\
D_t = 3/3600; \quad \text{3 sec in units of hours} \\
t = Dt:Dt:T_{\text{max}}; \quad \text{units of hours}

T_{\text{e}} = A - B \cos (\frac{2\pi t_n}{24}); \quad \text{external temperature}

T_{\text{c}} = \text{TH}*(\text{mod}(t,24)<12) + \text{TL}*(\text{mod}(t,24)\geq12); \quad \text{control temperature}

figure; plot(t,Tc,'b-');xlabel('t (hours)'); ylabel('degrees (^\circ F)');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] \quad % generate graphs for three values of k 
T(1) = T0; 
for n=1:length(t)-1 
R = R0*(T(n)<Tc(n)); \quad \text{control signal}
T(n+1) = T(n) - k*Dt*(T(n)-T_{\text{e}}(n)) + Dt*R; 
end
```

The essential MATLAB code for this problem is listed below:

```matlab
A = 40; B = 10; 
T0 = 35; T_H = 70; T_L = 60; 
R0 = 20; 

T_{\text{max}} = 48; \quad \text{hours} \\
D_t = 3/3600; \quad \text{3 sec in units of hours} \\
t = Dt:Dt:T_{\text{max}}; \quad \text{units of hours}

T_{\text{e}} = A - B \cos (\frac{2\pi t_n}{24}); \quad \text{external temperature}

T_{\text{c}} = \text{TH}*(\text{mod}(t,24)<12) + \text{TL}*(\text{mod}(t,24)\geq12); \quad \text{control temperature}

figure; plot(t,Tc,'b-');xlabel('t (hours)'); ylabel('degrees (^\circ F)');title('temperature control signal T_c(t)');yaxis(20,80,20:10:80); xaxis(0,48,0:6:48); grid on;
```

The essential MATLAB code for this problem is listed below:

```matlab
A = 40; B = 10; 
T0 = 35; T_H = 70; T_L = 60; 
R0 = 20; 

T_{\text{max}} = 48; \quad \text{hours} \\
D_t = 3/3600; \quad \text{3 sec in units of hours} \\
t = Dt:Dt:T_{\text{max}}; \quad \text{units of hours}

T_{\text{e}} = A - B \cos (\frac{2\pi t_n}{24}); \quad \text{external temperature}

T_{\text{c}} = \text{TH}*(\text{mod}(t,24)<12) + \text{TL}*(\text{mod}(t,24)\geq12); \quad \text{control temperature}

figure; plot(t,Tc,'b-');xlabel('t (hours)'); ylabel('degrees (^\circ F)');title('temperature control signal T_c(t)');yaxis(20,80,20:10:80); xaxis(0,48,0:6:48); grid on;
```

The essential MATLAB code for this problem is listed below:

```matlab
A = 40; B = 10; 
T0 = 35; T_H = 70; T_L = 60; 
R0 = 20; 

T_{\text{max}} = 48; \quad \text{hours} \\
D_t = 3/3600; \quad \text{3 sec in units of hours} \\
t = Dt:Dt:T_{\text{max}}; \quad \text{units of hours}

T_{\text{e}} = A - B \cos (\frac{2\pi t_n}{24}); \quad \text{external temperature}

T_{\text{c}} = \text{TH}*(\text{mod}(t,24)<12) + \text{TL}*(\text{mod}(t,24)\geq12); \quad \text{control temperature}

figure; plot(t,Tc,'b-');xlabel('t (hours)'); ylabel('degrees (^\circ F)');title('temperature control signal T_c(t)');yaxis(20,80,20:10:80); xaxis(0,48,0:6:48); grid on;
```

The essential MATLAB code for this problem is listed below:

```matlab
A = 40; B = 10; 
T0 = 35; T_H = 70; T_L = 60; 
R0 = 20; 

T_{\text{max}} = 48; \quad \text{hours} \\
D_t = 3/3600; \quad \text{3 sec in units of hours} \\
t = Dt:Dt:T_{\text{max}}; \quad \text{units of hours}

T_{\text{e}} = A - B \cos (\frac{2\pi t_n}{24}); \quad \text{external temperature}

T_{\text{c}} = \text{TH}*(\text{mod}(t,24)<12) + \text{TL}*(\text{mod}(t,24)\geq12); \quad \text{control temperature}

figure; plot(t,Tc,'b-');xlabel('t (hours)'); ylabel('degrees (^\circ F)');title('temperature control signal T_c(t)');yaxis(20,80,20:10:80); xaxis(0,48,0:6:48); grid on;
```

The essential MATLAB code for this problem is listed below:

```matlab
A = 40; B = 10; 
T0 = 35; T_H = 70; T_L = 60; 
R0 = 20; 

T_{\text{max}} = 48; \quad \text{hours} \\
D_t = 3/3600; \quad \text{3 sec in units of hours} \\
t = Dt:Dt:T_{\text{max}}; \quad \text{units of hours}

T_{\text{e}} = A - B \cos (\frac{2\pi t_n}{24}); \quad \text{external temperature}

T_{\text{c}} = \text{TH}*(\text{mod}(t,24)<12) + \text{TL}*(\text{mod}(t,24)\geq12); \quad \text{control temperature}

figure; plot(t,Tc,'b-');xlabel('t (hours)'); ylabel('degrees (^\circ F)');title('temperature control signal T_c(t)');yaxis(20,80,20:10:80); xaxis(0,48,0:6:48); grid on;
```
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-$R_1$ 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- \( R \)_\( H \)enderson 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 data smoothing with polynomials of degrees \( d = 0, 1, 2 \). The corresponding smoothed values are given by the 0th, 1st, and 2nd degree polynomials defined for \( m = -2, -1, 0, 1, 2 \):

\[
\hat{y}_m = \begin{cases} 
  c_0 & \text{(constant)} \\
  c_0 + c_1 m & \text{(linear)} \\
  c_0 + c_1 m + c_2 m^2 & \text{(quadratic)} 
\end{cases}
\]

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 mean-square error. For example, in the quadratic case, we have the performance index:

\[
J = \sum_{m=-2}^{2} e_m^2 = \sum_{m=-2}^{2} (y_m - (c_0 + c_1 m + c_2 m^2))^2 = \text{min} \tag{23.2.2}
\]

where the fitting errors are defined as

\[
e_m = y_m - \hat{y}_m = y_m - (c_0 + c_1 m + c_2 m^2), \quad m = -2, -1, 0, 1, 2
\]

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:

\[
y = \begin{bmatrix} y_{-2} \\ y_{-1} \\ y_0 \\ y_1 \\ y_2 \end{bmatrix}, \quad \hat{y} = \begin{bmatrix} \hat{y}_{-2} \\ \hat{y}_{-1} \\ \hat{y}_0 \\ \hat{y}_1 \\ \hat{y}_2 \end{bmatrix}, \quad e = \begin{bmatrix} e_{-2} \\ e_{-1} \\ e_0 \\ e_1 \\ e_2 \end{bmatrix} = y - \hat{y}
\]

Similarly, we define the five-dimensional polynomial basis vectors \( s_0, s_1, s_2 \), whose components are:

\[
s_0 (m) = 1, \quad s_1 (m) = m, \quad s_2 (m) = m^2, \quad -2 \leq m \leq 2
\]

Vectorially, we have:

\[
s_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \quad s_1 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \quad s_2 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 2 \end{bmatrix} \tag{23.2.3}
\]

In this notation, we may write the third of Eq. (23.2.1) vectorially:

\[
\hat{y} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} c = \begin{bmatrix} -2 \\ -1 \\ 0 \\ 1 \\ 2 \end{bmatrix} c_0 + \begin{bmatrix} 4 \\ 1 \\ 1 \\ 0 \\ 4 \end{bmatrix} c_1 + \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} c_2
\]

\[c = c_0 s_0 + c_1 s_1 + c_2 s_2
\]
Therefore,
\[ \hat{y} = c_0 s_0 + c_1 s_1 + c_2 s_2 = \begin{bmatrix} s_0, s_1, s_2 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = Sc \] (23.2.4)

The 5×3 basis matrix \( S \) has as columns the three basis vectors \( s_0, s_1, s_2 \). It is given explicitly as follows:
\[
S = \begin{bmatrix} s_0, s_1, s_2 \end{bmatrix} = \begin{bmatrix} 1 & -2 & 4 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 2 & 4 \end{bmatrix}
\]

Writing \( e = y - \hat{y} = y - Sc \), we can express the performance index (23.2.2) as the dot product:
\[
J = e^T e = (y - Sc)^T (y - Sc) = \min
\]
(23.2.6)

To minimize this expression with respect to \( c \), we set the gradient \( \frac{\partial J}{\partial c} \) to zero:
\[
\frac{\partial J}{\partial c} = -2S^T e = -2S^T (y - Sc) = -2 (S^T y - S^T Sc) = 0
\]
(23.2.7)

Therefore, the minimization condition gives the so-called orthogonality equations and the equivalent normal equations:
\[
\frac{\partial J}{\partial c} = 0 \iff S^T e = 0 \iff S^T Sc = S^T y
\]
(23.2.8)

with optimal solution:
\[
c = (S^T S)^{-1} S^T y = G^T y
\]
(23.2.9)

where we defined the 5×3 matrix \( G \) by
\[
G = S (S^T S)^{-1}
\]
(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 \( Sc = y^\dagger \), \( c = S^+ y \), where \( S^+ = (S^T S)^{-1} S^T \) is the corresponding pseudoinverse. Inserting the optimal coefficients \( c \) into Eq. (23.2.4), we find the smoothed values:
\[
\hat{y} = Sc = SG^T y = S (S^T S)^{-1} S^T y = B^T y
\]
(23.2.11)

where we defined the 5×5 matrix \( B \) by
\[
B = B^T = SG^T = GS^T = S (S^T S)^{-1} S^T
\]
(23.2.12)

\(^1\)see for example by [45]  
\(^2\)although \( B \) is symmetric, we prefer to write \( \hat{y} = B^T y \), which generalizes to the non-symmetric case of minimum-roughness filters of Sec. 23.12.

The symmetric 3×3 matrix \( F = S^T S \), which appears in the expressions for \( G \) and \( B \), has matrix elements that are the dot products of the basis vectors, that is, the \( ij \) matrix element is \( F_{ij} = (S^T S)_{ij} = s_i^T s_j \). Indeed, using Eq. (23.2.5), we find:
\[
F = S^T S = \begin{bmatrix} s_0^T \\ s_1^T \\ s_2^T \end{bmatrix} [s_0, s_1, s_2] = \begin{bmatrix} s_0^T s_0 & s_0^T s_1 & s_0^T s_2 \\ s_1^T s_0 & s_1^T s_1 & s_1^T s_2 \\ s_2^T s_0 & s_2^T s_1 & s_2^T s_2 \end{bmatrix}
\]
(23.2.13)

Using Eq. (23.2.5), we calculate \( F \) and its inverse \( F^{-1} \):
\[
F = \begin{bmatrix} 5 & 0 & 10 \\ 0 & 10 & 0 \\ 10 & 0 & 34 \end{bmatrix}, \quad F^{-1} = \begin{bmatrix} 17 & 0 & -5 \\ 0 & 3.5 & 0 \\ -5 & 0 & 2.5 \end{bmatrix}
\]
(23.2.14)

Then, we calculate the 5×3 matrix \( G = S (S^T S)^{-1} = SF^{-1} \):
\[
G = \begin{bmatrix} -3 & -7 & 5 \\ 12 & -3.5 & -2.5 \\ 17 & 0 & -5 \\ 12 & 3.5 & -2.5 \\ -3 & 7 & 5 \end{bmatrix} = [g_0, g_1, g_2]
\]
(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 \( B \):
\[
B = G S^T = \begin{bmatrix} -3 & -7 & 5 \\ 12 & -3.5 & -2.5 \\ 17 & 0 & -5 \\ 12 & 3.5 & -2.5 \\ -3 & 7 & 5 \end{bmatrix} = [b_{-2}, b_{-1}, b_0, b_1, b_2]
\]
(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:
\[
B = [b_{-2}, b_{-1}, b_0, b_1, b_2] = \begin{bmatrix} b_{-2}^T \\ b_{-1}^T \\ b_0^T \\ b_1^T \\ b_2^T \end{bmatrix} = B^T
\]
The five columns or rows of $B$ are the LPSM filters of length 5 and polynomial order 2. The corresponding smoothed values $\hat{y}$ can be expressed component-wise in terms of these filters, as follows:

$$
\begin{bmatrix}
\hat{y}_{-2} \\
\hat{y}_{-1} \\
\hat{y}_0 \\
\hat{y}_1 \\
\hat{y}_2
\end{bmatrix} = \hat{y} = B^T y = \begin{bmatrix}
b_0^T \\
b_1^T \\
b_0^T \\
b_1^T \\
b_2^T
\end{bmatrix} y = \begin{bmatrix}
b_1^T y \\
b_0^T y \\
b_1^T y \\
b_2^T y \\
b_2^T y
\end{bmatrix}
$$

or, for $m = -2, -1, 0, 1, 2$:

$$
\hat{y}_m = b_m^T y
$$

(23.2.17)

and more explicitly,

$$
\begin{bmatrix}
\hat{y}_{-2} \\
\hat{y}_{-1} \\
\hat{y}_0 \\
\hat{y}_1 \\
\hat{y}_2
\end{bmatrix} = \begin{bmatrix}
31 & 9 & -3 & -5 & 3 \\
9 & 13 & 12 & 6 & -5 \\
-3 & 12 & 17 & 12 & -3 \\
-5 & 6 & 12 & 13 & 9 \\
3 & -5 & -3 & 9 & 31
\end{bmatrix} \begin{bmatrix}
y_{-2} \\
y_{-1} \\
y_0 \\
y_1 \\
y_2
\end{bmatrix}
$$

(23.2.18)

Thus, the $m$th filter $b_m$ dotted into the data vector $y$ generates the $m$th 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

$$
\begin{bmatrix}
c_0 \\
c_1 \\
c_2
\end{bmatrix} = c = G^T y = \begin{bmatrix}
g_0^T \\
g_1^T \\
g_2^T
\end{bmatrix} y = \begin{bmatrix}
g_0^T y \\
g_1^T y \\
g_2^T y
\end{bmatrix}
$$

or, explicitly,

$$
\begin{bmatrix}
c_0 \\
c_1 \\
c_2
\end{bmatrix} = \begin{bmatrix}
-3 & 12 & 17 & 12 & -3 \\
-7 & -3.5 & 0 & 3.5 & 7 \\
5 & -2.5 & -5 & -2.5 & 5
\end{bmatrix} \begin{bmatrix}
y_{-2} \\
y_{-1} \\
y_0 \\
y_1 \\
y_2
\end{bmatrix}
$$

(23.2.19)

Thus, the coefficients $c_i$ can be expressed as the dot products of the columns of $G$ with the data vector $y$: $c_i = g_i^T y$, $i = 0, 1, 2$.

(23.2.20)

Of the five columns of $B$, the middle one, $b_0$, is the most important because it smooths the value $y_0$, which is symmetrically placed with respect to the other samples in $y$, as shown in Fig. 23.2.1.

In smoothing a long block of data, the filter $b_0$ 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 $b_0$ 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: $\hat{y}_0 = c_0 = b_0^T y$ (the middle column of $B$ and the first column of $G$ are always the same, $b_0 = g_0$).

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 $n$th time instant, that is, we make the substitution:

$$
[y_{-2}, y_{-1}, y_0, y_1, y_2] \rightarrow [y_{n-2}, y_{n-1}, y_n, y_{n+1}, y_{n+2}]
$$

The corresponding five smoothed values will be then:

$$
\begin{bmatrix}
\hat{y}_{n-2} \\
\hat{y}_{n-1} \\
\hat{y}_n \\
\hat{y}_{n+1} \\
\hat{y}_{n+2}
\end{bmatrix} = \begin{bmatrix}
31 & 9 & -3 & -5 & 3 \\
9 & 13 & 12 & 6 & -5 \\
-3 & 12 & 17 & 12 & -3 \\
-5 & 6 & 12 & 13 & 9 \\
3 & -5 & -3 & 9 & 31
\end{bmatrix} \begin{bmatrix}
y_{n-2} \\
y_{n-1} \\
y_n \\
y_{n+1} \\
y_{n+2}
\end{bmatrix}
$$

(23.2.21)

In particular, the middle sample $y_n$ is smoothed by the filter $b_0$:

$$
\hat{y}_n = \frac{1}{35} (-3y_{n-2} + 12y_{n-1} + 17y_n + 12y_{n+1} - 3y_{n+2})
$$

(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 $b_0$ is equal to the middle value of its impulse response, that is, the coefficient $b_0(0)$. Therefore,

$$
R = b_0^T b_0 = \sum_{m=-2}^{2} b_0(m)^2 = b_0(0)^2 = 17/35 = 17/5 = 2.43 = 0.49
$$

By comparison, the length-5 FIR averager operating on the same five samples is:

$$
\hat{x}_n = \frac{1}{5} (y_{n-2} + y_{n-1} + y_n + y_{n+1} + y_{n+2})
$$

(23.2.23)

with $R = 1/N = 1/5$. 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 $m = 0$:

$$
\frac{\hat{y}_0}{m} = \frac{d}{dm} y_0 = c_1, \quad \frac{d^2 \hat{y}_0}{dm^2} \bigg|_0 = 2c_2
$$
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 $G$:

$$
\mathbf{\hat{y}}_0 = c_1 = g_1^T \mathbf{y} \\
\mathbf{\hat{y}}_0 = 2c_2 = 2g_2^T \mathbf{y}
$$

(23.2.24)

Shifting these to the $r$th time sample, and denoting them by $\mathbf{\hat{x}}_n$ and $\mathbf{\hat{x}}_n$, we find the length-5 local polynomial filters for estimating the first and second derivatives of $x_n$:

$$
\mathbf{\hat{x}}_n = \frac{1}{35} (-7y_{n-2} - 3.5y_{n-1} + 3.5y_{n+1} + 7y_{n+2})
$$

$$
\mathbf{\hat{x}}_n = \frac{2}{35} (5y_{n-2} - 2.5y_{n-1} - 5y_n - 2.5y_{n+1} + 5y_{n+2})
$$

(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 $d \leq N - 1$, a restriction to be clarified later. Assuming that $N$ is odd, say, $N = 2M + 1$, the five-dimensional data vector $\mathbf{y} = [y_{-1}, y_0, y_1, y_2]^T$ is replaced by an $N$-dimensional one, having $M$ points on either side of $y_0$:

$$
\mathbf{y} = [y_{-M}, \ldots, y_{-1}, y_0, y_1, \ldots, y_M]^T
$$

(23.2.26)

The $N$ data samples in $\mathbf{y}$ are then fitted by a polynomial of degree $d$:

$$
\mathbf{\hat{y}}_m = c_0 + c_1 m + \cdots + c_dm^d, \quad -M \leq m \leq M
$$

(23.2.27)

In this case, there are $d+1$ polynomial basis vectors $s_i$, $i = 0, 1, \ldots, d$, defined to have components:

$$
s_i(m) = m^i, \quad -M \leq m \leq M
$$

(23.2.28)

The corresponding $N \times (d+1)$ basis matrix $S$ is defined to have the $s_i$ as columns:

$$
S = [s_0, s_1, \ldots, s_d]
$$

(23.2.29)

The smoothed values (23.2.27) can be written in the vector form:

$$
\mathbf{\hat{y}} = \sum_{i=0}^{d} c_i \mathbf{s}_i = [s_0, s_1, \ldots, s_d]
$$

(23.2.30)

$$
= S \mathbf{c}
$$

The design steps for the LPSM filters can be summarized then as follows:

$$
F = S^T S \quad \Leftrightarrow \quad F_{ij} = s_i^T s_j, \quad i,j = 0, 1, \ldots, d
$$

$$
G = SF^{-1} = [g_0, g_1, \ldots, g_d]
$$

$$
B = GS^T = SF^{-1}S^T = [b_0, \ldots, b_M]
$$

(23.2.31)

The corresponding coefficient vector $\mathbf{c}$ and smoothed data vector $\mathbf{\hat{y}}$ will be:

$$
\mathbf{c} = G^T \mathbf{y} \quad \Leftrightarrow \quad c_i = g_i^T \mathbf{y}, \quad i = 0, 1, \ldots, d
$$

$$
\mathbf{\hat{y}} = B^T \mathbf{y} \quad \Leftrightarrow \quad \mathbf{\hat{y}}_m = b_m \mathbf{y}, \quad -M \leq m \leq M
$$

(23.2.32)

The middle smoothed value $\mathbf{\hat{y}}_0$ is given in terms of the middle LPSM filter $b_0$:

$$
\mathbf{\hat{y}}_0 = b_0 \mathbf{y} = \sum_{k=-M}^{M} b_0(k) y_k
$$

The $N$-dimensional vector $\mathbf{y} = [y_{-M}, \ldots, y_{-1}, y_0, y_1, \ldots, y_M]^T$ can be shifted to the $r$th time instant by the replacement:

$$
[y_{-M}, \ldots, y_{-1}, y_0, y_1, \ldots, y_M] \rightarrow [y_{n-M}, \ldots, y_{n-1}, y_{n}, y_{n+1}, \ldots, y_{n+M}]
$$

The resulting length-$N$, order-$d$, LPSM filter for smoothing a noisy sequence $y_n$ will be, in its steady-state form (denoting again $\mathbf{\hat{x}}_n = \mathbf{\hat{y}}_n$):

$$
\mathbf{\hat{x}}_n = \mathbf{\hat{y}}_n = \sum_{k=-M}^{M} b_0(k) y_{n+k} = \sum_{k=-M}^{M} b_0(-k) y_{n-k}
$$

(23.2.33)

The second equation expresses the output in convolutional form. Because the filter $b_0$ is symmetric about its middle, we can replace $b_0(-k) = b_0(k)$. The non-central estimated values are obtained from the $b_m$ filters:

$$
\mathbf{\hat{y}}_{n+m} = \sum_{k=-M}^{M} b_m(k) y_{n+k} = \sum_{k=-M}^{M} b^R_m(k) y_{n-k}
$$

(23.2.34)

These filters satisfy the symmetry property $b^R_m(k) = b_m(-k) = b_{-m}(k)$ 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 $i = 0, 1, \ldots, d$. It follows by differentiating Eq. (23.2.27) $i$ times and setting $m = 0$:

$$
\mathbf{\hat{y}}^{(i)}_0 = \frac{d^i \mathbf{\hat{y}}}{dm^i} = b^i \mathbf{c} = b^i g^T \mathbf{y}
$$

(23.2.35)

Shifting these to time $n$, gives the differentiation convolutional filtering equations:

$$
\mathbf{\hat{x}}^{(i)}_n = b^i \sum_{m=-M}^{M} g^R_m(m) y_{n-m}, \quad i = 0, 1, \ldots, d
$$

where, $g^R_m(m) = g_i(-m)$ and as in Eq. (23.2.33), we reversed the order of writing the terms, but here the filters $g_i$ are not necessarily symmetric (actually, they are symmetric for even $i$, and antisymmetric for odd $i$).

$^1$We use the notation $b^R$ to denote the reverse of a double-sided filter $b$, 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 \(s_0\) defined in Eq. (23.2.3). The basis matrix \(S = [s_0]\) will have just one column, and the matrix \(F\) will be the scalar
\[
F = S^T S = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = 5
\]
The matrix \(G\) will then be
\[
G = SF^{-1} = \frac{1}{5} s_0 = \frac{1}{5} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}^T
\]
resulting in the LPSM matrix \(B\):
\[
B = G S^T = \frac{1}{5} s_0 s_0^T = \frac{1}{5} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}
\]
Thus, the steady-state LPSM filter is the length-5 averager:
\[
b_0 = \frac{1}{5} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}^T
\]
For the case \(d = 1\), corresponding to the linear fit \(\hat{y}_m = c_0 + c_1 m\), we have the two basis vectors \(s_0\) and \(s_1\), given in Eq. (23.2.3). We calculate the matrices \(S, F,\) and \(F^{-1}\):
\[
S = [s_0, s_1] = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \quad F = S^T S = \begin{bmatrix} 5 \\ 0 \\ 10 \\ 0 \\ 5 \\ 0.5 \end{bmatrix}, \quad F^{-1} = \frac{1}{5} \begin{bmatrix} 1 \\ 0 \\ 0.5 \end{bmatrix}
\]
This gives for \(G\) and \(B\):
\[
G = SF^{-1} = \frac{1}{5} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \quad B = G S^T = \frac{1}{5} \begin{bmatrix} 3 \\ 2 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}
\]
Thus, the steady-state LPSM filter \(b_0\) is still equal to the length-5 FIR averager. It is a general property of LPSM filters, that the filter \(b_0\) 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. \(\Box\)

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 \(s_3\) with components \(s_3(m) = m^3\). Therefore, the basis matrix \(S\) is:
\[
S = [s_0, s_1, s_2, s_3] = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \quad F = S^T S = \begin{bmatrix} 5 \\ 0 \\ 10 \\ 0 \\ 10 \\ 0 \\ 34 \\ 0 \\ 34 \\ 0 \end{bmatrix}
\]
Because of the checkerboard pattern of this matrix, its inverse can be obtained from the inverses of the two \(2\times2\) 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:
\[
F^{-1} = \begin{bmatrix} 34/70 & 0 & -10/70 & 0 \\ 10/70 & 0 & 5/70 & 0 \\ -34/144 & 0 & 10/144 & 0 \\ 0 & -34/144 & 0 & 10/144 \end{bmatrix}
\]
Then, we compute the derivative filter matrix \(G\):
\[
G = SF^{-1} = \frac{1}{5} \begin{bmatrix} -3 & 35/12 & 5 & -35/12 \\ 12 & -70/3 & -2.5 & 35/6 \\ 17 & 0 & -5 & 0 \\ -3 & -35/12 & 5 & 35/12 \end{bmatrix}
\]
and the LPSM matrix \(B\):
\[
B = SG^T = \frac{1}{5} \begin{bmatrix} 34.5 & -3 & 2 & -3 \\ 12 & 17 & 12 & -3 \\ -3 & -35/12 & 2 & -3 \\ -0.5 & 2 & -3 & 34.5 \end{bmatrix}
\]
As mentioned above, the steady-state LPSM filter \(b_0\) is the same as that of case \(d = 2\). But, the transient and differentiation filters are different. \(\Box\)

23.3 Exact Design Equations

In practice, the most common values of \(d\) are 0, 1, 2, 3, 4. For these \(d\)s 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 \(F^{-1}\), we can write
\[
B = SF^{-1} S^T = \sum_{i=0}^{d} \sum_{j=0}^{d} s_i s_i^T \Phi_{ij}
\]
23.3. EXACT DESIGN EQUATIONS

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 $\Phi$:

$$
\begin{bmatrix}
1/F_0 \\
1/F_0 & 0 & -F_2/D_4 \\
0 & 1/F_2 & 0 & -F_2/D_4 \\
-F_2/D_4 & 0 & F_0/D_4 & 0
\end{bmatrix}
$$

and for $d = 3$:

$$
\Phi = F^{-1} =
\begin{bmatrix}
F_4/D_4 & 0 & -F_2/D_4 & 0 \\
0 & F_0/D_8 & 0 & -F_4/D_4 \\
-F_2/D_4 & 0 & F_0/D_4 & 0 \\
0 & -F_4/D_8 & 0 & F_2/D_8
\end{bmatrix}
$$

where the $D_4$ and $D_8$ are determinants of the interlaced submatrices:

$$
D_4 = F_0 F_4 - F_2^2 = \frac{1}{43} M(M + 1)(2M + 1)(2M + 3)(4M^2 - 1)
$$

$$
D_8 = F_2 F_6 - F_4^2 = \frac{3}{35} M(M + 2)(M^2 - 1)D_4
$$

Inserting the above expressions for $\Phi$ into Eq. (23.3.3), we determine the corresponding LPSM filters. For $d = 0$, we find for $-M \leq m, k \leq M$:

$$
b_m(k) = B_{mk} = \frac{1}{F_0} = \frac{1}{N}
$$

For $d = 1$:

$$
b_m(k) = B_{mk} = \frac{1}{F_0} + \frac{mk}{F_2}
$$

For $d = 2$:

$$
b_m(k) = B_{mk} = \frac{F_4}{D_4} + \frac{F_6}{D_8} mk - \frac{F_2}{D_4} (m^2 + k^2) + \frac{F_0}{D_4} m^2 k^2
$$

For $d = 3$:

$$
b_m(k) = B_{mk} = \frac{F_4}{D_4} + \frac{F_6}{D_8} mk - \frac{F_2}{D_4} (m^2 + k^2) + \frac{F_0}{D_4} m^2 k^2 - \frac{F_2}{D_8} (km^3 + mk^3) + \frac{F_2}{D_8} m^3 k^3
$$

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:

$$
[F_0],
\begin{bmatrix}
F_0 & 0 & F_2 \\
0 & 0 & F_0 \\
F_0 & F_2 & 0
\end{bmatrix},
\begin{bmatrix}
F_0 & 0 & F_2 & 0 \\
0 & F_2 & 0 & F_4 \\
F_2 & 0 & F_4 & 0 \\
0 & F_4 & 0 & F_0
\end{bmatrix}
$$

which gives for the $mk$th matrix element

$$
B_{mk} = \sum_{i=0}^{d} \sum_{j=0}^{d} s_i(m) s_j(k) \Phi_{ij} = \sum_{i=0}^{d} \sum_{j=0}^{d} m^i k^j \Phi_{ij}, \quad -M \leq m, k \leq M
$$

(23.3.2)

Because of symmetry, $B_{mk} = B_{km}$, these matrix elements represent the $k$th component of the LPSM filter $b_m$ or the $m$th component of the filter $b_k$, that is,

$$
B_{mk} = B_{km} = b_m(k) = b_k(m) = \sum_{i=0}^{d} \sum_{j=0}^{d} m^i k^j \Phi_{ij}
$$

(23.3.3)

The matrix $\Phi$ can be determined easily for the cases $0 \leq d \leq 4$. 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:

$$
F_{ij} = (S^T S)_{ij} = s_i^T s_j = \sum_{m=-M}^{M} s_i(m) s_j(m) = \sum_{m=-M}^{M} m^{i+j}, \quad \text{or,}
$$

$$
F_{ij} = \sum_{m=-M}^{M} m^{i+j} \equiv F_{i+j}, \quad 0 \leq i, j \leq d
$$

(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 $d \leq 4$, 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:

$$
F_0 = \sum_{m=-M}^{M} m^0 = N = 2M + 1
$$

$$
F_2 = \sum_{m=-M}^{M} m^2 = \frac{1}{3} M(M + 1)(2M + 1)
$$

$$
F_4 = \sum_{m=-M}^{M} m^4 = \frac{1}{5} (3M^2 + 3M - 1) F_2
$$

$$
F_6 = \sum_{m=-M}^{M} m^6 = \frac{1}{7} (3M^4 + 6M^3 - 3M + 1) F_2
$$

$$
F_8 = \sum_{m=-M}^{M} m^8 = \frac{1}{15} (5M^6 + 15M^5 + 5M^4 - 15M^3 - M^2 + 9M - 3) F_2
$$

(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:
The required ratios are given explicitly as follows:

\[
\begin{align*}
F_4 &= \frac{3(3M^2 + 3M - 1)}{(2M + 3)(4M^2 - 1)} \\
D_4 &= \frac{15}{(2M + 3)(4M^2 - 1)} \\
F_2 &= \frac{45}{M(M + 1)(2M + 3)(4M^2 - 1)} \\
D_2 &= \frac{25(3M^4 + 6M^3 - 3M + 1)}{M(M + 2)(2M - 1)(2M + 3)(4M^2 - 1)} \\
F_6 &= \frac{35(3M^2 + 3M - 1)}{M(M + 2)(2M - 1)(2M + 3)(4M^2 - 1)} \\
D_6 &= \frac{175}{M(M + 2)(2M - 1)(2M + 3)(4M^2 - 1)}
\end{align*}
\] (23.3.11)

In a similar fashion, we also find for the case \(d = 4\):

\[
\begin{align*}
b_m(k) &= B_{mk} = \frac{D_{12}}{D} + \frac{F_6}{D_6}mk - \frac{F_2}{D_8}(km^2 + mk^3) + \frac{F_4}{D_8}m^2k^2 \\
&\quad - \frac{D_6}{D}m^2k^3 + \frac{D_4}{D_8}(m^4 + k^4)
\end{align*}
\] (23.3.12)

where

\[
\begin{align*}
D_6 &= F_0F_6 - F_2F_4 \\
D_{10} &= F_2F_6 - F_4F_6 \\
D_{12} &= F_4F_6 - F_6^2 \\
D &= F_0D_{12} - F_2D_{10} + F_4D_6
\end{align*}
\] (23.3.13)

These are given explicitly as follows:

\[
\begin{align*}
D_6 &= \frac{1}{7}(6M^2 + 6M - 5)D_4 \\
D_{10} &= \frac{1}{21}M(M + 2)(M^2 - 1)(2M^2 + 2M - 3)D_4 \\
E_8 &= \frac{1}{5}(4M^4 + 8M^3 - 4M^2 - 8M + 1)D_4 \\
D_{12} &= \frac{4}{11025}M(M + 2)(M^2 - 1)(15M^4 + 30M^3 - 35M^2 - 50M + 12)D_4 \\
D &= \frac{4}{11025}M(M + 2)(2M + 5)(4M^2 - 9)(4M^2 - 1)D_4
\end{align*}
\] (23.3.14)

and the required ratios are:

\[
\begin{align*}
D_{12} &= \frac{15(15M^4 + 30M^3 - 35M^2 - 50M + 12)}{4(2M + 5)(4M^2 - 1)(4M^2 - 9)} \\
D_{10} &= \frac{525(2M^2 + 2M - 3)}{4(2M + 5)(4M^2 - 1)(4M^2 - 9)} \\
E_8 &= \frac{2205(4M^4 + 8M^3 - 4M^2 - 8M + 5)}{4M(M + 2)(2M^2 - 1)(2M + 5)(4M^2 - 1)(4M^2 - 9)} \\
D_8 &= \frac{945}{4(2M + 5)(4M^2 - 1)(4M^2 - 9)} \\
D_6 &= \frac{1575(6M^2 + 6M - 5)}{4M(M + 2)(2M^2 - 1)(2M + 5)(4M^2 - 1)(4M^2 - 9)} \\
D_4 &= \frac{11025}{4M(M + 2)(2M^2 - 1)(2M + 5)(4M^2 - 1)(4M^2 - 9)}
\end{align*}
\] (23.3.15)

In this case, the matrix \(F\) and its two interlaced submatrices are:

\[
F = \\
\begin{bmatrix}
F_0 & F_2 & F_4 \\
F_0 & F_2 & F_4 & F_6 \\
F_0 & F_2 & F_4 & F_6 & F_8 \\
F_0 & F_2 & F_4 & F_6 & F_8 & F_{10}
\end{bmatrix}
\]

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):

\[
\phi = F^{-1} = \begin{bmatrix}
D_{12}/D & 0 & -D_{10}/D & 0 & D_8/D \\
D_8/D & 0 & -D_{10}/D & 0 & 0 \\
0 & -D_{10}/D & 0 & -D_6/D & 0 \\
0 & 0 & -D_6/D & 0 & 0 \\
0 & 0 & 0 & -D_6/D & 0
\end{bmatrix}
\]

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 \(b_0(k)\), \(-M \leq k \leq M\). For \(d = 0, 1\):

\[
b_0(k) = \frac{1}{2M + 1}
\] (23.3.16)

for \(d = 2, 3\):

\[
b_0(k) = \frac{3(3M^2 + 3M - 1 - 5k^2)}{(2M + 3)(4M^2 - 1)}
\] (23.3.17)

and for \(d = 4, 5\):

\[
b_0(k) = \frac{15(15M^4 + 30M^3 - 35M^2 - 50M + 12 - 35(2M^2 + 2M - 3)k^2 + 63k^4)}{4(2M + 5)(4M^2 - 1)(4M^2 - 9)}
\] (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 $-M \leq k \leq M$):

$$b_0(k) = \frac{12 - 5k^2}{35} = \frac{1}{35} [-3, 12, 17, 12, -3]$$

$$b_0(k) = \frac{7 - k^2}{21} = \frac{1}{21} [-2, 3, 6, 7, 6, 3, -2]$$

$$b_0(k) = \frac{59 - 5k^2}{231} = \frac{1}{231} [-21, 14, 39, 54, 59, 54, 39, 14, -21]$$

where the coefficients have been reduced to integers as much as possible. \(\square\)

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 $\mathcal{Y}$ be the vector space of the $N$-dimensional real-valued vectors $\mathbf{y}$, that is, the space $\mathbb{R}^N$, and let $S$ be the $(d+1)$-dimensional subspace spanned by all linear combinations of the basis vectors $s_i$, $i = 0, 1, \ldots, d$.

Thus, the matrix $S = [s_0, s_1, \ldots, s_d]$ is a (non-orthogonal) basis of the subspace $S$. The smoothed vector $\hat{y}$, being a linear combination of the $s_i$, belongs to the subspace $S$. Moreover, because of the orthogonality equations (23.2.8), $\hat{y}$ is orthogonal to the error vector $e$:

$$\hat{y}^T e = (Sc)^T e = c^T S^T e = 0$$

Then, the equation $e = y - \hat{y}$ can be rewritten as the orthogonal decomposition:

$$y = \hat{y} + e \quad (23.4.1)$$

which expresses $y$ as a sum of a part that belongs to the subspace $S$ and a part that belongs to the orthogonal complement subspace $S^\perp$. The decomposition is unique and represents the direct sum decomposition of the full vector space $\mathcal{Y}$:

$$\mathcal{Y} = S \oplus S^\perp$$

This geometric interpretation requires that the dimension of the subspace $S$ not exceed the dimension of the full space $\mathcal{Y}$, that is, $d + 1 \leq N$. The component $\hat{y}$ that lies in $S$ is the projection of $y$ onto $S$. The matrix $B$ in Eq. (23.2.11) is the corresponding projection matrix. As such, it will be symmetric, $B^T = B$, and idempotent:

$$B^2 = B \quad (23.4.2)$$

The proof is straightforward:

$$B^2 = (SF^{-1}S^T)(SF^{-1}S^T) = SF^{-1}(S^T S)F^{-1}S^T F = SF^{-1}S^T = B$$

The matrix $(I - B)$, where $I$ is the $N$-dimensional identity matrix, is also a projection matrix, projecting onto the orthogonal subspace $S^\perp$. Thus, the error vector $e$ belonging to $S^\perp$ can be obtained from $y$ by the projection:

$$e = y - \hat{y} = (I - B)y$$

Because $(I - B)$ is also idempotent and symmetric, $(I - B)^2 = (I - B)$, we obtain for the minimized value of the performance index $J$ of Eq. (23.2.6):

$$J_{\text{min}} = e^T e = y^T (I - B)^2 y = y^T (I - B) y = y^T y - y^T B y \quad (23.4.3)$$

23.5 Orthogonal Polynomial Bases

Computationally, the non-orthogonal basis $S = [s_0, s_1, \ldots, s_d]$ 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\(^1\) on $S$, that is,

$$S = QR \quad (23.5.1)$$

where $Q$ is an $N \times (d+1)$ matrix with orthonormal columns, that is, $Q^T Q = I$, and $R$ is a $(d+1) \times (d+1)$ non-singular upper-triangular matrix.

The columns of $Q = [q_0, q_1, \ldots, q_d]$, correspond to the (orthonormalized) discrete Chebyshev or Gram polynomials $q_i(n)$, $i = 0, 1, \ldots, d$, constructed from the monomial basis $s_i(n) = n^i$ by the Gram-Schmidt process. Noting that $S^T S = R^T (Q^T Q) R = R^T R$, the design of the filter matrices $B$, $G$ can be formulated more efficiently as follows:

$$F = S^T S = R^T R \quad (23.5.2)$$

$$G = S F^{-1} = Q R^{-T}$$

$$B = S F^{-1} S^T = Q Q^T$$

which lead to the explicit construction of the differentiation and LPSM filters in terms of the Chebyshev polynomials $q_i(n)$:

$$g_i = \sum_{j=0}^{d} q_j(R^{-1})_{ij} \Rightarrow g_i(n) = \sum_{j=0}^{d} q_j(n) (R^{-1})_{ij} \quad (23.5.3)$$

$$B = \sum_{i=0}^{d} q_i q_i^T \Rightarrow B_{km} = \sum_{i=0}^{d} q_i(k) q_i(m)$$

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 $B, G$:

\(^1\)see [45].
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:

```matlab
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
```

It is straightforward to show using the property (23.6.3) that $b_t(k)$ is given by the usual Lagrange interpolation formula:

$$b_t(k) = \prod_{m=-M \atop m \neq k}^{M} \left( \frac{t-m}{k-m} \right), \quad -M \leq k \leq M$$  \hspace{1cm} (23.6.4)

Indeed, Eq. (23.6.4) states that the $(2M)$ roots of $b_t(k)$ are the points $t = m$, for $-M \leq m \leq M$ and $m \neq k$, which fixes the polynomial up to a constant. That constant is determined by the condition $b_t(k) = 1$.

**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 $t$:

$$
\begin{bmatrix}
    b_t(-2) \\
    b_t(-1) \\
    b_t(0) \\
    b_t(1) \\
    b_t(2)
\end{bmatrix} = \frac{1}{24}
\begin{bmatrix}
    0 & 2 & -1 & 2 & 1 \\
    -1 & 16 & 16 & -4 & -4 \\
    -24 & 0 & -30 & 0 & 6 \\
    0 & 16 & -16 & -4 & 4 \\
    0 & -2 & -1 & 2 & 1
\end{bmatrix}
\begin{bmatrix}
    t^4 \\
    t^3 \\
    t^2 \\
    t \\
    1
\end{bmatrix}
$$

The coefficient matrix is recognized as the inverse transposed of the basis matrix $S$:

$$S = \begin{bmatrix}
1 & 2 & 4 & -8 & 16 \\
1 & -1 & 1 & -1 & 1 \\
0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 \\
2 & 4 & 8 & 16 & 16
\end{bmatrix} \Rightarrow S^+ = \frac{1}{24}
\begin{bmatrix}
0 & 2 & -1 & -2 & 1 \\
-1 & 16 & 16 & -4 & -4 \\
-24 & 0 & -30 & 0 & 6 \\
0 & 16 & -16 & -4 & 4 \\
0 & -2 & -1 & 2 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 \\
2 & 4 & 8 & 16 & 16
\end{bmatrix}
$$

which verifies Eq. (23.6.2). \hfill \Box

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-Filter:

$$b_t(k) = \Gamma(M+1+t)\Gamma(M+1-k) \sin(\pi(t-k)) \pi(t-k)$$  \hspace{1cm} (23.6.5)

Some alternative expressions are as follows:

$$b_t(k) = (-1)^{M+k} \sum_{m=-M}^{2M} \left( \binom{M+k}{m} \right) \left( \binom{m}{k} \right) (-1)^m$$  \hspace{1cm} (23.6.6)

and since the $b_t(k)$ sum up to one, we also have [740]:

$$b_t(k) = \left[ \sum_{n=-M}^{M} b_t(n) b_t(k) \right]^{-1} = \left[ \sum_{n=-M}^{M} (-1)^{k-n} \binom{M+k}{m} \binom{M-k}{n} t-k \binom{M+n}{n} \binom{M-n}{-n} \right]^{-1}$$  \hspace{1cm} (23.6.8)
23.6. POLYNOMIAL PREDICTIVE AND INTERPOLATION FILTERS

For polynomial orders \( d < N - 1 \), one can still interpolate approximately and smoothly between the samples \( y_m \). In this case, using \( c = G^T y = (S^T S)^{-1} S^T y \), we have:

\[
\hat{y}_t = \sum_{i=0}^{d} c_i t^i = c^T u_t = y^T G u_t = y^T b_t = \sum_{k=-M}^{M} b_t(k) y_k
\]

(23.6.9)

where now

\[
u_t = \begin{bmatrix} 1 \\ t^1 \\ t^2 \\ \vdots \end{bmatrix}, \quad b_t = G u_t = S(S^T)^{-1} u_t \Rightarrow b_t(k) = \sum_{i=0}^{d} G_{ki} t^i
\]

(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 \( n \):

\[
y_n = b^R_t \hat{y}_{n+t} = \sum_{k=-M}^{M} b_t(k) y_{n+k} = \sum_{k=-M}^{M} b_t^R(k) y_{n-k}
\]

(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:

\[
b_t(0) = \frac{1}{F_0} + \frac{t k}{F_2}
\]

\[
b_t(1) = \frac{F_4}{D_4} + \frac{1}{D_2} t k - \frac{F_2}{D_4} (t^2 + k^2) + \frac{F_0}{D_4} t^2 k^2
\]

\[
b_t(2) = \frac{F_4}{D_4} + \frac{F_6}{D_6} t k - \frac{F_2}{D_4} (t^2 + k^2) + \frac{F_0}{D_4} t^2 k^2 - \frac{F_4}{D_8} (kt^3 + tk^3) + \frac{F_2}{D_8} t^3 k^3
\]

(23.6.12)

\[
b_t(3) = \frac{D_{12}}{D_4} + \frac{F_6}{D_8} t k - \frac{D_{10}}{D_4} (t^2 + k^2) + \frac{F_8}{D_4} t^2 k^2 - \frac{F_4}{D_8} (kt^3 + tk^3) + \frac{F_6}{D_8} t^3 k^3 + \frac{F_6}{D_8} t^3 k^3 + \frac{D_8}{D_8} (t^4 + k^4) - \frac{D_6}{D_4} (t^2 k^4 + k^2 t^4) + \frac{D_4}{D_4} t^4 k^4
\]

(23.6.13)

where the required coefficient ratios are given by Eqs. (23.3.11) and (23.3.15). The interpolation formula (23.6.10) may be written in terms of the columns of \( G = [g_0, g_1, \ldots, g_d] \):

\[
b_t(k) = \sum_{i=0}^{d} G_{ki} t^i = \sum_{i=0}^{d} g_t(i) t^i \Rightarrow b_t = \sum_{i=0}^{d} g_t t^i
\]

(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

\[
b_t = g_0 + g_1 t + g_2 t^2 + g_3 t^3 = \left( (g_3 t + g_2) t + g_1 \right) t + g_0
\]

(23.7.1)

Fig. 23.7.1 shows this realization where we replaced \( g_t \) by their reversed versions \( g_t^R \), which appear in the convolutional filtering equations. The parameter \( t \) appears only in the lower multipliers and can be independently controlled.

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 \( N = 2M + 1 \):

\[
\hat{y}_{n+\tau} = \sum_{k=0}^{N-1} b_{\tau}(k) y_{n-k}, \quad \tau \geq 0
\]

(23.7.3)

Fig. 23.7.2 Double-sided and causal predictive FIR filters, with \( n' = n - M \) and \( t = M + \tau \). Since \( \tau \) 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,
The filters \( b_i(k) \) and \( h_i(k) \) satisfy the following polynomial-preserving moment constraints (being equivalent to \( S^T b_i = u_i \)), where \( i = 0, 1, \ldots, d \):

\[
\sum_{k=-M}^{M} k^M b_i(k) = t^i \quad \Rightarrow \quad \sum_{k=0}^{N-1} k^M h_i(k) = (-\tau)^i, \quad \sum_{k=0}^{N-1} k^M h_i(k) = \tau^i
\]

These constraints imply that Eqs. (23.7.3) and (23.7.4) are exact for polynomials of degree \( r \leq d \). For any such polynomial \( P(n) \), we have:

\[
\sum_{k=0}^{N-1} h_i(k) P(n-k) = P(n-\tau)
\]

For example, we have for the monomial \( P(n) = n^r \) with \( r \leq d \):

\[
\sum_{k=0}^{N-1} h_i(k) (n-k)^r = \sum_{k=0}^{N-1} h_i(k) \sum_{i=0}^{r} \binom{r}{i} n^{r-i} (-1)^i k^i
\]

\[
= \sum_{i=0}^{r} \binom{r}{i} n^{r-i} \sum_{k=0}^{N-1} k^i h_i(k) = \sum_{i=0}^{r} \binom{r}{i} n^{r-i} (-1)^i \tau^i = (n-\tau)^r
\]

It is in the sense of Eq. (23.7.6) that we may think of the transfer function of the filter \( h_i(k) \) as approximating the ideal fractional delay \( z^{-\tau} \):

\[
\sum_{k=0}^{N-1} h_i(k) z^{-k} \approx z^{-\tau}
\]

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 \( h_i(k) = b_{M-i} (M-k) \) and Eqs. (23.6.4) and (23.6.6), we obtain, for \( k = 0, 1, \ldots, N-1 \):

\[
h_i(k) = \prod_{i=0}^{N-1} \left( \frac{T_i - k}{T_i - k} \right) = \sum_{i=N-1-k}^{N-1} \binom{N-1-i}{k} \binom{i}{N-1-k} (-1)^{i-(N-1-k)}
\]

The \( z \)-transform of \( h_i(k) \) is then,

\[
\sum_{k=0}^{N-1} h_i(k) z^{-k} = \sum_{i=0}^{N-1} \sum_{i=0}^{N-1-k} \binom{N-1-i}{k} \binom{i}{N-1-k} (-1)^{i-(N-1-k)} z^{-k}
\]

Changing summation variables and using the binomial expansion of \((z-1)^i\), we obtain,

\[
\sum_{k=0}^{N-1} h_i(k) z^{-k} = z^{-(N-1)} \sum_{k=0}^{N-1} \sum_{i=0}^{m} \binom{N-1-i}{k} \binom{i}{N-1-k} (-1)^{i-(N-1-k)} z^{-k}
\]

\[
= z^{-(N-1)} \sum_{k=0}^{N-1} \binom{N-1-\tau}{i} (-1)^{i-(N-1-k)} z^{-k}
\]

For example, we have for the monomial \( P(n) = n^r \) with \( r \leq d \):

\[
\sum_{k=0}^{N-1} h_i(k) (n-k)^r = \sum_{k=0}^{N-1} h_i(k) \sum_{i=0}^{r} \binom{r}{i} n^{r-i} (-1)^i k^i
\]

\[
= \sum_{i=0}^{r} \binom{r}{i} n^{r-i} \sum_{k=0}^{N-1} k^i h_i(k) = \sum_{i=0}^{r} \binom{r}{i} n^{r-i} (-1)^i \tau^i = (n-\tau)^r
\]

This approximation becomes exact whenever \( \tau \) is an integer, say \( \tau = m \), with \( m = 0, 1, \ldots, N-1 \). Indeed in this case, the summation range \( 0 \leq i \leq N-1 \) in Eq. (23.7.9) can be restricted to \( 0 \leq i \leq N-1-m \) because the binomial coefficient vanishes whenever its (integer) arguments satisfy \( N-1-m < i \leq N-1 \). We then have an ordinary binomial expansion for an integer power:

\[
\sum_{k=0}^{N-1} h_i(k) z^{-k} = z^{-(N-1)} \sum_{i=0}^{m} \binom{N-1-i}{k} \binom{i}{N-1-k} (-1)^{i-(N-1-k)} z^{-k}
\]

which implies the expected result \( h_i(k) = \delta(k-m) \). 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, \( \Delta y_n = y_{n+1} - y_n \), and apply (23.7.9) in the time domain:

\[
\hat{y}_{n-\tau} = \sum_{k=0}^{N-1} h_i(k) y_{n-k} = \sum_{k=0}^{N-1} \binom{N-1-i}{k} \Delta^i y_{n-i-1} (N-1)
\]

This interpolates between the points \( \{y_{n-(N-1)}, \ldots, y_{n-1}, y_n\} \) with \( \tau \) 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,

\[
\hat{y}_{n-(N-1)+x} = \sum_{i=0}^{N-1} \binom{x}{i} \Delta^i y_{n-(N-1)}
\]
and setting $n = N - 1$ so that the data range is $[y_0, y_1, \ldots, y_{N-1}]$, we obtain the usual way of writing Newton's polynomial interpolation formula:

$$\hat{y}_x = \sum_{i=0}^{N-1} \left( \prod_{k=0}^{i-1} \frac{x - k}{(i-k)} \right) y_i = \sum_{i=0}^{N-1} \frac{x(x-1) \cdots (x-l+1)}{l!} \Delta^l y_0$$  \hspace{1cm} (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:

$$h[-(N-2)], h[-(N-1)], h[-N], h[-N+1], h[-N+2] = \frac{1}{2} (\tau - 1) (\tau - 2) \left( \frac{1}{\tau} \right) \sum \left( \begin{array}{c} \frac{1}{\tau} \\ \tau \\ \tau(\tau - 1) \end{array} \right) \frac{(-1)^{n-k}}{n-k} \left( \begin{array}{c} n-2 \\ k \end{array} \right) \left( \begin{array}{c} n-1 \\ k \end{array} \right)$$  \hspace{1cm} (23.7.16)

and the corresponding interpolation formulas:

$$\hat{y}_{n-\tau} = (1 - \tau) y_n + \tau y_{n-1}$$

$$\hat{y}_{n-\tau} = \frac{1}{2} (\tau - 1) (\tau - 2) y_n - \tau (\tau - 2) y_{n-1} + \frac{1}{2} \tau (\tau - 1) y_{n-2}$$

$$\hat{y}_{n-\tau} = \frac{1}{6} (\tau - 1) (\tau - 2) (\tau - 3) y_n + \frac{1}{2} \tau (\tau - 2) (\tau - 3) y_{n-1} - \frac{1}{6} \tau (\tau - 1) (\tau - 2) y_{n-2}$$  \hspace{1cm} (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 $\tau = m/10$, $m = 1, 2, \ldots, 10$.

The bottom row is the case $N = 5$ and $d = N - 1 = 4$ with delays $\tau$ extending over the interval $0 \leq \tau \leq 2$. This filter interpolates between the samples $[y_{n-4}, y_{n-3}, y_{n-2}, y_{n-1}, y_n]$. The chosen range of $\tau$ spans the gaps between $[y_{n-2}, y_{n-1}, y_n]$. The following MATLAB code segment illustrates the generation of the upper two graphs:

```matlab
f = linspace(0,1,1001); \% frequencies 0 \leq \omega \leq \pi
N=3; d=N-1; M = floor(N/2);
Hmag = []; Hdel = [];
for m=1:10,
    tau = m/10;
    h = flip(1interp(N,d,m-tau)); \% 1interp is discussed in Sec. 23.9
    H = freqz(h,1,w);
    Hmag = [Hmag; 10*log10(abs(H))]; \% magnitude responses in dB
    Hdel = [Hdel; tau]; \% phase delays
    figure; plot(f,Hmag); figure; plot(f,Hdel);
end
```

The filters were calculated with the function 1interp (from Sec. 23.9) with arguments $d = N-1$, $t = M - \tau$, with reversed output to account for the definition $h[-\tau](k) = b_{d-\tau}(M-k)$.

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 $f_s$. □


### 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,
23.8. MINIMUM VARIANCE FILTERS

such designs are referred to as “minimum $R_c$” or “minimum variance” filters, as opposed to the “minimum $R_s$” 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 $b_0$ is the equal to the middle value $b_0(0)$ follows from Eq. (23.4.2). Using the symmetry of $B$, we have

$$B^T = B^2 = B^TB$$

Taking matrix elements, we have $B_{km} = (B^T)_{mk} = (B^T)_{mk}$. But, $B_{km}$ is the $k$th component of the $m$th column $b_m$. Using a similar argument as in Eq. (23.2.13), we also have $(B^T)_{mk} = b_m^Tb_k$. Therefore,

$$b_m^Tb_k = b_m(k)$$

For $k = m$, we have the diagonal elements of $B^TB = B$:

$$R = b_m^Tb_m = b_m(m)$$

(23.8.1)

These are recognized as the NRRs of the filters $b_m$. In particular, for $m = 0$, we have $R = b_m^Tb_m = b_0(0)$. 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 $1/F_0, F_2/D_4,$ and $D_{12}/D$. Therefore:

$$(d = 0, 1) \quad R = \frac{1}{N}$$

$$(d = 2, 3) \quad R = \frac{3(3M^2 + 3M - 1)}{(2M + 3)(4M^2 - 1)}$$

$$(d = 4, 5) \quad R = \frac{15(15M^4 + 30M^3 - 35M^2 - 50M + 12)}{4(2M + 5)(4M^2 - 1)(4M^2 - 9)}$$

(23.8.2)

In the limit of large $N$ or $M$, we have the approximate asymptotic expressions:

$$(d = 0, 1) \quad R = \frac{1}{N}$$

$$(d = 2, 3) \quad R \approx \frac{9/4}{N} = \frac{2.25}{N}$$

$$(d = 4, 5) \quad R \approx \frac{225/64}{N} = \frac{3.52}{N}$$

(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:

$$B^TS = S, \quad S^TB = S$$

(23.8.4)

Indeed, $B^TS = BS = S(S^TS)^{-1}S^TS = S$. Column-wise the first equation states that:

$$B^T[s_0, s_1, \ldots, s_d] = [s_0, s_1, \ldots, s_d] \quad \Rightarrow \quad B^T_{si} = s_i, \quad i = 0, 1, \ldots, d$$

Thus, the basis vectors $s_i$ remain invariant under projection, but that is to be expected because they already lie in $S$. In fact, any other linear combination of them, such as Eq. (23.2.30), remains invariant under $B$, that is, $B^T\hat{y} = \hat{y}$.

This property answers the question: When are the smoothed values equal to the original ones, $\hat{y} = y$, or, equivalently, when is the error zero, $e = y - B^T\hat{y}$, the error will be zero if and only if $B^T\hat{y} = y$, which means that $y$ already lies in $S$, that is, it is a linear combination of $s_i$. This implies that the samples $y_m$ are already $d$th order polynomial functions of $m$, as in Eq. (23.2.27).

The second equation in (23.8.4) implies certain constraints on the filters $b_m$, 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:

$$S^T = [u_{-M}, \ldots, u_{-1}, u_0, u_1, \ldots, u_M]$$

(23.8.5)

For example, in the $N = 5, d = 2$ case, we have:

$$S^T = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ -2 & -1 & 0 & 1 & 2 \\ 4 & 1 & 0 & 1 & 4 \end{pmatrix} = [u_{-2}, u_{-1}, u_0, u_1, u_2]$$

It is easily verified that the $m$th column $u_m$ is simply

$$u_m = \begin{bmatrix} 1 \\ m \\ m^2 \\ \vdots \\ m^d \end{bmatrix}$$

$$-M \leq m \leq M$$

(23.8.6)

which is the same as $u_t$ at $t = m$, in terms of the definition (23.6.10). Using $B = GST$, we can express the LPSM filters $b_m$ in terms of $u_m$, as follows:

$$[b_{-M}, \ldots, b_{-1}, b_0, b_1, \ldots, b_M] = B = GST = G[u_{-M}, \ldots, u_{-1}, u_0, u_1, \ldots, u_M]$$

which implies:

$$b_m = G u_m = SF^{-1}u_m$$

(23.8.7)

Multiplying by $S^T$, we find $S^Tb_m = S^T SF^{-1}u_m = u_m$, or,

$$S^Tb_m = u_m \quad \Rightarrow \quad \begin{bmatrix} s_0^T b_m \\ s_1^T b_m \\ \vdots \\ s_d^T b_m \end{bmatrix} = \begin{bmatrix} 1 \\ m \\ \vdots \\ m^d \end{bmatrix}$$

(23.8.8)

These relationships are the column-wise equivalent of $S^TB = S$. Thus, each LPSM filter $b_m$ satisfies $(d+1)$ linear constraints:

$$s_i^T b_m = m^i, \quad i = 0, 1, \ldots, d$$

(23.8.9)
23.8. MINIMUM VARIANCE FILTERS

Writing the dot products explicitly, we have equivalently:

\[ \sum_{n=-M}^{M} n^i b_m(n) = m^i, \quad i = 0, 1, \ldots, d \]  

(23.8.10)

In particular, for the steady-state LPSM filter \( b_0 \), we have \( u_0 = [1, 0, 0, \ldots, 0]^T \), with \( i \)th component \( \delta(i) \). Therefore, the constraint \( S^T b_0 = u_0 \) reads component-wise:

\[ \sum_{n=-M}^{M} n^i b_0(n) = \delta(i), \quad i = 0, 1, \ldots, d \]  

(23.8.11)

For \( i = 0 \), this is the usual DC constraint:

\[ \sum_{n=-M}^{M} b_0(n) = 1 \]  

(23.8.12)

and for \( i = 1, 2, \ldots, d \):

\[ \sum_{n=-M}^{M} n^i b_0(n) = 0 \]  

(23.8.13)

The quantity in the left-hand side of Eq. (23.8.11) is called the \( i \)th 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:

\[ j^i B_0^{(i)}(0) = j^i \frac{d^i B_0(\omega)}{d\omega^i} \Bigg|_{\omega=0} = \sum_{n=-M}^{M} n^i b_0(n) \]  

(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 \( \omega = 0 \):

\[ B_0(0) = 1, \quad B_0^{(i)}(0) = 0, \quad i = 1, 2, \ldots, d \]  

(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 \( B_0^{(4)}(0) = 0 \). Similarly, the quartic filters are even flatter because they satisfy two flatness conditions: \( B_0^{(2)}(0) = B_0^{(4)}(0) = 0 \). The cutoff frequencies are approximately doubled and tripled in the cases \( d = 2 \) and \( d = 4 \), as compared to \( d = 0 \).

![Magnitude Response, N = 7](image1)
![Magnitude Response, N = 15](image2)

**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,

\[ \sum_n n^i x(n) = \sum_n n^i y(n), \quad i = 0, 1, \ldots, d \]  

(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:

\[ \hat{X}^{(i)}(0) = B_0(0) Y^{(i)}(0) = Y^{(i)}(0), \quad i = 0, 1, \ldots, d \]

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
\[
\mathcal{R} = b^Tb = \sum_{n=-M}^{M} b(n)^2 = \min
\] (23.8.17)
subject to the \( d + 1 \) constraints, with a given \( u = [u_0, u_1, \ldots, u_d]^T \):
\[
s_f^Tb = \sum_{n=-M}^{M} n^i b(n) = u_i, \quad i = 0, 1, \ldots, d \implies S_f^Tb = u \tag{23.8.18}
\]
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:
\[
\mathcal{J} = b^Tb + 2 \sum_{i=0}^{d} \lambda_i (u_i - S_f^Tb) = b^Tb + 2 \lambda^T(u - S_f^Tb) \tag{23.8.19}
\]
The gradient of \( \mathcal{J} \) with respect to the unknown filter \( b \) is:
\[
\frac{\partial \mathcal{J}}{\partial b} = 2b - 2S\lambda \]
Setting the gradient to zero, and solving for \( b \) gives:
\[
b = S\lambda = [s_0, s_1, \ldots, s_d] \begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_d \end{bmatrix} = \sum_{i=0}^{d} \lambda_i s_i
\]
Component-wise this means that \( b(n) \) has the polynomial form:
\[
b(n) = \sum_{i=0}^{d} \lambda_i s_i(n) = \sum_{i=0}^{d} \lambda_i n^i, \quad -M \leq n \leq M
\]
The Lagrange multiplier vector \( \lambda \) is determined by imposing the desired constraint:
\[
u = S^Tb = S^T S \lambda = F\lambda \implies \lambda = F^{-1}u
\]
resulting in the optimum \( b \):
\[
b = S\lambda = SF^{-1}u = S(S^T S)^{-1}u = Gu
\] (23.8.20)
Since the solution minimizes the norm \( b^Tb \), it is recognized to be the minimum-norm solution of the \((d+1) \times N\) full-rank under-determined linear system \( S_f^Tb = u \), which can be obtained by the pseudoinverse of \( S_f^T \), that is, \( b = (S_f^T)^+u \), \((S_f^T)^+ = S(S_f^TS_f)^{-1}\). In MATLAB, we can simply write \( b = \text{pinv}(S_f^T)u \).
Comparing this solution with Eqs. (23.8.7) and (23.8.8), we conclude that the LPSM filters \( b_m \) can be thought of as the optimum filters that have minimum NRR with constraint vectors \( u = u_m \), that is, the minimization problems,
\[
\mathcal{R} = b_m^Tb_m = \min, \quad \text{subject to } S_f^Tb_m = u_m \tag{23.8.21}
\]
have solutions,
\[
b_m = SF^{-1}u_m = Gu_m, \quad -M \leq m \leq M \tag{23.8.22}
\]
and putting these together as the columns of \( B \), we obtain Eq. (23.2.31):
\[
B = [\ldots, b_m, \ldots ] = G[\ldots, u_m, \ldots ] = GS_f = SF^{-1}S_f^T \tag{23.8.23}
\]
In particular, the steady-state LPSM filter \( b_0 \) minimizes the NRR with the constraint vector \( u = u_0 = [1, 0, \ldots, 0]^T \). This was precisely the problem first formulated and solved using Lagrange multipliers by Schiaparelli [620].
Similarly, the interpolating filter \( b_t = Gu_t \) of Eq. (23.6.10) can be thought of as the solution of the constrained minimization problem:
\[
\mathcal{R} = b_t^Tb = \min, \quad \text{subject to } S_f^Tb = u_t, \quad \text{where } u_t = [1, t, t^2, \ldots, t^d]^T
\]

### 23.9 Predictive Differentiation Filters

Going back to the polynomial fit of Eq. (23.6.9), that is,
\[
\hat{y}_t = \sum_{i=0}^{d} c_i t^i = c^T u_t = y^T Gu_t = y^T b_t, \quad \text{where } b_t = Gu_t, \tag{23.9.1}
\]
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 \( y \):
\[
\hat{y}_t |_{t=0} = c_0 = b_t^T y = g_0^T y
\]
\[
\dot{\hat{y}}_t |_{t=0} = c_1 = g_1^T y
\]
\[
\ddot{\hat{y}}_t |_{t=0} = 2c_2 = g_2^T y, \quad \text{etc.}
\]
The first derivative at an arbitrary value of \( t \) is given by:
\[
\dot{\hat{y}}_t = y^T b_t, \quad b_t = Gu_t
\]
where the differentiation operation can be expressed as matrix multiplication:
\[
u_t = \begin{bmatrix} 1 \\ t \\ t^2 \\ \vdots \\ t^d \end{bmatrix} \implies \bar{u}_t = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ t \\ t^2 \vdots \vdots \vdots \vdots \vdots \end{bmatrix} = D u_t \tag{23.9.2}
\]
where \( D \) is the \((d+1) \times (d+1)\) matrix with the sequence of numbers \( [1, 2, \ldots, d] \) along its first subdiagonal and zeros everywhere else. Such a matrix can be constructed trivially in MATLAB, for example, by:
\[
D = \text{diag}(1:d, -1);
\]
23.9. PREDICTIVE DIFFERENTIATION FILTERS

It follows that the first-order differentiation filter is \( \hat{b}_1 = GDu \). In particular, the differentiation filter at the sample point \( t = m \) is \( \hat{b}_m = GDu_m \) and the corresponding estimated derivative:

\[
\hat{y}_m = \hat{b}_m^T y = u_m^T D^T G^T y, \quad -M \leq m \leq M \tag{23.9.3}
\]

Stacking these together into a column vector, we obtain:

\[
\hat{y} = SD^T G^T y = B^T y, \quad \text{where} \quad B = GDS^T = SF^{-1}DS^T \tag{23.9.4}
\]

so that \( \hat{B} \) has the \( \hat{B} \) of the matrix \( u \) at time \( t \).

For the causal versions, we have for \( i \):

\[
\hat{b}(i) = \begin{cases} \frac{k}{F_2} & \text{for } d = 1, \\ \frac{F_1}{F_2} - \frac{F_2}{D_4} (2t) + \frac{F_0}{D_4} (2tk^2) & \text{for } d = 2, \\ \frac{F_6}{D_4} - \frac{F_2}{D_4} (2t) + \frac{F_0}{D_4} (2tk^2) & \text{for } d = 3, \\ \frac{F_6}{D_4} - \frac{D_10}{D} (2t) + \frac{F_8}{D} (2tk^2) - \frac{F_2}{D_8} (k3t^2 + k^3) & \text{for } d = 4, \end{cases} \tag{23.9.5}
\]

where \( k = 0, 1, \ldots, N - 1 \). We note that \( \hat{h} \) can be obtained by differentiating \( h \) with respect to \( \tau \). The derivative filter is independent of \( \tau \) because it corresponds to fitting a first-order polynomial. For \( d = 2 \), we have similarly,

\[
\hat{h}_2(k) = \frac{F_2}{D_4} - \frac{F_6}{D_4} (M + \tau) (M - k) \tag{23.9.6}
\]

As in Eq. (23.7.2), the redefinition \( \hat{h}(i)(k) = b(i)(M - k) \) will result into a causal version of the predictive differentiator filter, with Eq. (23.9.7) transforming into:

\[
\hat{y}_n^{(i)} = \sum_{k=-M}^{M} b_n^{(i)}(k)y_{n+k} = \sum_{k=-M}^{M} b_n^{(i)}(-k)y_{n-k} \tag{23.9.7}
\]

As in Eq. (23.7.2), the redefinition \( \hat{h}(i)(k) = b(M, \tau)(M - k) \) will result into a causal version of the predictive differentiator filter, with Eq. (23.9.7) transforming into:

\[
\hat{y}_n^{(i)} = \sum_{k=0}^{N-1} h_n^{(i)}(k)y_{n+k} \quad \text{(causal predictive differentiator)} \tag{23.9.8}
\]

One can easily obtain closed-form expressions for the differentiation filters \( b_n^{(i)}(k) \) 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 \( t \)-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:

\[
\hat{y}_n^{(i)} = \sum_{k=0}^{N-1} h_n^{(i)}(k)y_{n+k} \quad \text{(causal predictive differentiator)} \tag{23.9.8}
\]

\[
\hat{y}_n^{(i)} = \sum_{k=0}^{N-1} h_n^{(i)}(k)y_{n+k} \quad \text{(causal predictive differentiator)} \tag{23.9.8}
\]
23.9. PREDICTIVE DIFFERENTIATION FILTERS

The corresponding first- and second-order differentiation matrices will be:

$$\hat{B} = GD^1S^T = \frac{1}{35} \begin{bmatrix} -27 & -17 & -7 & 3 & 13 \\ 6.5 & 1.5 & -3.5 & -8.5 & -13.5 \\ 20 & 10 & 0 & -10 & -20 \\ 13.5 & 8.5 & 3.5 & -1.5 & -6.5 \\ -13 & -3 & 7 & 17 & 27 \end{bmatrix}, \quad D^1 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix}$$

$$\hat{B} = GD^2S^T = \frac{1}{35} \begin{bmatrix} 10 & 10 & 10 & 10 \\ -5 & -5 & -5 & -5 \\ -10 & -10 & -10 & -10 \\ -5 & -5 & -5 & -5 \\ 10 & 10 & 10 & 10 \end{bmatrix}, \quad D^2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix}$$

The central columns agree with Eq. (23.2.25). The interpolating smoothing and first-order differentiation filters are given by:

$$b = Gu = \frac{1}{35} \begin{bmatrix} -3 & -7 & 5 \\ 12 & -3.5 & -2.5 \\ 17 & 0 & -5 \\ 12 & 3.5 & -2.5 \\ -3 & 7 & 5 \end{bmatrix} t \begin{bmatrix} 1 \\ t^2 \end{bmatrix} = \frac{1}{35} \begin{bmatrix} -3 & -7t + 5t^2 \\ 12 & -3.5t - 2.5t^2 \\ 17 & -5t^2 \\ 12 & 3.5t - 2.5t^2 \\ -3 & 7 + 5t \end{bmatrix}$$

$$b = GDiST = \frac{1}{35} \begin{bmatrix} -3 & -7 & 5 \\ 12 & -3.5 & -2.5 \\ 17 & 0 & -5 \\ 12 & 3.5 & -2.5 \\ -3 & 7 & 5 \end{bmatrix} i \begin{bmatrix} 1 \\ t^2 \end{bmatrix} = \frac{1}{35} \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix} [t, t^2]$$

The MATLAB function `lpdiff` implements the design of the differentiation matrices:

```matlab
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:

$$S = QR, \quad Q^TQ = I, \quad R = \text{upper triangular}$$

$$G = S(S^T S)^{-1} = QR^{-T}$$

$$B(i) = GD^iS^T = Q(R^{-T}D^iR^T)Q^T$$

The predictive/interpolating differentiation filters $b(i)$ are the minimum-norm solution of the under-determined linear system $S^Tb = D^iu(i)$, or, equivalently the solution of the constrained minimization problem:

$$\mathcal{R} = b^Tb = \min, \quad \text{subject to} \quad S^Tb = D^iu(i)$$

The MATLAB function `lpinterp` implements the design of predictive and interpolating differentiation filters, essentially carrying out the operation $b = \text{pinv}(S^T)D^iu(i)$:

```matlab
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 $t = m, -M \leq m \leq M$, the filter $b$ 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.

![Differentiating noisy signals](image)

The noiseless signal is a raised cosine $s(t) = 0.5 - 0.5 \cos(\omega t)$, with $0 \leq t \leq T$ 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 $t_n = n\Delta t = nT/L, n = 0, 1, \ldots, L$, by adding zero-mean white gaussian noise $\nu_n$ of variance, say $\sigma^2$, so that the noisy observations are:

$$y_n = s(t_n) + \nu_n, \quad n = 0, 1, \ldots, L$$

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 $\Delta t$ 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 \( y_n \), as well as to the differenced signal \( \text{diff}(y) / \Delta t \), is illustrated.

The following MATLAB code illustrates the generation of the bottom-right graph:

```matlab
T = 10; L = 50; Dt = T/L; w = 2*pi/T; sigma = 0.1;

% noiseless signal

x = 0.5 - 0.5*cos(w*t);

% noisy signal

y = x + 0.5*w*sin(w*t);

% first-order differentiation filter
B1 = lpdiff(N,d,1);

% estimated derivative of \( x(t) \)
xd = lpfilt(B1,x)/Dt;

% estimated derivative from the noisy signal
x1 = lpfilt(B1,y)/Dt;

% differenced signal estimates the derivative
yd = diff(y)/Dt; td = t(2:end);

plot(t,xd,'--', t,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 `lpdiff`, which is explained in the next section.

### 23.10 Filtering Implementations

In smoothing a length-\( L \) signal block \( y_n, n = 0, 1, \ldots, L - 1 \), with a double-sided filter \( h_m, -M \leq m \leq M \), the output signal \( \hat{x}_n \) is given by the convolutional form:

\[
\hat{x}_n = \min(m,M) \sum_{m=-M}^{M} h_m y_{n-m}, \quad -M \leq n \leq L + M - 1 \tag{23.10.1}
\]

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, \( M \leq n \leq L - M - 1 \), correspond to the steady-state output computed from the steady-state version of Eq. (23.10.1):

\[
\hat{x}_n = \sum_{m=-M}^{M} h_m y_{n-m}, \quad M \leq n \leq L - M - 1 \tag{23.10.2}
\]

The range of the output index \( n \) and the limits of summation in (23.10.1) are determined from the inequalities \( -M \leq m \leq M \) and \( 0 \leq n - m \leq L - 1 \) that must be satisfied by the indices of \( h_m \) and \( y_{n-m} \). However, only the subrange \( \{\hat{x}_n, 0 \leq n \leq L - 1\} \) is of interest since these output samples represent the smoothed values of the corresponding input samples \( \{y_n, n = 0, 1, \ldots, L - 1\} \). This is illustrated in Fig. 23.10.1.

The first and last \( M \) samples in the subrange \( 0 \leq n \leq L - 1 \) 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:

\[
\begin{bmatrix}
\hat{x}_{-2} & \hat{x}_{-1} & \ldots & \hat{x}_0 & \hat{x}_1 & \ldots & \hat{x}_{L-2} & \hat{x}_{L-1} \\
\end{bmatrix}
\begin{bmatrix}
h_{-2} & h_{-1} & 0 & 0 & 0 & 0 & 0 & 0 \\
h_{-1} & h_{-2} & h_0 & 0 & 0 & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
h_0 & h_1 & h_2 & h_3 & h_4 & h_5 & y_0 & y_1 \\
\end{bmatrix}
\]

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 \( h = [h_{-M}, \ldots, h_0, \ldots, h_M]^T \), we have the syntax:

- \( H = \text{convmtx}(h,L) \); % built-in convolution matrix
- \( H = \text{convmat}(h,L) \); % sparse version of convmtx
- \( H = \text{datamat}(h,L-1) \); % used extensively in [45]
23.10. FILTERING IMPLEMENTATIONS

Dropping the first and last two outputs, we obtain the outputs in the subrange $0 \leq n \leq 7$:

$$\hat{x}_0 = \begin{bmatrix} h_0 & h_{-1} & h_{-2} & 0 & 0 & 0 & 0 & y_0 \\ h_1 & h_{-1} & h_{-2} & 0 & 0 & 0 & 0 & y_1 \\ \vdots \\ h_2 & h_1 & h_0 & h_{-1} & h_{-2} & 0 & 0 & 0 & y_2 \\ h_3 & 0 & h_2 & h_1 & h_0 & h_{-1} & h_{-2} & 0 & 0 & y_3 \\ \vdots \\ h_6 & 0 & 0 & 0 & h_2 & h_1 & h_0 & h_{-1} & h_{-2} & y_6 \\ \vdots \\ h_7 & 0 & 0 & 0 & 0 & h_2 & h_1 & h_0 & h_{-1} & y_7 \end{bmatrix} = Hy \ (23.10.3)$$

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 $-M \leq n \leq L - 1 + M$, and the desired output samples over the subrange $0 \leq n \leq L - 1$ 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 $\tilde{y}_{n-M}$ 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 $n = M$:

$$\hat{x}_M = \tilde{y}_M = \sum_{k=-M}^{M} b_0(k) y_{M+k}$$

Instead of calculating the previous output $\hat{x}_{M-1}$ using the transient version of $b_0(k)$,

$$\hat{x}_{M-1} = \sum_{k=-M}^{M} b_0(k) y_{M-1+k}$$

one could estimate $\hat{x}_{M-1}$ using $\tilde{y}_{M+m}$ with $m = -1$, that is, using $b_{-1}(k)$, and using $b_{-2}(k), b_{-3}(k), \ldots, b_{-M}(k)$ for the other initial $M$ outputs:

$$\hat{x}_{M-1} = \tilde{y}_{M-1} = \sum_{k=-M}^{M} b_{-1}(k) y_{M+k}$$

$$\hat{x}_{M-2} = \tilde{y}_{M-2} = \sum_{k=-M}^{M} b_{-2}(k) y_{M+k}$$

$$\vdots$$

$$\hat{x}_0 = \tilde{y}_M = \sum_{k=-M}^{M} b_{-M}(k) y_{M+k} \quad (23.10.4)$$

Similarly, one can use the filters $b_m(k)$ for $m = 1, 2, \ldots, M$ to calculate the last $M$ smoothed outputs, starting with the last steady-state output at $n = L - 1 - M$ and proceeding to the end $n = L - 1$:

$$\hat{x}_{L-M} = \tilde{y}_{L-1-M} = \sum_{k=-M}^{M} b_1(k) y_{L-1-M+k}$$

$$\hat{x}_{L-M+1} = \tilde{y}_{L-1-M+2} = \sum_{k=-M}^{M} b_2(k) y_{L-1-M+k}$$

$$\vdots$$

$$\hat{x}_{L-1} = \tilde{y}_{L-1-M+M} = \sum_{k=-M}^{M} b_M(k) y_{L-1-M+k} \quad (23.10.5)$$

The following example illustrates the computational steps for the input-on, steady, and input-off output samples, where we denoted $b_{n,k} = b_m(k)$ for simplicity:

$$\hat{x} = \begin{bmatrix} \hat{x}_0 \\ \hat{x}_1 \\ \vdots \\ \hat{x}_6 \\ \hat{x}_7 \end{bmatrix} = H_y$$

In particular, for $N = 5$ and $d = 2$, the convolutional filtering matrix will be:

$$\hat{x} = \begin{bmatrix} \hat{x}_0 \\ \hat{x}_1 \\ \vdots \\ \hat{x}_6 \\ \hat{x}_7 \end{bmatrix} = H_y$$

with entries obtained from the matrix $B$ of Eq. (23.2.16):

$$B = \begin{bmatrix} 31 & 9 & -3 & -5 & 3 & 0 & 0 \\ 13 & 12 & 6 & -5 & 0 & 0 & 0 \\ -3 & 12 & 17 & 12 & -3 & 0 & 0 \\ 0 & -3 & 12 & 17 & 12 & -3 & 0 \\ 0 & 0 & -3 & 12 & 17 & 12 & -3 \\ 0 & 0 & 0 & -5 & 6 & 12 & 13 \end{bmatrix}$$

$$= [b_{-2}, b_{-1}, b_0, b_1, b_2]$$

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 \( y \).

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 = \text{convmat}(f1p(B(:,1:M+1)), L); \quad \text{\% ordinary \((L+2M) \times L\) convolution matrix}
H = H(M+1:L+M,:); \quad \text{\% extract the \( L \times L \) convolution submatrix}
H(1:M, 1:N) = B(:,1:M)'; \quad \text{\% redefine upper-left \( M \times L \) corner}
H(L-M+1:L, L-N+1:N) = B(:,M+2:N)'; \quad \text{\% redefine lower-right \( M \times L \) corner}
\]

The function \( f1p \) reverses the central column of \( B \) because \text{convmat} expects as input the actual filter, not its reverse. The above steps have been incorporated into the function \text{lpmat} with syntax:

\[
H = \text{lpmat}(B,L); \quad \text{\% local polynomial filter matrix of size \( L \times L \)}
\]

Once the \( L \times L \) matrix \( H \) is constructed, the actual filtering of a length-\( L \) input block \( y \) is straightforward, that is, \( \hat{x} = H y \), 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:

\[
\hat{x}_n = \sum_{k=-M}^{M} b_0(k) y_{n+k}, \quad M \leq n \leq L - 1 - M \quad (23.10.6)
\]

The following MATLAB code illustrates this approach:

\[
y = B(:,1:M)’ * x(1:N); \quad \text{\% first \( M \) transient outputs}
\text{for } n = M+1:L-M,
\]

\[
y = y; \quad B(:,M+1)' * x(n-M:n+M); \quad \text{\% filtered by central column of \( B \)}
\]

\[
y = y; \quad B(:,M+2:N)' * x(L-N+1:L); \quad \text{\% last \( M \) transient outputs}
\]

These steps are implemented in the MATLAB function \text{lpfilt2}. A faster version is the function \text{lpfilt}, which uses MATLAB’s built-in filtering functions. Thus, three possible ways of computing the filtered output \( \hat{x} \) given a smoothing matrix \( B \) are as follows (assuming that \( y \) is a length-\( L \) column vector):

\[
x_{\text{hat}} = \text{lpmat}(B,L)y; \quad \text{\% use \( L \times L \) convolution matrix constructed from \( B \)}
\]

\[
x_{\text{hat}} = \text{lpfilt2}(B,y); \quad \text{\% use directly the filtering equations (23.10.4)–(23.10.6)}
\]

\[
x_{\text{hat}} = \text{lpfilt}(B,y); \quad \text{\% fast version using the function \text{filtdbl}}
\]

The function \text{lpfilt} internally calls the function \text{filtdbl}, which uses the built-in function \text{conv} to implement the FIR filtering by the steady-state double-sided central filter. The following code segment illustrates the essential part of \text{lpfilt}:

\[
x_{\text{hat}} = \text{filtdbl}(f1p(B(:,1:M+1)), y); \quad \text{\% filter with the central column of \( B \)}
\]

\[
x_{\text{hat}}(1:M) = B(:,1:M)' * y(1:N); \quad \text{\% correct the first \( M \) transient outputs}
\]

\[
x_{\text{hat}}(end-M+1:end) = B(:,M+2:N)' * y(end-N+1:end); \quad \text{\% correct the last \( M \) transient outputs}
\]

where the function \text{filtdbl} has usage:

\[
y = \text{filtdbl}(h,x); \quad \text{\% filtering by double-sided FIR filter}
\]

The function \text{filtdbl} is essentially the ordinary convolution of the length-(\( 2M+1 \)) filter \( h \) and the length-\( L \) signal \( x \), 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:

\[
B = \begin{bmatrix}
0 & h_0 & h_1 & h_2 & 0 & 0 \\
0 & h_{-1} & h_0 & h_1 & h_2 & 0 \\
0 & h_{-2} & h_{-1} & h_0 & h_1 & h_2 \\
0 & 0 & h_{-2} & h_{-1} & h_0 & h_1
\end{bmatrix}
\]

and contains the reversed filter \( h \) 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 \textit{minimum-revision} method that uses end-point asymmetric filters constructed from a given central filter \( h \). 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 \text{lpfilt}. The MATLAB function \text{minrev} implements Musgrave’s method:

\[B = \text{minrev}(h,R); \quad \text{\% Musgrave’s minimum revision asymmetric filters}\]

where \( R \) is a scalar parameter. The method is widely used in the X-11 method of seasonal adjustment and trend extraction.

\textbf{Example 23.10.1:} Schiaparelli was the first one to systematically pose and solve the minimum-NKR 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 \( y_{\text{n}} \) 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:

\[
b_0 = \frac{1}{143} \begin{bmatrix} -11, 0, 9, 16, 21, 24, 25, 24, 21, 16, 9, 0, -11 \end{bmatrix}
\]

\[
b_0 = \frac{1}{2431} \begin{bmatrix} 110, -198, -135, 110, 390, 600, 677, 600, 390, 110, -135, -198, 110 \end{bmatrix}
\]

The following program segment illustrates the computations:

\[
Y = \text{loadfile}(’schiaparelli.dat’); \quad \text{\% data file available in the ISP & OSP toolboxes}
n = Y(:,2); \quad \text{\% extract \( n \) and \( y_{\text{n}} \) from the columns of \( Y \)}
N=13; d=3; M=\text{floor}(N/2); \quad \text{\% filter length and polynomial order}
B = \text{lpsm}(N,d); \quad \text{\% construct LPSM matrix \( B \)}
x = \text{lpfilt}(B,y); \quad \text{\% filter noisy observations}
b0 = \text{B(:,M+1)}; \quad \text{\% middle column of \( B \)}
x0 = \text{filtdbl}(b0,y); \quad \text{\% filter with \( b_0 \) only}
pb(n,y,’.’, n,x,’.’, n,x0,’.’);
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 $b_0$, 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:

```matlab
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
x = lpfilt(B,y); % smooth the data vector
figure; plot(n,y,':', n,x,'-');

% middle-left graph uses the SVD signal enhancement method
M=10; r=2; K=40; x = svdenh(y,M,r,K); % SVD enhancement method
```

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.

The following MATLAB code segment illustrates the computation of the corresponding smoothed signals for these four methods:

```matlab
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);
D = diff(eye(N),s);

cvx_begin
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:

```matlab
N = 65; d=3; B = lpsm(N,d);x=y;
for i=1:3, x = lpfilt(B,x); end
s=3; W = diag(hend(N,s));

LPSM filter, repeated twice

Henderson filter, repeated twice
```

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, $\text{diff}(y)$, 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 $\hat{x} = \text{lpfilt}(B_1,y)$ with the differentiator matrix obtained from $B_1 = \text{lpdf}(N,d,i)$ 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 $\text{lpdf}$ is discussed in the next section.) Finally, the lower-right graph shows the second derivative computed with the filter $B_2 = \text{lpdf}(N,d,i)$ with $i = 2$, and compares it with the second difference signal, $\text{diff}(\text{diff}(y))$, which is even more noisy than the first difference. The following MATLAB code illustrates the computations:

```matlab
d=1; i=1; B1 = lpdiff(N,d,i);
plot(n, lpfilt(B1,y), n(2:end), diff(y),':');
d=2; i=1; B1 = lpdiff(N,d,i);
plot(n, lpfilt(B1,y), n(2:end), diff(y),':');
s=3; W = diag(hend(N,s));
d=2; i=1; B1 = lpdiff(N,d,i,W);
plot(n, lpfilt(B1,y), n(2:end), diff(y),':');
d=2; i=2; B2 = lpdiff(N,d,i);
plot(n, lpfilt(B2,y), n(3:end), diff(y),':');
```

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, $\text{diff}(\hat{x}_n)$. 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:

$$J = \sum_{m=M}^{M} e_m^2 = \sum_{m=-M}^{M} \left( y_m - \hat{y}_m \right)^2 = \sum_{m=-M}^{M} \left( y_m - \sum_{i=0}^{d} c_i m^i \right)^2 = \text{min}$$

This can be generalized by using unequal positive weights, $w_m$, $-M \leq m \leq M$:

$$J = \sum_{m=M}^{M} w_m e_m^2 = \sum_{m=-M}^{M} w_m \left( y_m - \sum_{i=0}^{d} c_i m^i \right)^2 = \text{min} \quad (23.11.1)$$

Introducing the diagonal matrix $W = \text{diag}(\{w_{-M},...,w_0,...,w_M\})$, we may write Eq. (23.11.1) compactly as:

$$J = \mathbf{e}^T W \mathbf{e} = (\mathbf{y} - SC)^T W (\mathbf{y} - SC) = \text{min} \quad (23.11.2)$$

where $\mathbf{y}, \mathbf{S}, \mathbf{c}$ 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:

$$S^T W e = S^T W (\mathbf{y} - SC) = 0 \iff (S^T W S) c = S^T W y \quad (23.11.3)$$
with solution for \( c \) and the estimate \( \hat{y} = Sc \):

\[
\begin{align*}
  c &= (S^T WS)^{-1} S^T Wy = G^T y \\
  \hat{y} &= Sc = S(S^T WS)^{-1} S^T Wy = B^T y
\end{align*}
\]  

(23.11.4)

where we defined

\[
\begin{align*}
  G &= W S (S^T WS)^{-1} \\
  B &= G S^T = W S (S^T WS)^{-1} S^T
\end{align*}
\]  

(23.11.5)

The matrix \( B \) satisfies the following properties:

\[
\begin{align*}
  S^T B &= S^T \\
  B^T &= W^{-1} B W \\
  B W B^T &= B W = W B^T
\end{align*}
\]  

(23.11.6)

The first implies the usual polynomial-preserving moment constraints \( S^T b_m = u_m \), for \(-M \leq m \leq M\), where \( b_m \) is the \( m \)th 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:

\[
J_{\text{min}} = e^T W e = y^T W y - y^T B W y - y^T W B^T y + y^T W B^T y = y^T W y - y^T B W y
\]  

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 \( J W = W J \), where \( J \) is the column-reversing matrix consisting of ones along its antidiagonal, that is, the reverse of a column vector is \( b^T \) of \( B \). Under this assumption on \( W \), it can be shown that \( B \) is also centro-symmetric:

\[
J B = B J \Rightarrow b^T_m = b_{-m}, \quad -M \leq m \leq M
\]  

(23.11.7)

This can be derived by noting that reversing the basis vector \( s \) simply multiplies it by the phase factor \((-1)^i\), so that \( J S = S \Omega \), where \( \Omega \) is the diagonal matrix of phase factors \((-1)^i\), \( i = 0, 1, \ldots, d \). This then implies Eq. (23.11.7). Similarly one can show that \( J G = G \Omega \), so that the reverse of each differentiation filter is \( g^i = (-1)^i g \).

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 \( S^T b_m = u_m \). To this end, we consider the constrained minimization of a generalized or “prefiltered” NRR:

\[
R = b^T V b = \min \quad \text{subject to} \quad S^T b = u
\]  

(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:

\[
R = \sum_{n,m=-M}^M b(n)V_{n-m}b(m) = \frac{1}{2\pi} \int_{-\pi}^{\pi} |B(\omega)|^2 V(\omega) d\omega
\]  

(23.11.9)

where we set \( V_{nm} = V_{n-m} \) because of the Toeplitz property, and introduced the corresponding DTFTs:

\[
B(\omega) = \sum_{n=-M}^M b(n)e^{-jn\omega}, \quad V(\omega) = \sum_{k=-\infty}^{\infty} V_k e^{-jnk}\omega
\]  

(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 \( R \) will be the ordinary NRR of the cascaded filter \( F(\omega) = D(\omega)B(\omega) \) or \( F(z) = D(z)B(z) \):

\[
R = \frac{1}{2\pi} \int_{-\pi}^{\pi} |B(\omega)|^2 V(\omega) d\omega = \frac{1}{2\pi} \int_{-\pi}^{\pi} |B(\omega)D(\omega)|^2 d\omega
\]  

(23.11.11)
23.11. MINIMUM ROUGHNESS WEIGHTED POLYNOMIAL FILTERS

The minimum-$R_e$ 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 $u$, the solution of the problem (23.11.8) is obtained by introducing a Lagrange multiplier vector $\lambda$: 

$$J = b^T V b + 2A^T (u - S^T b) = \min$$

leading to the solution:

$$A = (S^T V^{-1} S)^{-1} u$$
$$b = V^{-1} S \lambda = V^{-1} S (S^T V^{-1} S)^{-1} u$$  (23.11.12)

If we choose $u_m = [1, m, m^2, \ldots, m^d]^T$ as the constraint vectors and put together the resulting solutions as the columns of a matrix $B$, then,

$$B = [\ldots b_m \ldots] = V^{-1} S (S^T V^{-1} S)^{-1} [\ldots u_m \ldots]$$

or, because $S^T = [\ldots u_m \ldots]$,

$$B = V^{-1} S (S^T V^{-1} S)^{-1} S^T$$  (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]:

$$WS = SC \Rightarrow WS = V^{-1} SC$$  (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,

$$G = WS(S^T WS)^{-1} = V^{-1} S (S^T V^{-1} S)^{-1}$$

so that

$$B = WS(S^T WS)^{-1} S^T = V^{-1} S (S^T V^{-1} S)^{-1} S^T$$  (23.11.15)

For the minimum-$R_e$ filters, the particular choices for $W$ 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 $u = D^T u_t$ as the constraint vector in (23.11.12), we obtain,

$$b^{(i)}_m = V^{-1} S (S^T V^{-1} S)^{-1} D^T u_t = WS(S^T WS)^{-1} D^T u_t$$  (23.11.17)

and at the sample values $t = m$, $-M \leq m \leq M$, or, at $u_t = u_m$, we obtain the differentiation matrix having the $b^{(i)}_m$ as columns, $B^{(i)} = [\ldots b^{(i)}_m \ldots]$:

$$B^{(i)} = WS(S^T WS)^{-1} D^T S^T = V^{-1} S (S^T V^{-1} S)^{-1} D^T S^T$$  (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:

$$W = U^T U$$
$$US = Q_0^T Q_0, \quad \text{with}\quad Q_0^T Q_0 = I, \quad R_0 = (d+1) \times (d+1) \text{ upper-triangular}$$
$$B = U^T Q_0 Q_0^T U^{-T}$$
$$B^{(i)} = U^T Q_0 Q_0^T U^{-T} D^T u_t$$

The MATLAB functions $\text{lpsm}$, $\text{lpdf}$, $\text{linterp}$ 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:

$$[B,G] = \text{lpsm}(N,d,W);$$
$$B = \text{lpdf}(N,d,1,W);$$
$$b = \text{linterp}(N,d,t,1,W);$$

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:

$$a^T W b = \sum_{m=-M}^{M} w_m a(m) b(m)$$  (23.11.20)

Such polynomials may be constructed from the monomials $s_i(m) = m^i, i = 0, 1, \ldots, d$ via Gram-Schmidt orthogonalization applied with respect to the above inner product. The result of orthogonalizing the basis $S = [s_0, s_1, \ldots, s_d]$ is $Q = [q_0, q_1, \ldots, q_d]$ 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:

$$q_i^T W q_j = \delta_{ij} D_i, \quad i,j = 0,1,\ldots,d \Rightarrow Q^T W Q = D$$  (23.11.21)

where $D = \text{diag}([D_0, D_1, \ldots, D_d])$ 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 $D^{1/2} = \text{diag}([D_0^{1/2}, D_1^{1/2}, \ldots, D_d^{1/2}])$ to be its square root. Then we construct $Q, R$ in terms of the factors $U$, $Q_0$, $R_0$:

$$Q = U^{-1} Q_0 D^{1/2}, \quad R = D^{-1/2} R_0$$  (23.11.22)

where $R$ is still upper-triangular. Then, we have $Q^T W Q = D$ 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):

\[
S = QR, \quad \text{with} \quad Q^T W Q = D, \quad R = (d+1) \times (d+1) \text{ upper-triangular}
\]

\[
B = W Q D^{-1} Q^T
\]

\[
B^{(i)} = W Q D^{-1} (R^{-T} D^T R^T) Q^T
\]

(23.11.23)

\[
b^{(i)}_t = W Q D^{-1} R^{-T} D^T u_t
\]

Since \( Q = [q_0, q_1, \ldots, q_d] \), the matrix \( B \) can be expressed as,

\[
B = W Q D^{-1} Q^T = W \sum_{r=0}^d D_r^{-1} q_r q_r^T
\]

(23.11.24)

and for diagonal \( W \), we have component-wise:

\[
b_m(k) = B_{km} = w_k \sum_{r=0}^d \frac{q_r(k) q_r (m)}{D_r} \quad -M \leq m, k \leq M
\]

(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 \( b^{(i)}_t \) can also be expressed in a similar summation form:

\[
b^{(i)}_t (k) = w_k \sum_{r=0}^d \frac{q_r(k) q_r (t)}{D_r}
\]

(23.11.26)

where \( q_r^{(i)} (t) \) is the \( i \)th 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 \( m \)th rows of the matrices \( S \) and \( Q \) are the \( (d+1) \)-dimensional vectors:

\[
u^T_m = [s_0 (m), s_1 (m), \ldots, s_d (m)] = [1, m, \ldots, m^d]
\]

\[
p^T_m = [q_0 (m), q_1 (m), \ldots, q_d (m)]
\]

(23.11.27)

and since \( S = QR \), they are related by \( u^T_m = p^T_m R \). Replacing \( m \) by \( t \) preserves this relationship, so that \( u^T_t = p^T_t R \), or,

\[
u_t = R^T p_t , \quad \text{where} \quad p_t = [q_0 (t), q_1 (t), \ldots, q_d (t)]^T
\]

(23.11.28)

Differentiating \( i \) times, we obtain

\[
D^i u_t = u^{(i)}_t = R^T p^{(i)}_t \Rightarrow p^{(i)}_t = R^{-T} D^i u_t
\]

(23.11.29)

and therefore \( b^{(i)}_t \) from Eq. (23.11.23) can be written in the following form, which implies Eq. (23.11.26):

\[
b^{(i)}_t = W Q D^{-1} p^{(i)}_t
\]

(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 \( Q^T W Q = D \), which implies

\[
Q^{-1} = D^{-1} Q^T W
\]

(23.11.31)

from which we obtain the completeness property:

\[
Q D^{-1} Q^T = W^{-1}
\]

(23.11.32)

which shows that \( B = I \). Similarly, using \( W Q D^{-1} = Q^{-T} \), we obtain from (23.11.23) the usual Lagrange interpolation polynomials:

\[
b_t = W Q D^{-1} R^{-T} u_t = Q^{-T} R^{-T} u_t = S^{-T} u_t
\]

(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 \( y \), such as a row in a scanned image, we define the \( N \)-dimensional vector of moments with respect to the polynomials \( Q_s \),

\[
\mu_t = D^{-1} Q^T W y \Rightarrow \mu_t = \frac{1}{D_t} \sum_{n=-M}^M q_r (n) w_n y_n , \quad r = 0, 1, \ldots, N - 1
\]

(23.11.34)

Because of Eq. (23.11.31), we have \( \mu = Q^{-1} y \), which allows the reconstruction of \( y \) from its moments:

\[
y = Q \mu \Rightarrow y_n = \sum_{r=0}^{N-1} q_r (n) \mu_r , \quad -M \leq n \leq M
\]

(23.11.35)

### 23.12 Henderson Filters

All the results of the previous section find a concrete realization in the minimum-\( R \) filters that we discuss here. Consider the order-\( s \) backward difference filter and its impulse response defined by:

\[
D_s (z) = (1 - z^{-1})^s \Leftrightarrow d_s(k) = (-1)^k \binom{s}{k}, \quad k = 0, 1, \ldots, s
\]

(23.12.1)

This follows from the binomial expansion:

\[
(1 - z^{-1})^s = \sum_{k=0}^s (-1)^k \binom{s}{k} z^{-k}
\]

(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:

\[
g_n = \nabla^s f_n = \sum_{k=0}^s d_s(k) f_{n-k} = \sum_{k=0}^s (-1)^k \binom{s}{k} f_{n-k}
\]

(23.12.3)
If the signal \( f_n \) is restricted over the range \(-M \leq n \leq M\), then because \( 0 \leq k \leq s \) and \(-M \leq n-k \leq M\), the above equation can be written in the more precise form:

\[
g_n = \nabla^s f_n = \sum_{k=\max(0,n-M)}^{\min(n,n+M)} (-1)^k \binom{s}{k} f_{n-k}, \quad -M \leq n \leq M + s
\]  

(23.12.4)

Eq. (23.12.4) gives the full convolutional output \( g_n = (d_s \ast f)_n \), while (23.12.3) is the corresponding steady-state output, obtained by restricting the output index \( n \) to the range \(-M + s \leq n \leq M\). Defining the \((N+s)\)-dimensional output vector \( g \) and \( N\)-dimensional input vector \( f \), where \( N = 2M + 1 \),

\[
g = [g_{-M}, \ldots, g_M, \ldots, g_{M+s}]^T, \quad f = [f_{-M}, \ldots, f_M]^T,
\]

we may write the full filtering equation (23.12.4) in matrix form:

\[
g = D_s f
\]  

(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:

\[
(D_s)_{nm} = d_s(n-m), \quad -M \leq n \leq M + s, \quad -M \leq m \leq M
\]  

(23.12.6)

and subject to the restriction that only the values \( 0 \leq n - m \leq s \) will result in a non-zero matrix element. The MATLAB functions \text{binom} \text{ and } \text{diffmat} allow the calculation of the binomial coefficients \( d_s(k) \) and the convolution matrix \( D_s \):

\[
\begin{align*}
\text{d} &= \text{binom}(s,k); \\
\text{D} &= \text{diffmat}(s,N); \quad \text{binomial coefficients } d_s(k) \quad \text{and } (N+s)\times N \text{ difference convolution matrix}
\end{align*}
\]

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 \text{diffmat} is simply a call to \text{convmat}:

\[
\text{D} = \text{convmat} (\text{binom}(s,N));
\]

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 \( S^T b = u \), for a given constraint vector \( u \), where \( b \) denotes the impulse response of \( B(z) \) assumed to be double-sided, that is, \( b_n, -M \leq n \leq M \).

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 \( f = D_s b \), the corresponding cascaded NRR will be:

\[
R_s = f^T f = \sum_{n=-M}^{M+s} f_n^2 = \sum_{n=-M}^{M+s} (\nabla^s b_n)^2 = \frac{1}{2\pi} \int_{-\pi}^{\pi} |D_s(\omega)B(\omega)|^2 d\omega
\]

Since \( f = b^T (D_s^T D_s) b \), we can state the design condition of the minimum-\( R_s \) filters as

\[
R_s = \sum_{n=-M}^{M+s} (\nabla^s b_n)^2 = b^T (D_s^T D_s) b = \min, \quad \text{subject to } S^T b = u
\]  

(23.12.7)

This has exactly the same form as Eq. (23.11.8) with \( V = D_s^T D_s \). The minimization of \( R_s \) 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 \( R_s \) by a normalization factor:

\[
R_s = \frac{b^T (D_s^T D_s) b}{d_s^T d_s} = \frac{R_s}{d_s^T d_s} = \min
\]  

(23.12.8)

where \( R_s \) is referred to as the “smoothing coefficient”, \( d_s \) is the impulse response vector of the filter \( D_s(z) \), and \( d_s^T d_s \) is the NRR of \( D_s(z) \). Using a binomial identity (a special case of (23.12.13) for \( k = 0 \)), we have,

\[
\sum_{m=0}^{s} d_s^T (m) = \sum_{m=0}^{s} \binom{s}{m} 2^s = \binom{2s}{s}
\]  

(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 \( |D_s(z) B(\omega)|^2 S_{yy}(\omega) \), respectively. Therefore, the mean-square value of \( \nabla^s \hat{x}_n \) will be:

\[
E[|\nabla^s \hat{x}_n|^2] = \frac{1}{2\pi} \int_{-\pi}^{\pi} |D_s(z) B(\omega)|^2 S_{yy}(\omega) \; d\omega
\]  

(23.12.10)
If \( y_n \) is white noise of variance \( \sigma^2 \), or if we assume that \( S_{yy}(\omega) \) is bounded from above by a constant, such as \( S_{yy}(\omega) \leq \sigma^2 \), then we obtain:

\[
E\left[ (\nabla^2 \hat{x}_n)^2 \right] \leq \frac{1}{2\pi} \int_{-\pi}^{\pi} |D_1(\omega) B(\omega)|^2 \sigma^2 \, d\omega = \mathcal{R}_s \sigma^2 \tag{23.12.11}
\]

For white noise, \( S_{yy}(\omega) = \sigma^2 \), Eq. (23.12.11) becomes an equality. Thus, minimizing \( \mathcal{R}_s \) will minimize \( E[ (\nabla^2 \hat{x}_n)^2 ] \) 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 1879. Others, like Hardy and Henderson have considered the minimum-\( R_s \) 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 \( V = D_D^T D_3 \) 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 \( V = D_D^T D_3 \) are \( (D_D^T D_3)_{nm} = V_{nm} = V_{n-m} \), where \( V_k \) is the autocorrelation function of the power spectrum \( V(\omega) = |D_3(\omega)|^2 \). Working in the \( z \)-domain, we have the spectral density:

\[
V(z) = D_3(z) D_3(z^{-1}) = (1-z^{-1})^{s} (1-z)^{s} = (-1)^{s} z^{s} (1-z^{-1})^{2s} \tag{23.12.12}
\]

which shows that \( V(z) \) effectively acts as the \( (2s) \)-difference operation \( \nabla^{2s} \). Taking inverse \( z \)-transforms of both sides of (23.12.12), we obtain:

\[
V_k = \sum_{m=\max(0,k)}^{\min(s,k+s)} d_s(m) d_s(m-k) = (-1)^{s} d_{2s}(k+s), \quad -s \leq k \leq s \tag{23.12.13}
\]

or, explicitly in terms of the definition of \( d_s \):

\[
V_k = (-1)^{s} \sum_{m=\max(0,k)}^{\min(s,k+s)} \binom{s}{m} \binom{s}{m-k} = (-1)^{k} \binom{2s}{s+k}, \quad -s \leq k \leq s \tag{23.12.14}
\]

or,

\[
V_k = (-1)^{k} \frac{(2s)!}{(s+k)! (s-k)!}, \quad -s \leq k \leq s \tag{23.12.15}
\]

The \( V \) matrix is a banded Toeplitz matrix with bandwidth \( \pm s \), whose central row or central column consist of the numbers \( V_k, -s \leq k \leq s \), with \( V_0 \) positioned at the center of the matrix. As an example,

\[
V = D_D^T D_3 = \begin{bmatrix}
20 & -15 & 6 & -1 & 0 & 0 & 0 \\
-15 & 20 & -15 & 6 & -1 & 0 & 0 \\
6 & -15 & 20 & -15 & 6 & -1 & 0 \\
-1 & 6 & -15 & 20 & -15 & 6 & -1 \\
0 & -1 & 6 & -15 & 20 & -15 & 6 \\
0 & 0 & -1 & 6 & -15 & 20 & -15 \\
0 & 0 & 0 & -1 & 6 & -15 & 20
\end{bmatrix}
\]

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 \( \nabla^{2s} \), let \( f \) be an \( N \) dimensional vector indexed for \(-M \leq m \leq M\), and form the output \( N \)-dimensional vector:

\[
g = Vf \Leftrightarrow g_n = \sum_{m=-M}^{M} V_{n-m} f_m, \quad -M \leq n \leq M \tag{23.12.16}
\]

where \( n - m \) is further restricted such that \(-s \leq n - m \leq s\). Next, consider an extended version of \( f \) obtained by padding \( s \) zeros in front and \( s \) zeros at the end, so that the extended vector \( f^\text{ext} \) will be indexed over, \(-M - s \leq m \leq M + s\):

\[
f^\text{ext} = (0, \ldots, 0, f_{-M}, \ldots, f_0, \ldots, f_M, 0, \ldots, 0)^T
\]

Then, the summation in Eq. (23.12.16) can be extended as,

\[
g_n = \sum_{m=-M}^{M+s} V_{n-m} f^\text{ext}_m, \quad -M \leq n \leq M \tag{23.12.17}
\]

But because of the restriction \(-s \leq n - m \leq s\), the above summation can be restricted to be over \(-n \leq m \leq n + s\), which is a subrange of the range \(-M + s \leq m \leq M + s\) because we assumed \(-M \leq n \leq M\). Thus, we may write:

\[
g_n = \sum_{m=-n-s}^{n+s} V_{n-m} f^\text{ext}_m, \quad -M \leq n \leq M \tag{23.12.18}
\]

or, changing to \( k = n - m \),

\[
g_n = \sum_{k=-s}^{s} V_k f^\text{ext}_{n-k} = (-1)^{s} \sum_{k=-s}^{s} d_{2s}(k+s) f^\text{ext}_{n-k} = (-1)^{s} \sum_{i=0}^{2s} d_{2s}(i) f^\text{ext}_{n+s-i} \tag{23.12.18}
\]

but that is precisely the \( \nabla^{2s} \) operator:

\[
g_n = (-1)^{s} \nabla^{2s} f^\text{ext}_{n+s}, \quad -M \leq n \leq M \tag{23.12.19}
\]
If \( f_m^{\text{ext}} \) 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:

\[
f_m = w_m m^i, \quad -M \leq m \leq M
\]  

where the weighting function \(w_m\) is itself a polynomial of degree \(2s\), then in order for the extended vector \(f_m^{\text{ext}}\) to vanish over \(M < |m| \leq M + s\), the function \(w_m\) must have zeros at these points, that is,

\[
w_m = 0, \quad \text{for } m = \pm (M + 1), \pm (M + 2), \ldots, \pm (M + s)
\]

This condition fixes \(w_m\) uniquely, up to a normalization constant:

\[
w_m = \prod_{i=1}^{s} [(M + 1)^{2} - m^2] \quad \text{(Henderson weights)}
\]  

These are called \textit{Henderson weights}. Because the extended signal \(f_m^{\text{ext}}\) 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 \(W = \text{diag}([w_{-M}, \ldots, w_0, \ldots, w_M])\), we can write (23.12.20) vectorially in terms of the monomial basis vector \(s_i\) as \(f = WS_i\). We showed that the matrix operation \(g = VF = VWs_i\) results into a polynomial of degree \(i\), which therefore can be expanded as a linear combination of the monomials \(s_0, s_1, \ldots, s_i\) up to order \(i\), that is,

\[
VWs_i = \sum_{j=0}^{i} s_j C_{ji}
\]  

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 \(S = [s_0, s_1, \ldots, s_d]\) up to order \(d\), it follows that

\[
VWS = SC, \quad C = (d+1) \times (d+1) \text{ upper-triangular}
\]  

But, this is exactly the condition (23.11.14). Thus, we have shown the equivalence of the NRR minimization problem (23.12.7) with \(V = D_1^T D_4\) 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 \(\text{lprs}\) implements the design. It constructs the \(W\) matrix from the Henderson weights and passes it into the function \(\text{lpsm}\):

\[
[B, G] = \text{lprs}(N,d,s); \quad \% \text{local polynomial minimum } R_3 \text{ filters}
\]

The Henderson weights \(w_m, -M \leq m \leq M\) are calculated by the function \(\text{hend}\):

\[
w = \text{hend}(N,s); \quad \% \text{Henderson weights}
\]

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 \(k\)th component of the \(m\)th filter will be:

\[
b_{km}(k) = w_k \sum_{l=0}^{d} k^l \phi_{lj} = w_k u_k^T \Phi u_m
\]  

where \(u_k = [1, k, k^2, \ldots, k^d]^T\) and \(\Phi\) is the inverse of the Hankel matrix \(F = S^TWS\) whose matrix elements are the weighted inner products:

\[
F_{ij} = (S^TWS)_{ij} = s_i^T W s_j = \sum_{m=-M}^{M} w_m m^{i+j} = F_{i+j}, \quad i, j = 0, 1, \ldots, d
\]

Except for the factor \(w_k\) and the different values of \(\phi_{lj}\) the expressions are similar to those of the LPSM filters of Sec. 23.3. The matrix \(\Phi\) has a similar checkerboard structure. For example, we have for the commonly used case \(d = 3\) and \(s = 3\):

\[
b_m(k) = w_k \begin{bmatrix} 1 & m & m^2 \end{bmatrix}
\]

where \(w_k = [(M + 1)^{2} - k^2] [(M + 2)^{2} - k^2] [(M + 3)^{2} - k^2]\)

and

\[
F = \begin{bmatrix} F_0 & 0 & 0 & 0 \\ 0 & F_2 & 0 & 0 \\ 0 & 0 & F_4 & 0 \\ 0 & 0 & 0 & F_6 \end{bmatrix} \Rightarrow \Phi = F^{-1} = \begin{bmatrix} 1 & m & m^2 \end{bmatrix}
\]

where we obtain from the checkerboard submatrices:

\[
\begin{bmatrix} \phi_{00} & \phi_{02} \\ \phi_{20} & \phi_{22} \end{bmatrix} = \begin{bmatrix} F_0 & F_2 & 0 \\ F_2 & F_4 & 0 \\ 0 & 0 & F_6 \end{bmatrix}^{-1}, \quad \begin{bmatrix} \phi_{11} & \phi_{13} \\ \phi_{31} & \phi_{33} \end{bmatrix} = \begin{bmatrix} F_2 & F_4 & 0 \\ F_4 & F_6 & 0 \\ 0 & 0 & F_6 \end{bmatrix}^{-1}
\]

The corresponding \(F\)-factors for \(s = 3\) are:

\[
F_0 = \frac{2}{35} (2M + 7) (2M + 5) (2M + 3) (M + 1) (M + 2) (M + 1) \\
F_2 = \frac{1}{9} M (M + 4) F_0 \\
F_4 = \frac{1}{11} (3M^2 + 12M - 4) F_2 \\
F_6 = \frac{1}{143} (15M^4 + 120M^3 + 180M^2 - 240M + 68) F_2
\]
which give rise to the matrix elements of \( \Phi \):

\[
\begin{align*}
\Phi_{00} &= 315(3M^2 + 12M - 4)/D_1 \\
\Phi_{02} &= -3465/D_1 \\
\Phi_{22} &= 31185/D_1 \\
\Phi_{11} &= 1155(15M^2 + 120M^3 + 180M^2 - 240M + 68)/D_2 \\
\Phi_{13} &= -15015(3M^2 + 12M - 4)/D_2 \\
\Phi_{33} &= 165165/D_2
\end{align*}
\]

with the denominator factors:

\[
\begin{align*}
D_1 &= 8(2M + 9)(2M + 7)(2M + 5)(2M + 3)(M + 2)(M + 1)(4M^2 - 1) \\
D_2 &= 8M(M - 1)(M + 4)(M + 5)D_1
\end{align*}
\]

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 \( w_k = [(M+1)^2-k^2][(M+2)^2-k^2][(M+3)^2-k^2] \):

\[
b_0(k) = \frac{315(3M^2 + 12M - 4 - 11k^2)w_k}{8(2M+9)(2M+7)(2M+5)(2M+3)(M+2)(M+1)(4M^2-1)} \tag{23.12.29}
\]

The corresponding predictive/interpolating differentiation filters \( b^{(i)}_t(k) \) are given by a similar expression:

\[
b^{(i)}_t(k) = w_ku^i_1 \Phi D^iu_1 \tag{23.12.30}
\]

or, explicitly, for the \( d = s = 3 \) case and differentiation order \( i = 0, 1, 2, 3 \):

\[
b^{(i)}_t(k) = w_k \begin{bmatrix} 1 & k & k^2 & k^3 \end{bmatrix} \begin{bmatrix} \Phi_{00} & \Phi_{02} & 0 & 0 \\
0 & \Phi_{11} & \Phi_{13} & 0 \\
\Phi_{20} & \Phi_{22} & 0 & 0 \\
0 & \Phi_{31} & \Phi_{33} & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} t & t^2 & t^3 \end{bmatrix} \tag{23.12.31}
\]

**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-\( R \), 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 \( s = 3 \).

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. \(^1\) The \( s = 2 \) case represents the smooth signal in piece-wise linear form. The \( L_1 \) case is discussed further in Sec. 26.7.

\(^1\)http://cvxr.com/cvx/
The following MATLAB code illustrates the generation of the four graphs:

```matlab
Y = loadfile('exuseu.dat');
y = Y(:,4); t = taxi(y,12,1999);
N=19; d=3; x1 = lpfilt(lpsm(N,d),y);
s=3; x2 = lpfilt(lprs(N,d,s),y);
M=8; r=2; x3 = svdenh(y,M,r);
la=100; s=3; x4 = whsm(y,la,s);
cvx_begin
variable x5(N)
    minimize( sum_square(y-x5) + la * norm(D*x5,1) )
cvx_end
figure; plot(t,y,’.’, t,x3,’-’);
figure; plot(t,y,’.’, t,x4,’-’);
figure; plot(t,y,’.’, t,x5,’-’);

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. (23.12.23) 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),

$$VWQR = QRC \Rightarrow VWQ = QA, \quad A = RCR^{-1}$$

Multiplying both sides by $Q^TW$ and using the property $Q^TWQ = D$, we obtain:

$$Q^TWVWQ = Q^TWQA = DA$$

(23.12.33)

Because $R$ and $C$ are both upper-triangular, so will be $A$ and $DA$. But the left-hand side of (23.12.33) is a symmetric matrix, and so must be the right-hand side $DA$. This requires that $DA$ have the form of an eigenvector:

$$VWQ_r = \lambda_r Q_r, \quad r = 0, 1, \ldots, d$$

(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:

$$VW = QAQ^{-1} = QAD^{-1}Q^T \Rightarrow V = Q(AD^{-1})Q^T$$

We also find for the inverse of $V$:

$$V^{-1} = WQAQ^{-1}D^{-1}Q^T$$

There exist [677–679] similar and efficient ways to calculate $V^{-1} = (D^TD)^{-1}$. The eigenvalues $\lambda_r$ can be shown to be [707]:

$$\lambda_r = \frac{(2s+r)!}{r!} = \prod_{i=1}^{2s} (r+i), \quad r = 0, 1, \ldots, d$$

(23.12.35)

As we see in the next section, the $r$th column $Q_r(n)$ of $Q$ is a Hahn polynomial of degree $r$ in $n$, and hence $WQ_r$, or component-wise, $w_nQ_r(n)$, will be a polynomial of degree $2s+r$. Moreover, because of the zeros of $w_n$, the polynomial $f_n = w_nQ_r(n)$ can be extended to be over the range $-M - s \leq n \leq M + s$. Using the same reasoning as in Eq. (23.12.19), it follows that (23.12.34) can be written as

$$(-1)^s \sum_{n=M+s}^{M-s} f_n = \lambda_r q_r(n), \quad -M \leq n \leq M$$

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

$$f_n^{\text{ext}} = w_{n+s}q_r(n+s) = (-1)^s [(n + s)^r + \cdots ] [a_r(n + s)^{r+s} + \cdots ], \quad a_r(n) = a_r n^r + \cdots$$

where $a_r$ is the highest coefficient of $q_r(n)$ and the dots indicate lower powers of $n$. Dropping the $a_r$ constant, the eigenvector condition then becomes:

$$\nabla \sum_{n=M-s}^{M+s} f_n = \lambda_r n^r + \cdots$$

(23.12.36)

Each operation of $\nabla$ on $n$ lowers the power by one, that is, $\nabla (n^r) = in^{r-1} + \cdots$, $\nabla^2(n^r) = (i(i-1)n^{r-2} + \cdots$, $\nabla^3(n^r) = (i(i-1)(i-2)n^{r-3} + \cdots$, etc. Thus, we have:

$$\nabla^s [n^{2s+r} + \cdots ] = (2s + r)(2s + r - 1)(2s + r - 2) \cdots (r+1) n^r + \cdots$$

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-$R$ problem. For a review of the Hahn polynomials, see Karlin and McGregor [697].

The Hahn polynomials $Q_r(x)$ of a discrete variable $x = 0, 1, 2, \ldots, N - 1$ and orders $r \leq N - 1$ satisfy a weighted orthogonality property of the form:

$$\sum_{x=0}^{N-1} w(x)Q_r(x)Q_m(x) = D_r \delta_{rm}, \quad r, m = 0, 1, \ldots, N - 1$$

where $w(x)$ is a positive function.
23.13. HAHN ORTHOGONAL POLYNOMIALS

where the weighting function \( w(x) \) depends on two parameters \( \alpha, \beta \) and is defined up to a normalization constant as follows:

\[
w(x) = \frac{(\alpha + x)!}{x!} \cdot \frac{(\beta + N - 1 - x)!}{(N - 1 - x)!}, \quad x = 0, 1, \ldots, N - 1 \tag{23.13.1}
\]

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 \(-M \leq n \leq M \). Then, the weighting function becomes,

\[
w(n) = \frac{(\alpha + n + M + n)!}{(M + n)!} \cdot \frac{(\beta + M - n)!}{(M - n)!}, \quad -M \leq n \leq M \tag{23.13.2}
\]

Defining \( q_r(n) = Q_r(x) \big|_{x=n+M} \), the orthogonality property now reads:

\[
\sum_{n=-M}^{M} w(n) q_r(n) q_m(n) = D_r \delta_{rm}, \quad r, m = 0, 1, \ldots, N - 1 \tag{23.13.3}
\]

The minimum-\( R \) 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):

\[
w(n) = \frac{(s + M + n)!}{(M + n)!} \cdot \frac{(s + M - n)!}{(M - n)!} = \prod_{i=1}^{s} (M + n + i) \cdot \prod_{i=1}^{s} (M - n + i), \quad \text{or,}
\]

\[
w(n) = \prod_{i=1}^{s} [(M + i)^2 - n^2], \quad -M \leq n \leq M \tag{23.13.4}
\]

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 \( \genfrac{[}{]}{0pt}{}{r}{x} \) for \( \alpha = \beta = s \), they take the following explicit form:

\[
q_r(n) = Q_r(x) = \sum_{k=0}^{r} a_{rk} x^{[k]} \big|_{x=n+M} = \sum_{k=0}^{r} a_{rk} (n + M)^{[k]}, \quad -M \leq n \leq M \tag{23.13.5}
\]

where \( x^{[k]} \) denotes the falling-factorial power,

\[
x^{[k]} = x(x - 1) \cdots (x - k + 1) = \frac{x^k}{x^k} = \frac{\Gamma(x + 1)}{\Gamma(x + k + 1)} \tag{23.13.6}
\]

The polynomial coefficients are:

\[
a_{rk} = (-1)^k \prod_{m=0}^{r} \left[ \frac{(r - m + 1)(2s + r + m)}{(N - m)(s + m)m} \right], \quad k = 0, 1, \ldots, r \tag{23.13.7}
\]

where \( a_{r0} = 1 \). Expanding the product we have:

\[
a_{rk} = \frac{(-1)^k r(r - 1) \cdots (r - k + 1) \cdot (2s + r + 1) \cdot (2s + r + 2) \cdots (2s + r + k)}{(N - 1)(N - 2) \cdots (N - k)(s + 1)(s + 2) \cdots (s + k)k!} \tag{23.13.8}
\]

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 \( D_r \):

\[
D_r = \frac{(s!)^2}{(2M)!} \cdot \frac{r! (2M - r)!}{(2s + r + 1) \cdots (2s + r + N)!}, \quad 2s + 2r + 1 \tag{23.13.10}
\]

For minimum-\( R \) filter design with polynomial order \( d \leq N - 1 \), only polynomials up to order \( d \) are needed, that is, \( q_r(n), r = 0, 1, \ldots, d \). Arranging these as the columns of the \( N \times (d + 1) \) matrix \( Q = [q_0, q_1, \ldots, q_d] \), the orthogonality property can be expressed as \( Q^T W Q = D \), where \( D = \text{diag}([D_0, D_1, \ldots, D_d]) \).

The relationship to the monomial basis \( s = [s_0, s_1, \ldots, s_d] \) 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 \( i = 0, 1, \ldots, r \). In fact, \( R \) can be easily constructed from the Hahn coefficients \( a_{rk} \) and the Stirling numbers.

Thus, the construction of the minimum-\( R \) filters outlined in Eq. (23.11.23) is explicitly realized by the Hahn polynomial basis matrix \( Q \):

\[
B = W Q D^{-1} Q^T \tag{23.13.11}
\]

or, component-wise,

\[
b_m(n) = B_{nm} = w(n) \sum_{r=0}^{d} \frac{q_r(n) q_r(m)}{D_r}, \quad -M \leq n, m \leq M \tag{23.13.12}
\]

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:

\[
\hat{y}_m = \sum_{i=0}^{d} c_i m^i = \sum_{r=0}^{d} a_r q_r(m) \Rightarrow \hat{y} = Sc = Qa
\]

Then, minimize the weighted performance index with respect to \( a \):

\[
J = (y - Qa)^T W (y - Qa) = \min
\]
23.13. HAHN ORTHOGONAL POLYNOMIALS

Using the condition $Q^TWQ = D$, the solution leads to the same $B$:

$$a = D^{-1}Q^TWy = D^{-1}Q^TWy = B^Ty$$ (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:

$$x^{[k]} = \sum_{i=0}^{k} S_1(k,i)x^i \quad \leftrightarrow \quad x^k = \sum_{i=0}^{k} S_2(k,i)x^{[i]}$$ (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 $k = 0, 1, 2, 3$:

$$S_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad S_2 = S_1^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The MATLAB function `stirling` generates these matrices up to a desired order:

$$S = \text{stirling}(d, \text{kind}); \quad \%\text{ Stirling numbers up to order } d \text{ of kind } 1, 2$$

A polynomial can be expressed in falling factorial powers or in ordinary powers. The corresponding coefficient vectors are related by the Stirling numbers:

$$P(x) = \sum_{k=0}^{d} a_kx^{[k]} = \sum_{i=0}^{d} c_i x^i \quad \Rightarrow \quad c = S_1^T a, \quad a = S_2^T c$$

The function `polyval` allows the evaluation of a polynomial in falling (or rising) factorial powers or in ordinary powers at any vector of $x$ values:

$$P = \text{polyval}(a, z, \text{type}); \quad \%\text{ polynomial evaluation in factorial powers}$$

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 $Q = \{q_0, q_1, \ldots, q_d\}$:

$$[a, c] = \text{hahncoeff}(N, r, s); \quad \%\text{ Hahn polynomial coefficients } a_k$$

$$Q = \text{hahnpol}(N, r, s, x); \quad \%\text{ evaluate Hahn polynomial } Q_r(x)$$

$$[Q, 0, L] = \text{hahnbasis}(N, d, s); \quad \%\text{ Hahn basis } Q = \{q_0, q_1, \ldots, q_d\}$$

Like all orthogonal polynomials, the Hahn polynomials satisfy a three-term recurrence relation of the form:

$$nq_r(n) = \alpha_r q_{r+1}(n) + \beta_r q_r(n) + \gamma_r q_{r-1}(n)$$ (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

$$\langle a, b \rangle = \sum_{n=-M}^{M} w(n) a(n) b(n)$$ (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

$$nq_r(q_i) = \sum_{j=0}^{r+1} c_j D_i \delta_{ij} = D_i c_i \Rightarrow c_i = \frac{(nq_r, q_i)}{D_i}$$ (23.13.17)

This implies that $c_i = 0$ for $i \geq r - 2$, 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 $nq_r(n)$ has degree $(i + 1)$. Therefore, as long as $i + 1 < r$, or, $i \leq r - 2$, this inner product will be zero. It follows from (23.13.17) that:

$$\alpha_r = \frac{(nq_r, q_{r+1})}{D_{r+1}}, \quad \beta_r = \frac{(nq_r, q_r)}{D_r}, \quad \gamma_r = \frac{(nq_r, q_{r-1})}{D_{r-1}}$$ (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 $\gamma_r$ can be related to $\alpha_{r-1}$ by noting that

$$\alpha_{r-1} = \frac{(nq_{r-1}, q_r)}{D_r} = \frac{(nq_{r-1}, q_{r-1})}{D_{r-1}} = \frac{(nq_r, q_r-1)}{D_{r-1}} \Rightarrow \frac{(nq_r, q_{r-1})}{D_{r-1}} = D_r \alpha_{r-1}, \quad \text{and hence,}$$

$$\gamma_r = \frac{(nq_r, q_{r-1})}{D_{r-1}} = \frac{D_r \alpha_{r-1}}{D_{r-1}} = \frac{D_r \alpha_{r-1}}{D_{r-1}}$$ (23.13.19)

Moreover, $\alpha_r$ 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:

$$\alpha_r = \frac{(nq_r, q_{r+1})}{(q_{r+1}, q_{r+1})} = \frac{(a_{rr} n^{r+1} + np_{r-1}(n), q_{r+1})}{(a_{r+1, r+1} n^{r+1} + p_r(n), q_{r+1})} = \frac{a_{rr}(n^{r+1}, q_{r+1})}{a_{r+1, r+1}(n^{r+1}, q_{r+1})} = \frac{a_{rr}}{a_{r+1, r+1}}$$
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,

$$\alpha_r = \frac{a_{rr}}{a_{r+1,r+1}} \quad (23.13.20)$$

Using Eqs. (23.13.7) and (23.13.10), the expressions for $\alpha_r$ and $y_r$ simplify into:

$$\alpha_r = \frac{(2M - r)(2s + r + 1)}{2(2s + 2r + 1)}, \quad y_r = \frac{r(2M + 2s + r + 1)}{2(2s + 2r + 1)} \quad (23.13.21)$$

These satisfy the constraint $\alpha_r + y_r = -M$, which follows from the recurrence relation and the conditions $q_r(-M) = a_0 = 1$ 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 $y_r = \alpha_r D_r/D_{r-1}$ and dividing by $D_r$, the recurrence relation reads:

$$b(n) = \frac{\alpha_r}{D_r} q_{r+1}(n) + \frac{\alpha_{r-1}}{D_{r-1}} q_r(n) \quad (23.13.22)$$

Multiplying by $q_r(m)$, interchanging the roles of $n, m$, and subtracting, we obtain:

$$\frac{m q_r(m) q_r(n)}{D_r} = \frac{\alpha_r}{D_r} q_{r+1}(n) q_r(m) + \frac{\alpha_{r-1}}{D_{r-1}} q_{r-1}(n) q_r(m)$$

$$\frac{(n-m) q_r(n) q_r(m)}{D_r} = \frac{\alpha_r}{D_r} [q_{r+1}(n) q_r(m) - q_r(n) q_{r+1}(m)] - \frac{\alpha_{r-1}}{D_{r-1}} [q_{r-1}(n) q_r(m) - q_r(n) q_{r-1}(m)]$$

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:

$$(n-m) \sum_{r=0}^{d} \frac{q_r(n) q_r(m)}{D_r} = \frac{\alpha_d}{D_d} [q_{d+1}(n) q_d(m) - q_d(n) q_{d+1}(m)], \quad \text{or,}$$

$$\sum_{r=0}^{d} \frac{q_r(n) q_r(m)}{D_r} = \frac{\alpha_d}{D_d} \frac{q_{d+1}(n) q_d(m) - q_d(n) q_{d+1}(m)}{n-m} \quad (23.13.23)$$

Using this identity into the filter equations (23.13.12), we find

$$b_m(n) = w(n) \frac{\alpha_d}{D_d} \frac{q_{d+1}(n) q_d(m) - q_d(n) q_{d+1}(m)}{n-m} \quad (23.13.24)$$

This is valid for $-M \leq n, m \leq M$ and for orders $0 \leq d \leq N-2$. 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 $n$. In particular, the central Henderson filters are:

$$b_0(n) = w(n) \frac{\alpha_d}{D_d} \frac{q_{d+1}(n) q_d(0) - q_d(n) q_{d+1}(0)}{n-m} \quad (23.13.25)$$

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:

$$b_0(n) = \frac{\alpha_d}{D_d} q_{2r}(0) \frac{q_{2r+1}(n)}{n} = -\frac{\alpha_{2r+1}}{D_{2r+1}} q_{2r+2}(0) \frac{q_{2r+3}(n)}{n} \quad (23.13.26)$$

the equality of the coefficients follows 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 $-M \leq n \leq M$:

$$q_0(n) = 1$$

$$q_1(n) = -\frac{n}{M}$$

$$q_2(n) = \frac{(2s+3)n^2 - M(M+s+1)}{M(2M-1)(s+1)}$$

$$q_3(n) = -\frac{(2s+5)n^3 - \left[3M^2 + (s+1)(3M-1)\right]n}{M(M-1)(2M-1)(s+1)}$$

$$q_4(n) = \frac{(2s+5)(2s+7)n^4 - (2s+5)(6M^2 + 6(s+1)M - 4s-5)n^2}{M(M-1)(2M-1)(2M-3)(s+1)(s+2)}$$

$$q_5(n) = -\frac{(2s+7)(2s+9)n^5 - 5(2s+7)(2M^2 + 2(s+1)M - 2s-3)n^3}{M(M-1)(M-2)(2M-1)(2M-3)(s+1)(s+2)}$$

They are normalized such that $q_r(-M) = 1$. Setting $s = 0$, we obtain the corresponding discrete Chebyshev/Gram polynomials:

$$q_0(n) = 1$$

$$q_1(n) = -\frac{n}{M}$$

$$q_2(n) = \frac{3n^2 - M(M+1)}{M(2M-1)}$$

$$q_3(n) = -\frac{5n^3 - (3M^2 + 3M - 1)n}{M(M-1)(2M-1)}$$

$$q_4(n) = \frac{35n^4 - 5(6M^2 + 6M - 5)n^2 + 3M(M^2 - 1)(M+2)}{2M(M-1)(2M-1)(2M-3)}$$

$$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)(2M-1)(2M-3)} \quad (23.13.28)
23.13. HAHN ORTHOGONAL POLYNOMIALS

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 \( d = 0,1 \):

\[
b_0(n) = \frac{(2s + 1)!}{(2s)!} \frac{(2M)!}{(2M + 2s + 1)!} w(n) \tag{23.13.29}
\]

where \( w(n) \) is given by Eq. (23.13.4). For \( d = 2,3 \), we have:

\[
b_0(n) = \frac{(M+s+1)(2s+3)!(2M)!(3M^2 + (s+1)(3M-1)-(2s+5)n^2)}{(2M-1)!(2s+3)!} w(n) \tag{23.13.30}
\]

This generalizes Henderson’s ideal formula (23.12.29) to arbitrary \( s \). For \( s = 1,2, \) it simplifies to:

\[
s = 1, \quad b_0(n) = \frac{15(3M^2 + 6M - 2 - 7n^2)w_1(n)}{2(M + 1)(2M + 3)(2M + 5)(4M^2 - 1)}
\]

\[
s = 2, \quad b_0(n) = \frac{105(M^2 + 3M - 1 - 3n^2)w_2(n)}{2(M + 1)(M + 2)(2M + 3)(2M + 5)(4M^2 - 1)}
\]

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:

\[
b_0(n) = \frac{(M+s+1)(M+s+2)(2s+5)!(2M)!}{2(2M-1)(2M-3)(s+2)!(2M+2s+5)!} w(n) \cdot
\]

\[
\cdot \left[ (2s+7)(2s+9)n^4 - 5(2s+7)(2M^2 + 2(s+1)M - 2s-3)n^2 +
\right.
\]

\[
+ 15M^4 + 30(s+1)M^3 + 5(3s^2+s-7)M^2 + (s+1)(s+2)(25M-6) \right] \tag{23.13.31}
\]

Eqs. (23.13.29)-(23.13.31), as well as the case \( d = 6,7 \), have been implemented into the MATLAB function \texttt{1ps2}, with usage:

\[
\texttt{b0 = 1ps2(N,d,s);} \quad \text{exact forms of the Henderson filters } b_0(n) \text{ for } 0 \leq d \leq 6
\]

The asymmetric interpolation filters \( b_1(n) \) can be obtained by replacing the discrete variable \( m \) by \( t \) in Eqs. (23.13.12) and (23.13.24):

\[
b_1(n) = w(n) \sum_{r=0}^{d} \frac{q_r(t)q_{d+1}(n)q_{d+1}(t) - q_d(n)q_{d+1}(t)}{D_d} n - t \tag{23.13.32}
\]

Some specific cases are as follows. For \( d = 0, \) we have:

\[
b_1(n) = \frac{(2s + 1)!}{(s)!^2} \frac{(2M)!}{(2M + 2s + 1)!} w(n) \tag{23.13.33}
\]

For \( d = 1, \)

\[
b_1(n) = \frac{4(2s + 1)! (2M-1)!}{(s)!^2 (2M+2s+2)!} w(n) \left[ M^2 + (s+1)M + (2s+3)nt \right] \tag{23.13.34}
\]

For \( d = 2, \)

\[
b_1(n) = \frac{4(2s + 1)! (2M-1)!}{(s)!^2 (2M+2s+2)!} w(n) \left[ M^2 + (s+1)M + (2s+3)nt \right] \tag{23.13.35}
\]

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 procedure that define the Hahn coefficients \( a_r \), the Hahn polynomials \( q_{r}(n) \) and their

\[
\text{factpow} := \text{proc}(x,k) \text{product}((x-m), m=0..k-1); \text{end proc};
\]

\[
a := \text{proc}(M,r,s,k) (-1)^k \text{ product}((r-m+1)*(2s+r+m)/(2*M+1-m)/(s+m)/m, m=1..k);
\]

\[\text{end proc;}
\]

\[
Q := \text{proc}(M,r,s,n) \text{ if } r=0 \text{ then } 1; \text{ else } \sum(a(M,r,s,k) \text{ factpow}(n,M,k), k=0..r); \text{ end if; \text{ end proc;}
\]

\[
\text{Dr} := \text{proc}(M,r,s) \text{ GAMMA}(s+1)^2 \text{ GAMMA}(r+1) \text{ GAMMA}(2*M+1-r) * \text{ product}(2*s+i, i=1..(2*M+1)) / \text{ GAMMA}(2*M+1)^2 / (2*s^2*r+1); \text{ end proc;}
\]

\[
\text{B} := \text{proc}(M,d,s,n,t) \text{ sum}(Q(M,r,s,n)/Dr(M,r,s)*Q(M,r,s,t), r=0..d); \text{ end proc;}
\]

where \text{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 \( n \geq 0 \) 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:

\[
B = WS(S^TWS)^{-1}S^T = WQD^{-1}Q^T, \quad Q^TQ = D \tag{23.13.36}
\]

23.14 Maximally-Flat Filters

Greville [668] has shown that in the limit \( s \to \infty \) the minimum-\( R_A \) 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 \( i = 0,1,\ldots,d \), 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 \(-M \leq n \leq M \) by [717]

\[
\hat{q}_r(n) = \sum_{k=0}^{r} \frac{(-1)^k r (r - 1) \cdots (r - k + 1) p^k}{(N - 1) (N - 2) \cdots (N - k) k!} (n + M)^{[k]} \tag{23.14.1}
\]

where \( N = 2M + 1 \) and \( r = 0, 1, \ldots, N - 1 \). They satisfy the orthogonality property,

\[
\sum_{n=-M}^{M} \hat{w}(n) \hat{q}_r(n) \hat{q}_m(n) = \hat{D}_r \delta_{rm} \tag{23.14.2}
\]

with the following binomial weighting function and norms, where \( q = 1 - p \):

\[
\hat{w}(n) = \left( \frac{2M}{M+n} \right)^p q^{M-n} = \frac{(2M)!}{2^M (M+n)! (M-n)!} p^{M+n} q^{M-n} \tag{23.14.3}
\]

\[
\hat{D}_r = \frac{r! (2M - r)!}{(2M)!} q^r \tag{23.14.4}
\]

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 \( \hat{q}_r(M) = 1 \), 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

\[
\tilde{a}_r = \lim_{s \to \infty} a_r = \frac{(-1)^k r (r - 1) \cdots (r - k + 1) \cdot 2^k}{(N - 1) (N - 2) \cdots (N - k) k!} \tag{23.14.9}
\]

and therefore, the Hahn polynomials will tend to

\[
\tilde{q}_r(n) = \sum_{k=0}^{r} \frac{(-1)^k r (r - 1) \cdots (r - k + 1) \cdot 2^k}{(N - 1) (N - 2) \cdots (N - k) k!} (n + M)^{[k]} \tag{23.14.10}
\]

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:

\[
\tilde{w}(n) = \lim_{s \to \infty} \left[ \frac{(2M)! \hat{w}(n)}{2^M s^{2M} (s!)^2} \right] = \lim_{s \to \infty} \left[ \frac{(2M)! (s + M + n) (s + M - n)!}{2^M s^{2M} (s!)^2 (M+n)! (M-n)!} \right]
\]

\[
\tilde{D}_r = \lim_{s \to \infty} \left[ \frac{(2M)! \hat{D}_r}{2^M s^{2M} (s!)^2} \right] = \lim_{s \to \infty} \left[ \frac{r! (2M - r)!}{(2M)!} \cdot \frac{(2s + r + 1) (2s + r + 2) \cdots (2s + r + N)}{2^M s^{2M} (2s + 2r + 1)} \right]
\]

They are easily seen to lead to Eqs. (23.14.4) with \( p = 1/2 \), that is,

\[
\tilde{w}(n) = \frac{1}{2^M} \left( \frac{2M}{M+n} \right)^p = \frac{(2M)!}{2^M (M+n)! (M-n)!} \tag{23.14.6}
\]

\[
\tilde{D}_r = \frac{r! (2M - r)!}{(2M)!} \tag{23.14.7}
\]

The first few of the Krawtchouk polynomials are:

\[
\tilde{q}_0(n) = 1, \quad \tilde{q}_1(n) = -\frac{n}{M}, \quad \tilde{q}_2(n) = \frac{2n^2 - M}{M (M-1)}, \quad \tilde{q}_3(n) = \frac{-2n^3 - (3M-1)n}{M (M-1) (2M-1)}, \quad \tilde{q}_4(n) = \frac{4n^4 - (12M-8)n^2 + 3M (M-1)}{M (M-1) (2M-1) (2M-3)} \tag{23.14.8}
\]

These polynomials satisfy the three-term recurrence relation:

\[
\tilde{q}_r(n) = \alpha_r \tilde{q}_{r+1}(n) + \gamma_r \tilde{q}_{r-1}(n), \quad \alpha_r = -\frac{2M - r}{2}, \quad \gamma_r = -\frac{r}{2} \tag{23.14.9}
\]

with the coefficients \( \alpha_r, \gamma_r \), 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:

\[
\hat{b}_r(n) = \hat{w}(n) \frac{\sum_{r=0}^{d} \hat{q}_r(n) \hat{q}_r(t) \hat{D}_r}{\sum_{r=0}^{d} \hat{D}_r} = \hat{w}(n) \frac{\hat{d}_r \hat{q}_{d-1}(n) \hat{d}_d(t) - \hat{q}_{d}(n) \hat{d}_{d-1}(t)}{n-t} \tag{23.14.10}
\]

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

\[
\hat{b}_r(n) = \hat{w}(n), \quad \hat{b}_1(n) = \hat{w}(n) \frac{2nt + M}{M} \tag{23.14.11}
\]

For \( d = 2 \), the smoothing and first-order differentiation filters are:

\[
\hat{b}_1(n) = \hat{w}(n) \frac{4n^2 + 2M (n^2 + r^2) + 2(2M-1)nt + M (3M-1)}{M (2M-1)} \tag{23.14.12}
\]

\[
\hat{b}_1(n) = \hat{w}(n) \frac{2(2M-1)n - 4Mnt + 8n^2 t}{M (2M-1)} \tag{23.14.13}
\]

and setting \( t = 0 \), the central filters simplify into:
23.14. Maximally-Flat Filters

For \( d = 3 \), we have:

\[
\hat{b}_t(n) = \hat{w}(n) \frac{3M - 1 - 2n^2}{2M - 1}, \quad \hat{b}_0(n) = \hat{w}(n) \frac{2n}{M} \tag{23.14.12}
\]

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 \( \hat{w}(n) \) into the functions \( \text{lpsm}, \text{lpdfilt} \), and \( \text{linterp} \):

\[
\begin{align*}
W &= \text{diag}(\text{hend}(N,\infty)); & \text{Krawtchouk weights} \\
B &= \text{lpsm}(N,d,W); & \text{smoothing filters} \\
B1 &= \text{lpdfilt}(N,d,1,W); & \text{1st derivative filters} \\
b &= \text{linterp}(N,d,1,1,W); & \text{interpolation filters} \\
\end{align*}
\]

The function \( \text{hend}(N,s) \), 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 \( \text{lpdfilt} \).

Alternatively, one can call \( \text{lprs} \) with \( s = \infty \):

\[
B = \text{lprs}(N,d,9); \quad b_9 = B(:,M+1); \tag{23.14.14}
\]

The first implies the flatness constraints at dc, the second, the flatness constraints at Nyquist, and the third expressions have the following equivalent expressions:

\[
B_0(\omega) = \sum_{i=0}^{r} \binom{M}{i} x^i (1-x)^{M-i} = 1 - \sum_{i=r+1}^{M} \binom{M}{i} x^i (1-x)^{M-i}
\]

\[
(1-x)^{M-r} \sum_{i=0}^{r} \binom{M-r+i-1}{i} x^i, \quad \text{where } x = \sin^2 \left( \frac{\omega}{2} \right) \tag{23.14.14}
\]

Near \( \omega = 0 \) and near \( \omega \approx \pi \), the second and third expressions have the following expansions that exhibit the desired flatness constraints [707]:

\[
\begin{align*}
\omega \approx 0 & \Rightarrow B_0(\omega) \approx 1 - (\text{const}) \omega^{2r+2} = 1 - (\text{const}) \omega^{d+1} \\
\omega \approx \pi & \Rightarrow B_0(\omega) \approx (\text{const}) (\omega - \pi)^{2M-2r} = (\text{const}) (\omega - \pi)^{2M-d+1} \tag{23.14.15}
\end{align*}
\]

The first implies the flatness constraints at dc, \( B_0^{(i)}(0) = \delta(i) \), for \( i = 0, 1, \ldots, d \), and the second, the flatness constraints at Nyquist, \( B_0^{(i)}(\pi) = 0 \), for \( i = 0, 1, \ldots, 2M-d \).

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:

\[
B_0(z) = \left[ \frac{(1+z^{-1})(1+z)}{4} \right]^{M-1} \frac{1}{4} [2(M+1)-(M-1)(z+z^{-1})]
\]

With \( z = e^{i\omega} \) we may write

\[
\begin{align*}
x &= \sin^2 \left( \frac{\omega}{2} \right) = \frac{1 - (1-z^{-1})(1-1)}{4} = 2 - z - z^{-1} \Rightarrow z + z^{-1} = 1 - x \\
1 - x &= \cos^2 \left( \frac{\omega}{2} \right) = \frac{1 + (1+z^{-1})(1+z)}{4} \\
\end{align*}
\]

Thus, we may express \( B_0(z) \) in terms of the variable \( x \):

\[
B_0(z) = (1-x)^{M-1} [1 + (M-1)x]
\]

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 \( N = 13 \), \( r = 2 \), \( d = 4 \), 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):

\[
\begin{align*}
N &= 13; \quad r=2; \quad d = 2*r+1; \quad M = \text{floor}(N/2); \\
B &= \text{lprs}(N,d,9); \quad b_9 = B(:,M+1); \quad \text{LPRS filter with } s = 9 \\
B &= \text{lprs}(N,d,inf); \quad binf = B(:,M+1); \quad \text{LPRS with Krawtchouk weights} \\
f &= \text{onespace}(0,1,1001); \quad w = \text{pi}^f; \quad x = \sin(w/2).*A2; \\
B9 &= \text{real}(\text{exp}(j*w*M).*freqz(b9,1,w)); \quad \text{frequency responses} \\
Binf &= \text{real}(\text{exp}(j*w*M).*freqz(binf,1,w)); \\
\end{align*}
\]

\[
\begin{align*}
\text{Bth} &= 0; \\
\text{for } i=0:r, \\
\text{Bth} &= \text{Bth} + \text{ncgchoosek}(M,i) \ast x \ast i \ast (1-x) \ast M(i); \quad \text{Eq. (23.14.14)} \\
\text{norm}(\text{Bth-Binf}) &= \text{compare Eq. (23.14.14) with output of LPSM} \\
\end{align*}
\]

The calls to \( \text{lprs} \) and \( \text{lpsm} \) return the full smoothing matrices \( B \) from which the central column \( b_0 \) is extracted.

The frequency response function \( \text{freqz} \) expects its filter argument to be causal. The factor \( e^{i\omega M} \) compensates for that, corresponding to a time-advance by \( M \) units.

Finally, we note that the Krawtchouk binomial weighting function \( \hat{w}(n) \) tends to a gaussian for large \( M \), which is a consequence of the De Moivre-Laplace theorem,

\[
\hat{w}(n) = \frac{(2M)!}{2^M (M+n)! (M-n)!} \approx \frac{1}{\sqrt{\pi M}} e^{-n^2/M}, \quad -M \leq n \leq M \tag{23.14.16}
\]

In fact, the two sides of (23.14.16) are virtually indistinguishable for \( M \gtrsim 10 \).
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 \( y_n, n = 0, 1, \ldots, L - 1 \), 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 \(-M \leq m \leq M\), 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 \( m \).

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

\[
J = \sum_{m=-M}^{M} p_m w_m \left( y_{n_0+m} - \sum_{i=0}^{d} c_im^i \right)^2 = \min
\]

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 \( y = [y_{n_0-M}, \ldots, y_{n_0}, \ldots, y_{n_0+M}]^T \), and denote by \( W, P \) the corresponding diagonal matrices of the weights \( w_m, p_m \). Then, (23.15.1) reads:

\[
J = (y - Sc)^TPW(y - Sc) = \min,
\]

leading to the orthogonality conditions and the solution for \( c \):

\[
S^TPW(y - Sc) = 0 \quad \Rightarrow \quad c = (S^TPWS)^{-1}S^TPWy
\]

in the interval \(-M \leq m \leq M\), 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 \( m \).
23.15. MISSING DATA AND OUTLIERS

where we assumed that \( S^T P W S \) is invertible.\(^1\) The estimated samples will be:

\[
\hat{y} = S c = (S^T P W S)^{-1} S^T W P y = B^T y \tag{23.15.4}
\]

with the filter matrix,

\[
B = P W S (S^T P W S)^{-1} S^T \tag{23.15.5}
\]

We note that \( P \) is a projection matrix (\( P^T = P \) and \( P^2 = P \)) and commutes with \( W \), \( P W = W P \), because both are diagonal. Defining \( Q = I - P \) to be the complementary projection matrix, the estimated signal can be decomposed in two parts: \( \hat{y} = T \hat{y} + Q \hat{y} \), with \( Q \hat{y} \) being the part that contains the estimated missing values or adjusted outliers.

The quantity \( T \hat{y} \) represents the samples that are being used to make the estimates, whereas \( Q \hat{y} \) corresponds to the missing samples and can be set to zero or to an arbitrary vector \( Q y_{arb} \). In other words, we may replace \( y \) by \( T \hat{y} + Q y_{arb} \) without affecting the solution of Eq. (23.15.4). This so because \( P (T \hat{y} + Q y_{arb}) = T \hat{y} \).

Once the estimated missing values have been obtained, we may replace \( Q y_{arb} \) by \( Q \hat{y} \) and recompute the ordinary \( W \)-weighted least-squares estimate from the adjusted vector \( T \hat{y} + Q \hat{y} \). This produces the same \( \hat{y} \) as in Eq. (23.15.4). Indeed, one can show that

\[
\hat{y} = (S^T P W S)^{-1} S^T W P y = (S^T W S)^{-1} S^T W (T \hat{y} + Q \hat{y}) \tag{23.15.6}
\]

To see this, start with the orthogonality equation (23.15.3), and replace \( P \hat{y} = \hat{y} - Q \hat{y} \):

\[
S^T W P (y - \hat{y}) = 0 \Rightarrow S^T W P y = S^T W P \hat{y} = S^T W (\hat{y} - Q \hat{y}) \tag{23.15.7}
\]

or,

\[
S^T W (T \hat{y} + Q \hat{y}) = S^T W \hat{y} = S^T W S (S^T P W S)^{-1} W P \hat{y} \tag{23.15.8}
\]

from which Eq. (23.15.6) follows by multiplying both sides by \( S^T W S \)\(^{-1} \). The MATLAB function \texttt{lmissing} implements the calculation of \( B \) in (23.15.5):

\begin{verbatim}
B = lmissing(N,d,m,s); % filter matrix for missing data
\end{verbatim}

The following MATLAB code illustrates the generation of Fig. 23.15.1:

\begin{verbatim}
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 = [-10 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 = lmissing(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
hold on; plot(t,x0,'--', t,y,'o', t,xnew,'-'); % right graph
\end{verbatim}

\(^1\)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 \cite{cleveland1979} discussed in the next section, we may apply the following procedure.

Given a length-\( L \) signal \( y_n, n = 0, 1, \ldots, L - 1 \), with \( L \leq N \), 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,

\[
B = lprs(N,d,s)
\]

\[
\hat{x} = \text{lpfilt}(B,y)
\]

\[
e = y - \hat{x}
\]

From the error residual \( e \), 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:

\[
\mu = \text{median}(|e_n|), \quad r_n = W \left( \frac{e_n}{K \mu} \right), \quad n = 0, 1, \ldots, L - 1
\]

where \( K \) is a constant such as \( K = 2-6 \), and \( W (u) \) is the bisquare function,

\[
W (u) = \begin{cases} (1-u^2)^2, & |u| \leq 1 \\ 0, & \text{otherwise} \end{cases}
\]

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 \( c \) in Eq. (23.15.3), that is,

\[
P_n = \text{diag} ([r_n-M, \ldots, r_n, \ldots, r_n+M])
\]

\[
\hat{x}_n = c_0 = u_0^T (S^T P W S)^{-1} S^T W P y(n)
\]

where \( u_0 = [1, 0, \ldots, 0]^T \) and \( y(n) = [y_{n-M}, \ldots, y_n, y_{n+M}]^T \). Eq. (23.15.10) may be used for \( M \leq n \leq L - 1 - M \). For \( 0 < n < M \) 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 \( y \) in (23.15.4) applied to the first and last length-\( N \) data vectors and robustness weights:

\[
y = [y_0, y_1, \ldots, y_{N-1}]^T, \quad P = \text{diag} ([r_0, r_1, \ldots, r_{N-1}])
\]

\[
y = [y_{L-N}, y_{L-N+1}, \ldots, y_{L-1}]^T, \quad P = \text{diag} ([r_{L-N}, r_{L-N+1}, \ldots, r_{L-1}])
\]

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 \texttt{rlpfilt} implements the above steps:

\[
[x,r] = \text{rlpfilt}(y,N,d,s,Nit) \quad \text{% robust local polynomial filtering}
\]
23.15. MISSING DATA AND OUTLIERS

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 \( \sigma^2 \), 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 \( K = \sqrt{5} = 2.44 \) (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.

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:

\[
J_n = \sum_{m=-M}^{M} (y_{n+m} - p(m))^2 w(m) = \min, \quad p(m) = \sum_{r=0}^{d} c_r m^r \tag{23.16.1}
\]

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:

\[
J_n = \sum_{k=-n}^{n} (y_k - p(k-n))^2 w(k-n) = \min, \quad p(k-n) = \sum_{r=0}^{d} c_r (k-n)^r \tag{23.16.2}
\]

For a set of \( N \) unequally-spaced observations \( \{t_k, y(t_k)\}, k = 0, 1, \ldots, N - 1 \), 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 \( t_0 \leq t \leq t_{N-1} \). 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, \( |t - t_k| \leq h_t \), to perform the polynomial fit:

\[
J_t = \sum_{|t - t_k| \leq h_t} (y(t_k) - p(t_k - t))^2 w(t_k - t) = \min, \quad p(t_k - t) = \sum_{r=0}^{d} c_r (t_k - t)^r \tag{23.16.3}
\]

The estimated/interpolated value at \( t \) will be \( \hat{x}_t = c_0 = p(0) \), and the estimated first derivative, \( \hat{x}_t = c_1 = \hat{p}(0) \), and so on for the higher derivatives, with \( r! c_r \) representing the \( r \)th derivative. As illustrated in Fig. 23.16.1, the fitted polynomial,

\[
p(x - t) = \sum_{r=0}^{d} c_r (x - t)^r, \quad t - h_t \leq x \leq t + h_t
\]

is local in the sense that it fits the observations only within the local window \( [t - h_t, t + h_t] \). The quantity \( y_k = p(t_k - t) \) represents the estimated value of the \( k \)th 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 \( -1 \leq u \leq 1 \), and setting \( u = (t_k - t)/h_t \):

\[
w(t_k - t) = W \left( \frac{t_k - t}{h_t} \right) \tag{23.16.4}
\]
Some typical choices for $W(u)$ are as follows [808]:

1. Tricube, \( W(u) = (1 - |u|^3)^3 \)
2. Bisquare, \( W(u) = (1 - u^2)^2 \)
3. Triweight, \( W(u) = (1 - u^3)^3 \)
4. Epanechnikov, \( W(u) = 1 - u^2 \)
5. Gaussian, \( W(u) = e^{-\alpha u^2/2} \)
6. Exponential, \( W(u) = e^{-\alpha |u|} \)
7. Rectangular, \( W(u) = 1 \)

where all types have support $|u| \leq 1$ and vanish for $|u| > 1$. A typical value for $\alpha$ 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 \texttt{locw} generates the above functions at any vector of values of $u$:

\[
W = \text{locw}(u, \text{type});
\]

where \texttt{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 $W(u) = (1 - u^s)^s$, 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:

\[
J_t = \sum_{k=0}^{N-1} \left( y(t_k) - p(t_k - t) \right)^2 w(t_k - t) = \min \quad p(t_k - t) = \sum_{r=0}^{d} c_r (t_k - t)^r
\]

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 $y_t$, the $N_t \times (d+1)$ basis matrix $S_t$, and the $N_t \times N_t$ diagonal matrix of weights by

\[
y_t = [\cdots, y(t_k), \cdots]^T, \quad \text{for } t - h_t \leq t_k \leq t + h_t
\]

\[
S_t = \begin{bmatrix}
1 & (t_k - t) & (t_k - t)^2 & \cdots & (t_k - t)^d \\
0 & 1 & (t_k - t) & \cdots & (t_k - t)^{d-1} \\
0 & 0 & 1 & \cdots & (t_k - t)^{d-2} \\
0 & 0 & 0 & \cdots & 1
\end{bmatrix}
\]

\[
W_t = \text{diag}([\cdots, w(t_k - t), \cdots])
\]

where $u^f(t_k - t)$ is the $k$-th row of $S_t$, defined in terms of the $(d+1)$-dimensional vector $u^f(\tau) = [1, \tau, \tau^2, \ldots, \tau^d]$. For example, if $t - h_t < t_1 < t_2 < t_3 < t_4 < t_5 < t + h_t$, then $N_t = 4$ and for a polynomial order $d = 2$, we have:

\[
\begin{align*}
S_t &= \begin{bmatrix}
1 & (t_k - t) & (t_k - t)^2 \\
0 & 1 & (t_k - t) \\
0 & 0 & 1
\end{bmatrix} \\
W_t &= \text{diag}([w(t_1 - t), w(t_2 - t), w(t_3 - t), w(t_4 - t), w(t_5 - t)])
\end{align*}
\]

With these definitions, Eq. (23.16.3) can be written as

\[
J_t = (y_t - S_t c) ^T W_t (y_t - S_t c) = \min
\]

with solution for the coefficient vector $c = [c_0, c_1, \ldots, c_d]^T$:

\[
c = (S_t^TW_tS_t)^{-1} S_t^TW_t y_t
\]
The quantity \( \hat{y}_t = S_t c \) represents the polynomial estimate of the local observation vector \( y_t \). It can be written as

\[
\hat{y}_t = B_t^T y_t, \quad B_t = W_t S_t (S_t^T W_t S_t)^{-1} S_t^T
\]

(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),

\[
B_t^T S_t = S_t
\]

(23.16.11)

Defining the usual \((d+1)\)-dimensional unit vector \( u_0 = [1, 0, \ldots, 0]^T \), we obtain the estimated value at time \( t \) by \( \hat{x}_t = c_0 = u_0^T c \), and the first derivative by \( \hat{x}_t = c_1 = u_1^T c \), where \( u_1 = [0, 1, 0, \ldots, 0]^T \).

\[
\hat{x}_t = u_0^T (S_t^T W_t S_t)^{-1} S_t^T W_t y_t
\]

\[
\hat{x}_t = u_1^T (S_t^T W_t S_t)^{-1} S_t^T W_t y_t
\]

(23.16.12)

Thus, the effective estimation weights are:

\[
h(t) = W_t S_t (S_t^T W_t S_t)^{-1} u_0, \quad \hat{x}_t = h^T(t) y_t
\]

(23.16.13)

Component-wise, we can write:

\[
\hat{x}_t = h^T(t) y_t = \sum_{|t_k - t| \leq h_t} h_k(t) y_k
\]

(23.16.14)

where \( y_k = y(t_k) \) and

\[
h_k(t) = w(t_k - t) u^T(t_k - t) (S_t^T W_t S_t)^{-1} u_0
\]

(23.16.15)

We note that \( u_0, u_1 \) are related to the vector \( u(\tau) \) and its derivative by \( u_0 = u(0) \) and \( u_1 = u(0) \). We also have,

\[
S_t^T W_t S_t = \sum_{|t_k - t| \leq h_t} u(t_k - t) u^T(t_k - t) w(t_k - t)
\]

(23.16.16)

or, component-wise,

\[
(S_t^T W_t S_t)_{ij} = \sum_{|t_k - t| \leq h_t} (t_k - t)^{i+j} w(t_k - t), \quad i, j = 0, 1, \ldots, d
\]

(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 \textit{kernel smoothing} approach first proposed by Nadaraya and Watson [772,773]. In this case \( u(\tau) = [1] \) and we find:

\[
S_t^T W_t S_t = \sum_{|t_k - t| \leq h_t} w(t_k - t), \quad h_k(t) = \frac{w(t_k - t)}{\sum_{|t_k - t| \leq h_t} w(t_k - t)}
\]

(23.16.18)

In general, the invertibility of \( S_t^T W_t S_t \) requires that \( N_t \geq d + 1 \). 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 \( W(t) = \frac{1}{s_0(t)s_2(t)-s_1(t)^2} \), \( s_0(t), s_1(t), s_2(t) \) see Ref. [45], and carrying out the QR factorization of the matrix \( W(t) = \frac{1}{s_0(t)s_2(t)-s_1(t)^2} \), \( s_0(t), s_1(t), s_2(t) \) which gives for the filter weights \( h_k(t) \):

\[
h_k(t) = w(t_k - t) s_2(t) - s_1(t) s_0(t)
\]

(23.16.19)

or, component-wise,

\[
(S_t^T W_t S_t)^{-1} = \frac{1}{s_0(t)s_2(t)-s_1(t)^2} \begin{bmatrix} s_2(t) & -s_1(t) \\ -s_1(t) & s_0(t) \end{bmatrix}
\]

(23.16.20)

The above steps are equivalent to reducing the problem to an ordinary unweighted least-squares problem, that is, \( c \) is recognized to be the unique least-squares solution of the full-rank, overdetermined, \( N_t \times (d+1) \)-dimensional system \( (US_t)c = U y_t \). Indeed, \( c \) is given by:

\[
c = [(US_t)^T (US_t)]^{-1} (US_t)^T U y_t = (S_t^T W_t S_t)^{-1} S_t^T W_t y_t
\]

(23.16.21)

where \( [(US_t)^T (US_t)]^{-1} (US_t)^T \) is the pseudoinverse of \( US_t \). The corresponding performance indices are equivalent:

\[
J_t = (y_t - S_t c)^T W_t (y_t - S_t c) = \| U y_t - US_t c \|_2^2 = \min
\]

(23.16.22)

In MATLAB the least-squares solution (23.16.21) can be obtained by the backslash operation: \( c = (US_t) \backslash U y_t \). The construction of the quantities \( y_t, S_t, W_t \) is straightforward. Given the column vectors of observation times and observations, \( t_{obs} = [t_0, t_1, \ldots, t_{N-1}]^T, \quad y_{obs} = [y(t_0), y(t_1), \ldots, y(t_{N-1})]^T \)

we may determine, with the help of \( t_{obs} \), the column vector of indices \( k \) for which \( t_k \) lies in the local window, and then carry out the procedure (23.16.21):
23.16. WEIGHTED LOCAL POLYNOMIAL MODELING

Local linear regression can be used to fit the following general equation:

\[ y(t) = \sum_{k=1}^{K} c_k u(t-t_k) + \epsilon(t) \]

where \( c_k \) are the coefficients of the polynomial; \( u(t-t_k) \) are the basis functions; \( t_k \) are the fitting times; \( \epsilon(t) \) is the error term; \( K \) is the number of fitting times; and \( d \) is the order of the polynomial.

### 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 \( K = \alpha N = 0.25 \times 16 = 4 \):

\[
\begin{align*}
t_{obs} & = [0.5, 0.8, 1.1, 1.2, 1.8, 2.4, 2.5, 3.4, 3.5, 3.7, 4.0, 4.2, 4.9, 5.0, 5.1, 6.2] \\
t & = [0.5, 1.5, 2.9, 3.6, 5.1]
\end{align*}
\]

then one finds the corresponding bandwidths for each of the five \( t \)’s:

\[
h = \text{locband}(t_{obs}, t, 0.25, 0) = [0.7, 0.7, 0.6, 0.6, 0.9]
\]

### Remarks

In the fixed case, one chooses the same bandwidth at each fitting time, that is, \( h = h_1 = h_2 = \ldots = h_L \). The output \( x_{hat} \) is the same as \( x_1, x_2, \ldots, x_L \) vector of fitting times. In the nearest-neighbor case selects varying \( h_1 \) for all \( t \). In the nearest-neighbor case, \( K \) is the number of included observations. As an example, consider the following 16 observation times \( t_{obs} \), and 5 fitting times \( t \), and choose \( \alpha = 0.25 \) so that \( K = \alpha N = 0.25 \times 16 = 4 \):

\[
\begin{align*}
t_{obs} & = [0.5, 0.8, 1.1, 1.2, 1.8, 2.4, 2.5, 3.4, 3.5, 3.7, 4.0, 4.2, 4.9, 5.0, 5.1, 6.2] \\
t & = [0.5, 1.5, 2.9, 3.6, 5.1]
\end{align*}
\]

then one finds the corresponding bandwidths for each of the five \( t \)’s:

\[
h = \text{locband}(t_{obs}, t, 0.25, 0) = [0.7, 0.7, 0.6, 0.6, 0.9]
\]

and the corresponding local intervals, each containing \( K = 4 \) observation times:

\[
\begin{array}{cccc}
h_1 & t - h_1 & t & t + h_1 & \text{included} \\
0.7 & -0.2 & 0.5 & 1.2 & 0.5, 0.8, 1.1, 1.2 \\
0.7 & 0.8 & 1.5 & 2.2 & 0.8, 1.1, 1.2, 1.8 \\
0.6 & 2.3 & 2.9 & 3.5 & 2.4, 2.5, 3.4, 3.5 \\
0.6 & 3.3 & 4.1 & 4.7 & 3.3, 3.7, 4.0, 4.2 \\
0.9 & 4.2 & 5.1 & 6.0 & 4.2, 4.9, 5.0, 5.1 \\
\end{array}
\]

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:

\[
\begin{array}{cccc}
h_1 & t - h_1 & t & t + h_1 & \text{included} \\
0.7 & -0.2 & 0.5 & 1.2 & 0.5, 0.8, 1.1, 1.2 \\
0.7 & 0.8 & 1.5 & 2.2 & 0.8, 1.1, 1.2, 1.8 \\
0.7 & 2.2 & 2.9 & 3.6 & 2.4, 2.5, 3.4, 3.5 \\
0.7 & 2.9 & 3.6 & 4.3 & 3.4, 3.5, 3.7, 4.0, 4.2 \\
0.7 & 4.4 & 5.1 & 5.8 & 4.9, 5.0, 5.1 \\
\end{array}
\]

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:

\[
\begin{align*}
\hat{x}_i & = u_0^T (S_i^T W_i S_i)^{-1} S_i^T W_i y_i = h_i^T y_i, \\
h_i & = u_0^T (S_i^T W_i S_i)^{-1} S_i^T W_i
\end{align*}
\]

where \( \hat{x}_i, S_i, W_i, y_i \) are the quantities \( \hat{x}_i, S_i, W_i, y_i \) evaluated at \( t = t_i \). Component-wise,

\[
\hat{x}_i = \sum_{t_j-t_i \leq h} u_0^T (S_i^T W_i S_i)^{-1} u_j (t_j - t_i) w(t_j-t_i) y_j = \sum_{t_j-t_i \leq h} H_{ij} y_j
\]

where the matrix elements \( H_{ij} \) are,

\[
H_{ij} = h_j (t_j) = u_0^T (S_i^T W_i S_i)^{-1} u_j (t_j - t_i) w(t_j-t_i)
\]

Similarly, one may express \( S_i^T W_i S_i \) and \( S_i^T W_i y_i \) as,

\[
\begin{align*}
S_i^T W_i S_i & = \sum_{t_j-t_i \leq h} u_j (t_j - t_i) u_j^T (t_j - t_i) w(t_j-t_i) \\
S_i^T W_i y_i & = \sum_{t_j-t_i \leq h} u_j (t_j - t_i) w(t_j-t_i) y_j
\end{align*}
\]

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.
23.17. BANDWIDTH SELECTION WITH CV AND GCV

Defining the $N$-dimensional vectors $\hat{x} = [x_0, \hat{x}_1, \ldots, \hat{x}_{N-1}]^T$ and $y = [y_0, y_1, \ldots, y_{N-1}]^T$, we may write (23.16.24) in the compact matrix form:

$$\hat{x} = Hy$$  \hspace{1cm} (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 $w_0 = w(0)$.

$$H_{ii} = h_i(t_i) = w_0 u_0^T (S_i^T W_i S_i)^{-1} u_0$$  \hspace{1cm} (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:

$$CV(h) = \frac{1}{N} \sum_{i=0}^{N-1} (y_i - \hat{x}_i)^2 = \min$$  \hspace{1cm} (23.17.1)

where $\hat{x}_i$ is the estimate or prediction of the sample $x_i = x(t_i)$ obtained by deleting the $i$th 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 $\hat{x}$ based on all observations by the relationship:

$$\hat{x}_i = \frac{\hat{x} - H_{ii} y_i}{1 - H_{ii}}$$  \hspace{1cm} (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:

$$y_i - \hat{x}_i = \frac{y_i - \hat{x}}{1 - H_{ii}}$$  \hspace{1cm} (23.17.3)

and therefore, the CV index can be expressed as:

$$CV(h) = \frac{1}{N} \sum_{i=0}^{N-1} (y_i - \hat{x}_i)^2 = \frac{1}{N} \sum_{i=0}^{N-1} \left( \frac{y_i - \hat{x}}{1 - H_{ii}} \right)^2 = \min$$  \hspace{1cm} (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,

$$GCV(h) = \frac{1}{N} \sum_{i=0}^{N-1} \left( \frac{y_i - \hat{x}}{1 - H} \right)^2 = \min, \hspace{0.5cm} H = \frac{1}{N} \sum_{i=0}^{N-1} H_{ii} = \frac{1}{N} \text{tr}(H)$$  \hspace{1cm} (23.17.5)

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 $\alpha$ 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

$$c_\ast = (S_i^T W_i S_i)^{-1} (S_i^T W_i y_i) \rightarrow \hat{x}_i = u_0^T c_\ast$$

where the minus subscripts indicate that the $t_j = t_i$ terms are to be omitted. It follows from Eq. (23.16.26) that

$$S_i^T W_i S_i = (S_i^T W_i S_i)_{-} + w_0 u_0 u_0^T \hspace{1cm} (23.17.6)$$

and then,

$$c_\ast = [S_i^T W_i S_i - w_0 u_0 u_0^T]^{-1} [S_i^T W_i y_i - w_0 u_0 y_i] \hspace{1cm} (23.17.7)$$

Setting $F_i = S_i^T W_i S_i$ and noting that $c = F_i^{-1} S_i^T W_i y_i$ or $S_i^T W_i y_i = F_i c$, we may write,

$$c_\ast = [F_i - w_0 u_0 u_0^T]^{-1} [F_i c - w_0 u_0 y_i]$$

Using the matrix inversion lemma, we have,

$$[F_i - w_0 u_0 u_0^T]^{-1} = F_i^{-1} + \frac{w_0 F_i^{-1} u_0 u_0^T F_i^{-1}}{1 - w_0 u_0 F_i^{-1} u_0} \hspace{1cm} (23.17.8)$$

Noting that $H_{ii} = w_0 u_0^T F_i^{-1} u_0$, we obtain,

$$c_\ast = \left[ F_i^{-1} + \frac{w_0 F_i^{-1} u_0 u_0^T F_i^{-1}}{1 - H_{ii}} \right] [F_i c - w_0 u_0 y_i]$$

$$= \left[ I + \frac{w_0 F_i^{-1} u_0 u_0^T}{1 - H_{ii}} \right] [c - w_0 F_i^{-1} u_0 y_i]$$

$$= c - w_0 F_i^{-1} u_0 y_i + \frac{w_0 F_i^{-1} u_0 [u_0^T c - w_0 u_0^T F_i^{-1} u_0 y_i]}{1 - H_{ii}}$$

and since $\hat{x}_i = u_0^T c$, we find,

$$c_\ast = c - w_0 F_i^{-1} u_0 y_i + \frac{w_0 F_i^{-1} u_0 [\hat{x}_i - H_{ii} y_i]}{1 - H_{ii}} = c + w_0 F_i^{-1} u_0 [\hat{x}_i - y_i] \frac{1}{1 - H_{ii}}$$

from which we find for $\hat{x}_i = u_0^T c$,

$$\hat{x}_i = \hat{x}_i + \frac{H_{ii} (\hat{x}_i - y_i)}{1 - H_{ii}} = \hat{x}_i + \frac{\hat{x}_i - y_i}{1 - H_{ii}}$$  \hspace{1cm} (23.17.9)

In practice, the CV and GCV indices are evaluated over a certain range of the smoothing parameter $h$ or $\alpha$ to look for a minimum. The MATLAB function `locpcv` evaluates these indices at any vector of parameter values.
$$[\text{GCV, CV}] = \text{locgcv}(\text{tobs}, \text{yobs}, d, \text{type}, b, b\text{type});$$

where type is the window type, b is either a vector of hs or a vector of alphas 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 \( \hat{y} \) at a less dense grid of t's, 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 \( T = t_2 - t_1 \),

\[
P(t) = \left( \frac{t - t_2}{T} \right)^2 \left[ a_1 + (T b_1 + 2 a_1) \left( \frac{t - t_1}{T} \right) \right] + \left( \frac{t - t_1}{T} \right)^2 \left[ a_2 + (T b_2 - 2 a_2) \left( \frac{t - t_2}{T} \right) \right]
\]

(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,

\[
P(t) = a_1 + (a_2 - a_1) \left( \frac{t - t_1}{T} \right)
\]

(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 \( y_{\text{grid}} \) at the denser grid of points \( t_{\text{grid}} \):

\[
y_{\text{grid}} = \text{locval}(C, t, t_{\text{grid}}); \quad \text{interpolating local polynomial fits}
\]

The auxiliary function locgrid helps establish a uniform grid between the \( t \) points:

\[
t_{\text{grid}} = \text{locgrid}(t, \text{Ngrid}); \quad \text{uniform grid with respect to} \ t
\]

which is simply a shorthand for,

\[
t_{\text{grid}} = \text{inspace}(\text{min}(t), \text{max}(t), \text{Ngrid});
\]

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 \( \alpha \) on the left, and as a function of the fixed bandwidth \( h \) on the right, for the two polynomial orders \( d = 1, 2 \).

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:

\[
\text{Y} = \text{loadfile('mcyc.dat');}
\]

\[
\text{tobs} = \text{Y(:,1);} \quad \text{yobs} = \text{Y(:,2);}
\]

\[
\alpha = \text{inspace}(0.1, 0.5, 51); \quad \text{variable over } 0.1 \leq \alpha \leq 0.5
\]

\[
d=1; \quad \text{type}=1; \quad \text{gcv} = \text{locgcv}(\text{tobs}, \text{yobs}, d, \text{type}, \alpha, 'nn'); \quad \text{GCV as function of} \ \alpha
\]

\[
[F, I] = \text{min} (\text{gcv}); \quad \text{alpha} = \text{alpha}(I); \quad \text{minimum at} \ \alpha = \alpha_1
\]

\[
\text{figure; plot(alpha, gcv);}
\]

\[
h = \text{inspace}(2, 10, 51); \quad \text{GCV as function of} \ h
\]

\[
\text{gcv} = \text{locgcv}(\text{tobs}, \text{yobs}, d, \text{type}, h, 'f'); \quad \text{GCV as function of} \ h
\]

\[
[F, I] = \text{min} (\text{gcv}); \quad h_1 = h(I);
\]
23.18. LOCAL POLYNOMIAL INTERPOLATION

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).

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 \( \text{locval} \). Continuing with the above MATLAB code, the graphs were generated by

\[
\begin{align*}
\text{t} &= \text{locgrid(tobs,101)}; & \text{equally-spaced fitting times} \\
\text{h} &= \text{locband(tobs, t, alpha1, 0);} & \text{NN bandwidths at each } t \\
\text{x1} &= \text{locpol(tobs,yobs,t,h,1,type);} & \text{fit at times } t \text{ with } d = 1 \\
\text{h} &= \text{locband(tobs, t, alpha2, 0);} & \text{NN bandwidths at each } t \\
\text{x2} &= \text{locpol(tobs,yobs,t,h,2,type);} & \text{fit at times } t \text{ with } d = 2 \\
\text{figure; plot(tobs,yobs,'.', t,x1,'-', t,x2,'--');} & \text{left graph} \\
\text{h} &= \text{locband(tobs, t, 0, h1);} & \text{fixed bandwidths at each } t \\
\text{x1} &= \text{locpol(tobs,yobs,t,h1,type);} & \text{fit at times } t \text{ with } d = 1 \\
\text{h} &= \text{locband(tobs, t, 0, h2);} & \text{fixed bandwidths at each } t \\
\text{x2} &= \text{locpol(tobs,yobs,t,h2,type);} & \text{fit at times } t \text{ with } d = 2 \\
\text{figure; plot(tobs,yobs,'.', t,x1,'-', t,x2,'--');} & \text{right graph}
\end{align*}
\]

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:

\[
\begin{align*}
\text{tf} &= \text{locgrid(tobs,21)}; & \text{fitting times} \\
\text{h} &= \text{locband(tobs, tf, alpha1, 0);} & \text{NN bandwidths at } \text{tf} \\
\{\text{xf},\text{C}\} &= \text{locpol(tobs,yobs,tf,h,1,type);} & \text{fitted values and derivatives} \\
\text{tint} &= \text{locgrid(tf,101);} & \text{interpolation times} \\
\text{xint} &= \text{locval(C, tf, tint);} & \text{interpolated values} \\
\text{figure; plot(tobs,yobs,'.', tf,xf,'o', tint,xint,'-');} & \text{plot(tobs,yobs,'.', tf,xf,'o', tint,xint,'-');}
\end{align*}
\]

**Example 23.18.2:** The ethanol dataset [814] is also a benchmark example for smoothing techniques. The ordinate NOx 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 \( d = 1, 2 \).

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:

```matlab
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 \leq h \leq 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); % order 1, tricube window
h = locband(tobs, t, 0, h2); % fit with d = 1 and tricube window
x2 = locpol(tobs,yobs,t,h,2,1);
figure; plot(tobs,yobs,'.', t,x1,'-', t,x2,'--'); % right graph
```

The MATLAB code for generating Fig. 23.18.5 is as follows:

```matlab
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,'-', t,xl,'--'); % 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 window
figure; plot(tobs,yobs,'.', t,x1,'-', t,xl,'--'); % 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:

```matlab
tf = locgrid(tobs,10); % fitting points
alpha = 0.3; h = locband(tobs, tf, alpha, 0); % nearest-neighbor bandwidths
ti = locgrid(tf,101); yi = locval(C,tf,ti);
figure; plot(tobs,yobs,'.', ti,yi,'-', tf,xf,'o');
```

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 \( \kappa_t \), 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:

$$\kappa_t = \frac{|\mathring{\mathring{s}}_t|}{\max |\mathring{\mathring{s}}_t|} \quad (23.19.1)$$

The second derivative $\mathring{\mathring{s}}_t$ 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 $\alpha$. 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

$$h_t = h_{\max} \left( \frac{h_{\min}}{h_{\max}} \right)^\kappa_t \quad (23.19.2)$$

Other simple choices are possible, for example,

$$h_t = h_{\max} h_{\min}^{\kappa_t^p}$$

for some power $p$. Since $\kappa_t$ 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

$$s(t) = \sum_{i=1}^{11} \frac{a_i}{1 + |t-t_i|/w_i}^d, \quad 0 \leq t \leq 1$$

with the parameter values:

$$t_i = [10, 13, 15, 23, 25, 40, 44, 65, 76, 78, 81]/100$$
$$a_i = [40, 50, 30, 40, 50, 42, 21, 43, 31, 51, 42] \cdot 1.0523$$
$$w_i = [5, 5, 6, 10, 10, 30, 10, 10, 5, 8, 5]/10000$$

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 $y_n = s_n + \sigma_n$, 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 $\sigma_s$ 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.

The curvature $\kappa_t$, 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:

```matlab
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;
w = [5,5,6,10,10,30,10,10,5,8,5]/10000;

for i=1:length(ai),
    s = s + ai(i)*F((t-ti(i))/w(i));
end

y = s + randn(size(t));

hmax=20e-3; hmin=2.5e-4; h0=hmax;
seed=2009; randn('state',seed);
% noisy signal

for i=1:length(ai),
    s = s + ai(i)*F((t-ti(i))/w(i));
end

y = s + randn(size(t));

% fit with d = 2 and tricube window
[x,C] = locpol(t,y,t,h0,d,type);
k = abs(CC(:,3)); kt = k = max(kt);
h = hmax * (hmin/hmax).*kt;
```
23.19. VARIABLE AND ADAPTIVE BANDWIDTH

for i=1:length(ai),
    s = s + ai(i) * (1 + sign(t - ti(i)))/2;
end

% use fixed bandwidth
xf = locpol(t,y,t,h0,d,type);

hmax=0.03; hmin=0.0015; h0=hmax;

% upper graphs
figure; plot(t,xv); figure; plot(t,xf); figure; plot(t,ht/hmax);
% lower graphs
figure; plot(t,sn); figure; plot(t,yn);

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

$$s(t) = \sum_{i=1}^{11} a_i F(t - t_i), \quad F(t) = \frac{1}{2} (1 + \text{sign} t), \quad 0 \leq t \leq 1$$

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 $y_n = s_n + \nu_n$ 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 $h_{\text{max}} = 0.03$ and $h_{\text{min}} = 0.0015$:

$$N=2048; \quad t=\text{linspace}(0, 1, N); \quad s=\text{zeros}(\text{size}(t));$$
$$t_i = [10 13 15 23 25 40 44 65 76 81]/100; \quad a_i = [40, -50, 30, -40, 50, -42, 21, 43, -31, 21, -42] \cdot 0.3655;$$

Fig. 23.19.2 shows the “blocks” function defined by

$$s(t) = \sum_{i=1}^{11} a_i F(t - t_i), \quad F(t) = \frac{1}{2} (1 + \text{sign} t), \quad 0 \leq t \leq 1$$

where the factor 2.357 is adjusted to give $\text{std}(s)/\sigma = 7$. 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:

$$s(t) = \left[ 4 \sin(4\pi t) - \text{sign}(t - 0.3) - \text{sign}(0.72 - t) \right] \cdot 2.357, \quad 0 \leq t \leq 1$$

We note that the curvature $\kappa_t$ is significantly large—and the bandwidth $h_t$ is significantly
small—only near the discontinuity points. The fixed bandwidth case smooths
the discontinuities too much, whereas the variable bandwidth tends to preserve them
while reducing the noise equally well in the rest of the signal.

The “doppler” example shown in Fig. 23.19.4, noticing that the curvature $\kappa_t$ is significantly large only in the range $0 \leq t \leq 0.2$, 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 $0 \leq t \leq 0.2$ and transitioned gradually to a larger bandwidth for $0.2 \leq t \leq 1$. The signal is defined by

$$s(t) = 24\sqrt{t(1-t)} \sin \left( \frac{2.1\pi}{t+0.05} \right), \quad 0 \leq t \leq 1$$

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 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 $t = 14.6$.

Let $n_k$ denote the multiplicity of the observations at time $t_k$, that is, let $y_i(t_k)$, $i = 1, 2, \ldots, n_k$ 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 $y_i(t_k)$:

$$J_t = \sum_{|t_k-t|<h_t} \sum_{i=1}^{n_k} [y_i(t_k) - \mathbf{u}^T(t_k-t) \mathbf{c}]^2 w(t_k-t) = \min \quad (23.20.1)$$

Setting the gradient with respect to $\mathbf{c}$ to zero, gives the normal equations:

$$\sum_{|t_k-t|<h_t} \sum_{i=1}^{n_k} w(t_k-t) \mathbf{u}(t_k-t) \mathbf{u}^T(t_k-t) \mathbf{c} = \sum_{|t_k-t|<h_t} w(t_k-t) \mathbf{u}(t_k-t) \sum_{i=1}^{n_k} y_i(t_k)$$

Defining the average of the $n_k$ observations,

$$\hat{y}(t_k) = \frac{1}{n_k} \sum_{i=1}^{n_k} y_i(t_k)$$
and noting that the left-hand side has no dependence on \( i \), we obtain:

\[
\sum_{|t_k - t| \leq h_l} n_k w(t_k - t) u(t_k - t) u^T(t_k - t) c = \sum_{|t_k - t| \leq h_l} n_k w(t_k - t) u(t_k - t) \hat{\gamma}(t_k) \tag{23.20.2}
\]

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 \( k \)th observation replaced by the average \( \bar{y}(t_k) \), that is,

\[
\bar{y}_l = \sum_{|t_k - t| \leq h_l} [\hat{\gamma}(t_k) - u^T(t_k - t) c]^2 n_k w(t_k - t) = \min \tag{23.20.3}
\]

Internally, the function `locep01` 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);
```

For example, if

\[
tobs = [1 1 1 3 3 3 4 4 7 9 9 9 9 9];
yobs = [20 22 21 11 12 13 15 19 21 25 28 29 31 32];
\]

the function first sorts the `t`s in increasing order,

\[
tobs = [1 1 1 3 3 3 4 4 5 5 5 7 9 9 9 9 9];
yobs = [20 20 22 22 21 11 12 19 21 13 15 25 28 29 31 32];
\]

and then returns the output,

\[
ta = [1 3 4 5 5 7 9];
ya = [21 14 21 14 25 30];
na = [3 3 1 1 2 1 4];
\]

### 23.21 Loess Smoothing

Loess, which is a shorthand for local regression, 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 \( t_{obs}, y_{obs}, \) the nearest-neighbor bandwidth parameter \( \alpha \), 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:

\[
h = \text{locband}(t_{obs}, \alpha, 0); \quad \text{find local bandwidths at } t_{obs}
\]

\[
\hat{x} = \text{locpol}(t_{obs}, y_{obs}, t_{obs}, h, d, 1); \quad \text{perform fit at all } t_{obs} \tag{23.21.1}
\]

where the last argument of `locpol1` designates the use of the tricube window. From the resulting \( N \)-dimensional signal \( \hat{x}_k, k = 0, 1, \ldots, N - 1 \), we calculate the corresponding error residuals \( e_k \) and use their median to calculate “robustness” weights \( r_k \):

\[
e_k = y_k - \hat{x}_k, \quad k = 0, 1, \ldots, N - 1
\]

\[
\mu = \text{median}(\{|e_k|\})
\]

\[
r_k = W \left( \frac{e_k}{\mu} \right)
\]

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 \( w(t_k - t_{obs}) \) for the \( k \)th observation’s contribution to the fit, one uses the modified weights \( r_k w(t_k - t_{obs}) \). 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 \( r_k \), 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 \( k \)th observation, that is, the weight matrix \( W(t) \) in Eq. (23.16.7) is replaced by

\[
W(t) = \text{diag}(\{ \ldots, r_k w(t_k - t), \ldots \})
\]

The MATLAB function `loess` implements these steps:

```
[xhat, C] = loess(tobs, yobs, t, alpha, d, Nit);
```

where \( t \) are the final fitting times and \( xhat \) and \( C \) have the same meaning as in `locpol1`. 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 \( N_{it} = 0, 2, 4, 6 \) iterations—the first one corresponding to ordinary fitting with no robustness iterations. The MATLAB code for the top two graphs was:

```matlab
\[
t = (0:50); x0 = (1 - \cos(2*\pi*t/50))/2;
\]

\[
seed=2005; \text{randn(’state’,seed);
}\]

\[
y = x0 + 0.1 \times \text{randn(size(x0))};
\]

\[
m = [-1 0 1 3]; \quad \text{noisy signal}
\]

\[
n0=25; y(n0+1) = 0; \quad \text{outlier indices}
\]

\[
n1=10; y(n1+1) = 1; \quad \text{outlier values}
\]

\[
alpha=0.4; \quad d=2; \quad \text{bandwidth parameter and polynomial order}
\]

\[
\text{Nit}=0; x = \text{loess}(t,y,t,\alpha,d,\text{Nit}); \quad \text{loess fit at } t
\]

\[
\text{figure; plot(t,x,’-‘, t,y,’.’, t,x,’-‘); \quad \text{left graph}
\]

\[
\text{Nit}=2; x = \text{loess}(t,y,\alpha,d,\text{Nit}); \quad \text{loess fit at } t
\]

\[
\text{figure; plot(t,x,’-‘, t,y,’.’, t,x,’-‘); \quad \text{right graph}
\]

The loess fit was performed at all \( t \). We observe how successive iterations gradually diminish the distorting influence of the outliers.
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

\[ H_{ii} = H_{ii}^{-1} + H_{ii}^{-1}F_{i}^{-1}u_{0}, \quad F_{i}^{-1} = (S_{i}^T W_{i} S_{i})^{-1}, \]

then, argue that \( 0 \leq H_{ii} \leq 1. \)

---

Fig. 23.21.1 Loess smoothing with \( d = 2, \alpha = 0.4, \) and different iterations.

23.22 Problems

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.).

2. Prove the matrix inversion lemma identity (23.17.8). Using this identity, show that

\[ H_{ii} = H_{ii}^{-1} + H_{ii}^{-1}F_{i}^{-1}u_{0}, \quad F_{i}^{-1} = (S_{i}^T W_{i} S_{i})^{-1}, \]

then, argue that \( 0 \leq H_{ii} \leq 1. \)

---

4. 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:

\[ y_n = a_n + v_n \]

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:

\[ J = E[(y_n - a)^2] = \min \quad a_{opt} = m = E[y_n] \]

with minimized value \( J_{\text{min}} = \sigma^2_f \). This result is obtained by setting the gradient with respect to \( a \) to zero:

\[ \frac{\partial a}{\partial J} = -2E[y_n - a] = 0 \]

---

\( ^1 \)adapted from the author’s book on Applied Optimum Signal Processing [45]
In general, given a theoretical performance index $J$, one must replace it in practice by an experimental one, say $\hat{J}$, expressible in terms of the actual available data. The minimization of $\hat{J}$ provides then estimates of the parameters or signals to be estimated.

$$\hat{J} = \text{min}$$

Theoretical optimum solution

Data

Estimated optimum solution

Depending on the index $\hat{J}$, 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 $\hat{J}$ that imitate (24.1.2) are:

$$\hat{J} = \sum_{n=0}^{L-1} (y_n - \hat{a})^2 = \text{min}$$

$$\hat{J} = \sum_{k=0}^{n} (y_k - \hat{a})^2 = \text{min}$$

$$\hat{J} = \sum_{k=n-N+1}^{n} (y_k - \hat{a})^2 = \text{min}$$

$$\hat{J} = \sum_{k=0}^{n} \lambda^{n-k} (y_k - \hat{a})^2 = \text{min}$$

The first assumes a length-$L$ block of data $[y_0, y_1, \ldots, y_{L-1}]$. 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 $[y_0, y_1, \ldots, y_n]$, while the third uses a length-$N$ sliding window $[y_{n-N+1}, \ldots, y_{n-1}, y_n]$. 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" $\lambda$ be in the range $0 < \lambda < 1$. These time ranges are depicted below.

In order for the $\hat{J}s$ to be unbiased estimates of $J$, 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:

$$\hat{a} = \frac{y_0 + y_1 + \cdots + y_{L-1}}{L}$$;

(24.1.5a)

$$\hat{a}_n = \frac{y_0 + y_1 + \cdots + y_n}{n+1}$$;

(24.1.5b)

$$\hat{a}_n = \frac{y_n + y_{n-1} + \cdots + y_{n-N+1}}{N}$$;

(24.1.5c)

$$\hat{a}_n = \frac{y_n + \lambda y_{n-1} + \lambda^2 y_{n-2} + \cdots + \lambda^N y_0}{1 + \lambda + \lambda^2 + \cdots + \lambda^N}$$.

(24.1.5d)

We have hatched on a subscript $n$ to the last three to emphasize their dependence on the index $\hat{J}$, 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.

Eqs. (24.1.4c) and (24.1.5c) tentatively assume that

$$\hat{J}_n \geq 0 < \lambda < 1;$$

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):

$$\sigma^2_{\hat{a}} = \sigma^2_{\hat{y}} \frac{L}{L}$$;

(24.1.6a)

$$\sigma^2_{\hat{a}_n} = \sigma^2_{\hat{y}} \frac{n}{n+1}$$;

(24.1.6b)

$$\sigma^2_{\hat{a}_n} = \sigma^2_{\hat{y}} \frac{N}{N}$$;

(24.1.6c)

$$\sigma^2_{\hat{a}_n} = \sigma^2_{\hat{y}} \frac{1 - \lambda}{1 + \lambda} \frac{1 + \lambda^{n+1}}{1 + \lambda + \cdots + \lambda^{n+1}}$$.

(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 $\lambda$.

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:

$$\frac{\partial \hat{a}}{\partial \hat{J}} = -2E[y_n - \hat{a}] \Rightarrow \frac{\partial \hat{a}}{\partial \hat{J}} = -2[y_n - \hat{a}_{n-1}]$$.

(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 $\mu$:

$$\Delta a = -\mu \frac{\partial \hat{a}}{\partial \hat{J}}; \quad \hat{a}_n = \hat{a}_{n-1} + \Delta a$$.

(24.1.8)
24.1. MEAN TRACKING

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 $\mu(y_n)$ 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:

$$\hat{a}_n = (1 - \alpha_n) \hat{a}_{n-1} + \alpha_n y_n$$

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 $\alpha_n$. 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,

$$\hat{a}_n = \lambda \hat{a}_{n-1} + \alpha y_n = \hat{a}_{n-1} + \alpha (y_n - \hat{a}_{n-1})$$

Its transfer function and asymptotic variance are:

$$H(z) = \frac{\alpha}{1 - \lambda z^{-1}} , \quad \sigma_{\hat{a}_n}^2 = E[(\hat{a}_n - m)^2] = \sigma_y^2 \frac{1 - \lambda}{1 + \lambda}$$

The quantity $\sigma_{\hat{a}_n}^2 / \sigma_y^2$ 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:

$$\hat{J} = (y_n - \hat{a})^2 + (y_{n-1} - \hat{a})^2 + (y_{n-2} - \hat{a})^2 + \cdots + (y_0 - \hat{a})^2$$

$$\hat{J} = (y_n - \hat{a})^2 + \lambda (y_{n-1} - \hat{a})^2 + \lambda^2 (y_{n-2} - \hat{a})^2 + \cdots + \lambda^n (y_0 - \hat{a})^2$$

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:

$$n = \frac{\sum n \lambda^n}{\sum \lambda^n} = \frac{\lambda}{1 - \lambda}$$

This expression is only a guideline and other possibilities exist. For example, one can define $n$ to be the effective time constant of the filter:

$$\lambda^n = \epsilon \Rightarrow n = \frac{\ln \epsilon}{\ln \lambda} \approx \frac{\ln(\epsilon^{-1})}{1 - \lambda} , \quad \text{for } \lambda \leq 1$$

where $\epsilon$ 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:

$$\frac{1}{N} \sigma_{\hat{a}_n}^2 = \frac{1 - \lambda}{1 + \lambda} \sigma_y^2$$

$$\Rightarrow \lambda = \frac{N - 1}{N + 1} \Rightarrow \alpha = 1 - \lambda = \frac{2}{N + 1}$$

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:

$$\hat{n} = \frac{\sum n \mu_n}{\sum \mu_n}$$

(24.1.17)

where $\mu_n$ is the filter’s impulse response. Eq. (24.1.17) may also be expressed in terms of the filter’s transfer function $H(z) = \sum \mu_n z^{-n}$ and its derivative $H'(z) = dH(z)/dz$ evaluated at DC, that is, at $z = 1$:

$$\hat{n} = - \frac{H'(z)}{H(z)} \bigg|_{z=1}$$

(24.1.18)
24.1. MEAN TRACKING

Alternatively, \( n \) 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):

\[
\hat{n} = -\frac{d}{d\omega} \arg H(\omega) \bigg|_{\omega = 0} \quad \text{(group delay at DC)} \tag{24.1.19}
\]

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 \( h_n = \frac{1}{N} \delta_n \), where \( N = 1, 2, \ldots, N \).

\[
\hat{n} = 1 - \frac{\sum_n n}{N} = \frac{N - 1}{2}
\]

The FIR averager can be mapped into an “equivalent” exponential smoother by equating the \( \hat{n} \) lags of the two filters, that is,

\[
\hat{n} = N - 1 \times \frac{1 - \lambda}{2} \tag{24.1.20}
\]

This condition is exactly equivalent to condition (24.1.16) arising from matching the NRRs of the two filters. The two equations,

\[
E[(\hat{\alpha}_n - m)^2] = \frac{1 - \lambda}{1 + \lambda} \sigma_y^2 = \frac{1}{N} \sigma_y^2, \quad \hat{n} = \frac{\lambda}{1 - \lambda} = \frac{N - 1}{2}
\tag{24.1.21}
\]

capture the main tradeoff between variance and speed in using an exponential smoother or an equivalent FIR averager, that is, the closer \( \lambda \) 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),

\[
\hat{\alpha}_n = \lambda - \lambda^{n+1} \hat{\alpha}_n - 1 + \frac{\alpha}{1 - \lambda^{n+1}} y_n = \hat{\alpha}_n - 1 + \frac{\alpha}{1 - \lambda^{n+1}} (y_n - \hat{\alpha}_n) \tag{24.1.22}
\]

\[
\hat{\alpha}_n = \lambda \hat{\alpha}_n - 1 + \alpha y_n = \hat{\alpha}_n - 1 + \alpha (y_n - \hat{\alpha}_n)
\]

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 \( \hat{\alpha}_1 \). For the exact smoother, \( \hat{\alpha}_1 \) can have an arbitrary value because its coefficient vanishes at \( n = 0 \). This gives for the first smoothed value \( \hat{\alpha}_0 = 0 \cdot \hat{\alpha}_1 - 1 \cdot y_0 = y_0 \). For the steady smoother it would make sense to also require that \( \hat{\alpha}_0 = y_0 \), which would imply that \( \hat{\alpha}_1 = y_0 \).

\[
\hat{\alpha}_0 = \lambda \hat{\alpha}_1 - 1 + \alpha y_0 = \lambda y_0 + \alpha y_0 = y_0
\]

There are other reasonable ways of choosing \( \hat{\alpha}_1 \), 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:

\[
\hat{\alpha}_n = \alpha \sum_{k=0}^{n} \lambda^{n-k} y_k + \lambda^{n+1} \hat{\alpha}_1 \tag{24.1.23}
\]

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.](image)

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 \( \lambda_n = 1/(n+1) \) and initialized at \( \hat{\alpha}_1 = 0 \) (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 \( \hat{\alpha}_1 = 0 \) with the two values of the forgetting factor \( \lambda = 0.98 \) and \( \lambda = 0.995 \). For the smaller \( \lambda \)
24. EXPONENTIAL MOVING AVERAGE FILTERS

24.1. MEAN TRACKING

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 \( \lambda \) 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 (DJI) 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:

```matlab
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,':'); % discard first N-1 outputs

N = round((1+la)/(1-la));
h = ones(N,1)/N; % FIR averager
x = filter(h,1,y); % FIR averager
figure; plot(n(N:end),x(N:end),'-', n,y,':');
```

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 \( s_0 = \lambda y_0 \). 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:

\[
\hat{a}_n = s + \alpha y_n \\
\hat{a}_0 = s_0 = \lambda y_0 \\
\]

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 \( \bar{\tau} = (\lambda + 1)/\lambda (1-\lambda) \).

**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,

- \( 0 \leq n < 75 \) \( s(n) = 0 \)
- \( 75 \leq n < 225 \) \( s(n) = 20 \)
- \( 225 \leq n < 300 \) \( s(n) = 80 \)

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 \( \bar{\tau} \) of Eq. (24.1.20).

The delay \( \bar{\tau} \) 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, \( \bar{\tau} = 0 \). 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:

\[
y_n = \sum_{k=0}^{n} h_k x_{n-k} = \sum_{k=0}^{n} h_k [a + b(n-k)] = (a + bn) \sum_{k=0}^{n} h_k - b \sum_{k=0}^{n} k h_k
\]
24. EXPONENTIAL MOVING AVERAGE FILTERS

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:

\[ x_{n+1} = x_n + w_n \]
\[ y_n = x_n + v_n \]  

(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 \( \hat{x}_{n-1} \) 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 \( \alpha \) and \( q = \frac{\alpha^2}{1 - \alpha^2} \):

\[ q = \frac{\alpha^2}{1 - \alpha^2} \Rightarrow \alpha = \sqrt{q^2 + 4q^2 - q} \]  

(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 \( \{y_{n-M}, \ldots, y_n, \ldots, y_{n+M}\} \), the performance index was

\[ J = \sum_{k=M}^{k=M} [y_{n-k} - p(k)]^2 = \sum_{k=M}^{k=M} [y_{n-k} - u_k^T c]^2 = \min \]

where \( p(k) \) is a \( d \)th degree polynomial, representing the estimate \( \hat{y}_{n+k} = p(k) \),

\[ p(k) = u_k^T c = [1, k, \ldots, k^d] \]

and we defined the monomial basis vector \( u_k = [1, k, k^2, \ldots, k^d]^T \). The higher-order exponential smoother is obtained by restricting the data range to \( \{y_0, y_1, \ldots, y_n\} \) and using exponential weights, and similarly, the corresponding FIR version will be restricted to \( \{y_{n-N}, \ldots, y_n, y_{n+1}\} \). The resulting performance indices are then,

\[ J_n = \sum_{k=n}^{k=n} \lambda^{-k} [y_{n+k} - u_k^T c]^2 = \min \]
\[ J_n = \sum_{k=n-N+1}^{k=n} [y_{n+k} - u_k^T c]^2 = \min \]  

(24.3.1)

or, replacing the summation index \( k \) by \( -k \), the performance indices read,

\[ (EMA) \quad J_n = \sum_{k=0}^{k=n} \lambda^k [y_{n-k} - u_k^T c]^2 = \min \]  
\[ (FIR) \quad J_n = \sum_{k=0}^{k=n} [y_{n-k} - u_k^T c]^2 = \min \]  

(24.3.1)
In both cases, we may interpret the quantities \( p(\tau) = u^T c \) as the estimates \( \hat{y}_{n+\tau} \).
We will denote them by \( \hat{y}_{n+\tau/n} \) to emphasize their causal dependence only on data up to the current time \( n \). In particular, the quantity \( c_0 = u^T c = p(0) \) represents the estimate \( \hat{y}_n \), or \( \hat{y}_{n/n} \), that is, an estimate of the local level of the signal. Similarly, \( c_1 = p(0) = u^T c \) represents the local slope, and \( 2c_2 = p(0) \), 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:

\[
\mathcal{F} = (y - Se)^T W (y - Sc) = \min c = (S^T W S)^{-1} S^T W y
\] (24.3.2)

where the data vector \( y \) is defined as follows in the EMA and FIR cases,

\[
y(n) = \begin{bmatrix} y_n \\ y_{n-1} \\ \vdots \\ y_0 \end{bmatrix}, \quad \hat{y}(n) = \begin{bmatrix} y_n \\ y_{n-1} \\ \vdots \\ y_{n-N+1} \end{bmatrix}
\] (24.3.3)

with the polynomial basis matrices \( S \),

\[
S_n = [u_0, u_{-1}, \ldots, u_{-n}]^T, \quad S_N = [u_0, u_{-1}, \ldots, u_{-N+1}]^T
\] (24.3.4)

where the response coefficient:

\[
\hat{y}_{n+\tau/n} = u^T c(n) = h^T(n) y(n)
\] (24.3.6)

with \( u^T_k = [1, (-k), (-k)^2, \ldots, (-k)^d] \), and weight matrices \( W \) in the two cases,

\[
W_n = \text{diag}([1, \lambda, \ldots, \lambda^n]), \quad \text{or, } W = I_N
\] (24.3.5)

The exact solutions for 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,

\[
h_{\tau}(k) = \frac{2(N-1) (2N-1 - 3k) + 6(N-1 - 2k) \tau}{N(N^2 - 1)}, \quad k = 0, 1, \ldots, N - 1
\] (24.4.1)

A direct derivation of (24.4.1) is as follows. From the definition (24.3.4), we find:

\[
S_N^T S_N = \sum_{k=0}^{N-1} u_k u_k^T = \sum_{k=0}^{N-1} \begin{bmatrix} 1 \\ -k \end{bmatrix} \begin{bmatrix} 1 & -k \\ -k & k^2 \end{bmatrix}
\]

\[
= \begin{bmatrix} N & -N(N-1)/2 \\ -N(N-1)/2 & N(N-1)(2N-1)/6 \end{bmatrix}
\] (24.4.2)

then, from Eq. (24.3.8), because the \( k \)th row of \( S_N \) is \( u_k^T \), we obtain the \( k \)th impulse response coefficient:

\[
h_{\tau}(k) = u_{-k}^T (S_N^T S_N)^{-1} u_{\tau} = \frac{2}{N(N^2 - 1)} \begin{bmatrix} (N-1)(2N-1) \\ 3(N-1) \end{bmatrix} \begin{bmatrix} 1 \\ -k \end{bmatrix}
\]

which leads to Eq. (24.4.1). Thus, we obtain,

\[
h_{\tau}(k) = h_{\tau}(k) + h_{\tau}(k), \quad k = 0, 1, \ldots, N - 1
\] (24.4.3)
24.4. LINEAR TREND FIR FILTERS

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\(^1\) at time \( n \), we obtain from (24.3.8),

\[
\begin{bmatrix}
    a_n \\
    b_n 
\end{bmatrix} = \begin{bmatrix}
    S_N^T S_N \\
    S_N^T y(n)
\end{bmatrix}^{-1} \begin{bmatrix}
    \sum_{k=0}^{N-1} y_{n-k} \\
    (S_N^T S_N)^{-1} \sum_{k=0}^{N-1} u_k y_{n-k}
\end{bmatrix}
\]

which is equivalent, component-wise, to the filtering equations:

\[
a_n = \sum_{k=0}^{N-1} h_a(k) y_{n-k} = \text{local level} \tag{24.4.5}
b_n = \sum_{k=0}^{N-1} h_b(k) y_{n-k} = \text{local slope}
\]

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 \( \hat{y}_{n+\tau/n} \) 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):

\[
h_1(k) = \frac{2 \cdot (2N - 2 - 3k)}{N(N - 1)}, \quad k = 0, 1, \ldots, N - 1 \tag{predictive FIR filter}
\]

The latter implies that its lag parameter \( n \) 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 \( N \):

\[
\mathcal{R} = \frac{2 \cdot (2N - 1)}{N(N + 1)} \tag{24.4.9}
\]

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,

\[
\sum_{k=0}^{N-1} P(n-k) h_{\tau}(k) = P(n+\tau), \quad \text{with } \deg(P) \leq d \tag{24.4.10}
\]

24.5 Higher-Order Exponential Smoothing

For any value of \( d \), the FIR filters \( h_{\tau} \) have length \( N \) and act on the \( N \)-dimensional data vector \( y(n) = [y_n, y_{n-1}, \ldots, y_{n-N+1}]^T \). By contrast, the exponential smoother weights \( h_{\tau}(n) \) have an ever increasing length. Therefore, it proves convenient to recast them

\(^1a, b \) are the same as the components \( c_0, c_1 \) of the vector \( c \).
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,

\[ R_n = S_n^T W_n S_n = \sum_{k=0}^{n} \lambda^k u_{-k} u_{-k}^T = (d+1) \times (d+1) \text{ matrix} \]

\[ r_n = S_n^T W_n y(n) = \sum_{k=0}^{n} \lambda^k u_{-k} y_{-k} = (d+1) \times 1 \text{ vector} \] (24.5.1)

Then, the optimal polynomial coefficients (24.3.7) are:

\[ c(n) = R_n^{-1} r_n \] (24.5.2)

Clearly, the invertibility of \( R_n \) requires that \( n \geq d \), which we will assume from now
on. The sought recursions relate \( c(n) \) to the optimal coefficients \( c(n-1) = R_n^{-1} r_{n-1} \)
at the previous time instant \( n-1 \). Therefore, we must actually assume that \( n > d \).
To proceed, we note that the basis vector \( u_\tau = [1, \tau, \tau^2, \ldots, \tau^d]^T \) satisfies the time-
propagation property:

\[ u_{\tau+1} = F u_{\tau} \] (24.5.3)

where \( F \) is a \((d+1) \times (d+1)\) unit lower triangular matrix whose \( i \)th row consists of the
binomial coefficients:

\[ F_{ij} = \binom{i}{j}, \quad 0 \leq i \leq d, \quad 0 \leq j \leq i \] (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 \( d = 0, 1, 2, \):

\[ F = [1], \quad F = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 2 & 1 \end{bmatrix} \] (24.5.5)

It follows from Eq. (24.5.3) that \( u_\tau = F u_{\tau-1} \), and inverting \( u_{\tau-1} = F^{-1} u_{\tau} \). The
inverse matrix \( G = F^{-1} \) will also be unit lower triangular with nonzero matrix elements
obtained from the binomial expansion of \((\tau - 1)^i\):

\[ G_{ij} = (-1)^{i-j} \binom{i}{j}, \quad 0 \leq i \leq d, \quad 0 \leq j \leq i \] (24.5.6)

For example, we have for \( d = 0, 1, 2, \):

\[ G = [1], \quad G = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 1 & -2 & 1 \end{bmatrix} \] (24.5.7)

It follows from \( u_{\tau-1} = G u_{\tau-1} \) that \( u_{-k} = G u_{-k} \). This implies the following recursion
for \( R_n \):

\[
R_n = \sum_{k=0}^{n} \lambda^k u_{-k}^T \]

\[ = u_0 u_0^T + \lambda \sum_{k=1}^{n} \lambda^{k-1} u_{-k}^T \]

\[ = u_0 u_0^T + \lambda \sum_{k=0}^{n-1} \lambda^k u_{-k}^T \]

\[ = u_0 u_0^T + \lambda G \left( \sum_{k=0}^{n-1} \lambda^k u_{-k}^T \right) G^T = u_0 u_0^T + \lambda G R_{n-1} G^T \]

where in the third line we changed summation variables from \( k \) to \( k-1 \), and in the fourth,
we used \( u_{-k} = G u_{-k} \). Similarly, we have for \( r_n \),

\[ r_n = \sum_{k=0}^{n} \lambda^k u_{-k} y_{-k} = u_0 y_0 + \sum_{k=1}^{n} \lambda^k u_{-k} y_{-k} \]

\[ = u_0 y_0 + \lambda \sum_{k=0}^{n-1} \lambda^k u_{-k} y_{-k} \]

\[ = u_0 y_0 + \lambda G \left( \sum_{k=0}^{n-1} \lambda^k u_{-k} y_{-k-1} \right) \]

Thus, \( R_n, r_n \) satisfy the recursions:

\[ R_n = u_0 u_0^T + \lambda G R_{n-1} G^T \]

\[ r_n = u_0 y_0 + \lambda G r_{n-1} \] (24.5.8)

and they may be initialized to zero, \( R_{-1} = 0 \) and \( r_{-1} = 0 \). Using \( y_{n+1} = u_0^T c(n) \), we
may define the smoothed estimates, predictions, and the corresponding errors:

\[ \hat{y}_{n+1} = u_0^T c(n), \quad \epsilon_{n+1} = y_{n+1} - \hat{y}_{n+1} \]

\[ \hat{y}_{n+1} = u_0^T c(n), \quad \epsilon_{n+1} = y_{n+1} - \hat{y}_{n+1} \]

\[ \hat{y}_{n+1} = u_0^T c(n), \quad \epsilon_{n+1} = y_{n+1} - \hat{y}_{n+1} \] (24.5.9)

where we used \( u_1 = F u_0 \). In the language of RLS adaptive filters, we may refer to
\( \hat{y}_{n+1} \) and \( \hat{y}_{n+1} \) as the a priori and a posteriori estimates of \( y_{n+1} \), respectively. Using the recursions (24.5.8), we may now obtain a recursion for \( c(n) \). Using \( c(n-1) = R_{n-1} u_{n-1} \) and the matrix relationship \( G F = I \), we have,

\[ R_n c(n) = r_n = u_0 y_n + \lambda G R_{n-1} = u_0 y_n + \lambda G R_{n-1} c(n-1) \]

\[ = u_0 y_n + \lambda G R_{n-1} G^T F c(n-1) = u_0 y_n + (R_n - u_0 u_0^T) F c(n-1) \]

\[ = R_n F c(n-1) + u_0 (y_n - u_0^T F c(n-1)) = R_n F c(n-1) + u_0 (y_n - \hat{y}_{n-1}) \]

\[ = R_n F c(n-1) + u_0 \epsilon_{n-1} \]
where in the second line we used $\lambda GR_{n-1}G^T = R_n - u_nu_n^T$. Multiplying both sides by $R_n^{-1}$, we obtain,

$$c(n) = F^T c(n-1) + R_n^{-1} u_n e_{n/n-1}$$  \hspace{1cm} (24.5.10)

Again, in the language of RLS adaptive filters, we define the so-called a posteriori and a priori “Kalman gain” vectors $k_n$ and $k_{n/n-1}$,

$$k_n = R_n^{-1} u_0, \quad k_{n/n-1} = \lambda^{-1} F^T R_n^{-1} F u_0$$  \hspace{1cm} (24.5.11)

and the “likelihood” variables,

$$\nu_n = u_0^T k_{n/n-1} - \lambda^{-1} u_0^T F R_n^{-1} F u_0 \lambda^{-1} u_n^T R_n^{-1} u_1, \quad \mu_n = \frac{1}{1 + \nu_n}$$  \hspace{1cm} (24.5.12)

Starting with the recursion $R_n = u_0 u_0^T + \lambda GR_{n-1}G^T$ and multiplying both sides by $R_n^{-1}$ from the left, then by $F^T$ from the right, then by $R_n^{-1}$ from the left, and then by $F$ from the right, we may easily derive the equivalent relationship:

$$\lambda^{-1} F^T R_n^{-1} F = R_n^{-1} u_0 \lambda^{-1} u_0^T F R_n^{-1} F + R_n^{-1}$$  \hspace{1cm} (24.5.13)

Multiplying on the right by $u_0$ and using the definitions (24.5.11), we find

$$k_{n/n-1} = k_n \nu_n + k_n = (1 + \nu_n) k_n, \quad \text{or,}$$

$$k_n = \mu_n k_{n/n-1}$$  \hspace{1cm} (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:

$$R_n^{-1} = \lambda^{-1} F^T R_{n-1}^{-1} F - \mu_n k_{n/n-1} k_{n/n-1}^T$$  \hspace{1cm} (24.5.15)

This also implies that the parameter $\mu_n$ can be expressed as

$$\mu_n = 1 - u_0^T R_n^{-1} u_0 = 1 - u_0^T k_n$$  \hspace{1cm} (24.5.16)

The a priori and a posteriori errors are also proportional to each other. Using (24.5.16), we find,

$$\hat{y}_{n/n} = u_0^T c(n) = u_0^T (F^T c(n-1) + k_n e_{n/n-1}) = \hat{y}_{n/n-1} + (1 - \mu_n) e_{n/n-1} = y_n - \mu_n e_{n/n-1}$$

which implies that

$$e_{n/n} = \mu_n e_{n/n-1}$$  \hspace{1cm} (24.5.17)

The coefficient updates (24.5.10) may now be expressed as:

$$c(n) = F^T c(n-1) + k_n e_{n/n-1}$$  \hspace{1cm} (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 $n > d$.

For $0 \leq n \leq d$, the fitting may be done with polynomials of varying degree $d_n = n$, and the coefficient estimate computed by explicit matrix inversion, $c(n) = R_n^{-1} y_n$. The above computational steps and initialization have been incorporated into the MATLAB function `ema` with usage:

\[
\text{C = ema(y,d,lambda)}; \quad \% \text{exponential moving average - exact version}
\]

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 $c(n)^T$. 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 $\{y_0, y_1, y_2, \ldots\}$ and the $d = 1$ smoother. At $n = 0$, we use a smoother of order $d_0 = 0$, constructing the quantities $R_0, r_0$ using the definition (24.5.1):

$$R_0 = [1], \quad r_0 = [y_0] \Rightarrow c(0) = R_0^{-1} r_0 = y_0$$

Next, at $n = 1$ we use a $d_1 = 1$ smoother, and definition (24.5.1) now implies,

$$R_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + \lambda \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 + \lambda & -\lambda \\ -\lambda & \lambda \end{bmatrix} \Rightarrow c(1) = R_1^{-1} r_1 = \begin{bmatrix} y_1 \\ y_1 - y_0 \end{bmatrix}$$

$$r_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} y_1 + \lambda \begin{bmatrix} 1 \\ -1 \end{bmatrix} y_0 = \begin{bmatrix} y_1 + \lambda y_0 \\ -\lambda y_0 \end{bmatrix}$$

Starting with $R_1, r_1$, the recursion (24.5.8) may then be continued for $n \geq d + 1 = 2$. If we had instead $d = 2$, then there is one more initialization step, giving

\[
R_2 = \begin{bmatrix} 1 + \lambda & \lambda & +2 \lambda^2 \\ -\lambda - 2\lambda^2 & -\lambda + 4\lambda^2 & \lambda + 4\lambda^2 \\ -\lambda + 2\lambda^2 & -\lambda - 8\lambda^2 & \lambda - 16\lambda^2 \end{bmatrix}, \quad r_2 = \begin{bmatrix} y_2 + \lambda y_1 + \lambda^2 y_0 \\ -\lambda y_1 - 2\lambda^2 y_0 \\ \lambda y_1 + 4\lambda^2 y_0 \end{bmatrix}
\]

resulting in

$$c(2) = R_2^{-1} r_2 = \begin{bmatrix} 1.5 y_2 - 2 y_1 + 0.5 y_0 \\ 0.5 y_2 - y_1 + 0.5 y_0 \\ -y_1 + 0.5 y_0 \end{bmatrix}$$  \hspace{1cm} (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 \( h_n(k) \) 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 \( \text{upulse} \) which is a part of the OSP toolbox that generates a unit-pulse of prescribed duration.

```matlab
n = 0:300;
s = (20 + 0.8*n) .* upulse(n,75) + ... 
    (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
```

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.

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:

\[
\mathbf{c}(n) = [a_n, b_n, c_n] \quad \text{and} \quad \mathbf{c}(n) = \left[ \begin{array}{c} a_n \\ b_n \\ c_n \end{array} \right] \]

(24.6.1)

Then, with \( \mathbf{u}_T = [1] \), \( \mathbf{u}_T = [1, \tau]^T \), and \( \mathbf{u}_T = [1, \tau, \tau^2]^T \), the implied predicted estimates will be for arbitrary \( \tau \):

\[
\hat{y}_{n+\tau|n} = \mathbf{u}_T^T \mathbf{c}(n) = a_n \\
\hat{y}_{n+\tau|n} = \mathbf{u}_T^T \mathbf{c}(n) = a_n + b_n \tau \\
\hat{y}_{n+\tau|n} = \mathbf{u}_T^T \mathbf{c}(n) = a_n + b_n \tau + c_n \tau^2
\]

(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,

\[
\hat{y}_{n|n-1} = u_1^T c(n-1) = a_{n-1} \\
\hat{y}_{n|n-1} = u_1^T c(n-1) = a_{n-1} + b_{n-1} \\
\hat{y}_{n+1|n} = u_1^T c(n-1) = a_{n-1} + b_{n-1} + c_{n-1}
\]

(24.6.3)

Denoting the a posteriori gains \(k_n\) by,

\[
k_n = [\alpha(n)], \quad k_n = \begin{bmatrix} \alpha_1(n) \\ \alpha_2(n) \end{bmatrix}, \quad k_n = \begin{bmatrix} \alpha_1(n) \\ \alpha_2(n) \end{bmatrix}
\]

(24.6.4)

then, the coefficient updates (24.5.18) take the forms, where \(e_{n|n-1} = y_n - \hat{y}_{n|n-1}\),

\[
a_n = a_{n-1} + \alpha(n) e_{n|n-1}
\]

(24.6.5)

Since \(k_n = R_n^{-1} u_0\), the gains depend only on \(\lambda\) and \(n\) and converge to steady-state values for large \(n\). For example, for \(d = 0\), we have,

\[
R_n = \sum_{k=0}^{n} \lambda^k = \frac{1 - \lambda^{n+1}}{1 - \lambda} \Rightarrow k_n = R_n^{-1} = \frac{1 - \lambda}{1 - \lambda^{n+1}} \Rightarrow 1 - \lambda \equiv \alpha
\]

Thus, the steady-state form of the \(d = 0\) EMA smoother is as expected:

\[
e_{n|n-1} = y_n - \hat{y}_{n|n-1} = y_n - (a_{n-1} + b_{n-1})
\]

(24.6.6) (single EMA, \(d = 0\))

initialized as usual at \(a_{-1} = y_0\). The corresponding likelihood variable \(\nu_n = 1 - u_0^T k_n\) tends to \(\mu = 1 - (1 - \lambda) = \lambda\). Similarly, we find for \(d = 1\),

\[
R_n = \sum_{k=0}^{n} \lambda^k \begin{bmatrix} 1 - k \\ 1 \end{bmatrix} = \sum_{k=0}^{n} \lambda^k \begin{bmatrix} 1 - k \\ -k \end{bmatrix} = \begin{bmatrix} R_{00}(n) \\ R_{01}(n) \\ R_{10}(n) \\ R_{11}(n) \end{bmatrix}
\]

where

\[
R_{00}(n) = \frac{1 - \lambda^{n+1}}{1 - \lambda}, \quad R_{01}(n) = R_{10}(n) = -\frac{1 + n(1 - \lambda) + n^2(1 - \lambda)^2}{(1 - \lambda)^2}
\]

(24.6.7)

which have the limit as \(n \to \infty\),

\[
R_n \to R = \begin{bmatrix} (1 - \lambda^2) & -\lambda(1 - \lambda) \\ -\lambda(1 - \lambda) & \lambda(1 + \lambda) \end{bmatrix}
\]

\[
R^{-1} = \begin{bmatrix} 1 - \lambda^2 & (1 - \lambda)^2 \\ (1 - \lambda)^2 & \lambda^2(1 - \lambda)^3 \end{bmatrix}
\]

It follows that the asymptotic gain vector \(k = R^{-1} u_0\) will be the first column of \(R^{-1}\):

\[
k_n \to k = \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} 1 - \lambda^2 \\ (1 - \lambda)^2 \end{bmatrix}
\]

(24.6.8)

and the steady-state version of the \(d = 1\) EMA smoother becomes:

\[
e_{n|n-1} = y_n - \hat{y}_{n|n-1} = y_n - (a_{n-1} + b_{n-1})
\]

(24.6.9) (double EMA, \(d = 1\))

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 \(\mu = 1 - u_0^T k\) \(= 1 - (1 - \lambda^2) = \lambda^2\). The difference equation may be initialized at \(a_{-1} = y_0\) 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 \(c(1)\) as the exact smoother:

\[
\begin{bmatrix} a_0 \\ b_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a_{-1} \\ b_{-1} \end{bmatrix} + \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} \begin{bmatrix} e_0 \\ -1 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 - y_0 \end{bmatrix}
\]

\[
\begin{bmatrix} a_1 \\ b_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ b_0 \end{bmatrix} + \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} \begin{bmatrix} e_1 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_1 - y_0 \end{bmatrix}
\]

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 is

\[
R = \sum_{k=0}^{\infty} \lambda^k u_k u_k^T = \sum_{k=0}^{\infty} \lambda^k \begin{bmatrix} 1 \\ -k \\ -k \end{bmatrix} = \begin{bmatrix} 1 \\ -k \\ -k \end{bmatrix} \begin{bmatrix} 1 \\ -k \\ -k \end{bmatrix}
\]

which may be summed to

\[
R = \begin{bmatrix} 1 & \lambda & \lambda(1 + \lambda) \\ \lambda(1 - \lambda) & \lambda^2(1 - \lambda)^2 & \lambda(1 + 4\lambda + \lambda^2) \\ \lambda^2(1 - \lambda)^3 & \lambda^2(1 - \lambda)^4 & \lambda^2(1 + 10\lambda + \lambda^2) \end{bmatrix}
\]
24.6. STEADY-STATE EXPONENTIAL SMOOTHING

with an inverse

$$R^{-1} = \begin{bmatrix}
1 - \lambda^3 & \frac{3}{2} (1 + \lambda)(1 - \lambda)^2 & \frac{1}{2} (1 - \lambda)^3 \\
\frac{3}{2} (1 + \lambda)(1 - \lambda)^2 & (1 + \lambda)(1 - \lambda)^3 (1 + 9\lambda) & (1 - \lambda)^4 (1 + 3\lambda) \\
\frac{1}{2} (1 - \lambda)^3 & \frac{1}{4\lambda^2} & \frac{1}{2}\lambda^3
\end{bmatrix}
$$

The asymptotic gain vector \( \mathbf{k} = R^{-1}\mathbf{u}_0 \) and \( \mu \) parameter are,

$$\mathbf{k} = \begin{bmatrix}
\alpha_1 \\
\alpha_2 \\
\alpha_3
\end{bmatrix}
= \begin{bmatrix}
1 - \lambda^3 \\
\frac{3}{2} (1 + \lambda)(1 - \lambda)^2 \\
\frac{1}{2} (1 - \lambda)^3
\end{bmatrix}, \quad \mu = 1 - \alpha_1 = \lambda^3 \tag{24.6.10}
$$

and the steady-state \( d = 2 \) EMA smoother becomes:

$$\hat{y}_{n+1} = y_n - \hat{y}_n = y_n - (a_{n-1} + b_{n-1} + c_{n-1})$$

\[
\begin{bmatrix}
ap_n \\
b_p \\
c_n
\end{bmatrix} = \begin{bmatrix}1 & 1 & 1 \\0 & 1 & 2 \\0 & 0 & 1\end{bmatrix}
\begin{bmatrix}a_{n-1} \\b_{n-1} \\c_{n-1}\end{bmatrix} + \begin{bmatrix}\alpha_1 \\\alpha_2 \\\alpha_3\end{bmatrix} \epsilon_{n-1}
\] (triple EMA, \( d = 2 \)) \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:

$$\begin{bmatrix}a_{-1} \\
b_{-1} \\
c_{-1}
\end{bmatrix} = \begin{bmatrix}y_2 - 3y_1 + 3y_0 \\\-1.5y_2 + 4y_1 - 2.5y_0 \\0.5y_2 - y_1 + 0.5y_0
\end{bmatrix} \Rightarrow \begin{bmatrix}a_2 \\b_2 \\c_2
\end{bmatrix} = \begin{bmatrix}y_2 \\1.5y_2 - 2y_1 + 0.5y_0 \\0.5y_2 - y_1 + 0.5y_0
\end{bmatrix}
$$

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:

$$R = \mathbf{u}_0\mathbf{u}_0^T + \lambda GRG^T \tag{24.6.12}$$

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 \( \mathbf{u}_{-k} = G\mathbf{u}_{-k} \). Multiplying from the left by \( F \), and noting that \( F\mathbf{u}_0 = \mathbf{u}_1 \), we have

$$FR = \mathbf{u}_1\mathbf{u}_1^T + \lambda GRG^T \tag{24.6.13}$$

Taking advantage of the unit-lower-triangular nature of \( F \) and \( G \), this equation can be written component-wise as follows:

$$\sum_{k=0}^{i} F_{ik}R_{kj} = \mathbf{u}_i(1)\mathbf{u}_0(j) + \lambda \sum_{k=0}^{j} R_{ik}G_{kj}, \quad 0 \leq i, j \leq d \tag{24.6.14}$$

Noting that \( u_1(1) = 1 \) and \( u_0(j) = \delta(j) \), and setting first \( i = j = 0 \), we find

$$R_{00} = 1 + \lambda R_{00} \Rightarrow R_{00} = \frac{1}{1 - \lambda} \tag{24.6.15}$$

Then, setting \( i = 0 \) and \( 1 \leq j \leq d \),

$$R_{0j} = \lambda \sum_{k=0}^{j-1} R_{ik}G_{jk} + \lambda \sum_{k=0}^{j-1} R_{0k}G_{jk} \tag{24.6.16}$$

which can be solved recursively for \( R_{0j} \):

$$R_{0j} = R_{j0} = \frac{\lambda}{1 - \lambda} \sum_{k=0}^{j-1} R_{0k}G_{jk}, \quad j = 1, 2, \ldots, d \tag{24.6.16}$$

Next, take \( i \geq 1 \) and \( j \geq i \), 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 \( i = 1, 2, \ldots, d, \quad j = i, i+1, \ldots, d \),

$$R_{ij} = R_{ji} = 1 - \lambda \left[ \sum_{k=0}^{j-1} R_{ik}G_{jk} - \sum_{k=0}^{i-1} F_{ik}R_{kj} \right] \tag{24.6.17}$$

To clarify the computations, we give the MATLAB code below:

\[
\begin{align*}
R(1,2) &= 1/(1-\lambda); \\
R(j,1) &= \lambda R(j,1) + \lambda \sum_{k=0}^{j-1} R_{ik}G_{jk}; \\
end
\end{align*}
\]

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 \( n \geq 0 \),

\[
\hat{y}_{n+1} = \mathbf{u}_1^T \mathbf{c}(n - 1) \\
\epsilon_{n+1} = y_n - \hat{y}_{n+1} \\
\mathbf{c}(n) = F^T \mathbf{c}(n - 1) + \mathbf{k} \epsilon_{n+1} 
\]

(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):

\[
\mathbf{C}(z) = z^{-1} F^T \mathbf{C}(z) + \mathbf{k} \left( Y(z) - z^{-1} \mathbf{u}_1^T \mathbf{C}(z) \right), \quad \text{or,}
\]
The computational steps (24.6.18) have been incorporated in the MATLAB function stema, with usage,

\[
\text{C = stema(y,d,lambda,cinit);} \quad \% \text{steady-state exponential moving average}
\]

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 \(c(-1)\). 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 \(\lambda\), 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:

\[
cinit = \text{lpbasis}(L,d-1) \backslash y(1:L); \quad \% \text{fit order-}d \text{ polynomial to first} \ L \text{ inputs}
\]

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 \(y\). If not, another, shorter initialization possibilities for \(c_{\text{init}}\) are summarized in the help file for stema.

To clarify the fitting operation, we note that fitting the first \(L\) sample \(y_n, n = 0,1,\ldots,L - 1\), to a polynomial of degree \(d\) centered at \(n = -1\) amounts to the minimization of the performance index:

\[
J = \sum_{n=0}^{L-1} (y_n - p_n)^2 = \min \quad p_n = \sum_{i=0}^{d} (n+1)^i c_i = u_{n+1}^T c
\]

which can be written compactly as

\[
J = \| y - Sc \|^2 = \min \quad S = [u_1, u_2, \ldots, u_{n-1}, \ldots, u_L]^T
\]

with solution \(c = (S^T S)^{-1} S^T y = S^\dagger y\) in MATLAB notation.\(^7\) The actual fitted values \(p = [p_0, p_1, \ldots, p_{L-1}]^T\) are then computed by \(p = Sc\).

Selecting \(n = -1\) as the centering time, assumes that the filtering operation will start at \(n = 0\) requiring therefore the value \(c(-1)\). 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 then be

\[
J = \sum_{n=0}^{L-1} (y_n - p_n)^2 = \min \quad p_n = \sum_{i=0}^{d} (n-n_0)^i c_i = u_{n-n_0}^T \tilde{c}
\]

with another set of coefficients \(\tilde{c}\). The MATLAB implementation is in this case,

\[
cinit = \text{lpbasis}(L,d-1) \backslash y(1:L); \quad \% \text{fit order-}d \text{ polynomial to first} \ L \text{ inputs}
\]

From \(u_{n+1} = Fu_n\), we obtain \(u_{n+1} = F_{n+1} u_{n-n_0}\). By requiring that the fitted polynomials be the same, \(p_n = u_{n+1}^T c = u_{n-n_0}^T \tilde{c}\), it follows that,

\[
\tilde{c} = (F^T)^{n_0+1} c \quad \text{(24.6.20)}
\]

\(^7\)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 \(H(z) = \alpha / (1 - \alpha z^{-1})\), 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 \(c(n)\) 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 \(c(n)\), much like the case of the \(d = 0\) smoother. Starting with \(c(n) = R_n^{-1} r_n\) and \(r_n = S_n^T W_n y(n)\), we obtain for the correlation matrix,

\[
E[c(n)c^T(n)] = R_n^{-1} S_n^T W_n E[y(n)y^T(n)] W_n S_n R_n^{-1}
\]

and for the corresponding covariance matrix,

\[
\Sigma_{cc} = R_n^{-1} S_n^T W_n S_{yy} W_n S_n R_n^{-1} \quad \text{(24.7.1)}
\]

Under the typical assumption that \(y_n\) is white noise, we have \(S_{yy} = \sigma_y^2 I_{n+1}\), where \(I_{n+1}\) is the \((n+1)\)-dimensional unit matrix. Then,

\[
\Sigma_{cc} = \sigma_y^2 R_n^{-1} Q_n R_n^{-1}, \quad Q_n = S_n^T W_n S_n \quad \text{(24.7.2)}
\]

In the limit \(n \to \infty\), the matrices \(R_n, Q_n\) tend to steady-state values, so that

\[
\Sigma_{cc} \approx \sigma_y^2 R^{-1} Q R^{-1} \quad \text{(24.7.3)}
\]

where the limit matrices \(R, Q\) are given by

\[
R = \sum_{k=0}^{\infty} \lambda^k u_{-k} u_{-k}^T, \quad Q = \sum_{k=0}^{\infty} \lambda^{2k} u_{-k} u_{-k}^T \quad \text{(24.7.4)}
\]

Since \(\tilde{y}_{n\mid n} = u_{-n}^T c(n)\) and \(\tilde{y}_{n\mid n+1} = u_{-n}^T c(n)\), the corresponding variances will be:

\[
\sigma_{\tilde{y}_{n\mid n}}^2 = u_{-n}^T \Sigma_{cc} u_{-n}, \quad \sigma_{\tilde{y}_{n\mid n+1}}^2 = u_{-n}^T \Sigma_{cc} u_{-n} = \sigma_{\tilde{y}_{n\mid n}}^2 \quad \text{(24.7.5)}
\]

Because \(y_n\) was assumed to be an uncorrelated sequence, the two terms in the prediction error \(e_{n\mid n+1} = y_{n+1} - \tilde{y}_{n\mid n+1}\) will be uncorrelated since \(\tilde{y}_{n\mid n+1}\) depends only on data up to \(n\). Therefore, the variance of the prediction error \(e_{n\mid n+1}\) will be:

\[
\sigma_{\tilde{e}_{n\mid n+1}}^2 = \sigma_{\tilde{y}_{n\mid n}}^2 + \sigma_{\tilde{y}_{n\mid n+1}}^2 = \sigma_{\tilde{y}_{n\mid n}}^2 + \sigma_y^2 [1 + u_{-n}^T R^{-1} Q R^{-1} u_{-n}] \quad \text{(24.7.6)}
\]

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:

\[
\sigma_y^2 = \Sigma_{cc} = \frac{1 - \lambda}{1 + \lambda} \sigma_y^2, \quad \sigma_e^2 = \sigma_y^2 + \sigma_y^2 = \frac{2}{1 + \lambda} \sigma_y^2
\]
For \( d = 1 \), we have as in Eq. (24.6.7),

\[
R = \frac{1}{(1-\lambda)^2} \begin{bmatrix}
(1-\lambda)^2 & -\lambda(1-\lambda) \\
-\lambda(1-\lambda) & \lambda(1+\lambda)
\end{bmatrix},
\]

with the \( Q \) matrix being obtained from \( R \) by replacing \( \lambda \rightarrow \lambda^2 \),

\[
Q = \frac{1}{(1-\lambda^2)^2} \begin{bmatrix}
(1-\lambda^2)^2 & -\lambda^2(1-\lambda^2) \\
-\lambda^2(1-\lambda^2) & \lambda^2(1+\lambda^2)
\end{bmatrix}
\]

It follows then that

\[
\Sigma_c = \sigma_y^2 R^{-1} Q R^{-1} = \frac{1-\lambda}{(1+\lambda)^3} \begin{bmatrix}
1 + 4\lambda + 5\lambda^2 & (1-\lambda)(1+3\lambda) \\
(1-\lambda)(1+3\lambda) & 2(1-\lambda)^2
\end{bmatrix}
\]

(24.7.7)

The diagonal entries are the variances of the level and slope signals \( a_n, b_n \):

\[
\sigma_a^2 = \frac{(1-\lambda)(1+4\lambda + 5\lambda^2)}{(1+\lambda)^3} \sigma_y^2, \quad \sigma_b^2 = \frac{2(1-\lambda)^3}{(1+\lambda)^2} \sigma_y^2
\]

(24.7.8)

For the prediction variance, we find

\[
\sigma_y^2 = \sigma_y^2 u_1^T (R^{-1} Q R^{-1}) u_1 = \frac{(1-\lambda)(1+4\lambda + 5)}{(1+\lambda)^3} \sigma_y^2
\]

(24.7.9)

which gives for the prediction error:

\[
\sigma_p^2 = \sigma_y^2 + \sigma_y^2 = \left[ 1 + \frac{(1-\lambda)(\lambda^2 + 4\lambda + 5)}{(1+\lambda)^3} \right] \sigma_y^2 = \frac{2(3+\lambda)}{(1+\lambda)^3} \sigma_y^2
\]

(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 \( \lambda_0, \lambda_1 \) denote the equivalent \( d = 0 \) and \( d = 1 \) parameters, the condition reads:

\[
\frac{2}{1+\lambda_0} = \frac{2(3+\lambda_1)}{(1+\lambda_1)^3} \Rightarrow \lambda_0 = \frac{(1+\lambda_1)^3}{3+\lambda_1} - 1
\]

(24.7.11)

Eq. (24.7.11) may also be solved for \( \lambda_1 \) in terms of \( \lambda_0 \),

\[
\lambda_1 = \frac{1}{3} D_0 + \frac{1+\lambda_0}{D_0} - 1, \quad D_0 = \sqrt[3]{27(1+\lambda_0)+\sqrt{27(1+\lambda_0)^2(26-\lambda_0)}}
\]

(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 \( \lambda_0 \) is non-negative and the NRR \( \sigma_p^2/\sigma_y^2 \) 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):

\[
\frac{2(2N_1-1)}{N_1(N_1+1)} = 1 - \frac{\lambda_0}{1+\lambda_0}
\]

which gives,

\[
\lambda_0 = \frac{N_1^2 - 3N_1 + 2}{N_1^2 + 5N_1 - 2} \quad \Rightarrow \quad N_1 = \frac{3 + 5\lambda_0 + \sqrt{33\lambda_0^2 + 50\lambda_0 + 1}}{2(1-\lambda_0)}
\]

(24.7.13)

The MATLAB function emap implements Eq. (24.7.12),

\[
\lambda_1 = \text{emap}(\lambda_0); \quad \% \text{mapping equivalent } \lambda_0 \text{ between } d = 0 \text{ and } d = 1 \text{ EMAs}
\]

The computed \( \lambda_1 \) is an increasing function of \( \lambda_0 \) and varies over \( 0.5214 \leq \lambda_1 \leq 1 \) as \( \lambda_0 \) 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 \( \gamma_n = 20 + 0.2n + 4\nu_n \), for \( 0 \leq n \leq 300 \), with zero-mean, unit-variance, white noise \( \nu_n \).

![Fig. 24.7.1 Comparison of two equivalent steady-state EMAs with equivalent zero-lag FIR.](image-url)
24.7. SMOOTHING PARAMETER SELECTION

\( t = 0.300; \ s = 20 + 0.2*t; \)
\( \text{randn('state', 10000);} \)
\( y = s + 4 \times \text{randn(size(t))}; \) 
\% noisy input

\[ \text{NL} = 100; \]
\[ \text{la0} = (\text{NL}^2-3*\text{NL}+2)/(\text{NL}^2+5*\text{NL}-2); \]
\[ \text{la1} = \text{emap}(\text{la0}); \]

\( C = \text{stem}_a(y,0,\text{la0}); \ x0 = C(:,1); \) 
\% steady EMA with \( d = 0, \ \lambda = \lambda_0 \)
\( C = \text{stem}_a(y,1,\text{la1}); \ x1 = C(:,1); \) 
\% steady EMA with \( d = 1, \ \lambda = \lambda_1 \)

\( k=0:\text{NL}-1; \ h = 2*(2*\text{NL}-1-3*k)/\text{NL}/(\text{NL}+1); \)
\% zero-lag FIR of length NL
\( xh = \text{filter}(h,1,y); \)

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 \( \lambda \) is more of an art than science. There do exist, however, criteria that determine an “optimum” value. Given the prediction \( \hat{y}_n/n \) of \( y_n \), and prediction error \( e_n/n = y_n - \hat{y}_n/n-1 \), the following criteria, to be minimized with respect to \( \lambda \), are widely used:

\[
\text{MSE} = \text{mean}(e_n/n-1)^2, \quad \text{(mean square error)}
\]
\[
\text{MAE} = \text{mean}(|e_n/n-1|), \quad \text{(mean absolute error)}
\]
\[
\text{MAPE} = \text{mean}(100|e_n/n-1/y_n|), \quad \text{(mean absolute percentage error)}
\]

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 \( \lambda \)'s and the minimum is selected. Typically, the criteria tend to underestimate the value of \( \lambda \) that is, they produce too small a \( \lambda \) to be useful for smoothing purposes. Even so, the optimum \( \lambda \) has been used successfully for forecasting applications. The MATLAB function \text{emaerr} calculates these criteria for any vector of \( \lambda \)'s and determines the optimum \( \lambda \) in that range:

\[
[\text{err}, \text{lopt}] = \text{emaerr}(y,d,\text{lambda},\text{type}); \]
\% mean error criteria

where \text{type} takes one of the string values 'mse', 'mae', 'mape' and \text{err} is the criterion evaluated at the vector \text{lambda}, and \text{lopt} is the corresponding optimum \( \lambda \).

\textbf{Example 24.4.7.2:} 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 \( \leq \lambda \) \( \leq 0.9 \). The upper-left graph shows the MSE versus \( \lambda \). 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 MAF 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:

\begin{verbatim}
Y = loadfile('dow-oct07-dec09.dat');
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 \lambda's to search
[err,lopt] = emaerr(y,d,la,'mse'); \% evaluate MSE at this range of \lambda's
figure; plot(la,err, lopt,min(err),'.'); \% upper-left graph
C = ema(y,d,lopt); yhat = C*u1; \% use lopt=0.90 for bottom-right
figure; plot(n,y,':', n,yhat,'-'); \% bottom-right graph
C = ema(y,d,la); yhat = C*u1; \% upper-right graph
figure; plot(n,y,':', n,yhat,'-'); \% upper-right graph
C = ema(y,d,la0); yhat = C*u1; \% upper-right graph
figure; plot(n,y,':', n,yhat,'-'); \% upper-right graph
\end{verbatim}

We note that the \( d = 1 \) smoother is more capable in following the signal than the \( d = 0 \) one. We plotted the forecasted value \( \hat{y}_{n+1}/n = C(n+1)u_1 \) versus \( n \). Because the output matrix \( C \) from the ema function has the \( C(n) \) as its rows, the entire vector of forecasted values can be calculated by acting by \( C \) on the unit vector \( u_1 \), that is, \( yhat = C*u_1 \). \( \square \)
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:

\[
a_n^{[1]} = \lambda a_{n-1}^{[1]} + \alpha y_n, \quad H^{[1]}(z) = H(z) = \frac{\alpha}{1 - \lambda z^{-1}}
\]  

(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:

\[
a_n^{[2]} = \lambda a_{n-1}^{[2]} + \alpha a_n^{[1]}, \quad H^{[2]}(z) = \left(\frac{\alpha}{1 - \lambda z^{-1}}\right)^2
\]

(24.8.2)

\[
a_n^{[3]} = \lambda a_{n-1}^{[3]} + \alpha a_n^{[2]}, \quad H^{[3]}(z) = \left(\frac{\alpha}{1 - \lambda z^{-1}}\right)^3
\]

(24.8.3)

Thus, the filter \(H(z)\) acts once, twice, three times, or in general \(d+1\) times, in cascade, producing the outputs,

\[
y_n \rightarrow H \rightarrow a_1^{[1]} \rightarrow H \rightarrow a_2^{[2]} \rightarrow H \rightarrow a_3^{[3]} \rightarrow \cdots \rightarrow a_d^{[d]} \rightarrow H \rightarrow a_{d+1}^{[d+1]}
\]

(24.8.4)

The transfer function and the corresponding causal impulse response from \(y_n\) to the \(r\)-th output \(a_n^{[r]}\) are, for \(r = 1, 2, \ldots, d+1\) with \(u(n)\) denoting the unit-step function:

\[
H^{[r]}(z) = \left[H(z)\right]^r = \left(\frac{\alpha}{1 - \lambda z^{-1}}\right)^r, \quad \Rightarrow h^{[r]}(n) = \alpha^r \lambda^n \frac{(n + r - 1)!}{n!(r-1)!} u(n)
\]

(24.8.5)

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 \(H(z) = \left[H_a(z), H_b(z)\right]z\)

(24.8.6)

we may obtain the transfer functions from \(y_n\) to the outputs \(a_n\) and \(b_n\):

\[
H_a(z) = \frac{(1 - \lambda)(1 + \lambda - 2\lambda z^{-1})}{(1 - \lambda z^{-1})^2}, \quad H_b(z) = \frac{(1 - \lambda)^2(1 - z^{-1})}{(1 - \lambda z^{-1})^2}
\]

(24.8.7)

It is straightforward to verify that \(H_a\) and \(H_b\) are related to \(H\) and \(H^2\) by

\[
H_a = 2H - H^2 = (1 - H)^2
\]

(local level filter)

\[
H_b = \frac{\alpha}{\lambda} (H - H^2)
\]

(local slope filter)

In the time domain this implies the following relationships between the \(a_n\), \(b_n\) signals and the cascaded outputs \(a_n^{[1]}\), \(a_n^{[2]}\):

\[
a_n = 2a_n^{[1]} - a_n^{[2]} = \text{local level}
\]

\[
b_n = \alpha \left(a_n^{[1]} - a_n^{[2]}\right) = \text{local slope}
\]

(24.8.8)

which can be written in a 2×2 matrix form:

\[
\begin{bmatrix}
a_n \\
b_n
\end{bmatrix} = \begin{bmatrix}
2 & -1 \\
\alpha / \lambda & -\alpha / \lambda
\end{bmatrix} \begin{bmatrix}
a_n^{[1]} \\
a_n^{[2]}
\end{bmatrix} \Rightarrow \begin{bmatrix}
a_n^{[1]} \\
a_n^{[2]}
\end{bmatrix} = \begin{bmatrix}
1 & -\lambda / \alpha \\
1 & -2\lambda / \alpha
\end{bmatrix} \begin{bmatrix}
a_n \\
b_n
\end{bmatrix}
\]

(24.8.9)

Similarly, for the \(d = 2\) case, the transfer functions from \(y_n\) to \(a_n, b_n, c_n\) are:

\[
H_a(z) = \frac{\alpha[1 + \lambda + \lambda^2 - 3\lambda(1 + \lambda)z^{-1} + 3\lambda^2 z^{-2}]}{(1 - \lambda z^{-1})^3}
\]

(24.8.10)

\[
H_b(z) = \frac{1}{2} \left[\alpha^2 (1 - z^{-1}) [3(1 + \lambda) - 5(\lambda + 1)z^{-1}]\right]
\]

\[
H_c(z) = \frac{1}{2} \left[\alpha^3 (1 - z^{-1})^2\right]
\]

(24.8.11)

which are related to \(H, H^2, H^3\) by the matrix transformation:

\[
\begin{bmatrix}
H_a \\
H_b \\
H_c
\end{bmatrix} = \begin{bmatrix}
1 & -\lambda / \alpha & \lambda(\lambda + 1) / \alpha^2 \\
1 & -2\lambda / \alpha & 2\lambda(2\lambda + 1) / \alpha^2 \\
1 & -3\lambda / \alpha & 3\lambda(3\lambda + 1) / \alpha^2
\end{bmatrix} \begin{bmatrix}
a_n \\
b_n \\
c_n
\end{bmatrix}
\]

(24.8.12)

implying the transformation between the outputs:

\[
\begin{bmatrix}
a_n^{[1]} \\
a_n^{[2]} \\
a_n^{[3]}
\end{bmatrix} = \begin{bmatrix}
1 & -\lambda / \alpha & \lambda(\lambda + 1) / \alpha^2 \\
1 & -2\lambda / \alpha & 2\lambda(2\lambda + 1) / \alpha^2 \\
1 & -3\lambda / \alpha & 3\lambda(3\lambda + 1) / \alpha^2
\end{bmatrix} \begin{bmatrix}
a_n \\
b_n \\
c_n
\end{bmatrix}
\]

(24.8.13)

with corresponding inverse relationships,

\[
\begin{bmatrix}
H_a \\
H_b \\
H_c
\end{bmatrix} = \frac{1}{2\alpha^2} \begin{bmatrix}
6\alpha^2 & -6\alpha^2 & 2\alpha^2 \\
\alpha(1 + 5\lambda) & -2\alpha(1 + 4\lambda) & \alpha(1 + 3\lambda) \\
\alpha^2 & -2\alpha^2 & \alpha^2
\end{bmatrix} \begin{bmatrix}
H \\
H^2 \\
H^3
\end{bmatrix}
\]

(24.8.14)

\[
\begin{bmatrix}
\alpha \\
\alpha \\
\alpha
\end{bmatrix} = \frac{1}{2\alpha^2} \begin{bmatrix}
6\alpha^2 & -6\alpha^2 & 2\alpha^2 \\
\alpha(1 + 5\lambda) & -2\alpha(1 + 4\lambda) & \alpha(1 + 3\lambda) \\
\alpha^2 & -2\alpha^2 & \alpha^2
\end{bmatrix} \begin{bmatrix}
a_n^{[1]} \\
a_n^{[2]} \\
a_n^{[3]}
\end{bmatrix}
\]

(24.8.15)

In particular, we have:

\[
H_a = 3H - 3H^2 + H^3 = 1 - (1 - H)^3
\]

(24.8.16)

and

\[
\hat{y}_{n+1} = a_n = 3a_n^{[1]} - 3a_n^{[2]} + a_n^{[3]}
\]

(24.8.17)

More generally, for an order-\(d\) polynomial EMA, we have [827],

\[
H_a = 1 - (1 - H)^{d+1}
\]

(24.8.18)

\[
\hat{y}_{n+1} = a_n = -\sum_{r=1}^{d} (d + 1 - r) a_r^{[r]} a_r^{[r+1]}
\]
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 \( Y = H y \), 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 \( e = y - \hat{y} = (I - H)y \). In the twicing procedure, the residuals are filtered through the same smoothing operation, which will smooth them further, \( \hat{e} = He = H(I - H)y \), and the result is added to the original estimate to get an improved estimate:

\[
\hat{y}_{\text{impr}} = \hat{y} + \hat{e} = [H + H(I - H)]y = [2H - H^2]y
\]

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 \( r = e - \hat{e} = (I - H)e = (I - H)^2y \), then filter it through \( H \), \( \hat{r} = Hr = H(I - H)^2y \), and add it to get the “thriced” improved estimate:

\[
\hat{y}_{\text{impr}} = \hat{y} + \hat{e} + \hat{r} = [H + H(I - H) + H(I - H)^2]y = [3H - 3H^2 + H^3]y
\]

which is Eq. (24.8.14). The process can be continued \( d \) times, resulting in,

\[
\hat{y}_{\text{impr}} = H[I + (I - H) + (I - H)^2 + \ldots + (I - H)^d]y = [1 - (I - H)^{d+1}]y
\]

Tuwing and its generalizations can be applied with benefit to any smoothing operation, for example, if we used an LPRS filter designed by \( B = lprs(N,d,s) \), the computational steps for twicing would be:

\[
\hat{y} = \text{lpfilt}(B, y) \Rightarrow e = y - \hat{y} \Rightarrow \hat{e} = \text{lpfilt}(B, e) \Rightarrow \hat{y}_{\text{impr}} = \hat{y} + \hat{e}
\]

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, \( H_{\text{impr}}(z) = 1 - (1 - H(z))^{d+1} \). In the passband, \( H \) is near unity, say \( H \approx 1 - \epsilon \), with \(|\epsilon| \ll 1\), then,

\[
H_{\text{impr}} = 1 - (1 - (1 - \epsilon))^{d+1} = 1 - \epsilon^{d+1}
\]

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,

\[
H_{\text{impr}} = 1 - (1 \mp \epsilon)^{d+1} = 1 - (1 \mp (d + 1) \epsilon) = \pm (d + 1) \epsilon
\]

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,

\[
H_{\text{impr}} = H^2(3 - 2H) = 3H^2 - 2H^3
\]

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 \( p \)th order tangency at \( H = 0 \) and \( q \)th order tangency at \( H = 1 \) is given by

\[
H_{\text{impr}} = H^{p+1} \sum_{k=0}^{q} \frac{(p+k)!}{p!k!} (1-H)^k
\]

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, \( \hat{n} = 0 \).

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:

\[
H_{\text{impr}}'(z)\big|_{z=1} = 0
\]

which implies that its lag is zero, \( \hat{n} = 0 \), 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:

\[
H(z) = \frac{1}{N} \sum_{n=0}^{N-1} z^{-n} = \frac{1}{N} \frac{1-z^{-N}}{1-z^{-1}} \Rightarrow H_a(z) = 2H(z) - H^2(z)
\]

The impulse response of \( H_a(z) \) can be shown to be, where \( 0 \leq n \leq 2(N - 1) \),

\[
h_a(n) = \left( \frac{2N-1-n}{N^2} \right) \left[ u(n)-2u(n-N)+u(n-2N+1) \right]
\]

It is straightforward to show that \( \hat{n}_a = 0 \) and that its noise-reduction ratio is

\[
R = \frac{8N^2 - 6N + 1}{3N^2}
\]

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. (25.2.2) 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.
24.10. ZERO-LAG FILTERS AND TWICING

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 \( \bar{h} = 0 \). Using these two conditions, it is straightforward to verify that the second derivative of the magnitude response at DC is given by:

\[
\frac{d^2}{d\omega^2} |H(\omega)|^2 \bigg|_{\omega=0} = 2 \Re[H''(0)] + 2|H'(0)|^2 = 2 \Re[H''(0)] = -2 \sum_{n=0}^{\infty} n^2 h_n
\]

(24.10.5)

Because \( \bar{h} = \sum_n n h_n = 0 \), 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 \( \omega \)s immediately beyond DC. This is demonstrated for example in Problem 24.7 by Eq. (24.14.1), so that,

\[
\frac{d^2}{d\omega^2} |H(\omega)|^2 \bigg|_{\omega=0} = -2 \sum_{n=0}^\infty n^2 h_n = \frac{4\lambda^2}{(1-\lambda)^2}
\]

A similar calculation yields the result,

\[
\frac{d^2}{d\omega^2} |H(\omega)|^2 \bigg|_{\omega=0} = -2 \sum_{n=0}^\infty n^2 h_n = \frac{1}{3} (N - 1) (N - 2)
\]

for the optimum zero-lag FIR filter of Eq. (24.4.4),

\[
h_a(k) = \frac{2(2N-1-3k)}{N(N+1)}, \quad k = 0, 1, \ldots, N-1
\]

(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,

\[
\frac{d}{d\omega} |H(\omega)|^2 \bigg|_{\omega=0} = 2 \Re[H'(0)] = 0
\]

(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:

\[
H_a(z) = \frac{(1-\lambda)(1+\lambda - 2\lambda z^{-1})}{(1-\lambda z^{-1})^2}
\]

\[
|H_a(\omega)|^2 = \frac{(1-\lambda)^2[1 + 2\lambda + 5\lambda^2 - 4\lambda(1 + \lambda) \cos \omega]}{[1 - 2\lambda \cos \omega + \lambda^2]^2}
\]

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.

**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:

\[
\begin{align*}
\text{w} &= \text{linspace}(0,1,1001); \quad \% \omega \text{ in units of } \pi \\
N &= 15; \quad k = (0:N-1); \quad \% N = 15 \\
h &= 2^*(2^N-1-3*k)/N(N+1); \quad \% h_a(k) \\
H2 &= \text{abs}(freqz(h,1,\text{pi}.*w)); \quad \% \text{magnitude response squared}
\end{align*}
\]

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 \( H_z(z) = 2H(z) - H^2(z) \), it raises the question as to what would be the corresponding local-slope filter \( H_s(z) \), 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:

\[
\mu_1 = \bar{h} = \sum_{n=0}^{\infty} nh(n), \quad \mu_2 = \sum_{n=0}^{\infty} n^2 h(n)
\]

(24.11.1)
### 24.11. Local Level, Local Slope, Local Acceleration Filters

In terms of these parameters, the generalization of Eq. (24.8.6) is then,

\[
\begin{align*}
H_a(z) &= 2H(z) - H^2(z) \quad \text{(local level filter)} \\
H_b(z) &= \frac{1}{\mu_1} [H(z) - H^2(z)] \quad \text{(local slope filter)}
\end{align*}
\] (24.11.2)

while the generalization of (24.8.12) is,

\[
\begin{bmatrix}
H_a(z) \\
H_b(z) \\
H_c(z)
\end{bmatrix} = \frac{1}{2\mu_1^2} \begin{bmatrix}
6\mu_1^2 \\
5\mu_1^2 \\
\mu_1
\end{bmatrix} \begin{bmatrix}
6\mu_1^2 \\
-6\mu_1^2 \\
2\mu_1^2 \\
\mu_2 + 3\mu_1^2 \\
\mu_2 + 2\mu_1^2 \\
-2\mu_1 \\
\mu_1
\end{bmatrix} \begin{bmatrix}
H(z) \\
H^2(z) \\
H^3(z)
\end{bmatrix}
\] (24.11.3)

and in particular,

\[
H_a = 3H - 3H^2 + H^3 = 1 - (1 - H)^3 \quad \text{(24.11.4)}
\]

For an EMA filter, \(h(n) = (1 - \lambda)\lambda^n u(n)\), we have,

\[
\mu_1 = \frac{\lambda}{1 - \lambda}, \quad \mu_2 = \frac{(1 + \lambda)}{(1 - \lambda)^2}
\]

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,

\[
a_n = a + bn, \quad b_n = b
\]

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,

\[
\begin{align*}
a_n^{[1]} &= a + b(n - \mu_1) = (a - \mu_1 b) + bn = a + bn - \mu_1 b = a_n - \mu_1 b_n \\
a_n^{[2]} &= (a - \mu_1 b - \mu_1 b) + bn = (a - 2\mu_1 b) + bn = a_n - 2\mu_1 b_n
\end{align*}
\]

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),

\[
\begin{align*}
a_n &= 2a_n^{[1]} - a_n^{[2]} \\
b_n &= \frac{1}{\mu_1} (a_n^{[1]} - a_n^{[2]})
\end{align*}
\]

For the thrice rising 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,\(^1\)

\[
a_n = a + bn + cn^2, \quad b_n = b + 2cn, \quad c_n = c
\]

\(^1\)true acceleration would be represented by \(2c\).

Upon passing through the first stage of \(h(n)\), the output will be,

\[
a_n^{[1]} = \sum_k [a + b(n - k) + c(n - k)^2] h(k)
\]

\[
= \sum_k [a + b(n - k) + c(n^2 - 2nk + k^2)] h(k)
\]

\[
= a + b(n - \mu_1) + c(n^2 - 2\mu_1 n + \mu_2)
\]

\[
= (a - b\mu_1 + c\mu_2) + (b - 2c\mu_1) n + cn^2
\]

and applying the same formula to the second and third stages of \(h(n)\), we find the outputs,

\[
a_n^{[1]} = a - b\mu_1 + c\mu_2 + (b - 2c\mu_1) n + cn^2
\]

\[
a_n^{[2]} = a - 2b\mu_1 + 3c\mu_2 + 2c\mu^2 + (b - 4c\mu_1) n + cn^2
\]

\[
a_n^{[3]} = a - 3b\mu_1 + 3c\mu_2 + 6c\mu^2 + (b - 6c\mu_1) n + cn^2
\]

which can be re-expressed in terms of the \(a_n, b_n, c_n\) signals, as

\[
\begin{align*}
a_n^{[1]} &= a_n - \mu_1 b_n + \mu_2 c_n \\
a_n^{[2]} &= a_n - 2\mu_1 b_n + 2\mu_2 c_n \\
a_n^{[3]} &= a_n - 3\mu_1 b_n + 3\mu_2 c_n
\end{align*}
\]

Solving these for \(a_n, b_n, c_n\) leads to the time-domain equivalent of Eq. (24.11.3),

\[
\begin{bmatrix}
a_n \\
b_n \\
c_n
\end{bmatrix} = \frac{1}{2\mu_1^3} \begin{bmatrix}
6\mu_1^2 \\
5\mu_1^2 \\
\mu_1
\end{bmatrix} \begin{bmatrix}
6\mu_1^2 \\
-6\mu_1^2 \\
2\mu_1^2 \\
\mu_2 + 3\mu_1^2 \\
\mu_2 + 2\mu_1^2 \\
-2\mu_1 \\
\mu_1
\end{bmatrix} \begin{bmatrix}
a_n^{[1]} \\
a_n^{[2]} \\
a_n^{[3]}
\end{bmatrix}
\]

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 \(c(n) = [c_0(n), c_1(n), \ldots, c_d(n)]^T\) basis and the cascaded basis \(a(n) = [a_n^{[1]}, a_n^{[2]}, \ldots, a_n^{[d+1]}]^T\) can be written in the general form:

\[
a(n) = Mc(n) \Rightarrow a_n^{[r]} = \sum_{i=0}^d M_{ri} c_i(n), \quad r = 1, 2, \ldots, d+1 \quad (24.12.1)
\]

The matrix elements \(M_{ri}\) can be found by looking at the special case when the input is a polynomial of degree \(d\),

\[
X_{n+r} = \sum_{i=0}^d t^i c_i(n)
\]
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,

$$M_{r1} = \sum_{k=0}^{\infty} h^{[r]}(k)(-k)^i = \sum_{k=0}^{\infty} \alpha^r \lambda^k \frac{(k + r - 1)!}{k!(r - 1)!} (-k)^i \quad (24.12.2)$$

with $1 \leq r \leq d + 1$ and $0 \leq i \leq d$. 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,

$$M = \text{emat}(d,\lambda); \quad % \text{polynomial to cascaded basis transformation matrix}$$

One of the uses of this matrix is to determine the initial condition vector in the $a_n^{[r]}$ basis, $a_{\text{init}} = MC_{\text{init}}$, where $C_{\text{init}}$ 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 $a_n^{[r]}$:

$$[a, A] = \text{mema}(y, d, 1a, a_{\text{init}}); \quad % \text{multiple exponential moving average}$$

where $A$ is an $N \times (d + 1)$ matrix whose $n$th row is $a(n)^T = [a_1^{[1]}, a_2^{[1]}, \ldots, a_{d+1}^{[1]}]$, and $a$ is the $a_n$ output of Eq. (24.8.17). The $(d+1) \times 1$ vector $a_{\text{init}}$ represents the initial values of $a(n)$, that is, $a_{\text{init}} = [d_{\text{init}1}, d_{\text{init}2}, \ldots, d_{\text{init}d+1}]^T$. If the argument $a_{\text{init}}$ is omitted, it defaults to the following least-squares fitting procedure:

$$L = \text{round}(1+1\alpha)/(1-\alpha); \quad % \text{effective length of single EMA}$$
$$c_{\text{init}} = \text{lpbasis}(L, d+1, y(1:L)); \quad % \text{fit order-}d \text{ polynomial to first } L \text{ inputs}$$
$$M = \text{emat}(d, 1); \quad % \text{transformation matrix to cascaded basis}$$
$$a_{\text{init}} = Mc_{\text{init}}; \quad % (d+1) \times 1 \text{ initial vector}$$

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 $a_0^{[0]} = y_n$,

$$a_n^{[r]} = \text{filter}(\alpha, [1, -\lambda], a_{n-1}^{[r-1]}, \lambda a_{n-1}^{[r-1]}), \quad r = 1, 2, \ldots, d + 1 \quad (24.12.3)$$

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:

$$a_n^{[r]} = \lambda a_n^{[r-1]} + \alpha a_{n-1}^{[r-1]}, \quad r = 1, 2, \ldots, d + 1 \quad (24.12.4)$$

**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 $y$ 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 $c_{\text{init}}, a_{\text{init}}$ were determined at time $n = -1$ and processing began at $n = 0$. In the right graphs, the $c_{\text{init}}, a_{\text{init}}$ 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 \{ $y_n, 0 \leq n \leq 4$ \} were used to determine the initialization vectors $c_{\text{init}}$, which were then mapped into $a_{\text{init}} = MC_{\text{init}}$. 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.](image-url)
Alternatively, $c_{\text{init}}$ can be computed from $c_{\text{init}} = \text{lpbasis}(L,d,L-1)\gamma(1:L)$, i.e.,

\[
\begin{bmatrix}
1 \\
1 \\
1 \\
1 \\
7
\end{bmatrix} = c_{\text{init}} = (S^T S)^{-1} S^T y_{1:L} = \begin{bmatrix}
-0.2 & 0.0 & 0.2 & 0.4 & 0.6 \\
-0.2 & -0.1 & 0.0 & 0.1 & 0.2 \\
\end{bmatrix}
\]

The $a_{\text{init}}$ 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,

\[
d_{[1]}^{[1]} = \lambda a_{[1]}^{[1]} + \alpha y_0 = (0.9)(-5.2) + (0.1)(7) = -3.980
\]

\[
d_{[1]}^{[2]} = \lambda a_{[1]}^{[2]} + \alpha a_{[1]}^{[1]} = (0.9)(-18.7) + (0.1)(-3.98) = -17.228
\]

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:

\[
\begin{align*}
\vec{c}_{\text{init}} &= (F^T)^L c_{\text{init}} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 8.3 \\
1.5 \end{bmatrix} = \begin{bmatrix} 8.3 \\
1.5 \end{bmatrix} = \begin{bmatrix} 15.8 \\
1.5 \end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
\vec{a}_{\text{init}} &= M \vec{c}_{\text{init}} = \begin{bmatrix} 1 & 1 \\
1 & -9 \\
1 & -18 \end{bmatrix} \begin{bmatrix} 15.8 \\
1.5 \end{bmatrix} = \begin{bmatrix} 2.3 \\
-11.2 \end{bmatrix}
\end{align*}
\]
The first initialization scheme (finding $c_{\text{init}}$, $a_{\text{init}}$ 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

$$
\begin{bmatrix}
    a_n \\
    b_n
\end{bmatrix} = \begin{bmatrix}
    1 & 1 \\
    0 & 1
\end{bmatrix} \begin{bmatrix}
    a_{n-1} \\
    b_{n-1}
\end{bmatrix} + \begin{bmatrix}
    \alpha_1 \\
    \alpha_2
\end{bmatrix} (y_n - a_{n-1} - b_{n-1})
$$

(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 $\lambda$ parameter.

Holt [824] has generalized (24.13.1) to allow arbitrary values for $\alpha_1$, $\alpha_2$. 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

$$a_n = a_{n-1} + b_{n-1} + \alpha_1 (y_n - a_{n-1} - b_{n-1})$$

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:

$$
\begin{align*}
    a_n &= a_{n-1} + b_{n-1} + \alpha_1 (y_n - a_{n-1} - b_{n-1}) \\
    b_n &= b_{n-1} + \bar{\alpha}_2 (a_n - a_{n-1} - b_{n-1})
\end{align*}
$$

(24.13.2)

and defining the effective $\lambda$-parameters $\lambda_1 = 1 - \alpha_1$ and $\lambda_2 = 1 - \bar{\alpha}_2$,

$$
\begin{align*}
    a_n &= \lambda_1 (a_{n-1} + b_{n-1}) + \alpha_1 y_n \\
    b_n &= \lambda_2 b_{n-1} + \bar{\alpha}_2 (a_n - a_{n-1})
\end{align*}
$$

(Holt’s exponential smoothing) (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 $\hat{y}_{n+\tau/n} = a_n + b_n \tau$, and for the next time instant, $\hat{y}_{n+\tau/n} = a_n + b_n + D$, and for the next time instant, $\hat{y}_{n+\tau/n} = a_n + b_n \tau$. The MATLAB function $\text{holt}$ implements Eq. (24.13.1):

```
C = holt(y,a1,a2,cinit); % Holt's exponential smoothing
```

where $C$ has the same meaning as $\text{stema}$, its $n$th row $c^T(n) = [a_n, b_n]$ holding the local level and slope at time $n$. The initialization vector $c_{\text{init}}$ can be chosen as in stema by a linear fit to the first $L$ samples of $y$, where $L = (1 + \lambda)/(1 - \lambda)$, with $\lambda$ determined from $\alpha_1$ from the relationship $\alpha_1 = 1 - \lambda^2$ or $\lambda = \sqrt{1 - \alpha_1}$. Another possibility is to choose $c_{\text{init}} = [y_0, 0]^T$, or, $[y_0, y_1 - y_0]^T$.

Like $\text{emaerr}$, the MATLAB function $\text{holterr}$ evaluates the MSE/MAE/MAPE errors over any set of parameter pairs $(\alpha_1, \alpha_2)$ and produces the corresponding optimum pair $(\alpha_{1,\text{opt}}, \alpha_{2,\text{opt}})$:

```
[err,aloapt,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 $a_n, b_n$:

$$
H_a(z) = \frac{\alpha_1 + (\alpha_2 - \alpha_1)z^{-1}}{1 + (\alpha_1 + \alpha_2 - 2)z^{-1} + (1 - \alpha_1)z^{-2}}
$$

$$
H_b(z) = \frac{\alpha_2 (1 - z^{-1})}{1 + (\alpha_1 + \alpha_2 - 2)z^{-1} + (1 - \alpha_1)z^{-2}}
$$

(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 $\sigma_y^2 = \mathcal{R} \sigma_y^2$, where $\mathcal{R}$ is the NRR of $H(z)$:

$$
\mathcal{R} = \sum_{n=0}^{\infty} h^2(n) = \frac{2\alpha_1^2 + \alpha_4 \alpha_2 + 2\alpha_2}{\alpha_1 (4 - \alpha_1 - \alpha_4)}
$$

(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 $0 \leq \alpha_1 \leq 1$ and $0 \leq \alpha_2 \leq 2\alpha_1 (1 - \alpha_1)/(1 + \alpha_1)$, with $\mathcal{R}$ 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:

$$
\begin{align*}
    a_n &= \lambda_1 (a_{n-1} + b_{n-1}) + \alpha_1 (y_n - S_{n-D}) \\
    b_n &= \lambda_2 b_{n-1} + \bar{\alpha}_2 (a_n - a_{n-1}) \\
    s_n &= \lambda_3 s_{n-D} + \bar{\alpha}_3 (y_n - a_{n-1} - b_{n-1})
\end{align*}
$$

(Holt-Winters) (24.13.6)

where $D$ is the assumed periodicity of the seasonal data, and $\alpha_3$ 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 \( \hat{n} \) of Eq. (24.1.18) is the group delay at DC, that is, show Eq. (24.1.19),

\[
\hat{n} = \frac{d\theta(\omega)}{d\omega} \bigg|_{\omega=0}
\]

24.2 The lag of a filter was defined by Eqs. (24.1.17) and (24.1.18) to be,

\[
\hat{n} = \sum_{n} n h_n = -H'(\omega) \bigg|_{\omega=0}
\]

If the filter \( H(z) \) is the cascade of two filters, \( H(z) = H_1(z)H_2(z) \), with individual lags, \( \hat{n}_1, \hat{n}_2 \), 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,

\[
\hat{n} = \hat{n}_1 + \hat{n}_2
\]

24.3 Consider a low-frequency signal \( s(n) \) whose spectrum \( S(\omega) \) is limited within a narrow band around DC, \( |\omega| \leq \Delta \omega \), and therefore, its inverse DTFT representation is:

\[
s(n) = \left. \frac{1}{2\pi} \int_{-\Delta \omega}^{\Delta \omega} S(\omega) e^{j\omega n} \, d\omega \right. 
\]

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,

\[
y(n) = \left. \frac{1}{2\pi} \int_{-\Delta \omega}^{\Delta \omega} H(\omega)S(\omega) e^{j\omega n} \, d\omega \right. = s(n - \hat{n})
\]

24.4 Show that the general filter-sharpening formula (24.9.5) results in the following special cases:

\[
p = 0, \quad q = d \quad \Rightarrow \quad H_{\text{imp}} = 1 - (1 - H)^{d-1}
\]

\[
p = 1, \quad q = d \quad \Rightarrow \quad H_{\text{imp}} = 1 - (1 - H)^{d-1} [1 + (d + 1)H]
\]

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:

\[
h_a(n) = (1 - \lambda) \left[ 1 + \lambda - (1 - \lambda)n \right] \lambda^n u(n)
\]

b. Show the relationships:

\[
\sum_{n=0}^{\infty} n h_a(n) = 0, \quad \sum_{n=0}^{\infty} n^2 h_a(n) = -\frac{2\lambda^2}{(1 - \lambda)^2}
\]

c. Show that the NRR of the filter \( H_a(z) \) is:

\[
R = \sum_{n=0}^{\infty} h_a^2(n) = \frac{(1 - \lambda)(1 + 4\lambda + 5\lambda^2)}{(1 + \lambda)^3}
\]

d. Show that the magnitude response squared of \( H_a(z) \) is:

\[
|H_a(\omega)|^2 = \frac{(1 - \lambda)^2 [1 + 2\lambda + 5\lambda^2 - 4\lambda(1 + \lambda)]}{|1 - 2\lambda \cos \omega + \lambda^2|^2}
\]

e. Show that Eq. (24.14.2) has local minima at \( \omega = 0 \) and \( \omega = \pi \), and a local maximum at \( \omega = \omega_{\text{max}} \):

\[
\cos \omega_{\text{max}} = \frac{1 + 4\lambda - \lambda^2}{2(1 + \lambda)}
\]

and that the corresponding extremal values are:

\[
|H_a(0)|^2 = 1, \quad |H_a(\pi)|^2 = \frac{(1 - \lambda)^2 (1 + 3\lambda)^2}{(1 + \lambda)^4}
\]

24.8 Consider the modified EMA of Eq. (15.2.2) and its twicing,

\[
H(z) = \frac{(1 - \lambda)(1 + z^{-1})}{2(1 - \lambda z^{-1})}, \quad H_a(z) = 2H(z) - H^2(z) = \frac{(1 - \lambda)(1 + z^{-1})(3 + \lambda - z^{-1}(1 + 3\lambda))}{4(1 - \lambda z^{-1})^2}
\]

a. Show the relationships:

\[
\sum_{n=0}^{\infty} n h_a(n) = 0, \quad \sum_{n=0}^{\infty} n^2 h_a(n) = -\frac{(1 + \lambda)^2}{2(1 - \lambda)^2}
\]

b. Show that the NRR of the filter \( H_a(z) \) is:

\[
R = \sum_{n=0}^{\infty} h_a^2(n) = \frac{1}{8} (1 - \lambda)(3\lambda + 7)
\]

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 \( \tau \) by \( \hat{n} = -\tau \).

b. Show that its NRR is given by

\[
R = \sum_{k=0}^{N-1} h_\tau^2(k) = \frac{1}{N} + \frac{3(N - 1 + 2\tau)^2}{N(N^2 - 1)}
\]

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:

\[
\frac{d^2}{d\omega^2}H(\omega)_{\omega=0} = -\sum_{n=0}^{\infty} k^2 h_r(k) = \frac{1}{6} (N - 1) (N - 2 + 6\tau)
\]

Determine the range of \(\tau\) for which \(|H(\omega)|^2\) 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,

\[
h(k) = \frac{3N(N-1) - 2k(2N-1)}{N(N^2 - 1)}, \quad k = 0, 1, \ldots, N - 1
\]

and verify explicitly the results:

\[
\sum_{k=0}^{N-1} h(k) = 1, \quad \sum_{k=0}^{N-1} k h(k) = \frac{N - 2}{6}, \quad \sum_{k=0}^{N-1} k^2 h(k) = 0, \quad \sum_{k=0}^{N-1} h^2(k) = \frac{7N^2 - 4N - 2}{3N(N^2 - 1)}
\]

e. Show that \(h_r(k)\) interpolates linearly between the \(\tau = 0\) and \(\tau = 1\) filters, that is, show that for \(k = 0, 1, \ldots, N - 1\),

\[
h_r(k) = (1 - \tau) h_a(k) + \tau h_1(k) = h_a(k) + [h_1(k) - h_a(k)] \tau
\]

f. Another popular choice for the delay parameter is \(\tau = -(N-1)/3\). Show that,

\[
h(k) = \frac{2(N - k)}{N(N + 1)}, \quad k = 0, 1, \ldots, N - 1
\]

and that,

\[
\sum_{k=0}^{N-1} h(k) = 1, \quad \sum_{k=0}^{N-1} k h(k) = \frac{N - 1}{3}, \quad \sum_{k=0}^{N-1} k^2 h(k) = \frac{N(N-1)}{6}, \quad \sum_{k=0}^{N-1} h^2(k) = \frac{2(2N+1)}{3N(N+1)}
\]

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 \(h_r(k)\) 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 \(h_r(k)\), \(0 \leq k \leq N - 1\), and for each value of \(\tau\), calculate the filter lag, \(\hat{n}\), the NRR, \(R\), and the “curvature” parameter of Eq. (24.10.5). Recall from part (d) of the Problem 24.9 that the second \(\tau\) should result in zero curvature. Repeat all the questions for \(N = 18\).
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:

\[ h_{\text{(SMA)}}(n) = \frac{1}{N}, \quad 0 \leq n \leq N - 1 \]
\[ h_{\text{(WMA)}}(n) = \frac{2(N - n)}{N(N + 1)}, \quad 0 \leq n \leq N - 1 \]  
\[ (25.2.1) \]
\[ h_{\text{(TMA)}}(n) = \frac{N - |n - N + 1|}{N^2}, \quad 0 \leq n \leq 2N - 2 \]
\[ h_{\text{(EMA)}}(n) = (1 - \lambda)^n, \quad 0 \leq n < \infty \]

where \( N \) denotes the filter span for the SMA and WMA cases, while for the EMA case, \( \lambda \) is a forgetting factor such that \( 0 < \lambda < 1 \), which is usually specified in terms of 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,

\[ N = \frac{1 + \lambda}{1 - \lambda} \Rightarrow \lambda = \frac{N - 1}{N + 1} \Rightarrow \alpha = 1 - \lambda = \frac{2}{N + 1} \]  
\[ (25.2.3) \]

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 \( n \)th 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,

\[ a_{\text{(SMA)}} = \frac{y_n + y_{n-1} + y_{n-2} + \cdots + y_{n-N+1}}{N} \]
\[ (25.2.4) \]
\[ a_{\text{(WMA)}} = \frac{2}{N(N + 1)} \sum_{k=0}^{N-1} (N - k) y_{n-k} \]
\[ a_{\text{(TMA)}} = \frac{1}{N^2} \sum_{k=0}^{2N-2} (N - |k - N + 1|) y_{n-k} \]
\[ a_{\text{(EMA)}} = \alpha a_{n-1} + (1 - \lambda) y_n \]

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,

\[ R = \sum_{n=0}^{\infty} h^2(n) \]  
\[ (25.2.5) \]

with transfer functions,

\[ H_{\text{(SMA)}}(z) = \frac{1 + z^{-1} + z^{-2} + \cdots + z^{-(N+1)}}{N} = \frac{1}{N} \frac{1 - z^{-N}}{1 - z^{-1}} \]
\[ (25.2.2) \]
\[ H_{\text{(WMA)}}(z) = \frac{2}{N(N + 1)} \frac{N - (N + 1)z^{-1} + z^{-(N+1)}}{(1 - z^{-1})^2} \]
\[ H_{\text{(TMA)}}(z) = \left[ \frac{1}{N} \frac{1 - z^{-N}}{1 - z^{-1}} \right]^2 \]
\[ H_{\text{(EMA)}}(z) = \frac{\alpha}{1 - \frac{1}{N} \frac{1 - z^{-N}}{1 - z^{-1}}}, \quad \alpha = 1 - \lambda \]

\[ \hat{\text{adapted from the author’s book on Applied Optimum Signal Processing [45]}} \]
25.2. MOVING AVERAGE FILTERS – SMA, WMA, TMA, EMA

with smaller \( R \) corresponding to more effective noise reduction. By construction, the above filters are lowpass filters with unity gain at DC, therefore, satisfying the constraint,

\[
\sum_{n=0}^{\infty} h(n) = 1
\]

The “lag” is defined as the group delay at DC which, after using Eq. (25.2.6), is given by,

\[
\bar{n} = \sum_{n=0}^{\infty} nh(n)
\]

One can easily verify that the noise-reduction ratios and lags of the above filters are:

\begin{align*}
\text{(SMA)} & \quad R = \frac{1}{N}, \quad \bar{n} = \frac{N - 1}{2} \\
\text{(WMA)} & \quad R = \frac{4N + 2}{3N(N + 1)}, \quad \bar{n} = \frac{N - 1}{3} \\
\text{(TMA)} & \quad R = \frac{2N^2 + 1}{3N^2}, \quad \bar{n} = N - 1 \\
\text{(EMA)} & \quad R = \frac{1 - \lambda}{1 + \lambda} = \frac{1}{N}, \quad \bar{n} = \frac{\lambda}{1 - \lambda} = \frac{N - 1}{2}, \quad \text{for equivalent } N
\end{align*}

The tradeoff is evident, with \( R \) 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,

\[
h(n) = \frac{6(n + 1)(N - 1 - n)}{N(N^2 - 1)}, \quad n = 0, 1, \ldots, N - 1
\]

\[
H(z) = \frac{6}{N(2z - 1)(1 + z^{-1})} \left(1 + z^{-N} - (1 - z^{-N})(1 + z^{-1})(1 - z^{-1})\right) \\
\bar{n} = \frac{N - 2}{2}, \quad R = \frac{6(N^2 + 1)}{5N(N^2 - 1)}
\]

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, \( h = [h_0, h_1, \ldots] \), of the SMA, WMA, TMA, and ILRS filters for the case \( N = 5 \),

\begin{align*}
\text{(SMA)} & \quad h = \frac{1}{5}[1, 1, 1, 1, 1] \\
\text{(WMA)} & \quad h = \frac{1}{15}[5, 4, 3, 2, 1] \\
\text{(TMA)} & \quad h = \frac{1}{25}[1, 2, 3, 4, 5, 4, 3, 2, 1] \\
\text{(ILRS)} & \quad h = \frac{1}{10}[2, 3, 3, 2, 0]
\end{align*}

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 \( y \) represents the financial data to be filtered.

\[
a = \text{sma}(y, N, yin); \quad \% \text{simple moving average}
a = \text{wma}(y, N, yin); \quad \% \text{weighted moving average} \\
a = \text{tma}(y, N, yin); \quad \% \text{triangular moving average} \\
a = \text{ilrs}(y, N, yin); \quad \% \text{integrated linear regression slope}
\]

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,

\[
yin = 'f', \quad \% \text{progressive filtering (default method)}
yin = 'n', \quad \% \text{initial transients are NaNs (facilitates plotting)} \\
yin = 'c', \quad \% \text{standard convolutional transients}
\]

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:

\[
\begin{align*}
    a_n &= \sum_{k=0}^{N-1} h_a(k) y_{n-k} = \text{linear regression indicator} \\
    b_n &= \sum_{k=0}^{N-1} h_b(k) y_{n-k} = \text{linear regression slope indicator} \\
    a_n + b_n &= \sum_{k=0}^{N-1} h_1(k) y_{n-k} = \text{time-series forecast indicator}
\end{align*}
\]

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 \( \tau \) steps ahead from time \( n \) is given by the following indicator, which we will refer to as the predictive moving average (PMA),

\[
\hat{y}_{n+\tau/n} = a_n + \tau b_n = \sum_{k=0}^{N-1} h_\tau(k)y_{n-k} \tag{PMA} \]

where, as follows from Eq. (24.4.4), we have for \( n = 0, 1, \ldots, N-1, \)

\[
h_\tau(n) = h_a(n) + \tau h_b(n) = \frac{2(2N - 1 - 3n)}{N(N+1)} + \frac{6(N - 1 - 2n)}{N(N^2 - 1)} \tag{25.3.3}
\]

The time “advance” \( \tau \) can be non-integer, positive, or negative. Positive \( \tau \)'s correspond to forecasting, negative \( \tau \)'s 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 \( \tau \)):

\[
R = \sum_{n=0}^{N-1} h^2(n) = \min \quad \text{subject to} \quad \sum_{n=0}^{N-1} h(n) = 1, \quad \sum_{n=0}^{N-1} nh(n) = -\tau \tag{25.3.4}
\]

This was solved in Sec. 24.4. The noise-reduction ratio of these filters is,

\[
R_\tau = \sum_{n=0}^{N-1} h^2_\tau(n) = \frac{1}{N} + \frac{3(N-1+2\tau)^2}{N(N^2-1)} \tag{25.3.5}
\]

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 \( \tau = 0, \)

\[
R_{\text{SMA}} = \frac{1}{N}, \quad R_a = \frac{4N - 2}{N(N+1)}
\]

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, \( h = -H_a^\prime(z) \big|_{z=1} = 0. \)

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 \( 0 \leq n \leq N - 1 \). 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 \( H_p(z) \) 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, \( \text{pma} \), implements Eq. (25.3.2) and the related indicators, where the input array \( y \) represents the financial data to be filtered. The function, \( \text{pmainmp} \), implements the impulse response of Eq. (25.3.3).

\[
\begin{align*}
\text{at} &= \text{pma}(y,N,\text{tau},\text{yin}); \quad \% \text{at} = a + \tau b, \text{prediction distance tau} \\
\text{a} &= \text{pma}(y,N,0,\text{yin}); \quad \% \text{local-level indicator} \\
\text{b} &= \text{pma}(y,N,1,\text{yin}) - \text{pma}(y,N,0,\text{yin}); \quad \% \text{local-slope indicator} \\
\text{af} &= \text{pma}(y,N,1,\text{yin}); \quad \% \text{time-series forecast indicator, af} = a + b \\
\text{ht} &= \text{pmainmp}(N,\text{tau}); \quad \% \text{impulse response of predictive filter} \\
\text{ha} &= \text{pmainmp}(N,0); \quad \% \text{impulse response of local level filter} \\
\text{hb} &= \text{pmainmp}(N,1)-\text{pmainmp}(N,0); \quad \% \text{impulse response of local slope filter}
\end{align*}
\]

and again, the string variable \( \text{yin} \) specifies the way the filters are initialized and can take on the following values,

\[
\begin{align*}
\text{yin} &= 'f', \quad \% \text{progressive filtering (default method)} \\
\text{yin} &= 'n', \quad \% \text{initial transients are NaNs (facilitates plotting)} \\
\text{yin} &= 'c', \quad \% \text{standard convolutional transients}
\end{align*}
\]
A few examples of impulse responses are as follows, for $N = 5, 8, 11$,

$$N = 5, \quad h_a = \frac{1}{3} [3, 2, 1, 0, -1] \quad \text{ (local level)}$$
$$h_b = \frac{1}{10} [2, 1, 0, -1, -2] \quad \text{ (local slope)}$$
$$h_1 = \frac{1}{10} [8, 5, 2, -1, -4] \quad \text{ (time-series forecast)}$$

$$N = 8, \quad h_a = \frac{1}{12} [7, 5, 3, 1, -1, -3, -5, -7]$$
$$h_b = \frac{1}{84} [14, 11, 8, 5, 2, -1, -4, -7]$$
$$h_1 = \frac{1}{28} [20, 17, 14, 11, 8, 5, 2, -1, -4, -7, -10]$$

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,

- **SEMA** (25.4.1)
  \begin{align*}
  \hat{e}_{n-1} &= y_n - \hat{y}_{n-1} = y_n - a_{n-1} \\
  a_n &= a_{n-1} + (1 - \lambda) \hat{e}_{n-1}
  \end{align*}

- **DEMA** (25.4.2)
  \begin{align*}
  \left[ \begin{array}{c}
  a_n \\
  b_n
  \end{array} \right] &= \frac{1}{2} \left[ \begin{array}{c}
  a_{n-1} \\
  b_{n-1}
  \end{array} \right] + \left[ \begin{array}{c}
  \alpha_2
  \end{array} \right] \hat{e}_{n-1}
  \end{align*}

- **TEMA** (25.4.3)
  \begin{align*}
  \left[ \begin{array}{c}
  a_n \\
  b_n \\
  c_n
  \end{array} \right] &= \frac{1}{2 \lambda^2} \left[ \begin{array}{c}
  a_{n-1} \\
  b_{n-1} \\
  c_{n-1}
  \end{array} \right] + \left[ \begin{array}{c}
  \alpha_2
  \end{array} \right] \hat{e}_{n-1}
  \end{align*}

where $\alpha_1 = 1 - \lambda^3, \quad \alpha_2 = \frac{3}{2} (1 - \lambda) (1 - \lambda^2), \quad \alpha_3 = \frac{1}{2} (1 - \lambda)$

and $\hat{y}_{n-1}$ represents the forecast of $y_n$ based on data up to time $n-1$. More generally, the forecast ahead by a distance $\tau$ is given by,

- (SEMA) $\hat{y}_{n+\tau/n} = a_n$
- (DEMA) $\hat{y}_{n+\tau/n} = a_n + b_n \tau$
- (TEMA) $\hat{y}_{n+\tau/n} = a_n + b_n \tau + c_n \tau^2$

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 $\alpha = 1 - \lambda$,

$$a_n = 2a_n^{[1]} - a_n^{[2]} = \text{local level DEMA indicator}$$
$$b_n = \frac{\alpha}{\lambda} (a_n^{[1]} - a_n^{[2]}) = \text{local slope DEMA indicator}$$

The transfer functions from $y_n$ to the signals $a_n, b_n, c_n$ were given in Eq. (24.8.5), and are expressible as follows in terms of the transfer function of a single EMA, $H(z) = \frac{\alpha}{1 - \lambda z^{-1}}$,

$$H_a(z) = \frac{\alpha (1 + \lambda - 2\lambda z^{-1})}{1 - \lambda z^{-1}} = 2H(z) - H^2(z) = 1 - [1 - H(z)]^2$$

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,

$$a_n^{[1]} = \frac{\alpha (1 + 5\lambda)}{\lambda} a_n^{[1]} + \alpha y_n$$
$$a_n^{[2]} = \frac{\alpha (1 + 3\lambda)}{\lambda} a_n^{[1]} + \alpha a_n^{[1]}$$
$$a_n^{[3]} = \frac{\alpha (1 + \lambda)}{\lambda} a_n^{[1]} + \alpha a_n^{[2]}$$

where $\alpha = 1 - \lambda$. See also Eqs. (24.8.9)–(24.8.13). In particular, we have,

$$a_n = 3a_n^{[1]} - 3a_n^{[2]} + a_n^{[3]} = \text{local level TEMA indicator}$$

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 $y$ represents the financial data to be filtered.
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:

$$a_n - kb_n = y_{n-k}, \quad k = 0, 1, \ldots, N - 1 \quad (25.5.1)$$

which express the fitting of a straight line, $a + b\tau$, to the data $[y_{n+N-1}, \ldots, y_{n+1}, y_n]$, 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,

$$
\begin{bmatrix}
1 \\
1 \quad 1 \\
1 \quad 2 \\
\vdots \quad k \\
1 \\
N-1
\end{bmatrix}
\begin{bmatrix}
y_n \\
y_{n-1} \\
y_{n-2} \\
\vdots \\
y_{n-k} \\
y_{n-N+1}
\end{bmatrix}
= \begin{bmatrix}
[u,-k] \\
\vdots \\
1
\end{bmatrix}
\begin{bmatrix}
a_n \\
b_n
\end{bmatrix}
= y_n \quad (25.5.2)
$$

with the least-squares solution expressed in the following MATLAB-like vectorial notation using the backslash operator,

$$\begin{bmatrix}
a_n \\
b_n
\end{bmatrix} = [u,-k] \backslash y_n \quad (25.5.3)$$

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 $n$,

$$J_n = \sum_{k=0}^{N-1} (a - bk - y_{n-k})^2 = \min \quad (25.5.4)$$

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,

$$J_n = \sum_{k=0}^{\min(n,N-1)} (a - bk - y_{n-k})^2 = \min \quad (25.5.5)$$

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 $b_0 = 0$, then fit a line to $[y_0, y_1, y_2, y_3, \ldots, y_N]$ to get $a_1, b_1$, then fit a line to $[y_0, y_1, y_2, y_3, \ldots, y_N]$ to get $a_2, b_2$, and so on until $n = N - 1$, and beyond that, we continue with a length-$N$ data window.

---

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.

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].
Thus, we may state the complete solution for all $0 \leq n \leq L - 1$, where $L$ is the length of the data vector $y_n$, using the backslash notation.\footnote{The backslash solution also correctly generates the case $n = 0$, i.e., $a_0 = y_0$ and $b_0 = 0.$}

for each, $n = 0, 1, 2, \ldots, L - 1$, do:

\[
K_n = \min(n, N - 1) + 1 \quad \text{fitting length, } K_n = N \text{ when } n \geq N - 1
\]

\[
k = [0 : K_n - 1]^T \quad \text{column vector, length } K_n
\]

\[
y_n = y(n - k) \quad \text{column vector, } [y_n, y_{n-1}, \ldots, y_{n-K_n+1}]^T
\]

\[
\mathbf{u} = \text{ones}(K_n, 1) \quad \text{column vector}
\]

\[
\begin{bmatrix}
a_n \\ b_n
\end{bmatrix} = [\mathbf{u}, -k] \quad \text{linear regression indicators}
\]

\[
R^2(n) = (\text{corr}(-k, y_n))^2 = 1 - \det(\text{corrcoef}(-k, y_n)) = R^2 \quad \text{indicator}
\]

where we also included the so-called R-square indicator,\footnote{where \texttt{corr}, \texttt{det}, and \texttt{corrcoef}, are built-in MATLAB functions.} 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, \texttt{r2crit} in the AOSP toolbox, calculates the critical values $R^2_c$ of $R^2$ for a given $N$ and given confidence level $p$, such that if $R^2(n) > R^2_c$, then the linear fit is considered to be statistically significant for the $n$th segment. Some typical critical values of $R^2_c$ 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]),

\[
\begin{array}{cccc}
N & p = 0.95 & p = 0.99 \\
5 & 0.7711 & 0.9180 \\
10 & 0.3993 & 0.5846 \\
14 & 0.2835 & 0.4374 \\
20 & 0.1969 & 0.3152 \\
25 & 0.1569 & 0.2552 \\
30 & 0.1303 & 0.2143 \\
50 & 0.0777 & 0.1303 \\
60 & 0.0646 & 0.1090 \\
120 & 0.0322 & 0.0549
\end{array}
\]

(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),

\[
\mathbf{e}_n = y_n - [\mathbf{u}, -k] \begin{bmatrix} a_n \\ b_n \end{bmatrix} = \text{fitting error, column vector}
\]

\[
\sigma_e(n) = \sqrt{\frac{\mathbf{e}_n^T \mathbf{e}_n}{K_n - 2}} = \text{standard error}
\]

\[
\sigma_a(n) = \sqrt{\frac{2(2K_n - 1)}{K_n(K_n + 1)}} \sigma_e(n) = \text{standard error for } a_n
\]

\[
\sigma_b(n) = \sqrt{\frac{12}{K_n(K_n^2 - 1)}} \sigma_e(n) = \text{standard error for } b_n
\]

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, $(x_k, y_k), k = 0, 1, \ldots, K - 1$, 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],

\[
\sigma_e^2 = \frac{1}{K - 2} \sum_{k=0}^{K-1} e_k^2, \quad \sigma_a^2 = \sigma_e^2 \frac{\bar{x}^2}{K \sigma_e^2}, \quad \sigma_b^2 = \sigma_e^2 \frac{\bar{x}^2}{K \sigma_e^2} = \frac{12}{K(K^2 - 1)} \sigma_e^2
\]

(25.5.9)

For our special case of equally-spaced data, $x_k = -k$, we easily find,

\[
\bar{x} = -\bar{K} = -\frac{1}{K} \sum_{k=0}^{K-1} k = -\frac{K - 1}{2}
\]

\[
\bar{x}^2 = \bar{K}^2 = \frac{1}{K} \sum_{k=0}^{K-1} k^2 = \frac{(K - 1)(2K - 1)}{6}
\]

\[
\Rightarrow \sigma_a^2 = \frac{2(2K - 1)}{K(K + 1)} \sigma_e^2, \quad \sigma_b^2 = \frac{12}{K(K^2 - 1)} \sigma_e^2
\]

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, \texttt{lreg}, implements Eqs. (25.5.6) and (25.5.8) with usage,

\[
[a,b,R2,se,sa,sb] = \text{lreg}(y,N,\text{init}); \quad \% \text{linear regression indicators}
\]

where \text{init} specifies the initialization scheme and takes on the following values,

\text{init} = ‘f’, progressive linear fitting of initial N-1 samples, default \text{init} = ‘n’, replacing initial N-1 samples of a,b,R2,se,sa,sb by NaNs.
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,

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. 25.5.3 shows the PMA/linear regression indicator, $a_n$, for the same Nicor data, together with the corresponding $R^2$ signal, and the slope signal $b_n$, using again a filter length of $N = 20$. They were computed with the MATLAB code:

```matlab
[a,b,R2] = lreg(y,N); % local level, local slope, and R-squared signal
% 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 $R^2_c = 0.1969$, 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 $R^2_c$, 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, $0 \leq n < N - 1$, and then switches to fixed length $N$ for $n \geq N - 1$.

The same idea can be applied to all FIR filters, such as the SMA, WMA, TMA, and the PMA filter, $h_T(n)$, that is, to use the same type of filter, but of progressively increasing
length, during the period $0 \leq n < N - 1$, and then switch to using the filters of fixed length $N$ for $n \geq N - 1$. 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 $n \geq N - 1 = 4$, 

<table>
<thead>
<tr>
<th>progressive</th>
<th>convolutional</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_0 = y_0$</td>
<td>$a_0 = \frac{1}{5}y_0$</td>
</tr>
<tr>
<td>$a_1 = \frac{1}{2}(y_1 + y_0)$</td>
<td>$a_1 = \frac{1}{5}(y_1 + y_0)$</td>
</tr>
<tr>
<td>$a_2 = \frac{1}{3}(y_2 + y_1 + y_0)$</td>
<td>$a_2 = \frac{1}{5}(y_2 + y_1 + y_0)$</td>
</tr>
<tr>
<td>$a_3 = \frac{1}{4}(y_3 + y_2 + y_1 + y_0)$</td>
<td>$a_3 = \frac{1}{5}(y_3 + y_2 + y_1 + y_0)$</td>
</tr>
<tr>
<td>$a_4 = \frac{1}{5}(y_4 + y_3 + y_2 + y_1 + y_0)$</td>
<td>$a_4 = \frac{1}{5}(y_4 + y_3 + y_2 + y_1 + y_0)$</td>
</tr>
<tr>
<td>$a_5 = \frac{1}{5}(y_5 + y_4 + y_3 + y_2 + y_1)$</td>
<td>$a_5 = \frac{1}{5}(y_5 + y_4 + y_3 + y_2 + y_1)$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Similarly, the local level PMA filters, $h_b$, of lengths up to $N = 5$ can be determined from Eq. (25.3.3), leading to the following progressive initializations,

- $N = 1$, $h_a = [1]$, $a_0 = y_0$
- $N = 2$, $h_a = [1, 0]$, $a_1 = y_1$
- $N = 3$, $h_a = \frac{1}{6}[5, 2, -1]$, $a_2 = \frac{1}{6}(5y_2 + 2y_1 - y_0)$
- $N = 4$, $h_a = \frac{1}{10}[7, 4, 1, -2]$, $a_3 = \frac{1}{10}(7y_3 + 4y_2 + y_1 - 2y_0)$
- $N = 5$, $h_a = \frac{1}{5}[3, 2, 1, 0, -1]$, $a_4 = \frac{1}{5}(3y_4 + 2y_3 + y_2 - y_0)$

and for the local slope filters $h_b$,

- $N = 1$, $h_b = [0]$, $b_0 = 0$
- $N = 2$, $h_b = [1, -1]$, $b_1 = y_1 - y_0$
- $N = 3$, $h_b = \frac{1}{2}[1, 0, -1]$, $b_2 = \frac{1}{2}(y_2 - y_0)$
- $N = 4$, $h_b = \frac{1}{10}[3, 1, -1, -3]$, $b_3 = \frac{1}{10}(3y_3 + y_2 - y_1 - 3y_0)$
- $N = 5$, $h_b = \frac{1}{10}[2, 1, 0, -1, -2]$, $b_4 = \frac{1}{10}(2y_4 + y_3 - y_2 - 2y_0)$

where, we arbitrarily set $h_b = [0]$ 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 $n = 2$,

$$J_2 = (a - y_2)^2 + (a - b - y_1)^2 + (a - 2b - y_0)^2 = \min$$
25.6. INITIALIZATION SCHEMES

with minimization conditions,

\[
\begin{align*}
\frac{\partial J_2}{\partial a} &= 2(a - y_2) + 2(a - b - y_1) + 2(a - 2b - y_0) = 0 \quad \Rightarrow \quad 3a - 3b = y_2 + y_1 + y_0 \\
\frac{\partial J_2}{\partial b} &= -2(a - b - y_1) - 4(a - 2b - y_0) = 0 \quad \Rightarrow \quad 3a - 5b = y_1 + 2y_0
\end{align*}
\]

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 \( n = 1 \),

\[
J_0 = (a - y_0)^2 = \min \Rightarrow a = y_0, \quad b = \text{indeterminate}
\]

\[
J_1 = (a - y_1)^2 + (a - b - y_0)^2 = \min \Rightarrow a = y_1, \quad b = y_1 - y_0
\]

### 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,

\[
a_0 = \lambda a_{-1} + \alpha y_0 = \lambda y_0 + \alpha y_0 = y_0
\]

(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, \( a_{-1} = \text{mean}(\{y_0, y_1, \ldots, y_{N-1}\}) \).

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,

\[
\begin{bmatrix}
a_1^{[1]} \\
a_1^{[2]}
\end{bmatrix}
= \begin{bmatrix}
y_0 \\
y_0
\end{bmatrix} \Rightarrow
\begin{bmatrix}
a_{-1} \\
b_{-1}
\end{bmatrix}
= \begin{bmatrix}
2 & -1 \\
\alpha/\lambda & -\alpha/\lambda
\end{bmatrix}
\begin{bmatrix}
a_{-1}^{[1]} \\
a_{-1}^{[2]}
\end{bmatrix}
= \begin{bmatrix}
y_0 \\
0
\end{bmatrix}
\]

(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,

\[
\begin{align*}
n &= \lfloor 1 : N \rfloor \text{ = column vector} \\
y &= \{y_0, y_1, \ldots, y_{N-1}\} \text{ = column vector} \\
u &= \text{ones(size}(n)) \\
\begin{bmatrix}
a_{-1} \\
b_{-1}
\end{bmatrix}
= [u, n] \setminus y
\end{align*}
\]

If one wishes to use the cascade of two EMAs, then the EMA signals, \( a_0^{[1]}, a_0^{[2]} \), must be initialized by first applying Eq. (25.6.3), and then using the inverse matrix relationship of Eq. (25.4.6), i.e.,

\[
\begin{bmatrix}
a_1^{[1]} \\
a_1^{[2]}
\end{bmatrix}
= \begin{bmatrix}
1 & -\lambda/\alpha \\
1 & -2\lambda/\alpha
\end{bmatrix}
\begin{bmatrix}
a_{-1} \\
b_{-1}
\end{bmatrix}
\]

(25.6.4)

A third possibility \([864]\) is to initialize the first EMA with \( a_0^{[1]} = y_0 \), 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 \( a_{N-1}^{[1]} = a_{N-1}^{[1]} \). This value can be iterated backwards to \( n = -1 \) to determine the proper initial value \( a_0^{[2]} \) 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,

\[
\begin{align*}
a_0^{[1]} &= y_0 \\
a_0^{[2]} &= a_{N-1}^{[1]}
\end{align*}
\]

for \( n = 0, 1, \ldots, N - 1 \),

\[
a_n^{[1]} = \lambda a_n^{[1]} + \alpha y_n \Rightarrow a_{N-1}^{[2]} = \frac{\alpha}{\lambda} (a_{N-1}^{[2]} - \alpha a_n^{[1]})
\]

(25.6.5)

Upon exit from the second loop, one has \( a_0^{[2]} \), then, one can transform the calculated \( a_0^{[1]}, a_0^{[2]} \) 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,

\[
\begin{bmatrix}
a_{-1} \\
b_{-1}
\end{bmatrix}
= \begin{bmatrix}
0 \\
0
\end{bmatrix} \Rightarrow
\begin{bmatrix}
a_{-1}^{[1]} \\
a_{-1}^{[2]}
\end{bmatrix}
= \begin{bmatrix}
0 \\
0
\end{bmatrix}
\]

(25.6.6)

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,

\[
\begin{bmatrix}
a_{-1} \\
a_{-1}^{[1]} \\
a_{-1}^{[2]}
\end{bmatrix}
= \begin{bmatrix}
y_0 \\
y_0 \\
y_0
\end{bmatrix} \Rightarrow
\begin{bmatrix}
a_{-1} \\
b_{-1} \\
c_{-1}
\end{bmatrix}
= \begin{bmatrix}
y_0 \\
0 \\
0
\end{bmatrix}
\]

(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,
Upon exit from the loop, one has the proper starting value of $a_{-1}$. The following MATLAB function, \texttt{wema}, implements WEMA with such default initialization scheme,

\begin{verbatim}
  a = wema(y,N,ain);  \% Wilder's EMA
\end{verbatim}

$y =$ signal to be smoothed
$N =$ effective length, (EMA alpha = 1/N, lambda = 1-1/N)
$ain =$ any initial value

It is equivalent to an EMA with effective length, $N_e$, determined as follows,

\begin{equation}
  \lambda = \frac{N_e - 1}{N_e + 1} = 1 - \frac{1}{N} \implies N_e = 2N - 1
\end{equation}

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 $a_0$ 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,

\begin{equation}
  a_{N-1} = \frac{1}{N} (y_0 + y_1 + \cdots + y_{N-1})
\end{equation}

for $n = N-1, \ldots, 1, 0,$

\begin{equation}
  a_{n-1} = \frac{1}{\lambda} (a_n - \alpha y_n)
\end{equation}

\section*{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 samples/cycle, $f_s / 2\pi f_0$ is the sampling rate in Hz. We may also define the period of $f_0$ in units of samples/cycle, $N_e = 2\pi / f_0$ or, for $N_e \geq 1$.

Then, the transfer function can be expressed in the following cascaded and direct forms,

\begin{equation}
  H(z) = \left[ G_0 (1 + z^{-1}) \right] \prod_{i=1}^{K} \left[ 1 + a_i z^{-1} + a_2 z^{-2} \right] = \frac{G (1 + z^{-1})^M}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_M z^{-M}}
\end{equation}

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,

\begin{equation}
  G_0 = \frac{\Omega_0}{\Omega_0 + 1}, \quad a_0 = \frac{\Omega_0 - 1}{\Omega_0 + 1}
\end{equation}

The second-order coefficients are, for $i = 1, 2, \ldots, K,$
BUTTERWORTH MOVING AVERAGE FILTERS

FILTERING METHODS IN FINANCIAL MARKETS

\[ G_i = \frac{\Omega_0^2}{1 - 2\Omega_0 \cos \theta_i + \Omega_0^2} \]

\[ a_i = \frac{2(\Omega_0^2 - 1)}{1 - 2\Omega_0 \cos \theta_i + \Omega_0^2}, \quad a_{i+1} = \frac{1 + 2\Omega_0 \cos \theta_{i+1} + \Omega_0^2}{1 - 2\Omega_0 \cos \theta_i + \Omega_0^2} \]

(25.7.4)

\[ \theta_i = \frac{\pi}{2M} (M - 1 + i), \quad i = 1, 2, \ldots, K \]

(25.7.5)

and the quantity \( \Omega_0 \) is the equivalent analog 3-dB frequency defined as,

\[ \Omega_0 = \tan \left( \frac{\omega_0}{2} \right) = \tan \left( \frac{\pi f_0}{f_s} \right) = \tan \left( \frac{\pi}{N} \right) \]

(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:

\[ |H(\omega)|^2 = \frac{1}{1 + (\Omega/\Omega_0)^{2M}} = \frac{1}{1 + (\tan(\omega/2)/\Omega_0)^{2M}} \]

(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,

\[ \frac{4G_i}{1 + a_i + a_{i+1}} = 1 \quad \text{and} \quad \frac{2G_0}{1 + a_0} = 1 \]

Moreover, the filter lag can be shown to be (cf. Problem 24.12), for any \( M \geq 1 \) and \( N > 2 \),

\[ \hat{n} = \frac{1}{2\Omega_0 \sin \left( \frac{\pi}{2M} \right)} = \frac{1}{2 \tan \left( \frac{\pi}{N} \right) \sin \left( \frac{\pi}{2M} \right)} \]

(lag)

(25.7.8)

For \( M \geq 2 \) and \( N \geq 5 \), it can be approximated well by [868],

\[ \hat{n} \approx \frac{MN}{\pi^2} \]

The overall numerator gain in the direct form is the product of gains,

\[ G = G_0 G_1 G_2 \cdots G_K \]

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, \( a = [1] \) if \( M \) is even, and, \( a = [1, a_0] \) if \( M \) is odd, then the vector, \( a = [1, a_1, a_2, \ldots, a_M] \), can be constructed recursively by,

for \( i = 1, 2, \ldots, K \)

\[ a = \text{conv}(a, [1, a_1, a_2]) \]

(25.7.9)

For example, we have,

\[ M = 2, \quad a = [1, a_{11}, a_{12}] \]

\[ M = 3, \quad a = \text{conv}([1, a_{01}], [1, a_{11}, a_{12}]) = [1, a_{01} + a_{11}, a_{12} + a_{01} a_{11}, a_{01} a_{12}] \]

From these, we obtain the following explicit expressions, for \( M = 2 \),

\[ G = \frac{\Omega_0^2}{\Omega_0^2 + \sqrt{2}\Omega_0 + 1}, \quad a_1 = \frac{2(\Omega_0^2 - 1)}{\Omega_0^2 + \sqrt{2}\Omega_0 + 1}, \quad a_2 = \frac{\Omega_0^2 - \sqrt{2}\Omega_0 + 1}{\Omega_0^2 + \sqrt{2}\Omega_0 + 1} \]

(25.7.10)

\[ H(z) = G(1 + 2z^{-1} + z^{-2}), \quad \hat{n} = \frac{1}{\sqrt{2}\Omega_0} \]

and, for \( M = 3 \),

\[ G = \frac{\Omega_0^2}{(\Omega_0 + 1)(\Omega_0^2 + \Omega_0 + 1)}, \quad a_1 = \frac{(\Omega_0 - 1)(3\Omega_0^2 + 5\Omega_0 + 3)}{(\Omega_0 + 1)(\Omega_0^2 + \Omega_0 + 1)}, \quad a_2 = \frac{3\Omega_0^2 - 5\Omega_0 + 3}{\Omega_0^2 + \Omega_0 + 1}, \quad a_3 = \frac{(\Omega_0 - 1)(\Omega_0^2 + \Omega_0 + 1)}{(\Omega_0 + 1)(\Omega_0^2 + \Omega_0 + 1)} \]

(25.7.11)

\[ H(z) = G(1 + 3z^{-1} + 3z^{-2} + z^{-3}), \quad \hat{n} = \frac{1}{\Omega_0} \]

We note also that the \( M = 1 \) case has lag, \( \hat{n} = 1/(2\Omega_0) \), and is equivalent to the modified EMA of Eq. (15.2.2). This can be seen by rewriting \( H(z) \) in the form,

\[ H(z) = G_0(1 + z^{-1}) \frac{1 - \lambda(z^{-1})}{1 - \lambda z^{-1}}, \quad \lambda = \frac{1 - \Omega_0}{1 + \Omega_0} \]

where \( 0 < \lambda < 1 \) for \( \Omega_0 < 1 \), which requires \( N > 4 \).

The MATLAB function, \texttt{bma}, implements the design and filtering operations for any filter order \( M \) and any period \( N > 2 \), with usage,

\[ [y, nlag, b, a] = \text{bma}(x, N, M, yin); \quad \% \text{Butterworth moving average} \]

\[ [y, nlag, b, a] = \text{bma}(x, N, M); \]

where

\[ x = \text{input signal} \]

\[ N = \text{3-dB period}, \text{need not be integer, but } N>2 \]

\[ M = \text{filter order} \]

\[ yin = \text{any Mx1 vector of initial values of the output } y \]

\[ \text{default, } yin = \text{reppmat(x(1), M,1)} \]

\[ \text{yin = 'c' for standard convolutional output} \]

\[ y = \text{output signal} \]

\[ nlag = \text{filter lag} \]

\[ b = [b_0, b_1, b_2, \ldots, b_M], \text{numerator filter coefficients} \]

\[ a = [1, a_1, a_2, \ldots, a_M], \text{denominator filter coefficients} \]

\footnote{the sampling theorem requires, \( f_0 < f_s/2 \), or, \( N = f_s/f_0 > 2 \)}
25.8. MOVING AVERAGE FILTERS WITH REDUCED LAG

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,

```matlab
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
```

Below we list a number of examples that are special cases of the above constructions.

**Comparison of SMA and Butterworth filters of orders \( M = 2, 3 \).**

![Fig. 25.7.1](image-url)

**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, \( \hat{n}_1, \hat{n}_2, \hat{n}_3 \), 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),

\[
H_a(z) = H_1(z) \left[ (1 + v)H_2(z) - \nu H_3(z) \right] = \text{local level}
\]

\[
H_b(z) = \frac{1}{\hat{n}_3 - \hat{n}_2} H_1(z) \left[ H_2(z) - H_1(z) \right] = \text{local slope}
\]

where \( \nu \) is a positive constant. One may view \( H_a(z) \) as the smoothed, by \( H_1(z) \), version of \( (1 + v)H_2(z) - \nu H_3(z) \). The filter \( H_a(z) \) will still have unity DC gain as follows by evaluating Eq. (25.8.1) at \( z = 1 \), that is,

\[
H_a(1) = (1 + v)H_1(1)H_2(1) - \nu H_1(1)H_3(1) = (1 + v) - \nu = 1
\]

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,

\[
\hat{n}_a = (1 + v) (\hat{n}_1 + \hat{n}_2) - \nu (\hat{n}_1 + \hat{n}_3), \quad \text{or,}
\]

\[
\hat{n}_a = \hat{n}_1 + (1 + v) \hat{n}_2 - \nu \hat{n}_3
\]

By appropriately choosing the parameters \( v, \hat{n}_1, \hat{n}_2, \hat{n}_3 \), the lag \( \hat{n}_a \) can be made very small, even zero. Indeed, the following choice for \( v \) will generate any particular \( \hat{n}_a \),

\[
v = \frac{\hat{n}_1 + \hat{n}_2 - \hat{n}_a}{\hat{n}_3 - \hat{n}_2}
\]

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 \( \hat{n} \), 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, \( \hat{n} = r \cdot (N - 1) \), for example, \( r = 1/3 \), for WMA, and, \( r = 1/2 \), for EMA and SMA.

<table>
<thead>
<tr>
<th>reduced-lag filters</th>
<th>lag</th>
</tr>
</thead>
<tbody>
<tr>
<td>(twicing) ( H_a(z) = 2H(z) - H^2(z) ), ( \hat{n}_a = 0 )</td>
<td>( \hat{n}_a = 0 )</td>
</tr>
<tr>
<td>(GDEMA) ( H_a(z) = (1 + v)H(z) - \nu H^2(z) ), ( \hat{n}_a = (1 - \nu)\hat{n} )</td>
<td>( \hat{n}_a = (1 - \nu)\hat{n} )</td>
</tr>
<tr>
<td>(KH) ( H_a(z) = (1 + v)H^2(z) - \nu H^3(z) ), ( \hat{n}_a = (2 - \nu)\hat{n} )</td>
<td>( \hat{n}_a = (2 - \nu)\hat{n} )</td>
</tr>
<tr>
<td>(HMA) ( H_a(z) = H_{1/2}(z) [2H_{N/2}(z) - H_N(z)] ), ( \hat{n}_a = r \cdot (N - 2) )</td>
<td>( \hat{n}_a = r \cdot (N - 2) )</td>
</tr>
<tr>
<td>(ZEMA) ( H_a(z) = 2H(z) - z^{-d}H(z) ), ( \hat{n}_a = \hat{n} - d )</td>
<td>( \hat{n}_a = \hat{n} - d )</td>
</tr>
<tr>
<td>(ZEMA) ( H_a(z) = (1 + v)H(z) - \nu z^{-d}H(z) ), ( \hat{n}_a = \hat{n} - \nu d )</td>
<td>( \hat{n}_a = \hat{n} - \nu d )</td>
</tr>
</tbody>
</table>

The corresponding local-slope filters are as follows (they do not depend on \( v \),
25.8. MOVING AVERAGE FILTERS WITH REDUCED LAG

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, 
\[
H_{EMA}(z) = \frac{\alpha}{1 - \lambda z^{-1}}, \quad \alpha = 1 - \lambda, \quad \lambda = \frac{N - 1}{N + 1}, \quad n = \frac{N - 1}{2}
\]
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, \( H_{GDEMA}(z) = (1 + v)H(z) - vH^2(z) \), also has, \( H = H_{EMA} \), 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, 
\[
H_{T3}(z) = [H_{GDEMA}(z)]^3 (T3 \text{ indicator})
\]

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], \( H_a(z) = H_{\sqrt{N}}(z)\left[2H_{N/2}(z) - H_N(z)\right] \), 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,
\[
\hat{n}_1 = \frac{\sqrt{N} - 1}{3}, \quad \hat{n}_2 = \frac{N - 2}{2}, \quad \hat{n}_3 = \frac{N - 1}{3},
\]
and setting \( v = 1 \) in Eq. (25.8.2), we find,
\[
\hat{n}_a = \hat{n}_1 + 2\hat{n}_2 - \hat{n}_3 = \frac{\sqrt{N} - 1}{3} + \frac{N - 2}{2} - \frac{N - 1}{3} = \frac{\sqrt{N} - 2}{3}
\]

Thus, for larger \( N_s \), 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 \( \hat{n}_a \) 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),
\[
N_1 = \text{round}(\sqrt{N}), \quad \epsilon_1 = N_1 - \sqrt{N} = \text{rounding error}
\]
\[
N_2 = \text{round}\left(\frac{N}{2}\right), \quad \epsilon_2 = N_2 - \frac{N}{2} = \text{rounding error}
\]
\[
v = \frac{\hat{n}_1 + \hat{n}_2 - \hat{n}_a}{\epsilon_1 + \epsilon_2} = \frac{N/2 + \epsilon_1 + \epsilon_2}{N/2 - \epsilon_2}
\]

The following MATLAB code illustrates the computation of the local-level output signal \( a_n \) from the data \( y_m \) for the three versions of HMA and a given value of \( N > 1 \),
\[
N_1 = \text{round}((\sqrt{N}) \text{, } \epsilon_1 = N_1 - \sqrt{N} \text{, rounding error}
\]
\[
N_2 = \text{round}(N/2) \text{, } \epsilon_2 = N_2 - N/2 \text{, rounding error}
\]
\[
v = (N/2 + \epsilon_1 + \epsilon_2) / (N/2 - \epsilon_2)
\]
\[
a = \text{wma}((1+v)\text{wma}(y,N2) - v\text{wma}(y,N1), N1); \quad \% \text{ HMA}
\]
\[
a = \text{smw}((1+v)\text{smw}(y,N2) - v\text{smw}(y,N1), N1); \quad \% \text{ SHMA}
\]
\[
a = \text{sema}(2\text{sema}(y,N2) - \text{sema}(y,N), \sqrt{N}); \quad \% \text{ EHMA}
\]
The functions, \texttt{hma}, \texttt{shma}, \texttt{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 \cite{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. \eqref{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. \eqref{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, \( d = \text{round}(\bar{n}) \) = \( N/2 \), \( N = \text{even} \).

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,

\[
\begin{align*}
\bar{n} &= \frac{N - 1}{2}, \quad d = \text{round}(\bar{n}), \quad v = \frac{\bar{n}}{d} \\
H_a(z) &= (1 + v)H(z) - vzdH(z) \\
H_b(z) &= \frac{1}{d}[H(z) - z^{-d}H(z)]
\end{align*}
\tag{25.8.11}
\]

The code segment below illustrates the computation of the local-level ZEMA signal. It uses the function, \texttt{delay}, which implements the required delay.

```matlab
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.

```matlab
[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 \texttt{hma}, \texttt{shma}, and \texttt{zema}, \( yin = 'f' \), and to, \( yin = y_0 \), for \texttt{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.
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, $y_{\text{max}}$ or $y_{\text{min}}$, then, if $p < 1$ is the specified percentage amount ($p = \text{percentage}/100$), the following conditions must be satisfied by $y_{\text{max}}, y_{\text{min}},$

\[
\begin{align*}
y_{\text{max}} - y(n) &> p \cdot y_{\text{max}} & \text{(at local maximum)} \\
y(n) - y_{\text{min}} &> p \cdot y_{\text{min}} & \text{(at local minimum)}
\end{align*}
\]

The zigzag indicator can be implemented in the MATLAB function, \texttt{zigzag}, in the ISP2e toolbox, with usage,

\[
[z, nz] = \text{zigzag}(y, p); \quad % \text{zigzag indicator}
\]

\[
y = \text{closing values, or other indicator}
\]

\[
p = \text{percentage changes to be ignored, e.g., } p = 8 \text{ for } 8%
\]

\[
z = \text{zigzag points of swing-highs and swing-lows}
\]

\[
z = \text{zigzag time instants, note (MATLAB index) } = nz+1
\]

The MATLAB code for generating the above graphs was as follows:

\[
Y = \text{xlsread('IBM-sep00-jun01.xlsx');} \quad % \text{data file in ISP2e toolbox}
\]

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.
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, \( y_n, 0 \leq n \leq N - 1 \), the \( L_0 \) version minimizes the following \( L_0 \)-regularized least-squares performance index, for determining a length-\( N \) smoothed signal \( x_n \),

\[
\mathcal{J} = \sum_{n=0}^{N-1} |y_n - x_n|^2 + \lambda \sum_{n=s}^{N-1} |\nabla^s x_n|^0 = |y - x|^2 + \lambda \|D_s x\|_0 = \min
\]

(25.10.1)

where the \( L_0 \) norm\(^{1}\), \( \|D_s x\|_0 \), is the cardinality of the vector \( D_s x \), 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 \( \nabla^s \). It can be constructed in MATLAB by,

\[
D_s = \text{diff}(	ext{eye}(N),s); \quad \text{or, sparsely, } D_s = \text{diff}(	ext{sparsely}(N),s);
\]

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., \( D_s x \) is sparse, being almost zero essentially everywhere, except at the kink points. If \( s = 2 \), the kink points are where the slopes change — with \( D_s x \) being the 2nd difference of \( x \), the condition that \( D_s x \approx 0 \) 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 \( D_s x \) is the first difference of \( x \), and the sparsity of \( D_s x \) would imply that \( x(n) \) would be almost piecewise constant.

Larger values of the regularization parameter \( \lambda \) result in sparser \( D_s x \), and capture more broadly the trends in the data. Smaller values of \( \lambda \) 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, \texttt{l0trend} , in the ISP2e toolbox, implements this indicator with usage,

\[
[x,k] = \text{l0trend}(y,\lambda);
\]

There are some additional input parameters that allow one to set the value of \( \lambda \) (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, \texttt{l0trend} , 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 \( \lambda \) 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.

\[
\sum_{n=0}^{N-1} |y_n - x_n|^2 + \lambda \sum_{n=s}^{N-1} |\nabla^s x_n|^0 = |y - x|^2 + \lambda \|D_s x\|_0 = \min
\]

\[
\lambda \|D_s x\|_0
\]

\[
\text{Fig. 25.10.1 } L_0 \text{ trend indicator – almost piecewise linear } (s = 2).
\]

\[
\text{Fig. 25.10.2 } L_0 \text{ trend indicator – almost piecewise constants } (s = 1).
\]

1\( L_0 \) is not strictly-speaking a proper norm.
25.10. \( L_0 \) TRENDS INDICATOR

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,

\[
x, k = \text{l0trend}(y, \lambda, s); \quad \text{\textcopyright s=2 case by default}
\]

\[
\text{figure; plot(n,y, 'r-', n,x, 'b-', k(x+1), 'b.');}
\]

\[
\text{ek = 0.1; \quad \% kink threshold}
\]

\[
K = 10; \quad \% \text{number of IRLS iterations}
\]

\[
s = 1; \quad \% s can only be 1 or 2
\]

\[
[x, k] = \text{l0trend}(y, \lambda, ek, K, s); \quad \% s=1 case
\]

\[
\text{figure; plot(n,y, 'r-', n,x, 'b-', k(x+1), 'b.');}
\]

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,

\[
x, k = \text{l0trend}(y, \lambda, ek, K, s); \quad \% \text{estimate } x, \text{ and kink points } k
\]

\[
d = \ldots \quad \% \text{choose a value for } d
\]

\[
k2 = k(\text{end}); \quad \% \text{last kink point}
\]

\[
k1 = k(\text{end}-1); \quad \% \text{penultimate kink point}
\]

\[
x2 = x(k2+1); x1 = x(k1+1); \quad \% \text{estimated values at the kink points}
\]

\[
k\text{pred} = k2+1 : k2+d \quad \% \text{range of prediction times}
\]

\[
ypred = x2 + (s-1) * (k\text{pred}-k2) * (x2-x1)/(k2-k1) \quad \% \text{predictions}
\]

\[
M = \text{sma}(y, N); \quad \% \text{N-point SMA of closing prices } y
\]

\[
S = \text{stdev}(y, N); \quad \% \text{std-dev relative to } M
\]

\[
L = M - 2*S; \quad \% \text{lower bound}
\]

\[
U = M + 2*S; \quad \% \text{upper bound}
\]

where the function, \text{stdev}, uses the built-in function \text{std} to calculate the standard deviation over each length-\( N \) data window, and its essential code is,

\[
\text{for } n=1: \text{length}(y),
\]

\[
S(n) = \text{std}(y(\text{max}(1,n-N+1):n));
\]

\[
\text{end}
\]

25.11. ENVENPES, 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
- Standard-error bands
- Projection bands
- Donchian channels
- Fixed-width bands
- Keltner bands
- Starc 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 \text{SMA}, \text{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,

\[
M = \text{sma}(y, N); \quad \% \text{N-point SMA of closing prices } y
\]

\[
S = \text{stdev}(y, N); \quad \% \text{std-dev relative to } M
\]

\[
L = M - 2*S; \quad \% \text{lower bound}
\]

\[
U = M + 2*S; \quad \% \text{upper bound}
\]

where the function, \text{stdev}, uses the built-in function \text{std} to calculate the standard deviation over each length-\( N \) data window, and its essential code is,
**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,

\[
[a, -se, se] = \text{lreg}(y, N); \quad \% \text{N-point linear regression}
\]

\[
L = a - 2*se; \quad \% \text{lower bound}
\]

\[
U = a + 2*se; \quad \% \text{upper bound}
\]

---

![Fig. 25.11.1 Bollinger bands, standard-error bands, projection bands, and Donchian channels.](image1.png)

**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 \leq t \leq 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,
ENVELOPES, BANDS, AND CHANNELS

[~.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, \( n - N \leq t \leq n - 1 \), 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., \( \max[H_{n-1}, H_{n-2}, \ldots, H_{n-N}] \). The code below describes the procedure,

for \( n = 2:\text{length}(H) \) % 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,

\[
M = \text{sma}(C,N); \quad \% \text{N-point SMA of closing prices C}
L = M - p*M; \quad \% \text{lower band, e.g., } p = 0.03
U = M + p*M; \quad \% \text{upper band}
\]

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,

\[
M = \text{sma}(\text{sum}(H,L,C),N)/3; \quad \% \text{SMA of } (H+L+C)/3
D = \text{sma}(H-L,N); \quad \% \text{SMA of } (H-L)
L = M - D; \quad \% \text{lower band}
U = M + D; \quad \% \text{upper band}
\]

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\(^1\) 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,

\[
M = \text{sma}(C,N); \quad \% \text{SMA of closing prices}
R = \text{atr}(H,L,C,N_a); \quad \% \text{ATR = average true range}
L = M - 2*R; \quad \% \text{lower band}
U = M + 2*R; \quad \% \text{upper band}
\]

The ATR [865] is an \( N_a \)-point WEMA of the “true range”, defined as follows, at each time \( n \),

\[
T_n = \max[H_n - L_n, H_n - C_{n-1}, C_{n-1} - L_n] \quad \text{true range in } n \text{-th day}
R_n = \text{wema}(T_n, N_a) \quad \text{ATR} (25.11.1)
\]

and is implemented by the function \text{atr}, with the help of the \text{delay} function. Its essential MATLAB code is as follows, where \( H, L, C \) are column vectors,

\[
T = \max(H-L, H-\text{delay}(C,1), \text{delay}(C,1)-L, [], 2); \quad \% \text{row-wise max}
R = \text{wema}(T,N);
\]

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,

\[
[L,\bar{U},M] = \text{bbands}(y,N,d); \quad \% \text{Bollinger bands}
[L,\bar{U},\bar{a}] = \text{sebands}(y,N,d,\text{init}); \quad \% \text{standard-error bands}
[L,\bar{U},R,Rs] = \text{pbands}(Y,N,Ns); \quad \% \text{projection bands & oscillator}
[L,\bar{U},M] = \text{donch}(Y,N); \quad \% \text{Donchian channels}
[L,\bar{U},M] = \text{fbands}(Y,N,p); \quad \% \text{fixed-width bands}
[L,\bar{U},M] = \text{kbands}(Y,N); \quad \% \text{Keltner bands}
[L,\bar{U},M] = \text{stbands}(Y,N,Na); \quad \% \text{Starc bands}
S = \text{stdev}(y,N,\text{flag}); \quad \% \text{standard deviation}
[R,TR] = \text{atr}(Y,N); \quad \% \text{average true range}
\]

The essential MATLAB code for generating Figs. 25.11.1 and 25.11.2 is as follows,

\[
Y = \text{xlsread}('nicor.xls'); \quad \% \text{load Nicor-Gas data}
y = Y(:,1:4); \quad \% 130 trading days, and \{O,H,L,C\} prices
y = Y(:,4); \quad \% \text{closing prices}
t = 0:\text{length}(y)-1; \quad \% \text{trading days}
N = 20; \quad \% \text{used in Fig.6.22.1}
\]

\(^1\)Stoller Average Range Channels
Parabolic SAR

Wilders 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,

\[
S_{n+1} = \lambda_n S_n + \alpha_n E_n = (1 - \alpha_n) S_n + \alpha_n E_n, \quad \text{or,}
\]

\[
S_{n+1} = S_n + \alpha_n (E_n - S_n) \tag{25.11.2}
\]

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 \( \alpha_n \) increases linearly with time, starting with an initial value, \( \alpha_i \), 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,

\[
\alpha_{n+1} = \begin{cases} 
\alpha_n + \Delta \alpha, & \text{if } E_{n+1} \neq E_n \\
\alpha_n, & \text{if } E_{n+1} = E_n 
\end{cases} \tag{25.11.3}
\]

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 \( \alpha_n \) is not allowed to exceed a certain maximum value, \( \alpha_m \). 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 \( \alpha_n \) parameter, the EMA has a time-varying decreasing lag,\(^\dagger\) 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, \( S_n, E_n, \alpha_n, R_n \), where \( R_n \) holds the current position and is equal to \( \pm 1 \) for long/short.

\(^\dagger\)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 updater = 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, \(L_{n+1} \leq S_{n+1}\). 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, \(H_{n+1} \geq S_{n+1}\). 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 \(E_n\) is reset to be either \(L_n\) if reversing to short, or \(H_n\) if reversing to long, and the EMA parameter is reset to, \(\alpha_n = \alpha_i\).

An additional condition is that during an up-trend, the SAR for tomorrow, \(S_{n+1}\), is not allowed to become greater than 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,

\[
S_{n+1} = \min([S_{n+1}, L_{n-1}, L_n]) \cdot (r=-1) + \max([S_{n+1}, H_{n-1}, H_n]) \cdot (r=-1)
\]

The parabolic SAR is implemented with the MATLAB function \texttt{psar}, with usage,

\[
[S,E,a,R] = \text{psar}(H,L,Ri,ni,af,Hi,Li);
\]

\(H = \) 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 \(n<ni\)
\(af = [ai,da,am] = [\text{initial EMA factor, increment, maximum factor}]\)

\(Hi,Li = \) 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,\(^1\)

\[
Y = \text{xlsread}('psarexa.xls');
\]
\[
t = Y(:,1); H = Y(:,2); L = Y(:,3); \% extract H,L signals
\]
\[
Ri = 1; ni = 4; \% initialize SAR
[S,E,a,R] = psar(H,L,Ri,ni); \% extract H,L signals
\]
\[
\text{num2str([t, H, L, a, E, S, R], '8.2f');} \% extract H,L signals
\]
\[
\text{figure; ohlc(t,[H,L], t,S,'r.');} \% make OHLC plot, including SAR
\]

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,\(^2\)

\(^1\) data are from Wilder [865]
\(^2\) 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 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,

<table>
<thead>
<tr>
<th></th>
<th>( n )</th>
<th>( S_{\text{before}}(n) )</th>
<th>( S_{\text{after}}(n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>9</td>
<td>91.5448</td>
<td>95.1875 = ( H_6 )</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>92.3492</td>
<td>85.0625 = ( L_{12} )</td>
</tr>
<tr>
<td></td>
<td>27</td>
<td>89.8936</td>
<td>95.2500 = ( H_{18} )</td>
</tr>
</tbody>
</table>

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].

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,

\[
\begin{align*}
x &= \text{diff}(y); & \text{price differences} \\
x_u &= +x.*\text{d}(x>0); & \text{upward differences} \\
x_d &= -x.*\text{d}(x<=0); & \text{downward differences} \\
su &= \text{wema}(xu,N); & \text{smoothed differences} \\
sd &= \text{wema}(xd,N); \\
RSI &= 100*su/(su+sd); & \% \text{RSI}
\end{align*}
\]

Chande Momentum Oscillator, CMO

If the \text{wema} function is replaced by \text{sma}, one obtains the Chande momentum oscillator,
25.12. MOMENTUM, OSCILLATORS, AND OTHER INDICATORS

```matlab
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

CMO = 100*(su-sd)/(su+sd); % CMO
```

Thus, the SMA-based RSI is related to CMO via,

\[
\text{CMO} = 2 \times \text{RSI} - 100
\]

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 NaNs,

```matlab
x = [NaN; diff(y)]; % y = column of closing prices
for n=N+1:length(y),
y = y(n-N+1:n); % length-N look-back window
x = x(n-N+1:n);
R(n) = abs(max(y)-min(y)) / sum(abs(x)); % 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, \(\pm \text{DI}\), 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); % daily directional movements
Dm = DL ./ (DL>DH); % %

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,

\[
y_r(n) = x(n) - x(n-d) = \text{momentum}
\]

\[
y_r(n) = 100 \cdot \frac{x(n)}{x(n-d)} = \text{momentum as ratio}
\]

\[
y_p(n) = \frac{x(n)-x(n-d)}{x(n-d)} = \text{price rate of change}
\]

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

```

% percent-K
pK = 100 * sma(C-Lmin, Ks) ./ sma(Hmax-Lmin, Ks); %
```

Fast Stochastic has \(K_s = 1\), i.e., no smoothing, and Slow Stochastic has, typically, \(K_s = 3\).
**Accumulation/Distribution**

\[
H = Y(:,1) \quad L = Y(:,2) \quad C = Y(:,3) \quad V = Y(:,4) \quad \% \text{ extract } H,L,C,V
\]

\[
R = \text{cumsum}((2*C - H - L)./(H - L).*V); \quad \% \text{ ACCDIST}
\]

**Chaikin Oscillator**

\[
y = \text{accdist}(Y); \quad \% \text{ Y } = [H,L,C,V] \text{ data matrix}
\]

\[
R = \text{sema}(y,N1) - \text{sema}(y,N2); \quad \% \text{ CHOSC}, \text{ default, } N1=3, N2=10
\]

**Chaikin Money Flow**

\[
H = Y(:,1) \quad L = Y(:,2) \quad C = Y(:,3) \quad V = Y(:,4) \quad \% \text{ extract } H,L,C,V
\]

\[
R = \text{sma}((2*C - H - L)./(H - L).*V, N) \/. \text{sma}(V, N); \quad \% \text{ CMFLOW}
\]

**Chaikin Volatility**

\[
S = \text{sema}(H-L,N); \quad \% \text{ H,L given}
\]

\[
R = (S - \text{delay}(S,N)) \/. \text{delay}(S,N) * 100; \quad \% \text{ volatility}
\]

**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,

\[
P_n = P_{n-1} + (V_n > V_{n-1}) \cdot \frac{C_n - C_{n-1}}{C_{n-1}}, \quad P_{n-1} = P_{n-1} \left(\frac{C_n}{C_{n-1}}\right)^{(V_n > V_{n-1})}
\]

\[
N_n = N_{n-1} + (V_n < V_{n-1}) \cdot \frac{C_n - C_{n-1}}{C_{n-1}}, \quad N_{n-1} = N_{n-1} \left(\frac{C_n}{C_{n-1}}\right)^{(V_n < V_{n-1})}
\]

and initialized to some arbitrary initial value, such as, \(P_0 = N_0 = 1000\). The MATLAB implementation uses the function, \text{delay},

\[
P = P_0 * \text{cumprod}(C./\text{delay}(C,1)) \, ^{\wedge} (V>\text{delay}(V,1)) \); \quad \% \text{ PNVI}
\]

\[
N = P_0 * \text{cumprod}(C./\text{delay}(C,1)) \, ^{\wedge} (V<\text{delay}(V,1)) \); \quad \% \text{ PNVI}
\]

**Commodity Channel Index, CCI**

\[
T = (H+L+C)/3; \quad \% \text{ H,L,C given}
\]

\[
M = \text{sma}(T,N);
\]

\[
\text{for } n=N+1:1:\text{length}(C),
\]

\[
D(n) = \text{mean}(|\text{abs}(T(n-N+1:n) - M(n))|); \quad \% \text{ mean deviation}
\]

\[
\text{end}
\]

\[
R = (T-M)./D/0.015; \quad \% \text{ CCI}
\]

**Detrended Price Oscillator, DPO**

\[
S = \text{sma}(y,N,'n'); \quad \% y = \text{column of closing prices}
\]

\[
M = \text{floor}(N/2)+1; \quad \% \text{ advancing time}
\]

\[
R = y - \text{delay}(S, -M,'n'); \quad \% \text{ DPO, i.e., } R(n) = y(n) - S(n+M)
\]

**Dynamic Momentum Index, DMI**

\[
x = \text{[NaN; diff(y)]}; \quad \% y = \text{column of closing prices}
\]

\[
xu = x \,. \text{ (x>0)}; \quad \% \text{ upward differences}
\]

\[
xd = -x \,. \text{ (x=0)}; \quad \% \text{ downward differences}
\]

\[
S = \text{stdev}(y,Ns); \quad \% \text{ Ns-period stdev}
\]

\[
V = S \,./ \text{sma}(S,Nm); \quad \% \text{ volatility measure}
\]

\[
N = \text{floor}(Nr ./ V); \quad \% \text{ effective length}
\]

\[
N(N>Nmax) = Nmax; \quad \% \text{ restrict max and min N}
\]

\[
N(N<Nmin) = Nmin; \quad \% \text{ restrict max and min N}
\]

\[
N\text{start} = Nr + Ns + Nm;
\]

\[
sul = \text{mean}(\text{xu}(2:N\text{start})); \quad \% \text{ initialize at start time}
\]

\[
sd1 = \text{mean}(\text{xd}(2:N\text{start}));
\]

\[
\text{switch lower(type)}
\]

\[
\text{case 'wema'} \quad \% \text{ WEMA type}
\]

\[
\text{for } n=N\text{start+1:1:length(y),}
\]

\[
su(n) = sul + (\text{xu}(n) - sul) \, / N(n); sul = su(n);
\]

\[
\text{sd(n) = sd1 + (xd(n) - sd1) \, / N(n); sd1 = sd(n);}
\]

\[
\text{end}
\]

\[
\text{case 'sma'} \quad \% \text{ SMA type}
\]

\[
\text{for } n=N\text{start+1:1:length(y),}
\]

\[
su(n) = \text{mean}(\text{xu}(n-N(n)+1:n));
\]

\[
\text{sd(n) = mean}(\text{xd}(n-N(n)+1:n));
\]

\[
\text{end}
\]

\[
R = 100 \, * \text{su.} / \text{su+sd}; \quad \% \text{ DMI}
\]

**Forecast Oscillator**

\[
yp = \text{pma}(y,N,1); \quad \% \text{ time series forecast}
\]

\[
x = y - \text{delay}(yp,1);
\]

\[
R = 100 \, * x ./ y; \quad \% \text{ forecast oscillator}
\]

\[
Rs = \text{sma}(R,Ns); \quad \% \text{ trigger signal}
\]
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 parameters
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
% % ema    - multiple exponential moving average
% % holt   - Holt's exponential smoothing
% % holtorr - MSE, MAE, MAPE error criteria for Holt
%
% Technical Analysis Functions
% ------------------------------------
% % accdist - accumulation/distribution line
% % atr    - true range & average true range
% % ccil   - 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)
%
% Technical Analysis Functions
% ------------------------------------
% % accdist - accumulation/distribution line
% % atr    - true range & average true range
% % ccil   - 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
% % pnvv   - positive and negative volume indices, PVI, NVI
% % proxsc - price oscillator & MACD
% % psar   - Wilder's parabolic SAR
% % rsi    - relative strength index, RSI
% % stdev  - standard deviation index
% % stock  - stochastic oscillator, %S, %D oscillators
% % trix   - TRIX oscillator
% % vhfilt - Vertical Horizontal Filter
% % % % moving averages -------------------------------
% % bma    - Butterworth moving average
% % dema   - steady-state double exponential moving average
% % ema    - exponential Hull moving average
% % gdemax - 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
% % ireg   - linear regression, slope, and R-squared indicators
% % pma    - predictive moving average, linear fit
% % pmaintp - predictive moving average impulse response
% % pmaw2  - predictive moving average, polynomial order d=1,2
% % pmaw2mp2 - predictive moving average impulse response, d=1,2
% % sma    - single exponential moving average
% % sma    - SMA-based Hull moving average
% % sma    - simple moving average
% % t3     - Tillson's T3 indicator, triple dema
% % 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 --------------------------------------
% % bbbands - Bollinger bands
% % donch   - Donchian channels
% % fbbands - fixed-envelope bands
% % kbbands - 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
% % yyylim - 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.¹

Suppose one has identified $M$ stocks or assets into which to invest,

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{bmatrix}$$

where $y_i$ is a random variable that represents the value of the $i$th asset. From historical data, the expected returns of the individual assets are assumed to be known,

$$E[y] = m = \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ m_M \end{bmatrix}$$

as are the cross-correlations between the assets,

$$R_{ij} = E[(y_i - m_i)(y_j - m_j)] , \quad 1 \leq i, j \leq M$$

or, matrix-wise,

$$R = E[(y - m)(y - m)^T]$$

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 $i$th asset.

An investment portfolio is selected by choosing the percentage $a_i$ to invest in the $i$th asset $y_i$. The value of the portfolio is defined by the random variable:

$$y = \sum_{i=1}^{M} a_i y_i = [a_1, a_2, \ldots, a_M] \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{bmatrix} = a^T y$$

where the weights $a_i$ must add up to unity (negative weights are allowed, describing so-called “short sells”):$^2$

$$\sum_{i=1}^{M} a_i = [a_1, a_2, \ldots, a_M] \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} = a^T u = 1$$

The expected return of the portfolio and its variance or risk are given by:

$$\mu = E[y] = E[a^T y] = a^T m = \text{return}$$

$$\sigma^2 = E[(y - \mu)^2] = a^T R a = \text{risk}$$

An optimum portfolio may be defined by finding the weights that minimize the risk $\sigma^2$ for a given value of the return $\mu$, that is,

$$\sigma^2 = a^T R a = \min \quad \text{subject to} \quad a^T m = \mu, \quad a^T u = 1 \quad (25.14.1)$$

(a) Incorporate the constraints by means of two Lagrange multipliers, $\lambda_1, \lambda_2$, and minimize the modified performance index:

$$J = \frac{1}{2} a^T R a + \lambda_1 (\mu - a^T m) + \lambda_2 (1 - a^T u) = \min$$

Show that the quantities $[a, \lambda_1, \lambda_2]$ can be obtained from the solution of the $(M + 2) \times (M + 2)$ linear system of equations:

$$R \begin{bmatrix} -m & -u \\ m^T & 0 & 0 \\ u^T & 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} \mu \\ 0 \\ 1 \end{bmatrix} \quad (25.14.2)$$

where the invertibility of this matrix requires that the vectors $m$ and $u$ not be collinear.

¹Harry Markowitz received the Nobel prize in economics for this work.
(b) Show that the solution of this system for $a$ takes the form:

$$a = \lambda_1 R^{-1}m + \lambda_2 R^{-1}u$$  \hspace{1cm} (25.14.3)

and that $\lambda_1, \lambda_2$ can be obtained from the reduced 2×2 linear system:

$$\begin{bmatrix} A & B \\ B & C \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} \mu \\ 1 \end{bmatrix} \Rightarrow \begin{align*}
\lambda_1 &= \frac{\mu C - B}{D} \\
\lambda_2 &= \frac{A - \mu B}{D}
\end{align*}$$

where, $A, B, C, D$, are defined in terms of $m, R$ by

$$A = m^T R^{-1}m$$

$$B = m^T R^{-1}u$$

$$C = u^T R^{-1}u$$

$$D = AC - B^2$$

(c) Show that the quantities $A, C, D$ are non-negative.

(d) Show that the minimized value of the risk $\sigma^2 = a^T Ra$ can be written in the form:

$$\sigma^2 = \mu \lambda_1 + \lambda_2 = \frac{C \mu^2 - 2B \mu + A}{D}$$  \hspace{1cm} (25.14.4)

Thus, the dependence of the variance $\sigma^2$ on the return $\mu$ 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 $\mu$. Setting $\partial \sigma^2 / \partial \mu = 0$, show that the absolute minimum is reached for the following values of the return, risk, and weights:

$$\mu_0 = \frac{B}{C}, \quad \sigma^2_0 = \frac{1}{C}, \quad a_0 = \frac{R^{-1}u}{u^T R^{-1}u}$$  \hspace{1cm} (25.14.5)

(f) Show that Eq. (25.14.4) can be re-expressed as

$$\sigma^2 = \sigma^2_0 + \frac{C}{D} (\mu - \mu_0)^2$$  \hspace{1cm} (efficient frontier)  \hspace{1cm} (25.14.6)

which can be solved for $\mu$ in terms of $\sigma^2$, as is common in practice:

$$\mu = \mu_0 \pm \frac{\sqrt{D}}{C} \sqrt{\sigma^2 - \sigma^2_0}$$  \hspace{1cm} (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:

$$a = a_0 + \frac{C}{D} (\mu - \mu_0) R^{-1} (m - \mu_0 u)$$  \hspace{1cm} (25.14.8)

where $a_0$ was defined in Eq. (25.14.5). Thus, as expected, $a = a_0$ if $\mu = \mu_0$.

(h) Show that the optimum portfolio of Eq. (25.14.3) can be written in the form

$$a = \mu g + h$$

where $g, h$ depend only on the asset statistics $m, R$ and are independent of $\mu$. Moreover, show that, $m^T g = 1$, and, $m^T h = 0$, and that, $u^T g = 0$, and, $u^T h = 1$. In particular, show that $g, h$ are given by,

$$g = \frac{C}{D} R^{-1}m - \frac{B}{D} R^{-1}u$$

$$h = \frac{A}{D} R^{-1}u - \frac{B}{D} R^{-1}m$$

(i) Consider two optimal portfolios $a_1$ and $a_2$ having return-risk values that lie on the efficient frontier, $\mu_1, \sigma_1$ and $\mu_2, \sigma_2$, 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 $\mu, \sigma$, such that $\mu_1 < \mu < \mu_2$, can be constructed as a linear combination of $a_1, a_2$ as follows, with positive weights $p_1, p_2$, such that, $p_1 + p_2 = 1$,

$$a = p_1 a_1 + p_2 a_2, \quad p_1 = \frac{\mu - \mu_1}{\mu_2 - \mu_1}, \quad p_2 = \frac{\mu_2 - \mu}{\mu_2 - \mu_1}$$

Thus, the investor need only invest in the two standard portfolios $a_1$ and $a_2$ 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 co-variance matrix:

$$m = \begin{bmatrix} 0.02 \\ 0.03 \\ 0.01 \\ 0.05 \end{bmatrix}, \quad R = \begin{bmatrix} 0.10 & -0.02 & 0.04 & -0.01 \\ -0.02 & 0.20 & 0.05 & 0.02 \\ 0.04 & 0.05 & 0.30 & 0.03 \\ -0.01 & 0.02 & 0.03 & 0.40 \end{bmatrix}$$

Make a plot of the efficient frontier of $\mu$ versus $\sigma$ according to Eq. (25.14.7). To do so, choose 51 equally-spaced $\mu$s in the interval $[\min(m), \max(m)]$ and calculate the corresponding $\sigma$s.

Compute the risk $\sigma$ 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, \( \sigma^2 = \mathbf{a}^T \mathbf{R} \mathbf{a} \), and make a scatterplot of the points \((\mu, \sigma)\) to see that they lie on or below the efficient frontier.

\[ J = \frac{1}{2} \mathbf{a}^T \mathbf{R} \mathbf{a} - \mathbf{u}^T \mathbf{m} = \min, \quad \text{subject to } \mathbf{u}^T \mathbf{a} = 1 \]  

(25.14.9)

where \( \mathbf{y} \) is a positive parameter that quantifies the investor’s aversion for risk versus return—small \( \mathbf{y} \) emphasizes low risk, larger \( \mathbf{y} \), high return.

For various values of \( \mathbf{y} \), the optimum weights \( \mathbf{a} \) are determined and then the corresponding return, \( \mu = \mathbf{a}^T \mathbf{m} \), and risk, \( \sigma^2 = \mathbf{a}^T \mathbf{R} \mathbf{a} \), are calculated. The resulting plot of the pairs \((\mu, \sigma)\) is the efficient frontier in this case.

Given the parameters \( \mathbf{y}, \mathbf{m}, \mathbf{R}, \) incorporate the constraint, \( \mathbf{a}^T \mathbf{u} = 1 \), with a Lagrange multiplier and work with the modified performance index:

\[ J = \frac{1}{2} \mathbf{a}^T \mathbf{R} \mathbf{a} - \mathbf{y} \mathbf{a}^T \mathbf{m} + \lambda (1 - \mathbf{u}^T \mathbf{a}) = \min \]

Show that one obtains exactly the same solution for \( \mathbf{a} \) as in the previous problem and that the correspondence between the assumed \( \mathbf{y} \) and the realized return \( \mu \) is given by

\[ \mu = \mu_0 + \frac{D}{C} \mathbf{y} \]

3. **Portfolio with inequality constraints.** If short sales are not allowed, the portfolio weights \( a_i \) must be restricted to be in the interval, \( 0 \leq a_i \leq 1 \). In this case, the following optimization problem must be solved:

\[ \sigma^2 = \mathbf{a}^T \mathbf{R} \mathbf{a} = \min, \quad \text{subject to } \begin{cases} \mathbf{a}^T \mathbf{m} = \mu \\ \mathbf{a}^T \mathbf{u} = 1 \\ 0 \leq a_i \leq 1, \quad i = 1, 2, \ldots, M \end{cases} \]  

(25.14.10)

This is a convex optimization problem that can be solved conveniently using the CVX package.\(^1\) For example, given a desired value for \( \mu \), the following CVX code will solve for \( \mathbf{a} \):

```matlab
% define M, m, R, mu
cvx_begin
variable a(M)
minimize( a'*R*a );
subject to
\quad a'*u == 1;
\quad a'*m == mu;
\quad a <= ones(M,1);
-\mathbf{a} <= zeros(M,1);
cvx_end
```

(a) For the numerical example of Question (1.j), choose 51 equally-spaced \( \mu \) values of the quantities, \( \mu, \sigma \), and, \( \mathbf{a} \), and make a scatterplot of the points \((\mu, \sigma)\) and calculate the corresponding efficient frontier, that is the pairs \((\mu, \sigma)\). Superimpose on this graph the efficient frontier of the unconstrained case from Question 1.

(b) Compute the risk \( \sigma \) 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 \( \mu \), 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 \( \mathbf{y} = [y_1, y_2, \ldots, y_M]^T \) with expected returns \( E[\mathbf{y}] = \mathbf{m} = [m_1, m_2, \ldots, m_M]^T \), and known covariance matrix

\(^1\)http://cvxr.com/cvx
Clearly, we must assume that $\mu_f < m_i$, $i = 1, 2, \ldots, M$, otherwise, we would put all our money into the risk-free asset. We form an optimum portfolio of risky assets $y = a^T y$ 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 $\mu_f$ (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:

$$y_{tot} = w_f \mu_f + (1 - w_f) y = w_f \mu_f + (1 - w_f) a^T y$$

with mean and variance:

$$\mu = E[y_{tot}] = w_f \mu_f + (1 - w_f) a^T m$$

$$\sigma^2 = E[(y_{tot} - \mu)^2] = (1 - w_f)^2 a^T Ra$$  \hspace{1cm} (25.14.11)

The lowest location for $b$ is at the apex of the frontier, for which we have $a = a_0$. 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 $\mu$.

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 $a$ with mean and variance $\mu_a, \sigma_a$.

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 $\beta$, as the Sharpe ratio. Eqs. (25.14.11) become now:

$$\mu = w_f \mu_f + (1 - w_f) \mu_a$$

$$\sigma^2 = (1 - w_f) \sigma_a$$  \hspace{1cm} (25.14.12)

where $\mu_a = a^T m$ and $\sigma_a^2 = a^T Ra$. 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 $(\mu, \sigma)$ plane:

$$\mu = \mu_f + \beta \sigma, \quad \beta = \frac{\mu_a - \mu_f}{\sigma_a}$$  \hspace{1cm} (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 $\mu_a, \sigma_a$. 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 $(\mu_a, \sigma_a)$, we have:

$$\sigma_a^2 = \frac{\sigma_a^2}{\mu_a - \mu_f} + \frac{C}{D} (\mu_a - \mu_f)^2$$

$$a = a_0 + \frac{C}{D} (\mu_a - \mu_f) R^{-1} (m - \mu_f u)$$  \hspace{1cm} (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 $\beta$ of the line (25.14.13):

$$\frac{d\mu_a}{d\sigma_a} = \beta = \frac{\mu_a - \mu_f}{\sigma_a}$$  \hspace{1cm} (25.14.15)

(a) Using condition (25.14.15) and Eq. (25.14.14), show the following relationships:

$$\frac{C}{D} (\mu_a - \mu_f)(\mu_0 - \mu_f) = \frac{1}{C}$$

$$\sigma_a^2 = \frac{C}{D} (\mu_a - \mu_f)(\mu_a - \mu_f)$$

$$\beta = \frac{\sigma_a C}{\mu_a - \mu_f}$$  \hspace{1cm} (25.14.16)

(b) Using Eqs. (25.14.14) and (25.14.16), show that the optimum weights are given by

$$a = \frac{1}{C(\mu_0 - \mu_f)} R^{-1} (m - \mu_f u)$$  \hspace{1cm} (25.14.17)

and verify that they satisfy the constraints $a^T m = \mu_a$ and $a^T u = 1$.

(c) Show that the slope $\beta$ can also be expressed by:

$$\beta^2 = \left(\frac{\mu_a - \mu_f}{\sigma_a}\right)^2 = (m - \mu_f u)^T R^{-1} (m - \mu_f u)$$  \hspace{1cm} (25.14.18)

(d) Define $w = (1 - w_f) a$ to be the effective weight for the total portfolio:

$$y_{tot} = w_f \mu_f + (1 - w_f) a^T y = w_f \mu_f + w^T y$$  \hspace{1cm} (25.14.19)

Show that $w$ is given in terms of the return $\mu = w_f \mu_f + (1 - w_f) \mu_a$ as follows:

$$w = (1 - w_f) a = \frac{\mu - \mu_f}{\beta^2} R^{-1} (m - \mu_f u)$$  \hspace{1cm} (25.14.20)
(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 \( w \) be the weights to be assigned to the risk-free asset \( \mu_f \) and the risky assets \( \mathbf{y} \). Then the total portfolio, its mean \( \mu \), and variance \( \sigma^2 \) are given by:

\[
\begin{align*}
y_{\text{tot}} &= w_f \mu_f + w^T \mathbf{y} \\
\mu &= w_f \mu_f + w^T \mathbf{m} \\
\sigma^2 &= w^T \mathbf{R} w
\end{align*}
\]

(25.14.21)

where the weights must add up to unity: \( w_f + w^T \mathbf{u} = 1 \). Given \( \mathbf{\mu} \), we wish to determine the weights \( w_f, w \) to minimize \( \sigma^2 \). Incorporating the constraints by two Lagrange multipliers, we obtain the performance index:

\[
J = \frac{1}{2} w^T \mathbf{R} w + \lambda_1 (\mathbf{\mu} - w_f \mathbf{\mu}_f - w^T \mathbf{m}) + \lambda_2 (1 - w_f - w^T \mathbf{u}) = \min
\]

Show that the minimization of \( J \) with respect to \( w_f, w \) results in:

\[
w = \lambda_1 \mathbf{R}^{-1} (\mathbf{m} - \mathbf{\mu}_f \mathbf{u}) , \quad \lambda_2 = -\lambda_1 \mu_f
\]

(f) Imposing the constraints show that,

\[
w^T (\mathbf{m} - \mathbf{\mu}_f \mathbf{u}) = \mu - \mu_f , \quad \lambda_1 = \frac{\mu - \mu_f}{\beta^2}
\]

where \( \beta^2 \) is given as in Eq. (25.14.19),

\[
\beta^2 = (\mathbf{m} - \mathbf{\mu}_f \mathbf{u})^T \mathbf{R}^{-1} (\mathbf{m} - \mathbf{\mu}_f \mathbf{u})
\]

and hence, show that \( w \) is given by Eq. (25.14.21)

\[
w = \frac{\mu - \mu_f}{\beta^2} \mathbf{R}^{-1} (\mathbf{m} - \mathbf{\mu}_f \mathbf{u})
\]

(g) Show that \( \sigma^2 = w^T \mathbf{R} w \) is given by,

\[
\sigma^2 = \left( \frac{\mu - \mu_f}{\beta} \right)^2
\]

which corresponds to the straight-line frontier on the \( \mu, \sigma \) plane:

\[
\mu = \mu_f + \beta \sigma
\]

(b) For the numerical example of Question (1.j) and for a fixed-asset return of \( \mu_f = 0.005 \), connect the points \( (\mu_a, \sigma_a) \) 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 \( \mathbf{y}_a = \mathbf{a}^T \mathbf{y} \) be the linear combination of stocks for the market portfolio, and consider another stock \( \mathbf{y}_i \) with return and risk \( \mu_i, \sigma_i \). The stock \( \mathbf{y}_i \) 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 \( R_{ia} = \mathbf{E}[ (\mathbf{y}_i - \mu_i) (\mathbf{y}_a - \mu_a) ] \) and \( R_{aa} = \sigma_a^2 = \mathbf{E}[ (\mathbf{y}_a - \mu_a)^2 ] \).

Let us now make a portfolio \( \mathbf{y} \) consisting of a percentage \( w \) of the stock \( \mathbf{y}_i \) and a percentage \( (1 - w) \) of the market \( \mathbf{y}_a \). Then, \( \mathbf{y} \) and its mean and variance will be given as follows, where \( R_{ii} = \sigma_i^2 \).

\[
\mathbf{y} = w \mathbf{y}_i + (1 - w) \mathbf{y}_a \\
\mu = w \mu_i + (1 - w) \mu_a \\
\sigma^2 = w^2 R_{ii} + 2w(1 - w)R_{ia} + (1 - w)^2 R_{aa}
\]

(25.14.23)

As \( w \) varies, the pair \( (\mu, \sigma) \) varies over its own parabolic efficient frontier. The capital asset pricing model (CAPM) asserts that the tangent line at the market point \( \mathbf{y} = \mathbf{y}_a \) on this frontier obtained when \( w = 0 \), coincides with the capital market line between the risk-free asset \( \mu_f \) and the market portfolio. This establishes the following relationship to be proved below:

\[
\mu_i - \mu_f = \beta_i (\mu_a - \mu_f)
\]

(25.14.24)

so that the excess return above \( \mu_f \), called the risk premium, is proportional to the stock’s beta. The straight line of \( \mu_i \) vs. \( \beta_i \) 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

\[
\frac{d\sigma}{d\mu}\bigg|_{w=0} = \mu_i - \mu_a (\beta_i - 1) \sigma_a
\]

(25.14.25)


6. **Minimum-variance with multiple constraints.** A generalization of the portfolio constrained minimization problem involves more that two constraints:

\[
J = \frac{1}{2} a^T R a - b^T a = \min
\]

subject to \( K \) linear constraints:

\[
c_i^T a = \mu_i, \quad i = 1, 2, \ldots, K
\]

where \( R \) is an \( M \times M \) positive definite symmetric matrix, \( b \) is a given \( M \times 1 \) vector, and we assume that \( K < M \) and that the \( M \times 1 \) vectors \( c_i \) are linearly independent. Defining the \( M \times K \) matrix of constraints \( C \) and the \( K \)-dimensional vector of “returns” \( \mu \),

\[
C = [c_1, c_2, \ldots, c_K], \quad \mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_K \end{bmatrix}
\]

the above minimization problem can be cast compactly in the form:

\[
J = \frac{1}{2} a^T R a - b^T a = \min, \quad \text{subject to} \quad C^T a = \mu
\]

(a) Introduce a \( K \)-dimensional vector of Lagrange multipliers \( \lambda \) and replace the performance index by:

\[
J = \frac{1}{2} a^T R a - b^T a + \lambda^T (\mu - C^T a) = \min
\]

(b) Show that the above matrix has the following inverse:

\[
\begin{bmatrix}
R^{-1} - R^{-1} C (C^T R^{-1} C)^{-1} C^T R^{-1} \\
-C (C^T R^{-1} C)^{-1} C^T R^{-1}
\end{bmatrix}
\]

and explain why the assumed full rank of \( C \) guarantees the existence of the matrix inverse \((C^T R^{-1} C)^{-1}\).

(c) Show that the solution for \( a \) and \( \lambda \) can be obtained by:

\[
\lambda = (C^T R^{-1} C)^{-1} [\mu - C^T R^{-1} b]
\]

\[
a = R^{-1} [b + C \lambda]
\]

(d) Show that the “variance” \( \sigma^2 = a^T R a \) is parabolic in the “returns”, like Eq. (25.14.4), thus defining a generalized “efficient frontier”:

\[
\sigma^2 = \sigma_0^2 + \mu^T (C^T R^{-1} C)^{-1} \mu
\]

where the constant \( \sigma_0^2 \) is defined by:

\[
\sigma_0^2 = b^T [R^{-1} - R^{-1} C (C^T R^{-1} C)^{-1} C^T R^{-1}] b
\]

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 \(E[(\nabla^s \tilde{x})^2]\) 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, \(y(t) = x(t) + \nu(t)\), 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 \(n = 0, 1, \ldots, N - 1\), as shown below.

\[ \begin{array}{c}
\cdots \\
\vdots \\
y_{N-2} \\
y_N \\
y_0 \\
\end{array} \]

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 \([t_0, t_N]\), that is,

\[ t_0 < t_1 < t_2 < \cdots < t_N < t_0 \]

A smoothing spline fits a continuous function \(x(t)\), taken to be the estimate of the underlying smooth trend, by solving the optimization problem:

\[
J = \sum_{n=0}^{N-1} w_n (y_n - x(t_n))^2 + \lambda \int_{t_0}^{t_N} \left[ x^{(s)}(t) \right]^2 dt = \min
\]

(26.1.1)

where \(x^{(s)}(t)\) denotes the \(s\)-th derivative of \(x(t)\), \(\lambda\) 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 \(\lambda\); larger \(\lambda\) increases smoothing, smaller \(\lambda\) interpolates the data more closely.

Schoenberg [595] 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 \(n = 0, 1, \ldots, N - 2\), and it is a polynomial of order \(s - 1\) within the end subintervals \([t_0, t_1)\) and \((t_{N-1}, t_N]\). 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,\(^1\) 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,\(^2\) 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 \(y_n, n = 0, 1, \ldots, N - 1\), are the observations, and \(x_n\) the resulting smoothed estimate,

\[
J = \sum_{n=0}^{N-1} w_n |y_n - x_n|^2 + \lambda \sum_{n=s}^{N-1} |\nabla^s x_n|^2 = \min
\]

(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

\[
D_s(z) = (1 - z^{-1})^s
\]

\[
d_s(k) = (-1)^k \binom{s}{k}, \quad 0 \leq k \leq s
\]

\(^1\)adapted from the author’s book on Applied Optimum Signal Processing [45]

\(^2\)Bohlmann considered the case \(s = 1\), Whittaker and Henderson, \(s = 3\), and Hodrick-Prescott, \(s = 2\).
For example, we have for \( s = 1, 2, 3 \),

\[
d_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \quad d_2 = \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}, \quad d_3 = \begin{bmatrix} 1 \\ -3 \\ 3 \\ -1 \end{bmatrix}
\]

Because \( D_d(z) \) 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 \( D_d(z) \) may be replaced by any other (causal) FIR highpass filter \( D(z) \), or \( d_s \) in the time domain, with a similar result. Thus, a more general version of (26.2.1) would be:

\[
\mathcal{J} = \sum_{n=0}^{N-1} w_n |y_n - x_n|^2 + \lambda \sum_{n=s}^{N-1} |d_n \ast x_n|^2 = \min
\]

where \( d_n \ast x_n \) denotes convolution and \( s \) is the filter order, that is, we assume that the impulse response is \( d_n = [d_0, d_1, \ldots, d_s] \). 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 \([x_0, x_1, \ldots, x_{N-1}]\), the \( s \)-difference filter has the full convolutional output:

\[
\sum_{k=0}^{s} d_s(k) x_{n-k}, \quad 0 \leq n \leq N - 1 + s
\]

and the steady-state output (assuming \( N > s \)):

\[
g_n = \sum_{k=0}^{s} d_s(k) x_{n-k}, \quad s \leq n \leq N - 1
\]

Similarly, we have in the more general case,

\[
g_n = \sum_{k=0}^{s} d_s(k) x_{n-k}, \quad 0 \leq n \leq N - 1 + s
\]

\[
g_n = \sum_{k=0}^{s} d_s(k) x_{n-k}, \quad s \leq n \leq N - 1
\]

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 \textit{binom} and \textit{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 \( g_n = \mathcal{V}^s x_n \), \( 0 \leq n \leq N - 1 + s \), can be expressed vectorially as \( g = D_s x \), where \( x \) is the \( N \)-dimensional input vector \( x = [x_0, x_1, \ldots, x_{N-1}]^T \), and \( g = [g_0, g_1, \ldots, g_{N-1+s}]^T \), the \((N+s)\)-dimensional output vector. Similarly, the operation \( g_n = d_n \ast x_n \) can be expressed as \( g = D_{full} x \), where the \((N+s) \times N\) full convolution matrix can be constructed using \textit{convmat}—the sparse version of \textit{convmtx},

\[
D_{full} = \text{convmat}(d, N);
\]

where \( d = [d_0, d_1, \ldots, d_s]^T \). 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 \( d = [d_0, d_1, d_2]^T \),

\[
D_{full} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
0 & d_2 & d_1 & d_0 & 0 \\
0 & 0 & d_2 & d_1 & d_0 \\
0 & 0 & 0 & d_2 & d_1 \\
0 & 0 & 0 & 0 & d_2 \\
\end{bmatrix}
\]

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:

1) \( D_{full} = \text{convmat}(d, N); \quad D = D_{full}(s + 1 : N,:) \)

2) \( D = \text{convmat}(\text{flip}(d), N-s,:) \);

For the \( s \)-difference filter, we can use the equivalent (sparse) constructions:

1) \( D_{full} = \text{diffmat}(s, N); \quad D = D_{full}(s + 1 : N,:) \)

2) \( D = (-1)^s \ast \text{diffmat}(s, N-s,:) \);

3) \( D = \text{diff}(\text{speye}(N), s) \);

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 \( d \). As an example, we have for \( N = 7, s = 2 \), and \( d = [1, -2, 1]^T \):

\[
D = \begin{bmatrix}
1 & -2 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & -2 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & -2 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & -2 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & -2 & 1 \\
\end{bmatrix}
\]

The corresponding steady-state output vector \( g = [g_0, g_1, \ldots, g_{N-1}]^T \) is given by \( g = D x \), with squared norm,

\[
\sum_{n=0}^{N-1} (d_n \ast x_n)^2 = \sum_{n=0}^{N-1} g_n^2 = \mathcal{J} (D^T D) x = \min
\]

Therefore, the performance index (26.2.1) or (26.2.3) can be written compactly as:

\[
\mathcal{J} = (y - x)^T W (y - x) + \lambda x^T (D^T D) x = \min
\]
where $W$ is the diagonal matrix of the weights, $W = \text{diag}(\{w_0, w_1, \ldots, w_{N-1}\})$. The optimum solution is obtained by setting the gradient with respect to $x$ to zero,

$$\frac{\partial J}{\partial x} = -2W(y-x) + 2\lambda(D^T D)x = 0 \Rightarrow (W + \lambda D^T D)x = Wy$$

with solution, which may be regarded as the estimate of $x$ in the signal model $y = x + v$,

$$\hat{x} = (W + \lambda D^T D)^{-1}Wy$$  \hspace{1cm} (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 $\nabla^{2s}$, after ignoring the first $s$ and last $s$ rows. For example, we have for $N = 7$ and $s = 2$,

$$D^T D = \begin{bmatrix}
1 & -2 & 1 & 0 & 0 & 0 & 0 \\
-2 & 5 & -4 & 1 & 0 & 0 & 0 \\
1 & -4 & 6 & -4 & 1 & 0 & 0 \\
0 & 1 & -4 & 6 & -4 & 1 & 0 \\
0 & 0 & 1 & -4 & 6 & -4 & 1 \\
0 & 0 & 0 & 1 & -4 & 5 & -2 \\
0 & 0 & 0 & 0 & 1 & -2 & 1
\end{bmatrix}$$

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):

$$x = \text{whsm}(y, \lambda, s, w);$$  \hspace{1cm} % 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 $d$, whose steady convolution matrix $D$ is also computed by method (2):

$$x = \text{whgen}(y, \lambda, d, w);$$  \hspace{1cm} % generalized Whittaker-Henderson smoothing

Denoting the “hat” filtering matrix $H = (W + \lambda D^T D)^{-1}W$, and defining the error $e = y - \hat{x} = (I - H)y$, we may define a generalized cross-validation criterion for determining the smoothing parameter $\lambda$,

$$\text{GCV}(\lambda) = \frac{e^T We}{\text{tr}(I - H)} = \min$$  \hspace{1cm} (26.2.11)

The function whgcv calculates it at any vector of $\lambda$’s and finds the corresponding optimum:

$$[\text{gcv}, \text{lopt}] = \text{whgcv}(y, \lambda, s, w);$$  \hspace{1cm} % 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 $\lambda$.

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:


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 $\lambda_{opt} = 6.6$, 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:

```matlab
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 = 4.4331088700E+00; b5 = -1.6231428586E+00; b6 = 5.2554937560E-01; b7 = 2.6887614440E+00; b8 = 2.1232284848E-01; b9 = 1.4966870418E+00; yf = b1 + b2*cos(2*pi*t/12) + b3*sin(2*pi*t/12) + b4*cos(2*pi*t/b4) + b5*sin(2*pi*t/b4) + b6*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);  % GCV evaluation

Y = whsm(y,lopt,s);  % WH smoothing method
M=20; r=6; ye = svdenvh(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

% SVD enhancement method
figure; plot(t,y,':', t,ye,'.-', t,yf,'--');  % upper-right graph
```
26.3. REGULARIZATION FILTERS

26.3.1 WHITTAKER-HENDERSON SMOOTHING

Using the Parseval identity, the performance index (26.2.3) becomes:

\[
\mathcal{J} = \sum_{n=-\infty}^{\infty} |y_n - x_n|^2 + \lambda \sum_{n=-\infty}^{\infty} |d_n * x_n|^2 =
\]

\[
= \int_{-\pi}^{\pi} |Y(\omega) - X(\omega)|^2 \frac{d\omega}{2\pi} + \lambda \int_{-\pi}^{\pi} |D(\omega)X(\omega)|^2 \frac{d\omega}{2\pi} = \min
\]

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 \(X^*(\omega)\),

\[
\frac{\delta \mathcal{J}}{\delta X^*(\omega)} = X(\omega) - Y(\omega) + \lambda |D(\omega)|^2 X(\omega) = 0
\]

gives the effective equivalent smoothing filter \(H(\omega) = X(\omega)/Y(\omega)\):

\[
H(\omega) = \frac{1}{1 + \lambda |D(\omega)|^2}
\]

(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 \(z = e^{i\omega}\), so that,

\[
H(z) = \frac{1}{1 + \lambda D(z)D(z^{-1})}
\]

(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:

\[
H(z) = \frac{1}{1 + \lambda (1 - z^{-1})^s(1 - z)^s}
\]

(Whittaker-Henderson filter) (26.3.5)

Similarly, Eq. (26.3.2) can be written in the \(z\)-domain and converted back to the time domain. Noting that \(D_s(z)D(z^{-1}) = (1 - z^{-1})^s(1 - z)^s = (-1)^s s! (1 - z^{-1})^{2s} = (-1)^s s! D_{2s}(z)\), we have:

\[
X(z) - Y(z) + \lambda (-1)^s z^s (1 - z^{-1})^{2s} X(z) = 0, \quad \text{or,}
\]

\[-1)^s s! D_{2s}(z)X(z) = \lambda^{-1} (Y(z) - X(z)),
\]

resulting in the time-domain \(2s\)-difference equation:

\[
(-1)^s s! \nabla^{2s} x_{n+s} = \lambda^{-1} (y_n - x_n)
\]

(26.3.6)

In the early years, some ingenious methods were developed for solving this type of equation [1009–1018]. Noting that \(|D_s(\omega)| = |1 - e^{-j\omega}|^s = 2^s |\sin(\omega/2)|^s\), we obtain the frequency response of (26.3.5):

\[
H(\omega) = \frac{1}{1 + \lambda \left|\frac{\sin(\omega/2)}{2\sin(\omega/2)}\right|^{2s}}
\]

(26.3.7)

\[\text{Fig. 26.2.1 Smoothed ENSO signal and spectra.}\]
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:

$$H_c(z) = \frac{\lambda(1 - z^{-1})^s(1 - z^s)}{1 + \lambda(1 - z^{-1})^s(1 - z^s)}; \quad H_c(\omega) = \frac{\lambda}{1 + \lambda} \left( \frac{\omega_c}{\omega} \right)^{2s}. \tag{26.3.8}$$

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 $\lambda$ narrows the response of the lowpass filter and widens the response of the highpass one.

![Frequency responses of Whittaker-Henderson filters.](image)

### 26.4 Hodrick-Prescott Filters

In macroeconomic applications such as extracting business cycles from GDP data, the standard signal model $y_n = x_n + \nu_n$ is interpreted to consist of a long-term trend represented by $x_n$ and a short-term cyclical component $\nu_n$. 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 $\lambda$ can be fixed by specifying a desired value for the filter’s cutoff frequency $\omega_c$ corresponding to some standardized value of the gain. For the lowpass filter we have for general $s$, the condition:

$$\frac{1}{1 + \lambda} \left( \frac{\omega_c}{\omega} \right)^{2s} = G_c \tag{26.4.1}$$

where $G_c$ is desired value of the gain. The 3-dB cutoff frequency $\omega_c$ 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:

$$\frac{\lambda}{1 + \lambda} \left( \frac{\omega_c}{\omega} \right)^{2s} = G_c \left( 1 + 2^{2s} \lambda \right). \tag{26.4.2}$$

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 $\omega_c$ (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 $G_c = 0.702667 \approx -3.065 \text{ dB}$.

Using the same $\omega_c$ and $G_c$, but different values of $s$ requires adjusting the value of $\lambda$. For example, solving Eq. (26.4.2) for $\lambda$ 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 $\lambda$ must be adjusted if the sampling frequency is changed. For example, the same cutoff frequency expressed in different units is:

$$\omega_c = \frac{2\pi}{32} \text{ radians/quarter} = \frac{2\pi}{8} \text{ radians/year} = \frac{2\pi}{96} \text{ radians/month}. \tag{26.4.4}$$

The above value $G_c = 0.702667$ used in (26.4.2) with $s = 2$ then gives the following values of $\lambda$ 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 $\lambda$ 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],

$$\lambda = 1600 / 4^2 = 100 \ (\text{yearly}), \quad \lambda = 1600 \times 3^2 = 14400 \ (\text{monthly}) \tag{26.4.7}$$

$$\lambda = 1600 / 4^4 = 6.25 \ (\text{yearly}), \quad \lambda = 1600 \times 3^4 = 129600 \ (\text{monthly}) \tag{26.4.7}$$

The latter choice is essentially the same as that of Eq. (26.4.5) based on the criterion (26.4.2). Indeed, for small $\omega_c$, 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 $\omega_c \leq 0.3\pi$, 

$$\frac{\lambda \omega_c^2}{1 + \lambda \omega_c^2} = G_c \Rightarrow \lambda = \frac{G_c}{(1 - G_c) \omega_c^2} \quad (26.4.8)$$

If in this formula, we adjust $G_c$ to get $\lambda = 1600$ at $\omega_c = 2\pi/32$, we find $G_c = 0.70398 \approx -3.049$ dB, which in turn generates the second set of values in Eq. (26.4.7).

**Example 26.4.1: US GDP for investment.** A prototypical 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:

```matlab
Y = loadfile('USGDP_Inv.dat'); % data file in AOSP toolbox
y = log10(Y(:,2) * 1e9); % $\log_{10}(\text{GDP})$
% t-axis in quarters since 1947
t = taxis(y,4,1947);

s = 2; la = 1600; yt = whsm(y,la,s); % WH smoothing with $s = 2$
fig; plot(t,yt,'-'); % upper-left graph
yc = y-yt; % cyclical component
fig; plot(t,yc,'-'); % uppper-right graph

s = 3; la = 41640.16; yt = whsm(y,la,s);
fig; plot(t,yt,'-'); % WH smoothing with $s = 3$
fig; plot(t,yc,'-'); % upper-right graph

M=9; r=1; ye = svdenh(y,M,r); % SVD enhancement method
fig; plot(t,ye,'-'); % bottom-left graph
fig; plot(t,ye,'-'); % 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.

### 26.5 Poles and Impulse Response

Because of the invariance under the substitution $z \rightarrow z^{-1}$, 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:

$$H(z) = \frac{1}{1 + \lambda (1 - z^{-1})^2(1 - z)^2} = \prod_{k=1}^{s} \left(\frac{1 - Z_k^2}{1 - Z_k z^{-1}}\right) \quad (26.5.1)$$

where $z_k, k = 1, 2, \ldots, s$, 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]:

$$h_n = \int_{-\infty}^{\infty} H(z) z^n \frac{dz}{2\pi i z} \quad -\infty < n < \infty \quad (26.5.2)$$

Inserting the factored form (26.5.1) into (26.5.2), we find

$$h_n = \sum_{k=1}^{s} A_k z_k^n, \quad -\infty < n < \infty \quad (26.5.3)$$
where the coefficients $A_k$ are given by

$$A_k = \frac{1 - z_k^2}{1 + z_k^2} \prod_{i=1, i \neq k}^s \frac{(1 - z_i^2)}{(1 - z_i z_k)(1 - z_i z_k^{-1})}, \quad k = 1, 2, \ldots, s$$  \hspace{1cm} (26.5.4)$$

The poles can be obtained in the form $z_k = e^{i \omega_k}$, where $\omega_k$ are the complex frequencies of the denominator, that is, the frequencies that are solutions of the equation:

$$1 + \lambda \left[ 2 \sin \frac{\omega}{2} \right]^{2s} = 0$$  \hspace{1cm} (26.5.5)$$

where we note that even though $\omega$ is complex, we still have $s = \frac{\pi}{2}$ for $z = e^{i \omega}$. The solution of Eq. (26.5.5) is straightforward. The $s$ frequencies $\omega_k$ that lead to poles $z_k$ that are inside the unit circle can be parametrized as follows:

$$2 \sin \frac{\omega}{2} = \frac{-1}{\lambda} = \frac{e^{i \pi (2k - 1)}}{\lambda} \Rightarrow \sin \frac{\omega_k}{2} = \frac{e^{i \pi (2k - 1)/2s}}{2 \lambda^{1/2s}}, \quad k = 1, 2, \ldots, s$$  \hspace{1cm} (26.5.6)$$

Thus, the desired set of poles are:

$$z_k = e^{i \omega_k}, \quad \omega_k = 2 \arcsin \left[ \frac{e^{i \theta_k}}{2 \lambda^{1/2s}} \right], \quad \theta_k = \frac{\pi (2k - 1)}{2s}, \quad k = 1, 2, \ldots, s$$  \hspace{1cm} (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 $n$s and also produces the poles and residues $z_k, A_k, k = 1, 2, \ldots, s$.

\[
\text{[h,z,A] = whimp(lambda,s,n);}
\]

Fig. 26.5.1 shows the impulse responses for $s = 1, 2, 3$ with $\lambda$ 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,

\[
J = \sum_n |y_n - f_n * x_n|^2 + \lambda \sum_n |d_n * x_n|^2 = \text{min} \hspace{1cm} (26.6.1)
\]

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:

\[
J = \|y - Fx\|^2 + \lambda \|Dx\|^2 = (y - Fx)^T (y - Fx) + \lambda x^T (D^T D)x = \text{min} \hspace{1cm} (26.6.2)
\]

The solution is obtained from the gradient,

\[
\frac{\partial J}{\partial x} = -2F^T (y - Fx) + 2\lambda D^T D x = 0, \quad \text{or,}
\]

\[
x = (F^T F + \lambda D^T D)^{-1} F^T y \hspace{1cm} (26.6.3)
\]

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, $y = Fx$, may in general be overdetermined, or underdetermined, or rank deficient. 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, $x = (F^T F)^{-1} F^T y$, which is recognized as the unique pseudoinverse least-squares solution. In the square case, we have, $x = F^{-1} y$. We are envisioning a signal model of the form, $y = Fx + v$, and the objective is to determine an estimate of $x$. We have then,

\[
x = F^{-1} y = F^{-1} (F x + v) = x + F^{-1} v \hspace{1cm} (26.6.4)
\]

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 $u = F^{-1} v$ may be magnified to such an extent that it will mask the desired term $x$, rendering the estimate $x$ 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,

\[
x = (F^T F + \lambda I)^{-1} F^T y \hspace{1cm} (26.6.5)
\]
To see how regularization improves the condition number, let $\lambda_{\text{max}}, \lambda_{\text{min}}$ be the maximum and minimum eigenvalues of $F^T F$. Then, the condition numbers of $F^T F$ and $F^T F + \lambda I$ are $\lambda_{\text{max}}/\lambda_{\text{min}}$ and $(\lambda_{\text{max}} + \lambda)/(\lambda_{\text{min}} + \lambda)$. A highly ill-conditioned problem would have $\lambda_{\text{min}} \ll \lambda_{\text{max}}$. It is straightforward to verify that the larger the $\lambda$, the more the condition number of the regularized matrix is reduced:

$$
\lambda \gg 1 \Rightarrow \frac{\lambda_{\text{max}} + \lambda}{\lambda_{\text{min}} + \lambda} \ll \frac{\lambda_{\text{max}}}{\lambda_{\text{min}}}
$$

For example, if $\lambda_{\text{min}} = 10^{-3}$ and $\lambda_{\text{max}} = 10^3$, we have $\lambda_{\text{max}}/\lambda_{\text{min}} = 10^6$, but the regularized version $(\lambda_{\text{max}} + \lambda)/(\lambda_{\text{min}} + \lambda)$ 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 $\lambda$ causes more distortion and smoothing of the desired signal component $x$. As usual, choosing the proper value of $\lambda$ 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, $y_n = f(t_n) + v_n$, $n = 0, 1, \ldots, N - 1$.

$$
J = \sum_{n=0}^{N-1} [y_n - f(t_n)]^2 + \lambda \int_a^b [f(t)]^2 dt = \min
$$

(26.6.6)

The concept can be generalized from functions of time to multivariable functions of some independent variable, say $x$, such as three-dimensional space. The observed data samples are of the form, $y_n = f(x_n) + v_n$, and the objective is to learn the unknown function $f(x)$. The performance index (26.6.6) is replaced by,

$$
J = \sum_{n=0}^{N-1} [y_n - f(x_n)]^2 + \lambda \|f\|^2 = \min
$$

(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 $f(x)$. 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 $f(x)$ can be represented as a linear combination of a finite or infinite set of nonlinear basis functions $\phi_i(x)$, $i = 1, 2, \ldots, M$, where for now we will assume that $M$ is finite,

$$
f(x) = \sum_{i=1}^M \phi_i(x) c_i = \Phi^T(x) c, \quad \Phi(x) = \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \\ \vdots \\ \phi_M(x) \end{bmatrix}, \quad c = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_M \end{bmatrix}
$$

(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 $s_j(t) = t^j$. Here, we define the regularization norm in terms of the coefficients $c_i$ as follows:

$$
\|f\|^2 = \sum_{i=1}^M c_i^2 = c^T \Lambda^{-1} c
$$

(26.6.9)

where $\Lambda_i$ is a set of some given positive coefficients, and $\Lambda = \text{diag}([\lambda_1, \lambda_2, \ldots, \lambda_M])$. The function values at the $N$ observation points are $f(x_n) = \Phi^T(x_n) c$, and can be arranged into an $N$-dimensional column vector,

$$
f = \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_{N-1}) \end{bmatrix} = \Phi c, \quad \Phi = \Phi^T(x_n) \begin{bmatrix} \phi^T(x_0) \\ \phi^T(x_1) \\ \vdots \\ \phi^T(x_{N-1}) \end{bmatrix}
$$

(26.6.10)

where $\Phi$ has dimension $N \times M$. Thus, the performance index can be written compactly,

$$
J = (y - \Phi c)^T(y - \Phi c) + \lambda c^T \Lambda^{-1} c = \min
$$

(26.6.11)

The solution for the optimum coefficients $c$ is obtained by setting the gradient to zero:

$$
\frac{\partial J}{\partial c} = -2(\Phi^T(y - \Phi c) + 2\lambda \Lambda^{-1} c) = 0 \Rightarrow (\lambda \Lambda^{-1} + \Phi^T \Phi) c = \Phi^T y
$$

(26.6.12)

$$
c = (\lambda \Lambda^{-1} + \Phi^T \Phi)^{-1} \Phi^T y
$$

(26.6.13)

Using the matrix-inversion lemma, we have:

$$
(\lambda \Lambda^{-1} + \Phi^T \Phi)^{-1} = \frac{1}{\lambda} [\Lambda - \Lambda \Phi^T (\lambda I + \Phi \Lambda \Phi^T)^{-1} \Lambda] \Phi
$$

(26.6.14)

from which it follows that:

$$
(\lambda \Lambda^{-1} + \Phi^T \Phi)^{-1} \Phi^T y = \Lambda \Phi^T (\lambda I + \Phi \Lambda \Phi^T)^{-1} \Phi^T y
$$

(26.6.15)

where $I$ is the $N \times N$ identity matrix. Thus, the optimal coefficients are given by

$$
c = \Lambda \Phi^T (\lambda I + \Phi \Lambda \Phi^T)^{-1} y
$$

(26.6.16)

The observation vector $f = \Phi c$ and estimated function value $\hat{f}(x) = \Phi^T(x) c$ are then,

$$
\hat{f}(x) = \Phi \Lambda \Phi^T (\lambda I + \Phi \Lambda \Phi^T)^{-1} y
$$

(26.6.17)
The linear vector space of functions of the form $f(x) = \Phi^T(x)c$, spanned by the set of basis functions $\{\phi_i(x), i = 1, 2, \ldots, M\}$, 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 $f_1(x) = \Phi^T(x)c_1$ and $f_2(x) = \Phi^T(x)c_2$, we define the inner product:

$$\langle f_1, f_2 \rangle = c_1^T A^{-1} c_2$$  

(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, $K(x, x') = \Phi^T(x)A\Phi(x') = \Phi^T(x)c(x')$, with $c(x') = A\Phi(x')$, we see that, as a function of $x$ for each fixed $x'$, it lies in the space $\mathcal{H}$, and satisfies the two reproducing-kernel properties:

$$f(x) = \langle f, K(\cdot, x') \rangle, \quad K(x', x') = \langle K(\cdot, x), K(\cdot, x') \rangle$$  

(26.6.26)

These follow from the definition (26.6.25). Indeed, given $f(x) = \Phi^T(x)c$, we have,

$$\langle f(\cdot), K(\cdot, x') \rangle = c^T A^{-1} c(x') = c^T A^{-1} A\Phi(x') = c^T \Phi(x') = f(x')$$

$$\langle K(\cdot, x), K(\cdot, x') \rangle = c(x) A^{-1} c(x') = \Phi^T(x)A A^{-1} A\Phi(x') = \Phi^T(x)A\Phi(x') = K(x, x')$$

One can re-normalize the basis functions by defining $\bar{\phi}_i(x) = \lambda_i^{-1/2} \phi_i(x)$, or, vectorially $\bar{\Phi}(x) = A^{-1/2} \Phi(x)$, which imply the renormalized basis matrix $\bar{\Phi} = \Phi A^{-1/2}$ and coefficient vector $\bar{c} = A^{-1/2} c$. We obtain then the alternative expressions:

$$\bar{c} = \Phi^T a$$

$$f = \Phi c = \Phi \bar{c}$$

$$\|f\|^2 = c^T A^{-1} c = \bar{c}^T \bar{c}$$

(26.6.27)

and kernel function,

$$K(x, x') = \Phi^T(x)\Lambda\Phi(x') = \bar{\Phi}^T(x)\bar{\Phi}(x')$$  

(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 $K(x, x')$ 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],

$$f(x) = \sum_{n=0}^{N-1} K(x, x_n) a_n$$  

(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:

$$J = L(y - \Phi c) + \lambda c^T A^{-1} c = \min$$  

(26.6.30)
26.7 Sparse Whittaker-Henderson Methods

where \( L(z) \) is an arbitrary (convex, increasing, and differentiable) scalar function that replaces the quadratic norm \( L(z) = z^Tz \). Indeed, the vanishing of the gradient gives,

\[
\frac{\partial J}{\partial c} = -\phi^T \frac{\partial L}{\partial y} + 2\lambda \lambda^{-1} c = 0 \quad \Rightarrow \quad c = \Lambda \phi^T \frac{1}{2\lambda} \frac{\partial L}{\partial y}
\]

which implies for \( f = \phi c \) and \( f(x) = \phi^T(x) c \),

\[
f = Ka, \quad f(x) = \sum_{n=0}^{N-1} K(x, x_n) a_n, \quad \text{with} \quad a = \frac{1}{2\lambda} \frac{\partial L(y - Ka)}{\partial y} \quad (26.6.31)
\]

where the last equation is a nonlinear equation for the \( N \)-vector \( a \). Of course, in the quadratic-norm case, \( L(z) = z^Tz \), we obtain the equivalent of (26.6.19),

\[
a = \frac{1}{\lambda} (y - Ka)
\]

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:

\[
K(x, x') = (c + x \cdot x')^p \\
K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right) \quad (26.6.32)
\]

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 \( L_p \) and the \( L_{\infty} \) norms, and using unity weights \( w_n \) for simplicity,

\[
J_{fp} = \sum_{n=0}^{N-1} |y_n - x_n|^p + \lambda \sum_{n=1}^{N-1} |\nabla x_n|^p = \min \quad (26.7.1)
\]

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),

\[
J_{fp} = ||y - x||_p^p + \lambda ||Dx||_p^p = \min \quad (26.7.2)
\]

where \( ||x||_p \) denotes the \( L_p \) norm of the vector \( x = [x_0, x_1, \ldots, x_{N-1}]^T \),

\[
||x||_p = \left[ \sum_{n=0}^{N-1} |x_n|^p \right]^{1/p} \Rightarrow ||x||_p^p = \sum_{n=0}^{N-1} |x_n|^p
\]

and similarly for \( ||x||_r \). For \( p = \infty \), we have instead,

\[
||x||_\infty = \max_{0 \leq n < N-1} |x_n|
\]

For \( p = 0 \), we define \( ||x||_0 \) as the cardinality of the vector \( x \), that is, the number of nonzero elements of \( x \). We note that \( ||x||_p \) is a proper norm only for \( p \geq 1 \), however, the cases \( 0 \leq p < 1 \) have also been considered.

The case \( J_{11} \) was studied in [1024, 1028] and formulated as a linear programming problem, the case \( J_{pp} \), including the \( L_{\infty} \) norm case, \( p = \infty \), was studied in [1026], and the more general case, \( J_{rp} \), in [1027]. More recently, the case \( J_{21} \), called \( L_1 \) trend filtering, has been considered in [1070] and has received a lot of attention [1071–1080].

Generally, the cases \( J_{rp} \) 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 \( J_{22} \) criterion, and the \( J_{21} \) and \( J_{20} \) criteria,

\[
J_{22} = ||y - x||_2^2 + \lambda ||Dx||_2^2 = \min \\
J_{21} = ||y - x||_2^2 + \lambda ||Dx||_1 = \min \\
J_{20} = ||y - x||_2^2 + \lambda ||Dx||_0 = \min \quad (26.7.3)
\]

The \( J_{21} \) and \( J_{20} \) criteria tend to promote the sparsity of the regularizing term \( Dx \), that is, \( Dx \) will be a sparse vector consisting mostly of zeros with a few nonzero entries. Since \( Dx \) represents the \( s \)-differenced signal, \( \nabla x_n \), 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 \( \nabla x_n \) 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 differenting 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 \( J_{21} \) problem can be implemented easily in MATLAB with the CVX package. The \( J_{20} \) 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 \( J_{21} \) and the \( J_{2p} \) 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 \( L_p \)
norm with a weighted $L_2$ norm, which can be solved iteratively. Given any real number $0 \leq p \leq 2$, let $q = 2 - p$, and note that for any real number $x \neq 0$, we can write,

$$|x|^p = \frac{|x|^2}{|x|^{q+\varepsilon}} = \frac{|x|^2}{|x|^{q} + \varepsilon}$$

where $\varepsilon$ 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 $x \in \mathbb{R}^N$,

$$||x||_p^p = \sum_{i=0}^{N-1} |x_i|^p = \sum_{i=0}^{N-1} \frac{|x_i|^2}{|x_i|^{q} + \varepsilon} = x^T W(x) x$$

where $W(x) = \text{diag} \left[ \frac{1}{|x|^q + \varepsilon}, \frac{1}{|x_1|^q + \varepsilon}, \ldots, \frac{1}{|x_{N-1}|^q + \varepsilon} \right]$. This case has wide application in image denoising 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.

Next, we consider a number of computer experiment examples that illustrate and compare the performances 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.

### 26.8 Computer Experiments

The solution of the optimization problem in (26.7.6) at the $k$th step is:

$$x^{(k)} = (I + \lambda D^T W_{k-1} D)^{-1} y$$

Thus, the choices $p = 0$ and $p = 1$ should resemble the solutions of the $L_0$ and $L_1$ regularized problems.
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 $\lambda$, the best practice is by trial-and-error. There exist criteria for choosing $\lambda$, however, they are not particularly good. The higher the $\lambda$, the more emphasis is placed on minimizing $\nabla^s x_n$, and the smoother the solution. Too high a value of $\lambda$ 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 $\lambda$’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

$$y_n = x_n + v_n, \quad 0 \leq n \leq 600$$

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,

```matlab
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 $\hat{x}_n$, 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 $s, \lambda$, solve the $L_1$ problem using the CVX package, and then using the IRLS algorithm, and for both cases, plot the estimated signal $\hat{x}_n$, 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 $\hat{x}_n$, 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 $L_2, L_1, L_0$ 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.

### 26.8.2 Local Linear Trends

Consider a piecewise linear signal $x_n$ and its observed noisy version, $y_n$, defined by

$$y_n = x_n + v_n, \quad 0 \leq n \leq 600$$

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,

```matlab
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.
26.8. COMPUTER EXPERIMENTS

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.

```matlab
Y = loadfile('tavenh2v.dat'); % load temperature data file
n = Y(:,1); y = Y(:,14); N = length(y); % extract dates and data
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. 26.8.3 Total variation minimization example of Sec. 26.8.1.

Fig. 26.8.4 Total variation minimization example of Sec. 26.8.1.

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.
26.8. COMPUTER EXPERIMENTS

Fig. 26.8.7 Total variation minimization example of Sec. 26.8.1.

Fig. 26.8.8 Local linear trends – Sec. 26.8.2.

```matlab
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).

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, $y_n, 0 \leq n \leq N - 1$, the optimization criteria for determining a length-$N$ smoothed signal $x_n$ are,

$$(L_2): \quad J = \sum_{n=0}^{N-1} |y_n - x_n|^2 + \lambda \sum_{n=s}^{N-1} |\nabla^2 x_n|^2 = \|y - x\|^2 + \lambda \|D_2 x\|^2 = \min$$

$$(L_1): \quad J = \sum_{n=0}^{N-1} |y_n - x_n|^2 + \lambda \sum_{n=s}^{N-1} |\nabla^2 x_n| = \|y - x\|^2 + \lambda \|D_2 x\|_1 = \min$$

(26.8.3)
where $D_s$ is the $(N - s) \times N$ convolution matrix corresponding the $s$-difference operator $\nabla^s$. It can be constructed in MATLAB by:

$$D_s = \text{diff}(	ext{eye}(N), s); \quad \text{or, in sparse form, } D_s = \text{diff}(	ext{speye}(N), s);$$

The solution of problem $(L_2)$ is straightforward:

$$x = (I + \lambda D_s^T D_s)^{-1} y \tag{26.8.4}$$

The solution of problem $(L_1)$ can be obtained with the CVX package as follows:

```matlab
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.

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:

```matlab
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 $x$ together with the actual data $y$. 
b. Calculate the SVD enhanced version of \( y \) by the following steps: (i) remove and save the mean from \( y \), (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 \( y \). SVD enhancement is discussed in detail in [45].

c. For the value \( \lambda = \lambda/480 = 1600/480 \) and difference order \( s = 2 \), solve problem (\( L_1 \)), and plot the solution \( x \) together with the actual data \( y \). Moreover, on a separate graph, plot the differenced signal \( D_s x \) using a stem plot and observe its sparseness, which means that \( x \) is piece-wise linear. The particular choice for \( \lambda \) 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 \( D_s x \) 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 \( \lambda = 41640 \) 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, \( Ax = b \), 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.
In particular, Tikhonov regularization is defined by the following modified least-squares criterion, also known as ridge regression,

\[ J = \| b - Ax \|^2 + \lambda \| x \|^2 = \min \]  

(26.9.1)

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 \( \lambda \) 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 \( x \) of dimension \( 512^2 = 2.6\times10^5 \)), the solution can be obtained iteratively, for example, using conjugate-gradients [1081-1083].
Often, the second term, \( \| x \|_2^2 \), in (26.9.1) is replaced by the more general term, \( \| D x \|_p^2 = x^T D^T D x \), 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:

\[
J = \| b - Ax \|_2^2 + \lambda \| D x \|_2^2 = \min \quad x = (A^T A + \lambda D^T D)^{-1} A^T b \tag{26.9.2}
\]

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,

\[
b = A_1 x_1 + A_2 x_2
\]

and impose different regularization constraints on each part, for example, with positive parameters, \( \lambda_1, \lambda_2 \),

\[
J = \| b - A_1 x_1 - A_2 x_2 \|_2^2 + \lambda_1 \| D_1 x_1 \|_2^2 + \lambda_2 \| D_2 x_2 \|_2^2 = \min \tag{26.9.3}
\]

whose minimization with respect to \( x_1, x_2 \), leads to the solution,

\[
\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} A_1^T A_1 + \lambda_1 D_1^T D_1 \\ A_2^T A_2 + \lambda_2 D_2^T D_2 \end{bmatrix}^{-1} \begin{bmatrix} A_1^T b \\ A_2^T b \end{bmatrix} \tag{26.9.4}
\]

An example of such decomposition was the seasonal Whittaker-Henderson case discussed in Sec. 27.11 in which \( x_1 \) represented the seasonal component, and \( x_2 \), 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, \( \| D_1 x_1 \|_1 \).

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,

$$
\mathcal{J} = \| b - Ax \|_2^2 + \lambda \| x \|_p = \min
$$

$L_p$-regularized least-squares) \quad (26.9.5)

where the first term in (26.9.5) is still the $L_2$ norm of the modeling error, $b - Ax$, and $\| x \|_p$ denotes the $L_p$ norm of the vector $x = [x_1, x_2, \ldots, x_M]^T$.

$$
\| x \|_p = \left[ \sum_{n=1}^{M} |x_n|^p \right]^\frac{1}{p} \Rightarrow \| x \|_p^p = \sum_{n=1}^{M} |x_n|^p
$$

As mentioned before, even though $\| x \|_p$ 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 $x$, or rather, the sparsity of the vector, $Dx$, 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 $Ax = b$, with $A \in \mathbb{R}^{N \times M}$, $b \in \mathbb{R}^N$, and, $x \in \mathbb{R}^M$,

$$(L_0): \quad \mathcal{J} = \| b - Ax \|_2^2 + \lambda \| x \|_0 = \min
$$

$$(L_1): \quad \mathcal{J} = \| b - Ax \|_2^2 + \lambda \| x \|_1 = \min \quad (26.9.6)
$$

$$(L_2): \quad \mathcal{J} = \| b - Ax \|_2^2 + \lambda \| x \|_2 = \min
$$

where the $L_0$ norm, $\| x \|_0$, is the cardinality of the vector $x$, that is, the number of its non-zero entries. The criteria try to minimize the corresponding norm of $x$ 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\footnote{Least Absolute Shrinkage and Selection Operator} [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\footnote{http://cvxr.com/cvx/} as well as with the IRLS method for comparison.

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 $x \neq 0$,

$$
|\| x \|_p^p = \frac{|x|^p}{|x|^q} \approx \frac{|x|^2}{|x|^4 + \varepsilon}
$$

where $\varepsilon$ 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 $x \in \mathbb{R}^M$,

$$
\| x \|_p^p = \sum_{i=1}^{M} |x_i|^p \approx \sum_{i=1}^{M} |x_i|^q + \varepsilon = x^T W(x) x
$$

$$
W(x) = \text{diag} \left[ \frac{1}{|x|^q + \varepsilon}, \frac{1}{|x_2|^q + \varepsilon}, \ldots, \frac{1}{|x_M|^q + \varepsilon} \right] \quad (26.9.7)
$$

Alternatively, one can define $W(x)$ as the pseudo-inverse of the diagonal matrix of the powers $|x_i|^q$, $i = 1, 2, \ldots, M$, that is, in MATLAB language,*

$$
W(x) = \text{pinv} \left( \text{diag} \left[ |x_1|^q, |x_2|^q, \ldots, |x_M|^q \right] \right) \quad (26.9.8)
$$

Then, the $L_p$-regularized least-squares problem can be written in the form,

$$
\mathcal{J} = \| b - Ax \|_2^2 + \lambda \| x \|_p^p = \| b - Ax \|_2^2 + \lambda x^T W(x) x = \min
$$

(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,

$$
W_{k-1} = W(x^{(k-1)})
$$

$$
x^{(k)} = \arg \min_{x} \| b - Ax \|_2^2 + \lambda x^T W_{k-1} x
$$

(IRLS) \quad (26.9.10)

with the algorithm initialized to the ordinary least-squares solution of criterion $(L_2)$:

$$
x^{(0)} = (\lambda I + A^T A)^{-1} A^T b
$$

The solution of the optimization problem in (26.9.10) at the $k$th step is:

$$
x^{(k)} = (\lambda W_{k-1} + A^T A)^{-1} A^T b
$$

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 ($N,M > 10^5$), the successive least-squares problems could be solved with more efficient methods, such as conjugate gradients.

* e.g., \text{pinv}(\text{diag}(\{2, 0, 4\})) produces the diagonal matrix, \text{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,

\[
\mathcal{J} = \| \mathbf{b} - \mathbf{A} \mathbf{x} \|^2_2 + \lambda \| \mathbf{D} \mathbf{x} \|^2_\rho = \| \mathbf{b} - \mathbf{A} \mathbf{x} \|^2_2 + \lambda \mathbf{x}^T D^T W (D \mathbf{x}) D \mathbf{x} = \min \tag{26.9.11}
\]

which leads to the following iterative algorithm,

\[
\text{for } k = 1, 2, \ldots, K, \text{ do:} \\
W_{k-1} = W (D \mathbf{x}^{(k-1)}) \\
\mathbf{x}^{(k)} = \arg \min_{\mathbf{x}} \| \mathbf{b} - \mathbf{A} \mathbf{x} \|^2_2 + \lambda \mathbf{x}^T D^T W_{k-1} D \mathbf{x}
\]

with the algorithm initialized to the ordinary least-squares solution:

\[
\mathbf{x}^{(0)} = (\mathbf{A}^T \mathbf{A} + \lambda \mathbf{D}^T \mathbf{D})^{-1} \mathbf{A}^T \mathbf{b}
\]

The solution of the optimization problem at the \( k \)th step is:

\[
\mathbf{x}^{(k)} = (\mathbf{A}^T \mathbf{A} + \lambda \mathbf{D}^T W_{k-1} D)^{-1} \mathbf{A}^T \mathbf{b}
\]

### 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 \( \sigma^2_v \). 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:

\[
y = Hs + v
\]

where \( y, v \in \mathbb{R}^N, s \in \mathbb{R}^L \), and \( H \) is the \( N \times L \) convolution matrix corresponding to the filter. It can be constructed as a sparse matrix by the function:

\[
H = \text{convmat}(h, L); \quad \% H = \text{convmtx}(h, N) = \text{non-sparse version}
\]

The filter is taken to be:

\[
h_n = \cos(0.15(n - n_0)) \exp((-0.004(n - n_0)^2), \quad n = 0, 1, \ldots, M
\]

where \( M = 53 \) and \( n_0 = 25 \). The input is a sparse spike train consisting of \( S \) spikes:

\[
s_n = \sum_{i=1}^{S} a_i \delta(n - n_i), \quad n = 0, 1, \ldots, L - 1 \tag{26.9.13}
\]

where \( S = 8 \) and the spike locations and amplitudes are given as follows:

\[
n_i = [20, 40, 60, 70, 80, 100, 120, 140], \quad a_i = [10, 8, 4, -4, 5, 6, -2, 4]
\]

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, \( \text{SNR} = 20 \log_{10}(\max |Hs| / \sigma_v) = 38 \).

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 \( 0 \leq \omega \leq \pi \) rads/sample.

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,

\[
\mathcal{J} = \| y - Hx \|^2_2 + \lambda \| x \|_0 = \min \nwarrow \tag{26.9.14}
\]

The \( L_2 \) case with \( \lambda = 0 \) corresponds to the ordinary (full-rank overdetermined) least-squares solution of the linear system, \( Hx = y \), that is, \( x_{\text{ord}} = (H^T H)^{-1} H^T y \), or, \( x_{\text{ord}} = H^T y \), in MATLAB.

Similarly, the \( L_2 \)-regularized solution with non-zero \( \lambda \) is, \( x_2 = (\lambda I + H^T H)^{-1} H^T y \).
These two solutions are depicted below, displaying also the percent error of recovering the desired signal $s$, defined in terms of the $L_2$ norms by, $P_{\text{error}} = 100 \cdot \|x - s\|_2 / \|s\|_2$.

The MATLAB code for generating the above six graphs was as follows:

```matlab
g = @times(x) cos(0.15*pi*x).*exp(-0.004*x.*2);  % filter function
delta = @times(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+1;
N = (0:L-1);  % time indices for s(n) and y(n)
L = 141, N = 194

s = 0;
for i=1:length(ni),
    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;
Perr = 100 * norm(xord)/norm(s);

la = 0.01;
x2 = (la + eye(L) + H'*H) \ (H'*y);  % L2-regularized
Perr = 100 * norm(x2)/norm(s);
```

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 $\lambda$ 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,

$$
\mathcal{J} = \sum_{n=-\infty}^{\infty} |y_n - h_n \ast x_n|^2 + \lambda \sum_{n=-\infty}^{\infty} |x_n|^2
$$

or, (26.9.16)

$$
\int_{-\pi}^{\pi} \left| Y(\omega) - H(\omega) X(\omega) \right|^2 \frac{d\omega}{2\pi} + \lambda \int_{-\pi}^{\pi} |X(\omega)|^2 \frac{d\omega}{2\pi} = \min
$$

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,

$$
X(\omega) = \frac{H^*(\omega)}{\lambda + |H(\omega)|^2} Y(\omega)
$$

(regularized inverse filter)

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,

$$
Y(\omega) = H(\omega) S(\omega) + V(\omega) \Rightarrow X(\omega) = \frac{|H(\omega)|^2}{\lambda + |H(\omega)|^2} S(\omega) + \frac{H^*(\omega)}{\lambda + |H(\omega)|^2} V(\omega)
$$

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
26.9. SPARSE MODELING – LASSO AND BPDN

![Graphs showing deconvolved signals based on the L1 and L0 criteria.](image)

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 $k$th iteration by,

$$P(k) = 100 \cdot \frac{\|x^{(k)} - x^{(k-1)}\|_2}{\|x^{(k-1)}\|_2}.$$

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,

$$P_{\text{error}} = 100 \cdot \frac{\|x - s\|_2}{\|s\|_2},$$

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
variable x(L)
minimize( sum_square(H*x-y) + la * norm(x,1) )
cvx_end
Perr = 100*norm(x-s)/norm(s); % reconstruction errorfigure; 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);
x 0=( 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) = 100*norm(x-x0)/norm(x0); % iteration error
    x0 = x;
end
Perr = 100*norm(x-s)/norm(s); % reconstruction errorfigure; 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);
x 0=( 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 errorfigure; 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:

$$y = A s + v$$

where $A \in \mathbb{R}^{1000 \times 2000}$, $s \in \mathbb{R}^{2000}$, and $y,v \in \mathbb{R}^{1000}$. The matrix $A$ has full rank and consists of zero-mean, unit-variance, gaussian, independent random entries, and the 2000-long input signal $s$ is sparse with only $L = 100$ non-zero entries taken to be randomly positioned within its length, and constructed to have amplitudes $\pm 1$ with random signs and then weighted by a triangular window in order to get a variety of values.

The noise $v$ is zero-mean gaussian white noise with standard deviation $\sigma_v = 0.1$. The recovery criteria are as in Eq. (26.9.6),

$$J = \|y - Ax\|^2_2 + \lambda \|x\|_0 = \min$$
$$J = \|y - Ax\|^2_2 + \lambda \|x\|_1 = \min$$
$$J = \|y - Ax\|^2_2 + \lambda \|x\|_2^2 = \min$$

(26.9.18)
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 \( \lambda \) 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 \( Ax = y \), that is, \( x = A^+ y = (A A^T)^{-1} y \), in terms of the pseudo-inverse of \( A \).

Note that using \( \lambda = 0.1 \) 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 \( Ax = y \), that is, \( x = A^+ y = (A A^T)^{-1} y \), in terms of the pseudo-inverse of \( A \).

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, \( P_{\text{error}} = 100 \cdot \| x - s \|_2 / \| s \|_2 \).

The IRLS algorithms were run with parameters \( \lambda = 0.1 \), \( \varepsilon = 10^{-6} \), and \( K = 20 \) iterations. The successive iteration percentage errors, \( P(k) = 100 \cdot \| x(k) - x(k-1) \|_2 / \| x(k-1) \|_2 \), 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.

```matlab
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
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); % noisy observations
y = A*s + v;
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;
Perr = 100 * norm(x-s)/norm(s); % reconstruction error = 71.21 %
figure; stem(n,x,'r-');
% ----
la = 0.1;
cvx_begin
variable x(M)
% L1 - CVX solution
1
minimize( sum_square(A*x-y) + la * norm(x,1) )
cvx_end
Perr = 100 * norm(x-s)/norm(s); % reconstruction error = 2.54 %
```

Fig. 26.9.2 Recovered signals based on the \( L_2 \), \( L_1 \), and \( L_0 \) criteria.

The parameter \( \lambda \) 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 \( Ax = y \), that is, \( x = A^+ y = (A A^T)^{-1} y \), in terms of the pseudo-inverse of \( A \). Note that using \( \lambda = 0.1 \) 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 \( Ax = y \), that is, \( x = A^+ y = (A A^T)^{-1} y \), in terms of the pseudo-inverse of \( A \).
26.9. SPARSE MODELING – LASSO AND BPDN

$L_1$ – IRLS iteration error

$L_0$ – IRLS iteration error

Fig. 26.9.3 IRLS iteration error based on the $L_1$, and $L_0$ criteria.

```matlab
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-');
```

26.10 Problems

26.1 For the case $s=1$, show that the Whittaker-Henderson filter has poles $z_1, 1/z_1$, where

$$z_1 = e^{-\alpha}, \quad \alpha = 2 \sinh \left( \frac{1}{2\sqrt{\lambda}} \right)$$

For the case $s=2$, show that the filter has poles $\{z_1, z_1^*, 1/z_1, 1/z_1^*\}$, where

$$z_1 = \left( \sqrt{1 - a^2} + ja e^{\pi i/4} \right)^2 = \frac{1}{2} \sqrt{1 + \sqrt{1 + 4a^2}}, \quad a = \frac{1}{2\sqrt{\lambda}}$$

Show that in both cases $|z_1| < 1$.

26.2 Determine explicit expressions in terms of $\lambda$ for the quantities $\sigma^2$ and $z_1$ that appear in the factorization of the denominator of the Hodrick-Prescott filter:

$$\sigma^2 = \frac{1}{(1-z_1)(1-z_1^*)} \left( 1 - \frac{z_1}{z_1^*} \right) \left( 1 - \frac{z_1^*}{z_1} \right)$$

What are the numerical values of $\sigma^2, z_1$ for $\lambda = 1600$? What are the values of the coefficients of the second-order filter $(1 - 2Re(z_1)z^{-1} + |z_1|^2 z^{-2})$?

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,

$$J = \sum_n |y_n - f_n * x_n|^2 + \lambda \sum_n |d_n * x_n|^2$$

$$= \int_{-\pi}^{\pi} \left[ |Y(\omega) - F(\omega) X(\omega)|^2 + \lambda |D(\omega) X(\omega)|^2 \right] \frac{d\omega}{2\pi} = \min$$

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 $y_n$?
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 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 $\omega_1$. The noise reduction filter must be an ideal notch filter with notches at the harmonics $k\omega_1$, $k = 0, 1, \ldots$, 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.

![Notch filter for reducing periodic interference.](image)

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 $k = 0, 1, \ldots, D - 1$.

1 adapted from the author’s book on *Applied Optimum Signal Processing* [45]
27.2. NOTCH AND COMB FILTERS FOR PERIODIC SIGNALS

Notch and comb filters are used to extract periodic signals. Let $s(t)$ be a periodic discrete-time signal with period $D$.

**Combin Filter**: The comb filter is defined as:

$$ Y_c(\omega) = \sum_{n=-\infty}^{\infty} s(nT) e^{-j\omega n} $$

where $s(nT)$ is the periodic signal and $T$ is the sampling interval.

**Notch Filter**: The notch filter is defined as:

$$ Y_n(\omega) = \sum_{n=-\infty}^{\infty} s(nT) e^{-j\omega n} $$

Both filters can be implemented using discrete Fourier transforms (DFTs).

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 $H_{LP}(\omega)$, the frequency-scaled filter will be a comb filter, $H_{comb}(\omega) = H_{LP}(\omegaD)$. Similarly, a highpass filter is transformed into a notch filter $H_{notch}(\omega) = H_{HP}(\omegaD)$.

In the $z$-domain, we have the following simple prescriptions for turning lowpass and highpass filters into comb and notch filters:

(27.2.1)

$$ H_{comb}(z) = H_{LP}(z^D) $$

(27.2.4)

$$ H_{notch}(z) = H_{HP}(z^D) $$

For example, the simplest comb and notch filters are generated by,

$$ H_{LP}(z) = \frac{1}{2} (1 + z^{-1}) \quad H_{comb}(z) = \frac{1}{2} (1 + z^{-D}) $$

$$ H_{HP}(z) = \frac{1}{2} (1 - z^{-1}) \quad H_{notch}(z) = \frac{1}{2} (1 - z^{-D}) \quad (27.2.5) $$

Their magnitude responses are shown in Fig. 27.2.4 for $D=10$. The harmonics $\omega_k = 2\pi k/D = 2\pi k/10, k = 0, 1, \ldots, 9$ 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:

$$ |H_{comb}(\omega)|^2 = \cos^2(\omegaD/2) \quad |H_{notch}(\omega)|^2 = \sin^2(\omegaD/2) \quad (27.2.6) $$

The filters are complementary, as well as power-complementary, in the sense,

$$ H_{comb}(z) + H_{notch}(z) = 1 \quad |H_{comb}(\omega)|^2 + |H_{notch}(\omega)|^2 = 1 \quad (27.2.7) $$

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,

$$ H_{LP}(z) = b \frac{1 + z^{-1}}{1 - az^{-1}}, \quad b = \frac{1-a}{2} \quad H_{HP}(z) = b \frac{1-z^{-1}}{1-az^{-1}}, \quad b = \frac{1+a}{2} \quad (27.2.8) $$

where $S_k$ is the $D$-point DFT of one period $[s_0, s_1, \ldots, s_{D-1}]$ of the time signal. Because of 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. The sampling period $T_s = 1/f_s$ must be equal to the fundamental harmonic $f_1$, that is, $f_s = Df_1$. For example, the simplest comb and notch filters are generated by,

$$ H_{LP}(z) = \frac{1}{2} (1 + z^{-1}) \quad H_{comb}(z) = \frac{1}{2} (1 + z^{-D}) \quad (27.2.5) $$

$$ H_{HP}(z) = \frac{1}{2} (1 - z^{-1}) \quad H_{notch}(z) = \frac{1}{2} (1 - z^{-D}) \quad (27.2.5) $$

Their magnitude responses are shown in Fig. 27.2.4 for $D=10$. The harmonics $\omega_k = 2\pi k/D = 2\pi k/10, k = 0, 1, \ldots, 9$ 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:

$$ |H_{comb}(\omega)|^2 = \cos^2(\omegaD/2) \quad |H_{notch}(\omega)|^2 = \sin^2(\omegaD/2) \quad (27.2.6) $$

The filters are complementary, as well as power-complementary, in the sense,

$$ H_{comb}(z) + H_{notch}(z) = 1 \quad |H_{comb}(\omega)|^2 + |H_{notch}(\omega)|^2 = 1 \quad (27.2.7) $$

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,

$$ H_{LP}(z) = b \frac{1 + z^{-1}}{1 - az^{-1}}, \quad b = \frac{1-a}{2} \quad H_{HP}(z) = b \frac{1-z^{-1}}{1-az^{-1}}, \quad b = \frac{1+a}{2} \quad (27.2.8) $$
implementation of the multiple delay \( z^{-D} \) is as follows, where \( p \) is the circular pointer,

\[
\begin{align*}
\text{for each } x \\
& s_D = \ast (p + D) \\
& s_0 = bx + as_D \\
& y = s_0 + s_D \\
& \ast p = s_0 \\
& p = -- \\
\end{align*}
\]

**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).

**Example 27.2.2:** 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 \( \Delta \omega = 2\pi \Delta f / f_1 = 2\pi /(DQ) \approx 0.004\pi \), which results in the filter parameters \( a = 0.9391 \) and \( b = (1 + a)/2 \approx 0.9695 \). Thus, the designed notch filter was:

\[
H_{\text{notch}}(z) = 0.9695 \frac{1 - z^{-10}}{1 - 0.9391z^{-10}}
\]

The noise was simulated by adding the following harmonic components,

\[
v_n = \sum_{k=1}^{D/2-1} A_k \sin(\omega_k n), \quad \text{with } \omega_k = \frac{2\pi k}{D}, \quad A_k = \frac{1}{2k^2}
\]
27.2. NOTCH AND COMB FILTERS FOR PERIODIC SIGNALS

27. PERIODIC SIGNAL EXTRACTION

ECG + 60 Hz noise
filtered ECG

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 \( n_{e f f} = D \ln(0.01)/\ln(\pi) = 732 \) samples or equivalently, \( \tau = n_{e f f}/f_s = 732/600 = 1.22 \) 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:

\[
\text{nbbeats} = 2; \quad L = 600; \quad M = 15; \\
\text{s} = \text{ecgsim(nbeats,L,M);} \quad \% \text{simulated ECG} \\
\text{n} = (0:length(s)-1)^{\star}; \quad t = n/L; \quad \% \text{time in seconds} \\
\text{D} = 30; \quad v = 0; \\
\text{for } k=1:0/2-1, \quad v = v + (0.5/k^2) \times \sin(2\pi k n/D); \quad \% \text{generate noise} \\
\text{end} \\
\text{y} = s + v; \quad \% \text{noisy ECG} \\
\text{Q} = 50; \quad \text{beta} = \tan(pi/2)/Q; \\
\text{a} = (1-beta)/(1+beta); \quad b = (1+a)/2; \quad \% \text{filter parameters} \\
\text{aD} = \text{up}(a,-1,D); \quad \% \text{upsampled denominator coefficients} \\
\text{bD} = \text{up}(b,-b,D); \quad \% \text{upsampled numerator coefficients} \\
\text{x} = \text{filter(bD,aD,y);} \quad \% \text{filtered ECG} \\
\text{figure; plot(t,y);} \quad \% \text{left graph} \\
\text{figure; plot(t,x,t,s,');} \quad \% \text{right graph}
\]

The MATLAB function \text{ecgsim}, which is part of the AOSP toolbox [45], was used to generate the simulated ECG. It is based on the ISP function \text{ecg}. The function \text{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 \text{filter} can be used.

The upsampling operation used in the previous example is the time-domain equivalent of the transformation \( z \rightarrow z^D \) 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:

\[
[h_0, h_1, h_2, h_3] \rightarrow [h_0, 0, 0, h_1, 0, 0, 0, h_2, 0, 0, 0, h_3]
\]

The function \text{up} implements this operation,

\[
\mathbf{g} = \text{up}([h_1]), \quad \% \text{upsampled vector}
\]

It is similar to MATLAB's built-in function \text{upsample}, except it does not append \( D-1 \) zeros at the end. The difference is illustrated by the following example,

\[
\text{up}([1,2,3,4,4]) = [1 \ 0 \ 0 \ 0 \ 2 \ 0 \ 0 \ 0 \ 3 \ 0 \ 0 \ 0 \ 4 \ 0 \ 0 \ 0]
\]

In addition to enhancing periodic signals or removing periodic interference, comb and notch filters have many other applications. The transformation \( z \rightarrow z^D \) 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 \( z \rightarrow z^D \) 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 \( f' = f D \). 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 \( f_s ' = D f_s / k \) 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 \( D = f_s / f_s ' \) 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 \( f_k = k f_s / D \), \( k = 1, 2, \ldots \), 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:

\[
D = D_{\text{int}} + d \quad (27.3.1)
\]

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 \( z^{-D} \approx z^{-D_{\text{int}}} \). Then, the corresponding IIR comb/notch filters (27.2.9) will be approximated by

\[
H_{\text{comb}}(z) = b \frac{1 + z^{-D_{\text{int}}} H(z)}{1 - a z^{-D_{\text{int}}} H(z)}, \quad b = \frac{1 - a}{2} \\
H_{\text{notch}}(z) = b \frac{1 - z^{-D_{\text{int}}} H(z)}{1 - a z^{-D_{\text{int}}} H(z)}, \quad b = \frac{1 + a}{2}
\]

(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),

\[
H(z) = d + (1 - d) z^{-1} \\
H(z) = \frac{1}{2} (d - 1) (d - 2) - d (d - 2) z^{-1} + \frac{1}{2} d (d - 1) z^{-2}
\]

(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 \( e^{-j\omega_k D_{\text{int}}} = 1 \) we have,

\[
e^{-j\omega_k D_{\text{int}}} H(\omega_k) = 1 = e^{-j\omega_k D} = e^{-j\omega_k (D_{\text{int}} + d)} \Rightarrow H(\omega_k) = e^{-j\omega_k d}
\]

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, \(-\pi \leq \omega_k \leq \pi\), or,

\[-\pi \leq \frac{2\pi k}{D} \leq \pi \Rightarrow -\frac{D}{2} \leq k \leq \frac{D}{2}\]

Writing \( D_{\text{int}} = 2p + q \) and \( D = D_{\text{int}} + d = 2p + q + d \), with integer \( p \) and \( q \), the above condition reads:

\[-p - \frac{1}{2} (q + d) \leq k \leq \frac{1}{2} (q + d)\]

Since \( 0 < d < 1 \) and \( k \) must be an integer, we obtain,

\[-p \leq k \leq p\]

(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:

\[H(\omega_k) = e^{-j\omega_k d}, \quad -p \leq k \leq p\]

(27.3.5)

When \( q = d = 0 \), we must choose \( -p \leq k \leq p - 1 \), 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 \( \lambda_k \):

\[
J = \int_{-\alpha\pi}^{\alpha\pi} \left| H(\omega) - e^{-j\omega d} \right|^2 \frac{d\omega}{2\pi} + \sum_{k=-p}^{p} \left| e^{-j\omega_k d} - H(\omega_k) \right| \lambda_k^* + \text{c.c.} = \min
\]

(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, \( 0.9 \leq \alpha \leq 1 \), with \( \alpha = 1 \) covering the full interval. Assuming an \( M \)th order filter \( h = [h_0, h_1, \ldots, h_M]^T \), we can write the frequency response in terms of the \((M+1)\)-dimensional vectors,

\[H(\omega) = \sum_{n=0}^{M} h_n e^{-j\omega n} = s_\omega^* h, \quad s_\omega = \begin{bmatrix} e^{j\omega_0} \\ \vdots \\ e^{j\omega_M} \end{bmatrix}, \quad h = \begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_M \end{bmatrix}
\]

(27.3.7)

Similarly, we can express the gain constraints in the vector form,

\[S^\dagger h = g
\]

(27.3.8)

where \( S \) is an \((M+1) \times (2p+1)\) matrix and \( g \) a \((2p+1)\)-dimensional column vector defined component-wise by

\[S_{nk} = e^{jn\omega_k}, \quad 0 \leq n \leq M, \quad -p \leq k \leq p
\]

(27.3.9)

that is,

\[S = \begin{bmatrix} \vdots \\ s_{\omega_k}, \ldots \\ \vdots \end{bmatrix}, \quad g = \begin{bmatrix} e^{-j\omega d} \\ \vdots \\ \vdots \end{bmatrix}
\]

(27.3.10)
27.3. NOTCH AND COMB FILTERS WITH FRACTIONAL DELAY

It follows that the performance index can be written compactly as,
\[ J = \int_{-\alpha \pi}^{\alpha \pi} |s_{\omega}^{\ast} h - e^{-j \omega d}|^2 \frac{d \omega}{2 \pi} + \lambda^T (g - S^T) h + (g - S^T) h^T \lambda = \min \] (27.3.11)
where \( \lambda = [\ldots, \lambda_k, \ldots]^T \) is the vector of Lagrange multipliers. Expanding the first term of \( J \), we obtain,
\[ J = h^T R h - h^T r - r^T h + \alpha + \lambda^T (g - S^T h) + (g - h^T S) \lambda = \min \] (27.3.12)
where the matrix \( R \) and vector \( r \) are defined by,
\[ R = \frac{1}{2 \pi} \int_{-\alpha \pi}^{\alpha \pi} s_{\omega} s_{\omega}^\ast d \omega, \quad r = \frac{1}{2 \pi} \int_{-\alpha \pi}^{\alpha \pi} s_{\omega} e^{-j \omega d} d \omega \] (27.3.13)
and component-wise,
\[ R_{nm} = \frac{1}{2 \pi} \int_{-\alpha \pi}^{\alpha \pi} e^{j \omega (n - m)} d \omega = \frac{\sin(\alpha \pi (n - m))}{\pi (n - m)} \] (27.3.14)
\[ r_n = \frac{1}{2 \pi} \int_{-\alpha \pi}^{\alpha \pi} e^{j \omega (n - d)} d \omega = \frac{\sin(\alpha \pi (n - d))}{\pi (n - d)} \]

We note that for \( \alpha = \pi, R \) reduces to the identity matrix. The optimal solution for \( h \) is obtained by setting the gradient of \( J \) to zero:
\[ \frac{\partial J}{\partial h^T} = R h - r - S \lambda = 0 \Rightarrow h = R^{-1} r + R^{-1} S \lambda = h_u + R^{-1} S \lambda \]
where \( h_u = R^{-1} r \) is the unconstrained solution of the least-squares problem. The Lagrange multiplier \( \lambda \) can be determined by multiplying both sides by \( S^T \) and using the constraint (27.3.8):
\[ g = S^T h = S^T h_u + S^T R^{-1} S \lambda \Rightarrow \lambda = (S^T R^{-1} S)^{-1}(g - S^T h_u) \]
Finally, substituting \( \lambda \) into the solution for \( h \), we obtain,
\[ h = h_u + R^{-1} S (S^T R^{-1} S)^{-1}(g - S^T h_u) \] (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 optimal stock portfolio design.

The MATLAB function \texttt{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 \( \alpha \),
\[
[b0,a0,h,zmax] = \texttt{combfd}(D,M,a,\alpha); \quad \% \text{comb/notch filter design with fractional delay}
\]

Entering the parameter \( a \) as negative indicates the design of a notch filter. The outputs \( b0, a0 \) are the coefficients of the numerator and denominator polynomials of the comb/notch filters (27.3.2):
\[ B_D(z) = b[1 \pm z^{-D}H(z)] \]
\[ A_D(z) = 1 - a z^{-D}H(z) = 1 - az^{-D}(h_0 + h_1 z^{-1} + \cdots + h_M z^{-M}) \] (27.3.16)

The output \( h \) is the impulse response vector \( h \), and \( z_{\text{max}} \) 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 \( D_{\text{int}} = 9 \) 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.](image)

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 \(-\pi \leq \omega \leq \pi \). 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.
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 \( k \)th harmonic \( \omega_k = k\omega_1 = 2\pi k/D, \) \( k = 0, 1, \ldots, p \) and its negative \(-\omega_k \). Then, the transfer functions of the comb and notch filters will be:

\[
H_{\text{comb}}(z) = \sum_{k=0}^{p} H_k(z), \quad H_{\text{notch}}(z) = \prod_{k=0}^{p} H_k(z) \tag{27.4.1}
\]

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:

\[
z \rightarrow z' = \frac{z(\cos \omega_k - z)}{1 - z \cos \omega_k} \tag{27.4.2}
\]

The resulting second-order peaking and notch filters are, for \( k = 1, 2, \ldots, p \):

\[
\begin{align*}
H_k(z) &= b \frac{1 - z^2}{1 - (1 + a) \cos \omega_k z^{-1} + a z^{-2}}, \quad b = 1 - a \quad \text{peak} \quad \tag{27.4.3} \\
H_k(z) &= b \frac{1 - 2 \cos \omega_k z^{-1} + z^{-2}}{1 - (1 + a) \cos \omega_k z^{-1} + a z^{-2}}, \quad b = 1 + a \quad \text{notch}
\end{align*}
\]

The filter parameter \( a \) is fixed in terms of the 3-dB width of the peak or the notch by:

\[
\tan \left( \frac{\Delta \omega}{2} \right) = \frac{1 - a}{1 + a} = \beta \tag{27.4.4}
\]
27.4. PARALLEL AND CASCADE REALIZATIONS

Comb, $D = 8.9$, $M = 12$, $\alpha = 1$

Notch, $D = 8.9$, $M = 12$, $\alpha = 1$

Parallel comb, $D = 9.9$, $N = 1$

Cascaded notch, $D = 8.9$, $N = 1$

Parallel comb, $D = 8.1$, $N = 1$

Cascaded notch, $D = 8.1$, $N = 1$

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:

$$\tan \left( \frac{\Delta \omega}{4} \right) = \frac{1 - a}{1 + a} = \beta$$  \hspace{1cm} (27.4.5)

To clarify the construction, we give below the MATLAB code for generating the left graphs of Fig. 27.4.1:

```matlab
f = linspace(-1,1,4001); w = pi*f; % frequency range
D = 9.9; p = floor(D/2); w1 = 2*pi/D; % design parameters
R = 0.95; a = R^2;

beta = (1-a)/(1+a); % 3-dB width calculation
beta0 = tan(w1/4); a0 = (1-beta0)/(1+beta0);
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;

A = [1, -a, 0]; % denominator coefficients for k = 0
Bcomb = [1, 0, -1] * (1-a)/2; % numerator coefficients for k = 0
Bnotch = [1, -2*cos(k*wl), 1] * (1+a)/2;
Hcomb = freqz(Bcomb,A,w); % denominator of Hk(z)
Hnotch = freqz(Bnotch,A,w); % numerator of peak Hk(z)

for k=1:p,
    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*wl), 1] * (1+a)/2;
    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,'-'); % k = 0 section, \(H_0(\omega)\)
figure; plot(w/pi, abs(Hnotch).^2,'-'); % non-zero harmonics
```

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 \( N = 1 \).
27.5. SIGNAL AVERAGING

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:

\[ y_n = s_n + v_n \]  

(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:

\[ H_{LP}(z) = \frac{1}{N} \left[ 1 + z^{-1} + z^{-2} + \cdots + z^{-(N-1)} \right] = \frac{1}{N} \frac{1 - z^{-N}}{1 - z^{-1}} \]  

(27.5.2)

The definition \( H (z) = H_{LP}(z^D) \), then gives the comb filter:

\[ H(z) = \frac{1}{N} \left[ 1 + z^{-D} + z^{-2D} + \cdots + z^{-(N-1)D} \right] = \frac{1}{N} \frac{1 - z^{-ND}}{1 - z^{-D}} \]  

(27.5.3)

The latter equation shows that \( H (z) \) has zeros at all the \( (ND) \)-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 \( \omega_k = 2\pi k / D = 2\pi k / 10 \), \( k = 0, 1, \ldots, 9 \). 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.

The 3-dB width is given by

\[ \Delta \omega = 0.886 \frac{2\pi}{ND} \]  

(27.5.4)

which follows from the frequency response of \( H (z) \):

\[ H(\omega) = \frac{1}{N} \frac{1 - e^{-j\omega ND}}{1 - e^{-j\omega D}} = \frac{1}{N} \frac{\sin(ND\omega/2)}{\sin(D\omega/2)} e^{-j(\omega(N-1)D/2)} \]  

(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,

\[ \hat{s}_n = \frac{1}{N} \left[ y_n + y_{n-D} + y_{n-2D} + \cdots + y_{n-(N-1)D} \right] \]  

(27.5.6)

Inserting \( y_n = s_n + v_n \) and using the periodicity property \( s_{n-D} = s_n \), we obtain,

\[ \hat{s}_n = s_n + \frac{1}{N} \left[ y_{n-D} + \cdots + y_{n-(N-1)D} \right] \equiv s_n + \hat{v}_n \]  

(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 \( N \),

\[ \sigma_{\hat{v}}^2 = \frac{1}{N^2} (\text{var}(v_n) + \text{var}(v_{n-D}) + \cdots + \text{var}(v_{n-(N-1)D})) \]  

(27.5.8)

which implies that the NRR of the comb filter is \( R = 1/N \). 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, \( 0 \leq n \leq ND - 1 \), and divide the signal into length-\( D \) period segments as shown below,

\[ \vdots \quad y_0 \quad y_1 \quad y_2 \quad \cdots \quad y_{D-1} \quad \vdots \]

\[ n \]

The filtering operation (27.5.6) can be thought of as the averaging the periodic segments of \( y_n \), that is, \( 0 \leq n \leq ND - 1 \), and divide the signal into length-\( D \) period segments as shown below:

\[ y_0 \quad y_1 \quad y_2 \quad \cdots \quad y_{D-1} \]

\[ n \]

The MATLAB implementation of signal averaging is straightforward, for example, assuming that the array \( y \) has length at least \( ND \),

\[
\begin{align*}
&\text{s} = 0; \\
&\text{for } i=0:N-1, \quad y_{iD} = y(i*D+1 : i*D+D); \quad \% \text{ extract } i\text{-th period} \\
&\quad \text{s} = \text{s} + y_{iD}; \quad \% \text{ accumulate } i\text{-th period} \\
&\text{end} \\
&\text{s} = \text{s}/N; \quad \% \text{ average of } N \text{ periods}
\end{align*}
\]
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:

$$s_n = \frac{1}{N} \sum_{i=0}^{N-1} [y_i(n) - \mu_i] \quad \mu_i = \frac{1}{D} \sum_{n=0}^{D-1} y_i(n)$$

(27.5.11)

which does satisfy $S_0 = 0$. By replicating the $\mu_i$ 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:

```matlab
% number of periods in y
L = length(y);
N = floor(L/D);
% L = ND + r
r = mod(L,D);
s = 0;
for i=0:N-1,
yi = y(i*D+1 : i*D+D);
m(i+1) = mean(yi);
s = s + yi - m(i+1);
end
s = s / N;

ys = repmat(s,N,1); % replicate N periods
yt = repmat(m,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
ys(end+1:end+r) = ys(end); % extend to length L by appending a portion of s

ys = repmat(ys,N,1); % step-wise estimate of the trend component
yt = repmat(yt,N,1); yt = yt(:); % extend to length L by replicating last mean r times
yt(end+1:end+r) = yt(end); % extend to length L by appending a portion of s

ys = 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 $n = 0,1,\ldots,ND-1$, 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:

```matlab
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,'-');
xlim([0,200]);
figure; plot(n,ys, '-');
xlim([0,200]);
figure; plot(n,yt, '-');
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-
27.5 SIGNAL AVERAGING

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:

```matlab
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 \( \lambda \)'s
[gcv,lopt] = whgcv(y,la,s); % optimum smoothing parameter, \( \lambda_{opt} = 6850 \)
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.

![Ideal filters for decomposition into trend and seasonal components.](image)

Let \( H_{\text{trend}}(z) \) be the trend-extraction filter and \( H_{\text{comb}}(z) \) 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:

\[
T(z) = H_{\text{trend}}(z)Y(z)
\]
\[
R(z) = S(z) + V(z) = Y(z) - T(z) = \left( 1 - H_{\text{trend}}(z) \right) Y(z)
\]
\[
S(z) = H_{\text{comb}}(z)R(z) = H_{\text{comb}}(z) \left[ 1 - H_{\text{trend}}(z) \right] Y(z) \equiv H_S(z)Y(z)
\]
\[
V(z) = R(z) - S(z) = \left[ 1 - H_{\text{comb}}(z) \right] \left[ 1 - H_{\text{trend}}(z) \right] Y(z) \equiv H_I(z)Y(z)
\]
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:

\[
H_T(z) = H_{\text{trend}}(z) \quad \text{(trend)}
\]

\[
H_S(z) = H_{\text{comb}}(z) \left[ 1 - H_{\text{trend}}(z) \right] \quad \text{(seasonal)}
\]

\[
H_I(z) = \left[ 1 - H_{\text{comb}}(z) \right] \left[ 1 - H_{\text{trend}}(z) \right] \quad \text{(irregular)}
\]

The three filters satisfy the complementarity property:

\[
H_T(z) + H_S(z) + H_I(z) = 1 \quad \text{(27.6.2)}
\]

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.

![Spectra of monthly housing data with and without trend.](image)

The following MATLAB code illustrates the computation of the spectra:

```matlab
Y = loadfile('newhouse.dat'); % data file available in the AOSP toolbox
i = find(Y(:,1) == 1984); % 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 \(\lambda\) from Example 27.5.2

% Whittaker-Henderson smoothing
yt = whsm(y, lopt, s);

% frequency
k = linspace(-6.6, 1201); w = 2*pi*k/12; % frequency \(w = k\omega_1\)
L = length(y);
wind = 0.54 - 0.46*cos(2*pi*(0:L-1)/(L-1))';

% seasonal + irregular component
ysl = y - yt;

% Hamming window
wind = 0.54 - 0.46*cos(2*pi*(0:L-1)/(L-1))';

% frequency
k = linspace(-6.6, 1201); w = 2*pi*k/12; % frequency \(w = k\omega_1\)
L = length(y);
wind = 0.54 - 0.46*cos(2*pi*(0:L-1)/(L-1))';

% normalized spectrum
Ysi = abs(freqz(ysi.*wind, 1, w)); Ysi = Ysi/max(Ysi);

% left and right graphs
figure; plot(k,Y); figure; plot(k,Ysi);
```

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 \(H_T(z)\) by the presence of the factor \(1 - H_{\text{trend}}(z)\). 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,

\[
S_{\text{prelim}}(z) = H_{\text{comb}}(z)R(z)
\]

\[
S(z) = S_{\text{prelim}}(z) - H_{\text{trend}}(z)S_{\text{prelim}}(z) = H_{\text{comb}}(z) \left[ 1 - H_{\text{trend}}(z) \right]^2 Y(z)
\]

This results in the modified extraction filters, which still satisfy (27.6.2):

\[
H_T(z) = H_{\text{trend}}(z) \quad \text{(trend)}
\]

\[
H_S(z) = H_{\text{comb}}(z) \left[ 1 - H_{\text{trend}}(z) \right]^2 \quad \text{(seasonal)}
\]

\[
H_I(z) = \left[ 1 - H_{\text{trend}}(z) \right] \left[ 1 - H_{\text{comb}}(z) \left[ 1 - H_{\text{trend}}(z) \right] \right] \quad \text{(irregular)}
\]

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:

\[
H_{\text{trend}}(z) = \frac{1}{D} \left[ 1 + z^{-1} + z^{-2} + \cdots + z^{-(D-1)} \right]
\]

\[
H_{\text{trend}}(z) = \frac{1}{D} \left[ 1 + z^{-1} + \cdots + z^{-(D-1)} \right] \frac{1}{2} (1 + z^{-1})
\]

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:

\[
\frac{1}{N_1} \left[ 1 + z^{-1} + \cdots + z^{-(N_1-1)} \right] \cdot \frac{1}{N_2} \left[ 1 + z^{-1} + \cdots + z^{-(N_2-1)} \right]
\]
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 \( \omega_k = k\omega_0 = 2\pi k/D, \) \( k = 1, 2, \ldots, D - 1. \) Their 3-dB cutoff frequency is about one-half the fundamental harmonic \( \omega_1, \) 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 \( \omega \) over the symmetric Nyquist interval, \( -\pi \leq \omega \leq \pi. \) 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 \( z^p: \)
\[
D = 2p + 1, \quad H_{\text{trend}}(z) = z^p \frac{1}{D} \left[ 1 + z^{-1} + \cdots + z^{-(D-1)} \right]
\]
\[
D = 2p, \quad H_{\text{trend}}(z) = z^p \frac{1}{D} \left[ 1 + z^{-1} + \cdots + z^{-(D-1)} \right] \cdot \frac{1}{2} (1 + z^{-1}) \tag{27.7.4}
\]

The corresponding frequency responses are obtained by setting \( z = e^{j\omega}: \)
\[
D = 2p + 1, \quad H_{\text{trend}}(\omega) = \frac{\sin(D\omega/2)}{D \sin(\omega/2)}
\]
\[
D = 2p, \quad H_{\text{trend}}(\omega) = \frac{\sin(D\omega/2)}{D \sin(\omega/2)} \cdot \cos(\omega/2) \tag{27.7.5}
\]

where we used the identity \( 1 + z^{-1} + \cdots + z^{-(D-1)} = (1 - z^{-D}) / (1 - z^{-1}). \) The symmetric impulse responses are:
\[
D = 2p + 1, \quad h_{\text{trend}} = \frac{1}{D} \left[ 1, 1, \ldots, 1, 1 \right]_{2p-1 \text{ ones}}
\]
\[
D = 2p, \quad h_{\text{trend}} = \frac{1}{D} \left[ 0.5, 1, \ldots, 1, 0.5 \right]_{2p-1 \text{ ones}} \tag{27.7.6}
\]

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 \( p \leq \tau \leq L - 1 - p. \) 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 \( h_{\text{trend}}. \)

The calculation of the trend estimate, incorporating also the end-point asymmetric filters, can be carried out with the MATLAB functions \( \text{trendma}, \) \( \text{minrev}, \) and \( \text{lpfilt}, \)

\[
\begin{align*}
htrend &= \text{trendma}(D); & \text{trend filters of Eq. (27.7.6)} \\
B &= \text{minrev}(htrend,R); & \text{corresponding smoothing matrix} \\
t_hat &= \text{lpfilt}(B,y); & \text{filtering operation}
\end{align*}
\]

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 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 \( r_n = y_n - \hat{t}_n \) to calculate the seasonal component. The following computational steps describe the method:

\[
\begin{align*}
B &= \text{minrev}(\text{trendma}(D),R); & \text{trend moving-average, with minimum-revision end-filters} \\
yt &= \text{lpfilt}(B,y); & \text{trend component} \\
yr &= y - yt; & \text{seasonal + irregular components} \\
ys &= \text{sigav}(yr,D); & \text{seasonal component} \\
yi &= yr - ys; & \text{irregular component}
\end{align*}
\]

For a multiplicative decomposition, \( y_n = s_n t_n v_n, \) the last three steps are replaced by,

\[
\begin{align*}
yr &= y ./ yt; & \text{seasonal + irregular components} \\
ys &= \text{sigav}(yr,D); & \text{seasonal component} \\
yi &= yr ./ ys; & \text{irregular component}
\end{align*}
\]

The function \( \text{cldec} \) implements the above steps,

\[
[yt,ys,yi] = \text{cldec}(y,D,R,type); \quad \text{classical decomposition method}
\]

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,
27.7. CLASSICAL SEASONAL DECOMPOSITION

27.7.2: Global Carbon Dioxide Data. Figure 27.7.3 shows on the upper-left the monthly global CO₂ 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 CO₂ 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 \( \times \) 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 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:

```matlab
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 \( \omega \)
L = length(y); Rdb = 80; % Kaiser window parameters
wind = kwindow(L,Rdb)'; % Kaiser window in the AOSP toolbox
Y = abs(freqz(y.*wind,1,w)); Y = Y/max(Y); % DTFT computation
Ysi = abs(freqz(ysi.*wind,1,w)); Ysi = Ysi/max(Ysi); % middle-left graph
figure; plot(k, 20*log10(Y)); % middle-right graph
figure; plot(k, Ysi); % bottom-left graph
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); % bottom-right graph

The signal \( y_t \) 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 \( 100 \times \text{norm}(y_t-y_{t0})/\text{norm}(y_{t0}) \) is found to be 0.05%, 0.07%, and 0.05% for the classical, LPSM, and WH methods, respectively. \( \square \)

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 \( H_T(\omega) \) is given by Eq. (27.7.5) 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.

The sidelobe behavior of 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 seasonalization transformation \( z \rightarrow z^D \) 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 quite 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 \( w(n), -M \leq n \leq M \), where \( N = 2M + 1 \), is defined by

\[
W(z) = \sum_{n=-M}^{M} w(n)z^{-n} \Rightarrow H_{\text{comb}}(z) = W(z^D) = \sum_{n=-M}^{M} w(n)z^{-nD} \tag{27.7.8}
\]

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...
27.7. CLASSICAL SEASONAL DECOMPOSITION

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:

```matlab
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_{\text{trend}}(\omega)
hc = upones(1,N)/N; Hc = abs(freqz(hc,1,w)); % comb filter H_{\text{comb}}(\omega)
hs = conv(hc, compl(ht)); Hs = abs(freqz(hs,1,w)); % seasonal filter H_{\text{s}}(\omega)
ha = compl(ht); Ha = abs(freqz(ha,1,w)); % seasonal adjustment filter

figure; plot(k, Hs); figure; plot(k, Hi, k,Ht,'r--'); % left graph in Fig. 27.7.5
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 was described in Sec. 27.2. The function *compl* computes the impulse response of the complement of a double-sided symmetric filter, that is, $H(z) = 1 - H(z)$, or $h_n = -h_n$. The function *kwindow* computes the Kaiser window (for spectral analysis) [224] for a given length $N$ and sidelobe level $R_{\text{db}}$ 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 $H_{\text{s}}(\omega) = 1 - H_{\text{s}}(\omega) = H_{\text{T}}(\omega) + H_{\text{I}}(\omega)$, 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 $H_{\text{s}}(\omega) = 1 - H_{\text{s}}(\omega) = H_{\text{T}}(\omega) + H_{\text{I}}(\omega)$.

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 $L = ND$.

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 CO₂ data are used as in Example 27.7.2. The classical decomposition method is applied first to determine the trend $r_n$, 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 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 $(N - 1)/2$ 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 $3 \times 3$ double-sided seasonal moving-average filter, which is discussed below. The MATLAB code for generating these graphs is as follows:

```matlab
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;
[t, ys, yr] = cldec(y,D,R); yr = y - yr;

h = ones(1,N)/N; % length-N moving-average
hc = up(h, D); % seasonalized comb filter obtained from h
Bc = upmat(minrev(hc,R), D); % seasonalized minimum-revision filter matrix

ys1 = filter(hc,1,ys); % ordinary causal filtering by the comb filter hc
ys2 = filtdbl(hc,yr); % double-sided filtering
ys3 = 1pfilter(Bc, yr); % double-sided filtering and end-point filters
[ys4,ys5] = smadec(y, D, M, R); % ys4 is the 3×3 moving-average output
```

27. PERIODIC SIGNAL EXTRACTION
27.8. SEASONAL MOVING-AVERAGE FILTERS

The seasonal moving-average filters are widely used in de-seasonalizing 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 \( z \rightarrow z^D \). For example, the resulting \( 3 \times 3 \) and \( 3 \times 5 \) comb filters are:

\[
H_{33}(z) = \frac{1}{3} \left(z^D + 1 + z^{-D}\right) \cdot \frac{1}{3} \left(z^D + 1 + z^{-D}\right)
\]

\[
H_{35}(z) = \frac{1}{3} \left(z^D + 1 + z^{-D}\right) \cdot \frac{1}{5} \left(z^{2D} + z^D + 1 + z^{-D} + z^{-2D}\right)
\]

with symmetric impulse responses,

\[
h_{33} = \frac{1}{9} \left[1, 0, \ldots, 0, 2, 0, \ldots, 0, 3, 0, \ldots, 0, 2, 0, \ldots, 0, 1\right]_{D-1 \text{ zeros}}
\]

\[
h_{35} = \frac{1}{15} \left[1, 2, 0, \ldots, 0, 3, 0, \ldots, 0, 3, 0, \ldots, 0, 2, 0, \ldots, 0, 1\right]
\]

The MATLAB function \texttt{smav} calculates such impulse responses,

\[
h = \text{smav}(N_1,N_2,D);
\]

It is simply:

\[
h = \text{up}(\text{conv(ones(1,N1), ones(1,N2))} / (N1*N2), D);
\]

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 $3 \times 3$, $3 \times 5$, and $3 \times 9$ 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 $3 \times 3$ filter matrix before and after it is upsampling.

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 seasonalyzed $3 \times 3$, $3 \times 5$, or $3 \times 9$ lowpass filter, or a more general seasonalyzed filter such as one obtained from a non-rectangular window. The de-seasonalyzed, 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 $3 \times 3$, $3 \times 5$, or $3 \times 9$ comb filters for the seasonal part. It has usage:

\[
[yt,ys,yi] = \text{smadec}(y,D,M,R,iter,type);
\]

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 $3 \times 3$, $3 \times 5$, or $3 \times 9$ 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:

\[
F = \text{minrev}(\text{trendma}(D),R); \quad \% \text{trend moving-average, with minimum-revision end-filters}
B = \text{smat}(D,M,R); \quad \% \text{seasonal moving-averages, with end-filters}
\]

\[
yt = y; \quad \% \text{initialize iteration}
\]

\[
\text{for } i=1:\text{iter}, \quad \% \text{T component}
\]

\[
yt = \text{lpfilt}(F,yt); \quad \% \text{I component}
\]

\[
yr = y - yt; \quad \% \text{S component}
ys = \text{lpfilt}(B,yr); \quad \% \text{S+I component}
yi = yr - ys; \quad \% \text{I component}
\]

end

The function smat generates the filtering matrix of the seasonalyzed comb filters, including the specific asymmetric filters for the $3 \times 3$, $3 \times 5$, or $3 \times 9$ cases, as well for other cases.


The upper graphs of Fig. 27.8.2 illustrate the application of the smadec function using $D = 12$, $M = 35$ (which selects the $3 \times 5$ 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].

```
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(:,1)/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 $t_n$, $s_n$
```

**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, $\lambda_{opt} = 2039$. The Kaiser window had length $N = 15$ and relative sidelobe level of $R_{db} = 50 \text{ dB}$. The MATLAB code for generating these graphs was as follows:
27.8. SEASONAL MOVING-AVERAGE FILTERS

\[
\begin{align*}
\text{k} &= \text{linspace}(-6,6,1201); \quad w = 2\pi k/12; \quad \% \text{frequency range} \\
\text{ht} &= \text{trendma}(12); \quad \text{Ht} = \text{abs(freqz(ht,1,w)}; \quad \% 2 \times 12 \text{trend filter} \\
\text{hc} &= \text{smav}(3,12,5); \quad \text{hs} = \text{conv}(hc,\text{compl}(ht)); \quad \% \text{upsampled 3} \times 5 \text{comb filter} \\
\text{hs} &= \text{compl}(hs); \quad \text{ha} = \text{compl}(hs); \quad \% \text{seasonal filter, } H_S(\omega) \\
\text{yt} &= \text{whsm}(y,\text{lopt},s); \quad \% \text{extract trend component} \\
\text{yr} &= y-\text{yt}; \quad \% \text{residual component} \\
\end{align*}
\]

The Whittaker-Henderson trend filter was computed using Eq. (26.3.7). The frequency response of the Kaiser comb filter was multiplied by \(e^{io(N-1)/2}\) to make the filter symmetric.

The steps implementing the Whittaker-Henderson/Kaiser decomposition have been incorporated into the MATLAB function \text{whkdec},

\[
[yt,ys,yi] = \text{whkdec}(y,D,s,la,N,Rdb,R,type); \quad \% \text{WH/Kaiser decomposition}
\]

The WH parameters \(s, la\) must be selected in advance, for example, \(\lambda\) can be tentatively estimated using the GCV function \text{whgcv}, but it should be noted that the GCV does not always give a “good” value for \(\lambda\). 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 \text{type} is as in the function \text{smade}.

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 \(2 \times 12\) trend filter of Eq. (27.7.1), the \(3 \times 3\) and \(3 \times 5\) 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 \(2 \times 12\) 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:

1. \[ Y_{T}^{pre} = FY \]
2. \[ Y_{R}^{pre} = Y - Y_{T}^{pre} = (1 - F)Y \]
3. \[ Y_{S}^{pre} = H_{33}Y_{R}^{pre} = H_{33} (1 - F)Y \]
4. \[ Y_{S}^{imp} = Y_{S}^{pre} - F Y_{S}^{pre} = H_{33} (1 - F)^2 Y \]
5. \[ Y_{A}^{pre} = Y - Y_{S}^{imp} = [1 - H(1 - H_{33}(1 - F)^2)] Y \]
6. \[ Y_{T}^{imp} = H Y_{A}^{pre} = H[1 - H_{33}(1 - F)^2] Y \]
7. \[ Y_{R}^{imp} = Y - Y_{T}^{imp} = [1 - H(1 - H_{33}(1 - F)^2)] Y \]
8. \[ Y_{S}^{imp} = H_{33} Y_{R}^{imp} = H_{33} [1 - H(1 - H_{33}(1 - F)^2)] Y \]
9. \[ Y_{S} = Y_{S}^{imp} - F Y_{S}^{imp} = (1 - F)H_{33} [1 - H(1 - H_{33}(1 - F)^2)] Y = H_{55} Y \]
10. \[ Y_{A} = Y - Y_{S} = (1 - H_{55})Y = H_{44} Y \]
11. \[ Y_{T} = H Y_{A} = H(1 - H_{55})Y = H_{44} Y \]
12. \[ Y_{I} = Y_{A} - Y_{T} = (1 - H)(1 - H_{44})Y = H_{44} Y \]

where \( Y_{T}^{pre} = FY \) stands for \( Y_{T}^{pre}(z) = F(z)Y(z) \), etc., and \( F(z), H_{33}(z), H_{44}(z), H(z) \) 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:

\[
\begin{align*}
H_{S} &= (1 - F)H_{33} [1 - H(1 - H_{33}(1 - F)^2)] \quad \text{(seasonal)} \\
H_{A} &= 1 - H_{55} \quad \text{(seasonally-adjusted)} \\
H_{T} &= H(1 - H_{55}) \quad \text{(trend)} \\
H_{I} &= (1 - H)(1 - H_{44}) \quad \text{(irregular)}
\end{align*}
\]

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:

\[
k = linspace(-6,6,1201); \quad w = 2*pi*k/12; \quad \text{frequency axis } -\pi \leq \omega \leq \pi
\]

\[
hf = trendma(12); \quad \text{2×12 trend filter, } F\]
\[
hfc = compl(hf); \quad \text{complement of trend filter, } 1 - F\]
\[
h33 = smav(3,3,12); \quad \text{3×3 comb filter, } H_{33}\]
\[
h35 = smav(3,5,12); \quad \text{3×5 comb filter, } H_{35}\]
\[
N=13; \quad he = lpr2s(N,3,3); \quad \text{13-term Henderson filter, } H\]
\[
g = conv(hfc,hfc); \quad \text{G = } (1 - F)^2, \quad \text{G is temporary variable}\]
\[
g = compl(conv(h33, g)); \quad \text{G = } 1 - H(1 - H_{33}(1 - F)^2)\]
\[
g = compl(conv(h35, g)); \quad \text{G = } 1 - H(1 - H_{33}(1 - F)^2)\]

\[
hs = conv(hfc,g); \quad \text{hs = abs(freqz(hs,1,w)); \quad \text{seasonal}}
\]
\[
ha = compl(hs); \quad \text{ha = abs(freqz(ha,1,w)); \quad \text{adjustment}}
\]
\[
h = conv(he,ha); \quad \text{ht = abs(freqz(ht,1,w)); \quad \text{trend}}
\]
\[
hv = conv(compl(he),ha); \quad \text{hv = abs(freqz(hv,1,w)); \quad \text{irregular}}
\]

\[
figure; plot(k, Hs); \quad \text{figure; plot(k, Ht); \quad \text{upper graphs}}
\]
\[
figure; plot(k, ha); \quad \text{figure; plot(k, Hv); \quad \text{lower graphs}}
\]

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,

\[
[yt,ys,yi] = x11dec(y,D,M1,M2,N1,N2,R,type); \quad \text{X-11 decomposition method}
\]

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]:

\[
\begin{array}{c|c}
N & R \\
\hline
5 & 0.001 \\
7 & 4.5 \\
9 & 1.0 \\
13 & 3.5 \\
23 & 4.5 \\
\end{array}
\]

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/, under category: Unemployment > Labor Force Statistics > on-screen data search. The already seasonally adjusted data are also available as series LNU03000025. 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 \( s_n \).
The upper-right graph shows the seasonally-adjusted signal $d_n = y_n - s_n = t_n + v_n$ 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:

```matlab
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
Y = loadfile('unemp-20-sa.dat'); % seasonally adjusted data
i = find(Y==1980); Y = Y(i:end,2:13)';
yadj = Y(:)/1000;
% available seasonally adjusted
D=12; ML=33; M2=35; NL=13; N2=13; R=3.5; type='a';
[yt,ys,yi] = x11dec(y,D,ML,M2,NL,N2,R,type);
% X-11 parameters
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
%[yt,ys,yi] = x11dec(y,D,ML,M2,NL,N2,R,type);
% X-11 method
%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
```

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 $R_{DB} = 50$ dB. The code for that is included above but it is commented out. \[\square\]

### 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 $x_n, 0 \leq n \leq L-1$, by a double-sided filter $h_m, -M \leq m \leq M$, 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:

$$y_n = \sum_{m=-M}^{M} h_m x_{n-m}, \quad M \leq n \leq L - 1 - M$$

(27.10.1)
The overall operation can be cast in convolution matrix form. For example, for $L = 8$ and $M = 2$, we have:

\[
\begin{bmatrix}
0 \\
y_0 \\
y_1 \\
\vdots \\
y_2 \\
y_3 \\
\vdots \\
y_5 \\
0 \\
y_7
\end{bmatrix} = \begin{bmatrix}
h_0 & h_1 & h_2 & 0 & 0 & 0 & 0 & 0 \\
h_1 & h_0 & h_2 & 0 & 0 & 0 & 0 & 0 \\
\vdots \\
h_2 & h_1 & h_0 & h_2 & 0 & 0 & 0 & 0 \\
0 & h_2 & h_1 & h_0 & h_2 & 0 & 0 & 0 \\
0 & 0 & h_2 & h_1 & h_0 & h_2 & 0 & 0 \\
\vdots \\
0 & 0 & 0 & h_2 & h_1 & h_0 & h_2 & 0 \\
0 & 0 & 0 & 0 & h_2 & h_1 & h_0 & h_2 \\
0 & 0 & 0 & 0 & 0 & h_2 & h_1 & h_0 \\
0 & 0 & 0 & 0 & 0 & 0 & h_2 & h_1
\end{bmatrix} \begin{bmatrix}
x_0 \\
x_1 \\
x_2 \\
\vdots \\
x_5 \\
x_6 \\
\vdots \\
x_9
\end{bmatrix}
\]

(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,

\[
B = \begin{bmatrix}
h_0 & h_1 & h_2 & 0 & 0 \\
h_1 & h_0 & h_2 & 0 & 0 \\
h_2 & h_1 & h_0 & h_2 & 0 \\
0 & h_2 & h_1 & h_0 & h_2 \\
0 & 0 & h_2 & h_1 & h_0 \\
0 & 0 & 0 & h_2 & h_1 \\
0 & 0 & 0 & 0 & h_2 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

(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 in a prediction matrix $B$.

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 $h_m$, $-M \leq m \leq M$, one must use appropriately designed end-point filters. For example, Eq. (27.10.2) would change to:

\[
\begin{bmatrix}
y_0 \\
y_1 \\
\vdots \\
y_2 \\
y_3 \\
\vdots \\
y_5 \\
y_6 \\
y_7
\end{bmatrix} = \begin{bmatrix}
f_{i-2}^0 & f_{i-1}^1 & f_i^2 & 0 & 0 & 0 & 0 & 0 \\
f_{i-1}^1 & f_i^2 & f_{i+1}^1 & f_{i+2}^0 & 0 & 0 & 0 & 0 \\
\vdots \\
h_{i-2} & h_{i-1} & h_i & h_{i+1} & h_{i+2} & 0 & 0 & 0 \\
0 & h_{i-2} & h_{i-1} & h_i & h_{i+1} & h_{i+2} & 0 & 0 \\
0 & 0 & h_{i-2} & h_{i-1} & h_i & h_{i+1} & h_{i+2} & 0 \\
\vdots \\
0 & 0 & 0 & h_{i-2} & h_{i-1} & h_i & h_{i+1} & h_{i+2} \\
0 & 0 & 0 & 0 & h_{i-2} & h_{i-1} & h_i & h_{i+1} \\
0 & 0 & 0 & 0 & 0 & h_{i-2} & h_{i-1} & h_i \\
0 & 0 & 0 & 0 & 0 & 0 & h_{i-2} & h_{i-1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & h_{i-2}
\end{bmatrix} \begin{bmatrix}
x_0 \\
x_1 \\
\vdots \\
x_5 \\
x_6 \\
\vdots \\
x_9
\end{bmatrix}
\]

(27.10.4)

where the filters $f_m^i$ and $f_{i-1}^j$ are used for computing the first two transient outputs $y_0, y_1$, and the filters $g_m^i$ and $g_{i-1}^j$ are for the last two outputs $y_6, y_7$. The corresponding $B$ matrix would be in this case:

\[
B = \begin{bmatrix}
f_0^0 & f_1^1 & h_2 & 0 & 0 \\
f_1^1 & f_0^2 & h_1 & g_1^0 & 0 \\
f_0^2 & f_1^1 & h_0 & g_1^2 & 0 \\
0 & f_2^1 & h_1 & g_1^0 & 0 \\
0 & 0 & h_2 & g_1^1 & 0 \\
0 & 0 & 0 & g_1^0 & 0
\end{bmatrix}
\]

(27.10.5)

More generally, the filters $f_m^i, g_m^i, i = 0, 1, \ldots, M - 1$, compute the first $M$ and last $M$ output samples $y_1, y_{L-1-i}$, respectively, through the convolutional equations:

\[
y_i = \sum_{m=-M}^{i} f_m^i x_{i-m} = \sum_{m=-M}^{i} f_m^i x_{0} + \cdots + f_m^i x_{1} + \cdots + f_m^i x_{i-M} + \cdots + f_m^i x_{i-1} + \cdots + f_m^i x_{i-M}
\]

(27.10.6)

for $i = 0, 1, \ldots, M - 1$, 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, such as Musgrave’s method

\[
\sum_{m=-M}^{M} h_m = 1
\]

(27.10.7)

The asymmetric filters $f_m^i, g_m^i$ are required to satisfy similar moment constraints:

\[
\sum_{m=-M}^{i} f_m^i = 1, \quad \sum_{m=-i}^{M} g_m^i = 1
\]

(27.10.8)

The design is based on a minimum-revision criterion. When the data record has length $L$, the $i$th 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 $L-1+i$, 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,

\[
y_{L-1-i}^{rev} = \sum_{m=-M}^{M} g_m^i x_{L-1-i-m}, \quad y_{L-1-i} = \sum_{m=-M}^{M} h_m x_{L-1-i-m}
\]

Muscgrave’s criterion selects $g_m^i$ to minimize the mean-square revision error $E[e_{L-1-i}^2]$, where $e_{L-1-i} = y_{L-1-i}^{rev} - y_{L-1-i}$, under the assumption that locally the input series is linear, that is, $x_{L-1-i-m} = a + b m + v_m$, with $a$, $b$ constant parameters, and $v_m$ zero-mean
white noise with variance $\sigma^2$. The mean-square error becomes then,

$$ E[e_{i}^{2}] = E \left[ \sum_{m=-M}^{M} g_{m}^{i}(a + bm + v_{m}) - \sum_{m=-M}^{M} h_{m}(a + bm + v_{m}) \right]^2 $$

and summing up over $m$, results in two equations for the two unknowns $\lambda, \beta$:

$$ E[\epsilon] = E \left[ \left( \sum_{m=-M}^{M} mg_{m}^{i} + \sum_{m=-M}^{M} (g_{m}^{i} - h_{m})v_{m} - \sum_{m=-M}^{M} h_{m}v_{m} \right)^2 \right] $$

$$ = \sigma^2 \sum_{m=-M}^{M} (g_{m}^{i} - h_{m})^2 + b^2 \left( \sum_{m=-M}^{M} mg_{m}^{i} \right)^2 + \text{const.} $$

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 $\sum_{m=-M}^{M} mh_{m} = 0$, which follows from the assumed symmetry of $h_{m}$. Defining the constant $\beta^2 = b^2/\sigma^2$, it follows that the optimum filter $g_{m}^{opt}$ will be the solution of the following optimization criterion, which incorporates the constraint (27.10.8) by means of a Lagrange multiplier $\lambda$:

$$ J = \sum_{m=-M}^{M} (g_{m}^{i} - h_{m})^2 + \beta^2 \left( \sum_{m=-M}^{M} mg_{m}^{i} \right)^2 + \lambda \left( 1 - \sum_{m=-M}^{M} g_{m}^{i} \right) = \min $$

In a similar fashion, we can show that the filters $f_{m}^{i}$ are the solutions of

$$ J = \sum_{m=-M}^{M} (f_{m}^{i} - h_{m})^2 + \beta^2 \left( \sum_{m=-M}^{M} mf_{m}^{i} \right)^2 + \lambda \left( 1 - \sum_{m=-M}^{M} f_{m}^{i} \right) = \min $$

Because $h_{m}$ is even in $m$, it follows (by changing variables $m \rightarrow -m$ 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]:

$$ g_{m}^{i} = h_{m} + \frac{A_{i}}{M + i + 1} + \frac{\beta^2 B_{i}}{D_{i}} (m - \mu_{i}), \quad -i \leq m \leq M $$

for $i = 0, 1, \ldots, M - 1$, with the constants $A_{i}, B_{i}, D_{i}, \mu_{i}$ defined by,

$$ A_{i} = \sum_{m=-M}^{-i-1} h_{m}, \quad B_{i} = \sum_{m=-M}^{-i-1} (m - \mu_{i})h_{m}, \quad i = 0, 1, \ldots, M - 1 $$

$$ \mu_{i} = \frac{M - i}{2}, \quad D_{i} = 1 + \frac{\beta^2}{12} (M + i)(M + i + 1)(M + i + 2) $$

To show Eq. (27.10.12), we set the gradient of $J$ in (27.10.10) to zero, $\partial J/\partial g_{m} = 0$, to get,

$$ \frac{\partial J}{\partial g_{m}} = g_{m} - h_{m} + \lambda - \beta^2 Gm, \quad G = \sum_{m=-M}^{M} mg_{m} $$

and summing up over $m$ and using the constraint (27.10.8), and then, multiplying by $m$ and solving Eqs. (27.10.15) for the constants $\lambda, \beta$ and substituting them in (27.10.14), gives the solution (27.10.12). The parameter $\beta$ is usually computed in terms of the Musgrave parameter $R$, the two being related by

$$ R^2 = \frac{4}{\pi \beta^2} \Rightarrow \beta^2 = \frac{4}{\pi R^2} $$

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`,

```matlab
8 = 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).
27.10. MUSGRAVE ASYMMETRIC FILTERS

\[ h = \text{lprs2}(7,3,3) = \{-0.0587, 0.0587, 0.2937, 0.4126, 0.2937, 0.0587, -0.0587\} \]

\[
\begin{bmatrix}
0.5345 & 0.2892 & 0.0336 & -0.0587 & 0 & 0 & 0 \\
0.3833 & 0.4103 & 0.2747 & 0.0587 & -0.0531 & 0 & 0 \\
0.1160 & 0.2937 & 0.3997 & 0.2937 & 0.0582 & -0.0542 & 0 \\
0 & -0.0338 & 0.0610 & 0.2870 & 0.4126 & 0.2870 & 0.0610 & -0.0338 \\
0 & 0 & -0.0542 & 0.0582 & 0.2937 & 0.3997 & 0.2937 & 0.1160 \\
0 & 0 & 0 & -0.0587 & 0.0336 & 0.0587 & 0.2747 & 0.4103 & 0.3833 \\
0 & 0 & 0 & 0 & -0.0587 & 0.0336 & 0.2892 & 0.5345 \\
\end{bmatrix}
\]

\[
B = \minrev(h, 4, 5) = \]

\[
\begin{bmatrix}
0.8182 & 0.1836 & -0.0587 & -0.0587 & 0.0336 & 0.0682 & 0.0610 & -0.1049 \\
0.4895 & 0.4510 & 0.2741 & 0.0587 & -0.0951 & -0.0874 & -0.0874 & 0.1818 \\
0.2797 & 0.1049 & 0.3357 & 0.4126 & 0.3357 & 0.1049 & -0.2797 & 0.0951 \\
-0.2448 & 0.4283 & 0.5245 & 0.2937 & -0.0140 & -0.4186 & 0.0140 & 0.3999 \\
0.1399 & -0.1486 & -0.0140 & 0.2937 & 0.5245 & 0.4283 & -0.2448 & 0.0140 \\
0.1818 & -0.0874 & -0.0951 & 0.0587 & 0.2741 & 0.4510 & 0.4895 & -0.0874 \\
0.1049 & 0.0682 & 0.0336 & -0.0587 & 0.0587 & 0.1836 & 0.8182 & -0.0587 \\
\end{bmatrix}
\]

The mean-square revision error (27.10.9) was calculated assuming a linear function of time for the input. The criterion can be generalized to higher-order polynomials. For example, for a second-order polynomial,

\[ x_{t-1-t} = a + bm + cm^2 + \nu_m \]

the mean-square revision error will be:

\[
E[e_{t-1-t}^2] = E\left[ \left( \sum_{m=-M}^{M} g_m(a + bm + cm^2 + \nu_m) - \sum_{m=-M}^{M} h_m(a + bm + cm^2 + \nu_m) \right)^2 \right] = \sigma^2 \sum_{m=-M}^{M} (g_m - h_m)^2 + c^2 \left( \sum_{m=-M}^{M} m^2 g_m \right)^2 + \text{const.}
\]

(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:

\[
h_m = h_{-m}, \quad \sum_{m=-M}^{M} \left[ \begin{array}{c}
1 \\
m \\
m^2 \\
\end{array} \right] h_{m} = \left[ \begin{array}{c}
1 \\
0 \\
0 \\
\end{array} \right], \quad \sum_{m=-M}^{M} \left[ \begin{array}{c}
1 \\
m \\
m^2 \\
\end{array} \right] g_{m} = \left[ \begin{array}{c}
1 \\
0 \\
0 \\
\end{array} \right]
\]

(27.10.19)

Defining \( y^2 = c^2 / \sigma^2 \), we obtain the following optimization criterion, which incorporates the above constraints on \( g_m \) with two Lagrange multipliers \( \lambda_1, \lambda_2 \):

\[
J = \sum_{m=-M}^{M} (g_m - h_m)^2 + y^2 \left( \sum_{m=-M}^{M} m^2 g_m \right)^2 + \lambda_1 \left( 1 - \sum_{m=-M}^{M} g_m \right) - \lambda_2 \left( \sum_{m=-M}^{M} mg_m \right)
\]

(27.10.20)

The vanishing of the gradient gives:

\[
g_m = h_m + \lambda_1 + \lambda_2 m - y^2 G m^2, \quad G = \sum_{m=-M}^{M} m^2 g_m
\]

(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 \( \lambda_1, \lambda_2, G \),

\[
1 = \sum_{m=-M}^{M} h_m + \sum_{m=-M}^{M} 1 \lambda_1 + \left( \sum_{m=-M}^{M} m \right) \lambda_2 - \left( \sum_{m=-M}^{M} m^2 \right) y^2 G
\]

(27.10.22)

\[
0 = \sum_{m=-M}^{M} m h_m + \left( \sum_{m=-M}^{M} m \right) \lambda_1 + \left( \sum_{m=-M}^{M} m^2 \right) \lambda_2 - \left( \sum_{m=-M}^{M} m^3 \right) y^2 G
\]

(27.10.23)

\[
G = \sum_{m=-M}^{M} m^2 h_m + \left( \sum_{m=-M}^{M} m^2 \right) \lambda_1 + \left( \sum_{m=-M}^{M} m^3 \right) \lambda_2 - \left( \sum_{m=-M}^{M} m^4 \right) y^2 G
\]

Substituting the solutions for \( \lambda_1, \lambda_2, G \) into (27.10.21) gives the solution:

\[
g_m = \frac{A_i}{M + i + 1} + \frac{B_i}{\Sigma_i} (m - \mu_i) + \frac{y^2 C_i}{\Delta_i} \left[ (m - \mu_i)^2 - v_i^2 \right], \quad -i \leq m \leq M
\]

(27.10.24)

for \( i = 0, 1, \ldots, M - 1 \), with the same constants \( A_i, B_i, \mu_i \) as in Eq. (27.10.13), and with \( \Sigma_i, \Delta_i, \nu_i, C_i \) are defined by

\[
\Sigma_i = \frac{1}{12} (M + i + 1) (M + i + 2) (M + i + 2), \quad v_i^2 = \frac{1}{12} (M + i + 1) (M + i + 2)
\]

\[
\Delta_i = 1 + \frac{y^2}{180} (M + i - 1) (M + i) (M + i + 1) (M + i + 2)
\]

(27.10.25)

\[
C_i = \frac{\sum_{i=-M}^{-i-1} [(m - \mu_i)^2 - v_i^2] h_m}{M - i - 1}
\]

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 ARMA-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].
27.11. SEASONAL WHITTAKER-HENDERSON DECOMPOSITION

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 is,

$$J = \sum_{n=0}^{N-1} (y_n - t_n)^2 + \lambda \sum_{n=0}^{N-1} (\nabla^2 t_n)^2 \min \quad (27.11.1)$$

where $s$ is the smoothing order and $N$, the length of $y_n$. The seasonalized version with period $D$ replaces this by,

$$J = \sum_{n=0}^{N-1} (y_n - t_n - s_n)^2 + \lambda \sum_{n=0}^{N-1} (\nabla^2 t_n)^2 + \alpha \sum_{n=0}^{N-1} (s_n + s_{n-1} + \cdots + s_{n-D+1})^2 \min \quad (27.11.2)$$

where $s$ is the smoothing order and $N$, the length of $y_n$. The seasonalized version with period $D$ replaces this by,

$$J = \sum_{n=0}^{N-1} (y_n - t_n - s_n)^2 + \lambda \sum_{n=0}^{N-1} (\nabla^2 t_n)^2 + \alpha \sum_{n=0}^{N-1} (s_n + s_{n-1} + \cdots + s_{n-D+1})^2 \min \quad (27.11.2)$$

A fourth term, $\beta \sum_{n=0}^{N-1} (s_n - s_{n-D})^2$, may be added [1215,1216], but it is generally not necessary for the following minimization. The minimization of $J$ forces the sum

$$S_n = s_n + s_{n-1} + \cdots + s_{n-D+1} \quad (27.11.3)$$

to become small, ideally zero, and as a consequence the quantity $s_n - s_{n-D} = s_{n-D} - s_{n-1}$ 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 vector form as,

$$J = (y - t - s)^T (y - t - s) + \lambda t^T (D^2 t J) + t + \alpha s^T (A^T A) s = \min \quad (27.11.4)$$

where, as discussed in general terms in Sec. 26.2, the matrices $D$, $A$ 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,

$$D_s(z) = (1 - z^{-1})^s, \quad D_t = \text{binom}(s), \quad A(z) = \sum_{k=0}^{D-1} z^{-k}, \quad A = \text{convmat}(\text{flip}(d), N-s)^T$$

For example, we have for $N = 7$, $s = 2$, and $D = 4$:

$$D_s = \begin{bmatrix} 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 & 1 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \quad (27.11.5)$$

The solution of the minimization problem (27.11.4) is obtained from the vanishing of the gradient of $J$ with respect to $t$ and $s$, which results in the system and its solution:

$$\begin{align*}
(I + P) t + s &= y \\
(t + (I + Q) s &= y) \\
s &= y - (I + P) t \quad (27.11.7)
\end{align*}$$

where we defined $P = \lambda (D^2 J D_s)$ and $Q = \alpha (A^T A)$. 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, $t = (I + P)^{-1} (y - s)$, 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 $(y - s)$, 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 $\beta$-term mentioned above:

$$[yt, ys, yi] = \text{swhdec}(y, d, s, \lambda, \alpha, \beta); \quad \% \text{seasonal Whittaker-Henderson}$$

The larger the parameters $\alpha, \beta$, 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 $\lambda$. The latter, can be estimated using the GCV criterion. The simultaneous estimation of $\lambda, \alpha, \beta$ 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 $\ell_1$-regularized version can be obtained by replacing the $\ell_2$ norms of the regularizing parts by their $\ell_1$ norms, that is,

$$J = \sum_{n=0}^{N-1} (y_n - t_n - s_n)^2 + \lambda \sum_{n=0}^{N-1} |\nabla^2 t_n| + \alpha \sum_{n=0}^{N-1} |s_n + s_{n-1} + \cdots + s_{n-D+1}| = \min \quad (27.12.1)$$

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 $L_2$ version (bottom graphs). The input parameters were as follows, where $\lambda$ was determined in Example 27.8.2 by the GCV criterion,

$$D = 12, \quad s = 2, \quad \lambda = 2039, \quad \alpha = 10, \quad \beta = 0$$
The MATLAB code used to generate the six graphs was as follows:

```matlab
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

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:

\[ H(\omega) = \frac{1}{1 + \lambda |D_s(\omega)|^2}, \quad \text{where} \quad D_s(\omega) = (1 - e^{-j\omega})^3 \]  

(27.12.2)

and the state-space model was defined by

\[ y_n = t_n + v_n, \quad \nabla^s t_n = w_n \]  

(27.12.3)

where, \(v_n, w_n\) were zero-mean, mutually-uncorrelated, white-noise signals of variances \(\sigma_v^2, \sigma_w^2\), 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:

\[ J = \int_{-\pi}^{\pi} \left[ |Y(\omega) - T(\omega) - S(\omega)|^2 + \lambda |D_s(\omega) T(\omega)|^2 + \alpha |A(\omega) S(\omega)|^2 \right] \frac{d\omega}{2\pi} \]  

(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 \( \partial J / \partial T^* \) and \( \partial J / \partial S^* \), we obtain the equations:

\[
T(\omega) + \lambda |D_s(\omega)|^2 T(\omega) + S(\omega) = Y(\omega) \tag{27.12.5}
\]

\[
S(\omega) + \alpha |A(\omega)|^2 S(\omega) + T(\omega) = Y(\omega)
\]

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,

\[
H_T(\omega) = \frac{\alpha |A(\omega)|^2}{\lambda |D_s(\omega)|^2 + \alpha |A(\omega)|^2 + \lambda |D_s(\omega)|^2 |A(\omega)|^2}
\]

\[
H_S(\omega) = \frac{\lambda |D_s(\omega)|^2}{\lambda |D_s(\omega)|^2 + \alpha |A(\omega)|^2 + \lambda |D_s(\omega)|^2 |A(\omega)|^2}
\]

with \( |D_s(\omega)|^2 \) and \( |A(\omega)|^2 \) given by,

\[
|D_s(\omega)|^2 = |1 - e^{-j\omega/2s}|^2 = |2 \sin(\omega/2)|^2
\]

\[
|A(\omega)|^2 = |1 + e^{-j\omega} + \cdots + e^{-j(D-1)\omega}|^2 = \frac{\sin(\omega D/2)}{\sin(\omega/2)}^2 \tag{27.12.7}
\]

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:

\[
y_n = t_n + s_n + \nu_n, \quad \nabla^D t_n = w_n, \quad s_n + s_{n-1} + \cdots + s_{n-D+1} = u_n \tag{27.12.8}
\]

where \( \nu_n, w_n, u_n \) are mutually-uncorrelated, zero-mean, white noises. The model can be written symbolically in operator form:

\[
y_n = t_n + s_n + \nu_n, \quad D_s(z) t_n = w_n, \quad A(z) s_n = u_n \tag{27.12.9}
\]

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,

\[
\tilde{y}_n = D_s(z) A(z) y_n = \tilde{t}_n + \tilde{s}_n + \tilde{\nu}_n = A(z) w_n + D_s(z) u_n + D_s(z) A(z) \nu_n
\]

\[
\tilde{t}_n = D_s(z) A(z) t_n = A(z) w_n
\]

\[
\tilde{s}_n = D_s(z) A(z) s_n = D_s(z) u_n
\]

\[
\tilde{\nu}_n = D_s(z) A(z) \nu_n \tag{27.12.10}
\]

with spectral densities:

\[
S_{\tilde{y}y}(\omega) = S_{\tilde{t}t}(\omega) = \sigma^2_w |A(\omega)|^2
\]

\[
S_{\tilde{y}y}(\omega) = S_{\tilde{s}s}(\omega) = \sigma^2_w |D_s(\omega)|^2
\]

\[
S_{\tilde{y}y}(\omega) = \sigma^2_w |D_s(\omega)|^2 + \sigma^2_w |A(\omega)|^2 + \sigma^2_w |D_s(\omega) A(\omega)|^2
\]

It follows from [1235–1241] that the optimum Wiener filters for estimating \( t_n, s_n \) will be:

\[
H_T(\omega) = \frac{S_{\tilde{y}y}(\omega) = \sigma^2_w |A(\omega)|^2}{\sigma^2_w |D_s(\omega)|^2 + \sigma^2_w |A(\omega)|^2 + \sigma^2_w |D_s(\omega) A(\omega)|^2} \tag{27.12.11}
\]

\[
H_S(\omega) = \frac{S_{\tilde{y}y}(\omega) = \sigma^2_w |D_s(\omega)|^2}{\sigma^2_w |D_s(\omega)|^2 + \sigma^2_w |A(\omega)|^2 + \sigma^2_w |D_s(\omega) A(\omega)|^2}
\]

It is evident that these are identical to (27.12.6) with the identifications \( \lambda = \sigma^2_s/\sigma^2_w \) and \( \alpha = \sigma^2_s/\sigma^2_w \). 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.](image-url)
In the lower graphs, we increased the parameter $\alpha$ 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:

```matlab
k = linspace(-6,6,2401); w = 2*pi*k/12; % frequencies
D = 12; s = 2; la = 2039; alpha = 10;
a = ones(D,1); A = freqz(a,1,w); % calculate $A(\omega)$
P = la * abs(1 - exp(-j*w)).^(2*s); % evaluate $P(\omega) = \lambda |D_s(\omega)|^2$
Q = alpha * abs(A).^2; % evaluate $Q(\omega) = \alpha |A(\omega)|^2$
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,

$$\sum_{m=0}^{D-1} s_n e^{j\omega_m m} = e^{j\omega_n k} S_k, \quad k = 0, 1, \ldots, D - 1, \quad -\infty < n < \infty$$

27.2 Consider the analog signal $s(t) = \cos(2\pi f_1 t)$ and its sampled version $s_n = \cos(2\pi f_1 n T)$, 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, $\cos(2\pi f_1 (n + D) T) = \cos(2\pi f_1 n T)$, 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).

27.4 Show that the solution of the system (27.11.7) can be written in the more symmetric, but computationally less efficient, form:

$$t = (Q + P + QP)^{-1} Qy$$

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
- Boltzmann 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.

\[
\begin{align*}
\text{logistic: } & f(u) = \frac{1}{1 + e^{-u}}, & f'(u) &= f(u) [1 - f(u)] \\
\text{tanh: } & f(u) = \tanh(u), & f'(u) &= 1 - f^2(u) \\
\text{softplus: } & f(u) = \ln(1 + e^u), & f'(u) &= \frac{1}{1 + e^{-u}} \\
\text{ReLU: } & f(u) = \max(0, u), & f'(u) &= 1 \\
\text{leaky ReLU: } & f(u) = \max(u, \alpha u), & f'(u) &= \begin{cases} 1 & (u \geq 0) \\ \alpha & (u < 0) \end{cases} \\
\text{sinusoid: } & f(u) = \sin u, & f'(u) &= \cos u
\end{align*}
\]

where for the ReLU cases, the notations, \((u \geq 0)\) and \((u < 0)\), denote MATLAB-like logical operations, and the parameter \(\alpha\) must be in the range, \(0 < \alpha < 1\).

During the \textit{training phase} of the neural network, a set of training input/output patterns, \(\{x_0, d\}\), are applied to the network, and the connection matrices and bias weights are adapted to minimize the square deviations of the actual outputs \(x_3\) from the desired outputs \(d\), that is, minimizing the following typical performance index,\(^1\)

\[
\sum_{\text{patterns}} (d - x_3)^T (d - x_3) = \min
\]

\[\text{(28.2.2)}\]

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 \(\mu\), 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 \(b_r\),

\[
\Delta W_{rij} = -\mu \frac{\partial J}{\partial W_{rij}}, \quad \Delta b_{ri} = -\mu \frac{\partial J}{\partial b_{ri}}, \quad r = 0, 1, 2
\]

\[\text{(28.2.3)}\]

The updated weights are then,

\[
W_r = W_r + \Delta W_r, \quad r = 0, 1, 2
\]

\[\text{(28.2.4)}\]

The updates (28.2.4) can be applied in two ways:

\(^1\)being equal to 1 if their argument is true, and 0, otherwise.

\(^\dagger\)more generally, the negative-log-likelihood function based on maximum likelihood is used.
28.3. BACKPROPAGATION ALGORITHM

(i) On a pattern-basis, that is, \( J \) arises only from one pattern pair in the sum (28.2.2), that is, \( J_{\text{patt}} = (d - x_i)^T (d - x_i) \), 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, \( J(W, b) \), to first-order in Taylor series (justified if \( \mu \) is sufficiently small), and apply Eq. (28.2.3), so that component-wise we have,

\[
J(W + \Delta W, b + \Delta b) = J(W, b) + \sum_{r,i,j} \frac{\partial J}{\partial W_{rij}} \Delta W_{rij} + \sum_{r,i} \frac{\partial J}{\partial b_{ri}} \Delta b_{ri}
\]

\[
= J(W, b) - \mu \sum_{r,i,j} \left| \frac{\partial J}{\partial W_{rij}} \right|^2 + \mu \sum_{r,i} \left| \frac{\partial J}{\partial b_{ri}} \right|^2 \leq J(W, b), \quad \text{since } \mu > 0
\]

Thus, the value of \( J \) at the updated weights is smaller than its value at the original weights, \( J(W + \Delta W, b + \Delta b) \leq J(W, b) \), 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 \([x_0, d]\). It makes use of the following gradients of the performance index,

\[
e_r = -\frac{\partial J}{\partial u_r}, \quad r = 1, 2, 3, \quad \text{or, component-wise,} \quad e_{ri} = -\frac{\partial J}{\partial u_{ri}} \quad (28.3.1)
\]

To see how this works, start with the updates of the last connection weights, \( W_2, b_2 \). Given an input pattern, \( x_0 \), then after the forward pass of Eq. (28.2.1), we have computed all layer vectors, \( u_r, x_r, r = 1, 2, 3 \). Because \( J \) depends on \( W_2, b_2 \) only through the variable \( u_1 \), we have for the partial derivatives of the updates,

\[
\Delta W_{2ij} = -\mu \frac{\partial J}{\partial W_{2ij}} = -\mu \frac{\partial J}{\partial u_{3i}} \frac{\partial u_{3i}}{\partial W_{2ij}} = \mu e_{3i} x_{2j} \quad (28.3.2)
\]

\[
\Delta b_{2i} = -\mu \frac{\partial J}{\partial b_{2i}} = -\mu \frac{\partial J}{\partial u_{3i}} \frac{\partial u_{3i}}{\partial b_{2i}} = \mu e_{3i}
\]

where we used the definitions (28.3.1) and the partial derivatives of the connection formula,

\[
u_3 = W_2 x_2 + b_2 \quad \Rightarrow \quad u_{3i} = \sum_j W_{2ij} x_{2j} + b_{2i}
\]

\[
\frac{\partial u_{3i}}{\partial W_{2ij}} = x_{2j}, \quad \frac{\partial u_{3i}}{\partial b_{2i}} = 1
\]

Eqs. (28.3.2) can be written in the compact matrix forms,\(^\dagger\)

\[
\Delta W_2 = \mu e_3 x_2^T, \quad \Delta b_2 = \mu e_3 \quad (28.3.3)
\]

The other updates can similarly be expressed in terms of the gradients (28.3.1). In summary, we have,

\[
\Delta W_0 = \mu e_1 x_0^T, \quad \Delta b_0 = \mu e_1 \quad (28.3.4)
\]

The backpropagation algorithm efficiently calculates the quantities, \( e_1, e_2, e_3 \), 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,

\[
D(u) = \text{diag}[f'(u)], \quad \text{or, element-wise,} \quad D(u) = \delta_{ij} f'(u_i) \quad (28.3.5)
\]

Given the output \( x_3 \) of the network corresponding to the input pattern \( x_0 \), the error relative to the training pattern, \( d \), will be, \( e_3 = d - x_3 \), and its contribution to the performance index \( J \) of Eq. (28.2.2), will be, \( J_{\text{patt}} = (d - x_3)^T (d - x_3) = e_3^T e_3 \). Starting with \( e_3 \), we have component-wise,

\[
e_{3i} = -\frac{\partial J_{\text{patt}}}{\partial u_{3i}} = -\frac{\partial x_{3i}}{\partial u_{3i}} \frac{\partial J_{\text{patt}}}{\partial x_{3i}} = f'(u_{3i}) (d_i - x_{3i}) \quad \text{or,}
\]

\[
e_3 = D(u_3) (d - x_3) = D(u_3) e_3
\]

Next, for \( e_2 \), because \( J_{\text{patt}} \) depends on \( x_2 \) through \( u_1 \), and for \( e_1 \), because \( J_{\text{patt}} \) depends on \( x_1 \) through \( u_2 \), we have,

\[
e_{2i} = \frac{\partial J_{\text{patt}}}{\partial u_{2i}} = \frac{\partial x_{2i}}{\partial u_{2i}} \frac{\partial J_{\text{patt}}}{\partial x_{2i}} = -D(u_2) \frac{\partial J_{\text{patt}}}{\partial x_{2i}}
\]

\[
= -D(u_2) \sum_j \frac{\partial J_{\text{patt}}}{\partial u_{3j}} \frac{\partial u_{3j}}{\partial x_{2i}} = D(u_2) \sum_j W_{2ij} e_{3j}
\]

\[
e_{1i} = \frac{\partial J_{\text{patt}}}{\partial u_{1i}} = \frac{\partial x_{1i}}{\partial u_{1i}} \frac{\partial J_{\text{patt}}}{\partial x_{1i}} = -D(u_1) \frac{\partial J_{\text{patt}}}{\partial x_{1i}}
\]

\[
= -D(u_1) \sum_j \frac{\partial J_{\text{patt}}}{\partial u_{3j}} \frac{\partial u_{3j}}{\partial x_{1i}} = D(u_1) \sum_j W_{1ij} e_{3j}
\]

\(^\dagger\) superscript \( T \) denotes transposition
These can be written in matrix forms using the transposed connection matrices,

\[ e_2 = D(u_1)W_1^T e_1 \]
\[ e_1 = D(u_1)W_0^T b_0 \]

Finally, putting all the computational steps together, we have for the two-hidden-layer network, for each input/output pattern, \( [x_0, d] \),

\[
\begin{align*}
\text{forward pass} & \\
\mathbf{u}_1 &= W_0 x_0 + b_0 \\
x_1 &= f(u_1) \\
\mathbf{u}_2 &= W_1 x_1 + b_1 \\
x_2 &= f(u_2) \\
\mathbf{u}_3 &= W_2 x_2 + b_2 \\
x_3 &= f(u_3)
\end{align*}
\]

\[
\begin{align*}
\text{backpropagation} & \\
e_3 &= d - x_3 \\
e_2 &= D(u_1) e_3 \\
e_2 &= W_1^T e_3 \\
e_1 &= D(u_2) e_2 \\
e_1 &= W_0^T e_2 \\
e_1 &= D(u_1) e_1 
\end{align*}
\]

\[
\begin{align*}
\text{weight updates} & \\
\Delta W_2 &= \mu e_1 x_2^T \\
\Delta b_2 &= \mu e_2 \\
\Delta W_1 &= \mu e_2 x_1^T \\
\Delta b_1 &= \mu e_2 \\
\Delta W_0 &= \mu e_1 x_0^T \\
\Delta b_0 &= \mu e_1
\end{align*}
\]

If the last activation function is omitted, then we simply set, \( x_3 = u_3 \) and \( e_3 = e_3 \). For a single hidden layer network depicted in Fig. 28.3.1, we have the simpler version,

\[
\begin{align*}
\text{forward pass} & \\
\mathbf{u}_1 &= W_0 x_0 + b_0 \\
x_1 &= f(u_1) \\
\mathbf{u}_2 &= W_1 x_1 + b_1 \\
x_2 &= f(u_2) \\
\mathbf{u}_3 &= W_2 x_2 + b_2 \\
x_3 &= f(u_3)
\end{align*}
\]

\[
\begin{align*}
\text{backpropagation} & \\
e_2 &= d - x_2 \\
e_2 &= D(u_2) e_2 \\
e_1 &= W_0^T e_2 \\
e_1 &= D(u_1) e_1 
\end{align*}
\]

\[
\begin{align*}
\text{weight updates} & \\
\Delta W_1 &= \mu e_2 x_1^T \\
\Delta b_1 &= \mu e_2 \\
\Delta W_0 &= \mu e_1 x_0^T \\
\Delta b_0 &= \mu e_1
\end{align*}
\]

Fig. 28.3.1 Neural network with one hidden layer.

In practice, the weight updates, \( \Delta W_r, \Delta b_r \), are smoothed using a simple EMA with a forgetting factor \( \lambda \) (referred to as momentum updating) before the new weights are computed, that is, instead of using the corrections of Eq. (28.3.4),

\[
\begin{align*}
\Delta W_r &= \lambda \Delta W_{r+1} + (1 - \lambda) \mu e_{r+1} x_r^T, \quad r = 0, 1, 2 \\
\Delta b_r &= \lambda \Delta b_{r+1} + \mu e_{r+1}
\end{align*}
\]

we use their EMA-smoothed versions, with some forgetting factor, \( 0 < \lambda < 1 \),

\[
\begin{align*}
\Delta W_r &= \lambda \Delta W_r + \mu e_{r+1} x_r^T, \quad r = 0, 1, 2 \\
\Delta b_r &= \lambda \Delta b_r + \mu e_{r+1}
\end{align*}
\]

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 \( 0 \leq \lambda < 1 \) for pattern updating, and the weights are usually initialized to random values prior to starting the iteration,

\[
\begin{align*}
\text{for each epoch, do} & \\
\text{for each pattern, do} & \\
\quad \text{forward pass, calculate, } x_r, \ r = 1, 2, 3 & \\
\quad \text{backpropagation pass, calculate, } e_{r+1}, \ r = 2, 1, 0 & \\
\quad \text{weight updates, update now} & \\
\quad \text{if pattern-updating, update now} & \\
\text{end pattern loop} & \\
\text{if epoch-updating, update after all epoch patterns} & \\
\text{end} & \\
\text{end epoch loop}
\end{align*}
\]

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

\[ x_0, x_1, x_2 \]

\[ d \]

\[ e_2 = d - x_2 \]
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 $2 \times 8$ 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.

The complete MATLAB code for this example can be found in the file, nn332.m, located in the NN folder of the ISP2e toolbox.

### 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 ($\lambda < 1$) updating with sigmoid vs. tanh activation functions, but same initializations.

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].
28.4. COMPUTER EXPERIMENTS

3:3:2:2 net, $\mu=0.4$, $\lambda=0.1$, sigmoid activation

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 $N_t + p \leq n \leq N - 1$, 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, $p \leq n \leq N_t + p - 1$,

\[
\text{input vectors} = \begin{bmatrix} y_{n-1} \\ y_{n-2} \\ \vdots \\ y_{n-p} \end{bmatrix}, \quad \text{scalar outputs} = y_n
\]

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.

Fig. 28.4.3  Times series prediction with a neural network.
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.

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, \( 4 \leq n \leq 299 \).

The first \( N_t = 250 \) of these, that is, for \( 4 \leq n \leq 253 \), will be taken as the training set, and the remaining 46 ones, \( 254 \leq n \leq 299 \), 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 \( \mu \) 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 (color online).

The complete MATLAB code for this example is in the file, \texttt{nn441sun.m}, located in the NN folder of the ISP2e toolbox.
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.

![Uniform and Gaussian probability distributions](image)

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:

```c
u = ran(&iseed);
```

where the output is a real number in the interval $0 \leq u < 1$.

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.

![Single and successive calls to routine ran](image)

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 $0 \leq I_0 \leq m - 1$, the LCG algorithm is the recursion:

$$
I_n = (aI_{n-1} + c) \mod m
$$

(LCG algorithm) \hspace{1cm} (A.1)

Because of the modulo-$m$ operation, all the seeds $I_n$ are restricted to the interval:

$$
0 \leq I_n \leq m - 1
$$

This implies that $u_n$ will be in the interval $0 \leq u_n < 1$, 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:

$$
a = 7^5 = 16807, \quad m = 2^{31} - 1 = 2147483647
$$

(A.2)

† In some implementations, the seed is hidden from the user.

‡ If $c = 0$, one must pick $I_0 \neq 0$. 

---

1390 29. APPENDICES
With these parameters, Eq. (A.1) cannot be implemented in a straightforward fashion because the product $al$ can take on extremely large values that exceed the integer range of many computers. For example, if $I = m - 1 = 2147483647$, then $al \approx 3.6 \times 10^{13} \approx 2^{45}$, which exceeds the typical signed “long” (4-byte) integer range of most micros:

$$-2^{31} \leq I \leq 2^{31} - 1$$

(A.3)

A portable implementation suggested by Schrage [1258,1270] rearranges the computation of $(al) \mod(m)$ 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,

$$m = aq + r$$

(A.4)

where $r$ is in the range $0 \leq r < a - 1$. The key requirement for the method to work is that $r$ satisfy the additional constraint:

$$r < q$$

(A.5)

For the choice (A.2), we have the values for $q$ and $r$ satisfying (A.4) and (A.5):

$$q = 127773, \quad r = 2836$$

(A.6)

Given an integer seed $I$ in the range $0 \leq I \leq m - 1$, the quantity $J = (al) \mod(m)$ is the remainder of the division of $al$ by $m$, that is,

$$al = aK + J, \quad \text{where} \quad 0 \leq J \leq m - 1$$

(A.7)

Schrage’s method calculates $J$ without directly performing the multiplication $al$. As a preliminary step, the seed $I$ is divided by $q$, giving the quotient and remainder:

$$I = qk + j, \quad \text{where} \quad 0 \leq j < q - 1$$

(A.8)

Then, the quantity $al$ can be expressed as follows:

$$al = a(qk + j) = aqk + aj = (m - r)k + aj = mk + (aj - rk)$$

(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 $0 \leq aj - rk \leq m - 1$. Note that both quantities $aj$ and $rk$ lie in this range:

$$0 \leq aj \leq m - 1, \quad 0 \leq rk \leq m - 1$$

(A.10)

Indeed, the first follows from the fact that $j < q$ from Eq. (A.8), so that

$$0 \leq aj < aq = m - r < m$$

The second one follows from Eqs. (A.5) and (A.8):

$$0 \leq rk < qk = I - j \leq I \leq m - 1$$

Combining the inequalities (A.10) we find the range of the quantity $aj - rk$:

$$-(m - 1) \leq aj - rk \leq m - 1$$

If $aj - rk$ lies in the positive half, $0 \leq aj - rk \leq m - 1$, then we must necessarily have $J = aj - rk$ and $K = k$. But, if it lies in the negative half, $-m + 1 \leq aj - rk \leq -1$, we must shift it by $m$ so that $1 \leq aj - rk + m \leq m - 1$. In this case, we have $al = mk + aj - rk = m(k - 1) + (m + aj - rk)$; therefore, $J = m + aj - rk$ and $K = k - 1$.

Denoting $K = \left\lfloor I/q \right\rfloor$ and $j = I \mod q$, we can summarize the computation of the new seed $J = (al) \mod(m)$ as follows:

```c
#define m 2147483647
#define q 127773
#define r 2836

double ran(iseed) {
    long iseed; /* ran.c - uniform random number generator in [0, 1) */

    iseed passed by address
    double ran(iseed) {
        long *iseed; 
        usage: u = ran(&iseed);
        Given an integer seed I in the range 0 ≤ I ≤ m - 1 do:
        compute J = a(I \mod q) - r[I/q];
        if J < 0, then shift
        J = J + m;
        return (double) J / (double) 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 // long iseed; that is, a = 2^5
define m 2147483647 // note, q = m/a - quotient
define q 127773 // note, r = m/a - remainder
define r 2836
double ran(iseed) {
    long *iseed; 
    usage: u = ran(&iseed);
    Given an integer seed I in the range 0 ≤ I ≤ m - 1 do:
    compute J = a(I \mod q) - r[I/q];
    if J < 0, then shift
    J = J + m;
    return (double) J / (double) m;
}
```

The generated random numbers $u$ are uniformly distributed over the interval $0 \leq u < 1$, 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:
A. RANDOM NUMBER GENERATORS

\[
E[u] = \int_0^1 u p(u) \, du = \int_0^1 u \, du = \frac{1}{2}
\]

\[
\sigma_u^2 = E[u^2] - E[u]^2 = \int_0^1 u^2 \, du - \frac{1}{4} = \frac{1}{3} - \frac{1}{4} = \frac{1}{12}
\]

(A.11)

To generate a random number which is uniformly distributed over a different interval, say \( a \leq v < b \), we generate a uniform \( u \) over \([0,1)\) and then shift and scale it to obtain:

\[
v = a + (b - a)u
\]

(A.12)

The mean and variance of \( v \) will be:

\[
E[v] = a + (b - a) \frac{1}{2} = \frac{a + b}{2}, \quad \sigma_v^2 = \frac{(b - a)^2}{12}
\]

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:

\[
v = u_1 + u_2 + \cdots + u_{12}
\]

(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:

\[
E[v] = E[u_1] + \cdots + E[u_{12}] = \frac{1}{2} + \cdots + \frac{1}{2} = 12 \times \frac{1}{2} = 6
\]

\[
\sigma_v^2 = \sigma_{u_1}^2 + \cdots + \sigma_{u_{12}}^2 = 1 + \cdots + 1 = 12 \times \frac{1}{12} = 1
\]

Because each \( u_i \) has finite range \( 0 \leq u_i < 1 \), the range of \( v \) will also be finite: \( 0 \leq v < 12 \), with mean at 6. Even though \( v \) has finite range, it has 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 \( \pm 40\sigma_v \).

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 \texttt{gran.c} implements this method using the uniform routine \texttt{ran}. Its inputs are \( \{m,s\} \) and a seed. Its outputs are the random number \( x \) and the updated seed:

```c
/* gran.c - gaussian random number generator */

double ran();

double gran(m, s, iseed);

double m, s;

long *iseed;

{ double v = 0;
  val, say
  for (i = 0; i < 12; i++)
    v += ran(iseed);
  return s * (v - 6) + m;
}
```

Its usage is demonstrated by the following program segment. As in the case of \texttt{ran}, the seed must be declared to be \texttt{long}:

```c
iseed = 123456;

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 \texttt{ran}, such as,

\[ u_n = \texttt{ran}(&\text{iseed}) - 0.5, \quad n = 0, 1, 2, \ldots \]

corresponds to a white noise signal because the generated numbers are mutually independent. The autocorrelation function and power spectral density of such signal are,

\[
R_{uu}(k) = \sigma_u^2 \delta(k), \quad S_{uu}(f) = \sigma_u^2
\]

(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 instant, that is, at times \( n = 0, D, 2D, 3D, \ldots \), 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 \( D = 5 \).
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 \texttt{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.

```c
double ranh(D, u, &q, &iseed){
  double y;
  hold sample for \( D \) calls
  cdelay2(D-1, q);  // decrement \( q \) and wrap mod-\( D \)
  if \((\ast q == 0)\) {
    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 \texttt{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 \texttt{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:

```c
double u[2];  // \( u \) is a 2-dimensional array
int D, q;
long iseed = 654321;
u[0] = ran(&iseed) - 0.5;  // initialize \( u[0] \) (zero mean)
q = 0;
for \((n=0; n<D; n++)\)
  \(y[n] = \text{ranh}(D, u, \ &q, \ &iseed)\);  // \( q, \text{iseed} \) 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

\[
y = u[0] + (u[1] - u[0]) \frac{i}{D}, \quad i = 0, 1, \ldots, D - 1
\]

Because we use the routine \texttt{cdelay2}, the circular index \( q \) takes periodically the successive values: \( q = 0, D - 1, D - 2, \ldots, 1 \). These may be mapped to the interpolation index \( i \) by:

\[
i = (D - q) \% D = 0, 1, \ldots, D - 1
\]

where the modulo-\( D \) operation is effective only when \( q = 0 \) giving in that case \( i = D \% D = 0 \). The following routine \texttt{ranl.c} implements the linearly interpolated periodic generator, where now the temporary variable \( u \) is a two-dimensional array:

```c
double ranl(D, u, q, iseed){
  double y;
  linear interpolation
  cdelay2(D-1, q);  // decrement \( q \) and wrap mod-\( D \)
  if \((\ast q == 0)\) {
    u[0] = u[1];  // set new \( u[0] \) and
    u[1] = ran(iseed) - 0.5;  // get new \( u[1] \) (zero mean)
  }
  return y;
}
```

The initialization and usage of the routine are illustrated by the following program segment:

```c
double u[2];  // \( u \) is a 2-dimensional array
int D, q;
u[0] = ran(&iseed) - 0.5;  // initialize \( u[0] \) (zero mean)
q = 0;
for \((n=0; n<D; n++)\)
  \(y[n] = \text{ranl}(D, u, \ &q, \ &iseed)\);  // \( q, \text{iseed} \) are passed by address
```
assuming that the random numbers \( u_m, m = 0, 1, \ldots \) 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 \( \{0, 1, \ldots, D - 1\} \) 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:

\[
x(n) = \sum_{m=\infty}^{\infty} \delta(n - mD - d) u_m
\]

\[
y(n) = \sum_{m=\infty}^{\infty} h(n - mD - d) u_m
\]

(A.15)

Each realization of \( x(n) \) and \( y(n) \) is defined by a random value of \( d \) from the set \( \{0, 1, \ldots, D - 1\} \) and the random numbers \( u_m \). The particular case \( D = 5 \) and \( d = 3 \) is shown in Fig. A.6 for the hold interpolator.

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 \( \sigma_y^2 = \sigma_n^2/D \), 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):

\[
S_{yy}(f) = |H(f)|^2 \sigma_n^2 \frac{D}{f^2}
\]

(A.16)

where for the hold and linear interpolators, we have from Section 14.3:

\[
H(f) = \frac{\sin(\pi fD/f_s)}{\sin(\pi f/f_s)} e^{-j\pi(D-1)/f_s}, \quad H(f) = \frac{1}{D} \left[ \frac{\sin(\pi fD/f_s)}{\sin(\pi f/f_s)} \right]^2
\]

(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 0, but at \( q = d \), where the random integer \( d = 0, 1, \ldots, D - 1 \) 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 \( f_c / 2D \). 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:

```c
repeat forever:
    if \( q \leq 0 \) then
        w[0] = ran(&iseed) - 0.5
        i = (D - q) % D
        y = dot(P, b, w) = output
        cdelay2(D - 1, &q)
        if \( q \leq 0 \) then
            delay(P, w)
    else
        yb = y 

where \( b \) is the \( i \)th polyphase subfilter of order \( P \), \( w \) 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 \( w \). See Problem A.5 for a simulation.

A.3 1/f Noise Generators

1/f-noise is also known as \textit{flicker} or \textit{pink} noise, depending on the context. It is characterized by a power spectrum that falls in frequency like 1/f:

\[
S(f) = \frac{A}{f}
\]

To avoid the infinity at \( f = 0 \), this behavior is assumed valid for \( f \geq f_{\text{min}} \), where \( f_{\text{min}} \) is a desired minimum frequency. The spectrum (A.19) is characterized by a 3-dB \textit{per octave} drop, that is, whenever \( f \) doubles, \( S(f) \) drops by a factor of 1/2. Indeed, we have:

\[
S(2f) = \frac{A}{2f} = \frac{1}{2}S(f) \Rightarrow 10 \log_{10} \left( \frac{S(f)}{S(2f)} \right) = 10 \log_{10}(2) = 3 \text{ dB}
\]

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 \textit{any} octave interval is the same. That is, if \( f_1 = 2f_2 \), then \( A \ln(f_2/f_1) = A \ln(2f_1/f_1) = A \ln(2) \), which is \textit{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 \textit{pink noise} and is used to test the frequency response of audio equipment such as loudspeakers. It represents the \textit{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 \textit{average} of several periodically held random numbers with periods that form a geometric progression, \( B = 2^b \), with \( b = 0, 1, 2, \ldots, B - 1 \). That is, we define the random number sequence:

\[
y(n) = \frac{1}{B} \sum_{b=0}^{B-1} y_b(n) = \frac{1}{B} \left[ y_0(n) + y_1(n) + \cdots + y_{B-1}(n) \right]
\]

where the term \( y_b(n) \) is a periodically held random number with period \( B = 2^b \), produced by calling the routine ranh:

\[
y_b(n) = \text{ranh}(B, u, &q[b], &\text{iseed})
\]

Each term \( y_b(n) \) must have its own 1-dimensional array \( u[b] \) and circular index \( q[b] \). The same seed variable \( \text{iseed} \) 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:

```c
/* ran1f.c - 1/f random number generator */

double ranh();
double ran1f(B, u, q, &iseed);
int B, *q;
double *u;
long *iseed;

random hold periodic generator
usage: y = ran1f(B, u, q, &iseed);
q, u are B-dimensional
passed by address
```
A. RANDOM NUMBER GENERATORS

1401

{ 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 \( \sigma^2_{y_b} = \sigma^2_u = 1/12 \), it follows that the mean and variance of the \( 1/f \)-noise sequence \( y(n) \) will be:

\[
E[y] = 0, \quad \sigma^2_y = \frac{\sigma^2_u}{B} = \frac{1}{12B}
\]

(A.22)

The initialization and usage of the routine are illustrated by the following program segment, which generates \( N \) random numbers \( y(n) \):

```c
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;
  random initial u's
  q[b] = (1<<b) * ran(&iseed);
  random initial q's
}
for (n=0; n<N; n++)
  y[n] = ranl(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 \( \{0, 1, \ldots, D-1\} \). To see how the various terms \( y_b(n) \) combine to generate \( y(n) \), consider the case \( B = 4 \):

\[
y(n) = \frac{1}{4}[y_0(n) + y_1(n) + y_2(n) + y_3(n)]
\]

Figures A.7 and A.8 show the generated signal \( y(n) \) and its four component signals \( y_b(n) \), for \( n = 0, 1, \ldots, 199 \). 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, \( q[0] = 0 \), \( b = 0, 1, 2, 3 \).

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 \( f \gtrsim f_{\text{min}} \), where:

\[
f_{\text{min}} = \frac{f_s}{2^B}
\]

(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 \( B = \log_2(f_s/f_{\text{min}}) \).

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:
\[
S_{yy}(f) = \frac{1}{B'} \sum_{b=0}^{B-1} S_{y(b)f}(f) = \frac{1}{B'} \sum_{b=0}^{B-1} \frac{1}{2} \sin^2\left(\frac{\pi f}{2} \frac{b}{f_s} \right) \sigma_u^2
\]
(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: \( S_{yy}(0) = (2\pi - 1) \sigma_u^2 / B' \). 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.

### 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:

\[
\sigma_y^2 = \sigma_u^2 \quad \text{(hold)}, \quad \sigma_y^2 = \frac{2D^2 + 1}{3D^2} \sigma_u^2 \quad \text{(linear)}
\]

A.2 For the hold interpolator case of Eq. (A.15), show that the autocorrelation function of the output signal is:

\[
R_{yy}(k) = E[y(n+k)y(n)] = \left(1 - \frac{|k|}{D}\right) \sigma_y^2, \quad -D \leq k \leq D
\]

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 \( R_{yy}(k) \) of the block for \( k = 0, 1, \ldots, 99 \), 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 \( 0 \leq f \leq f_s \). 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 ran1f.c, that implements this algorithm. Its inputs should be the \( L \) (designed independently), the low-rate delay line vector \( \{w[n]\} \), \( n = 0 \), \( \ldots, 255 \), compute and plot the sample autocorrelation of the sequence for lags 0 \( \leq k \leq 255 \). Compare it Eq. (A.25).

A.6 Using the result of Problem A.2, show that the autocorrelation of the \( 1/f \) noise model of Eq. (A.20) is:

\[
R_{yy}(k) = \left[1 - \frac{b(k)}{B} - \frac{2|k|}{B} (2^{-b(k)} - 2^{-B}) \right] \sigma_y^2
\]

where \( b(k) \) is the ceiling quantity \( b(k) = \lfloor \log_2(|k| + 1) \rfloor \). Draw a sketch of \( R_{yy}(k) \) for \( B = 4 \). Show that the maximum correlation length is \( k_{\text{max}} = 2B - 1 - 1 \).

A.7 Computer Experiment: Autocorrelation Function of \( 1/f \) Noise. Generate a block of \( 1/f \) noise samples \( y(n), n = 0, 1, \ldots, N - 1 \), where \( N = 2000 \) assuming \( B = 8 \). Using the correlation routine corr.c, compute and plot the sample autocorrelation of the sequence for lags 0 \( \leq k \leq 150 \). 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, \( \tau_p = \tau_0 c^b \), \( b = 0, 1, \ldots \), where \( c > 1 \). Discrete-time versions of such filters are of the form \( H(z) = G(1 - a \z^{-1}) \), 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 \( T/\tau \), we may use the first-order approximation \( a = 1 - T/\tau \). The generation model is based on summing the outputs of \( B \) such filters:

\[
y'(n) = \frac{1}{B} \sum_{b=0}^{B-1} y_b(n)
\]

\[
y_b(n) = a_0 y_b(n-1) + G_b x_b(n), \quad b = 0, 1, \ldots, B - 1
\]

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 \texttt{randn}. The \( 1/B \) factor normalizes \( y'(n) \) to unit variance. The power spectrum of the signal \( y(n) \) will be:

\[
S(\omega) = \frac{1}{B} \sum_{b=0}^{B-1} |H_b(\omega)|^2 = \frac{1}{B} \sum_{b=0}^{B-1} \frac{1 - a_b^2}{1 - 2a_b \cos \omega + a_b^2}
\]

The filter parameters can be expressed as \( a_b = e^{-T/\tau_b} = 1 - T/\tau_b = 1 - c^{-b} T/\tau_0 \). As a practical matter, we would like to be able to approximate \( 1/f \) noise over a given interval \( \omega_{\min} \leq \omega \leq \omega_{\max} \). These limits are inversely proportional to the longest and shortest time constants \( \tau_{\max} [1287] \). Thus, we may set \( c^b = \omega_{\max}/\omega_{\min} \). Using the approximation \( a = 1 - \omega/\omega_c \) of Example 15.2, we obtain the "design" equations for the filter parameters:

\[
c = \left( \frac{\omega_{\max}}{\omega_{\min}} \right)^{1/(B - 1)}, \quad a_b = 1 - \omega_{\min} c^{-b} = 1 - \omega_{\max} c^{-b},
\]

for \( b = 0, 1, \ldots, B - 1 \). The model works well over a wide range of frequencies, especially when \( \omega_{\min} \) is very small [1288,1289]. The positivity of \( a_b \) requires \( a_{\min} < 1 \). This requires that \( \omega_{\max} < 2 \) or in terms of the sampling frequency \( f_{\max} \), which is so large 

\[
\omega_{\min} = 0.001 \pi, \quad 0.001 \pi
\]

\[
\omega_{\max} = 0.1 \pi, \quad 0.2 \pi, \quad 0.3 \pi, \quad 0.4 \pi, \quad 0.5 \pi, \quad 0.6 \pi
\]

For each pair \( \{\omega_{\min}, \omega_{\max}\} \), compute the model spectrum (A.27) over the interval \( \omega_{\min} \leq \omega \leq \omega_{\max} \) 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, \( \omega = 2\pi L/K \)).

**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 \( \sigma_x^2 \) through the following third-order filter [1289]:

\[
H(z) = G \left( 1 - 0.98444 z^{-1} \right) \left( 1 - 0.83392 z^{-1} \right) \left( 1 - 0.07568 z^{-1} \right)
\]

\[
= 0.99574 z^{-1} \left( 1 - 0.9791 z^{-1} \right) \left( 1 - 0.53568 z^{-1} \right)
\]

The resulting output sequence \( y(n) \) has power spectrum \( S_y(\omega) = |H(\omega)|^2 \sigma_x^2 \omega_c^2 \), according to Eq. (9.7.1). The filter is lowpass with finite gain at DC; its 3-dB frequency is approximately \( \omega_c = 0.001 \pi \) (e.g., 30 Hz at the audio rate \( f_c = 40 \) kHz). Beyond \( \omega_c \), the filter’s magnitude response squared behaves approximately like \( 1/\omega^2 \) over the rest of the Nyquist interval, that is,

\[
|H(\omega)|^2 = \frac{\text{const}}{\omega^2}, \quad \text{for } \omega_c \leq \omega \leq \pi
\]

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 \( n_{\text{eff}} \) outputs must be discarded, where \( n_{\text{eff}} = \log e / \log a \) is the \( e \)-level time constant of the filter (for example, with \( e = 0.05, a = \max |p| = 0.999574 \), 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:

\[
H(z) = G \left( 1 - b_1 z^{-1} \right) \left( 1 - b_2 z^{-1} \right) \left( 1 - b_3 z^{-1} \right)
\]

\[
= 1 - c z^{-1} \left( 1 - c z^{-1} \right) \left( 1 - c z^{-1} \right)
\]

\[
= 1 - c z^{-1} \left( 1 - c z^{-1} \right) \left( 1 - c z^{-1} \right)
\]

The time constants \( n_{\text{eff}} \) of the three poles are in geometric proportions 1 : \( c : c^2 \).

a. Determine the parameters \( \{a, b_1, c\} \) of the model (A.30) by matching them to those of the model (A.29), that is, set \( a = 0.999574, a^2 = 0.994791 \), and \( b = 0.98844 \). Then solve for \( c \) and determine the remaining pole and zeros: \( a_1^2, b_1^2, b_2^2 \).

b. Evaluate \( |H(\omega)|^2 \) of the filters (A.29) and (A.30) at 500 equally spaced frequencies over the interval \( 0.002 \pi \leq \omega \leq \pi \) 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.
and its dual, supergain, and their relationship to superoscillations. Further details on superoscillations may be found in references [1327–1340]. More information on super-resolution, 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 \(L^2_{\omega_0}\) of functions \(f(t)\) that are square-integrable over the real line, \(-\infty < t < \infty\), (b) the space \(L^2_{t_0}\) of functions \(f(t)\) that are square-integrable over the finite interval \([-t_0, t_0]\), and (c) the subspace \(B_{\omega_0}\) of \(L^2_{\omega_0}\) 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,

\[
f(t) = \int_{-\omega_0}^{\omega_0} \hat{f}(\omega) e^{j\omega t} d\omega, \quad \hat{f}(\omega) = \int_{-\infty}^{\infty} f(t) e^{-j\omega t} dt \tag{B.1}
\]

The PSWF functions, denoted here by \(\psi_n(t)\) with \(n = 0, 1, 2, \ldots\), belong to the subspace \(B_{\omega_0}\) and are defined by the following equivalent expansions in terms of Legendre polynomials or spherical Bessel functions:

\[
\psi_n(t) = \sqrt{\frac{\lambda_n}{t_0}} \sum_k \beta_{nk} \sqrt{k + \frac{1}{2}} P_k \left(\frac{\eta}{\omega_0}\right) = \sqrt{\frac{c}{2\pi t_0}} \sum_k \beta_{nk} \sqrt{k + \frac{1}{2}} 2^{k-n} j_k (\omega_0 t) \tag{B.2}
\]

for \(n = 0, 1, 2, \ldots\), where \(\beta_{nk}\) are the expansion coefficients, \(c = t_0 \omega_0\), and \(\lambda_n\) 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:

\[
\int_{-t_0}^{t_0} \frac{\sin((\omega_0 t - t'))}{\pi (t - t')} \psi_n(t') dt' = \lambda_n \psi_n(t) \quad n = 0, 1, 2, \ldots, \text{ for all } t \tag{B.3}
\]

The Legendre polynomial expansion in (B.2) is numerically accurate for \(|t| \leq t_0\), 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],

\[
P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n} (x^2 - 1)^n, \quad n = 0, 1, 2, \ldots \tag{B.4}
\]

The first few of them are listed below,

\[
P_0(x) = 1 \tag{B.5}
\]

\[
P_1(x) = x \tag{B.5}
\]

\[
P_2(x) = \frac{3}{2} \left[ x^2 - \frac{1}{2} \right] \tag{B.5}
\]

\[
P_3(x) = \frac{5}{2} \left[ x^3 - \frac{3}{5} x \right] \tag{B.5}
\]

\[
P_4(x) = \frac{35}{8} \left[ x^4 - \frac{6}{7} x^2 + \frac{3}{35} \right] \tag{B.5}
\]

They are normalized such that \(P_n(1) = 1\) and satisfy the orthogonality property:

\[
\int_{-\infty}^{\infty} P_n(x) P_m(x) dx = \frac{2}{2n+1} \delta_{nm} \tag{B.6}
\]

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 \(\lambda_n\) are distinct and lie in the interval, \(0 < \lambda_n < 1\). Typically, they have values near unity, \(\lambda_n \approx 1\), for \(n = 0, 1, 2, \ldots\), 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, \(j_k(\omega_0 t) = j_k(c t/t_0)\), 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, \(\psi_n(t) = t_0^{n/2} \phi_n(c, t/t_0)\), where,

\[
\phi_n(c, \eta) = \sqrt{\lambda_n} \sum_k \beta_{nk} \sqrt{k + \frac{1}{2}} j_k (\omega_0 t) = \sqrt{\frac{c}{2\pi}} \sum_k \beta_{nk} \sqrt{k + \frac{1}{2}} 2^{k-n} j_k (c\eta) \tag{B.10}
\]

We will also use the following notation to indicate explicitly the dependence on the parameters \(t_0, \omega_0\) 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 \( 0 \leq k < \infty \). However, \( k \) takes only even values, \( k = 0, 2, 4, \ldots \), when \( n \) is even or zero, and only odd values, \( k = 1, 3, 5, \ldots \), 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 \( \beta_{nk} \) are real-valued and because \( n, k \) have the same parity, i.e., \( n - k \) is even, it follows that the number \( \hat{B}^{k-n} \) will be real, \( \hat{B}^{k-n} = (-1)^{(k-n)/2} \). Therefore, all \( \psi_{n}(t) \) are real-valued.

The \( k \)-summation can be extended to all \( k \geq 0 \) by redefining the expansion coefficients \( \beta_{nk} \) for all \( k \) by appropriately interlacing zeros as follows:

\[
\beta_{nk} = \begin{cases} 
\beta_{n0}, & 0, \beta_{n2}, 0, \beta_{n4}, 0, \cdots \quad \text{(n even)} \\
0, \beta_{n1}, 0, \beta_{n3}, 0, \beta_{n5}, \cdots \quad \text{(n odd)}
\end{cases}
\] (B.12)

The expansion coefficients are chosen to satisfy the orthogonality property,

\[
\sum_{k=0}^{\infty} \beta_{nk} \beta_{mk} = \delta_{nm}, \quad n, m = 0, 1, 2, \ldots
\] (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 \( \beta_{nk} \) 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],

\[
\psi_{n}(t) = \frac{2c}{\pi \lambda_{n}} e^{-i \frac{\pi}{2} t} \sin \left( \omega_{0} t - \frac{1}{2} \pi n \right), \quad \text{for large } |t|
\] (B.14)

This follows from the asymptotic expansion of the spherical Bessel functions,

\[
j_{k}(x) = \frac{\sin \left( x - \frac{1}{2} \pi k \right)}{x}, \quad \text{for large } |x|
\]

### 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]:

\[
\int_{-\infty}^{\infty} e^{-i\omega t} \frac{\pi i^{-k} P_{k}(\omega)}{2\pi} \frac{d\omega}{\omega} = j_{k}(t), \quad \text{for all real } t
\] (B.15)

\[
\int_{-\infty}^{\infty} e^{-i\omega t} j_{k}(t) \frac{d\omega}{\omega} = \pi i^{-k} P_{k}(\omega) \cdot \chi_{1}(\omega) = \begin{cases} 
\pi i^{-k} P_{k}(\omega), & |\omega| < 1 \\
0, & |\omega| > 1
\end{cases}
\]

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 \([-\omega_{0}, \omega_{0}]\),

\[
\chi_{\omega_{0}}(\omega) = u(\omega_{0} - |\omega|) = \begin{cases} 
1, & |\omega| < \omega_{0} \\
0, & |\omega| > \omega_{0}
\end{cases}
\] (B.16)

\(^1\chi_{\omega_{0}}(\omega)\) 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 \( j_{k}(\omega) \) of \( j_{k}(\omega_{0} t) \) is bandlimited over \([-\omega_{0}, \omega_{0}]\),

\[
j_{k}(\omega) = \int_{-\infty}^{\infty} e^{-i\omega t} j_{k}(\omega_{0} t) \frac{d\omega}{\w 2\pi} = \frac{\pi i}{\omega_{0} \sqrt{\omega}} \frac{P_{k}\left(\frac{\omega_{0}}{\omega_{0}}\right)}{\omega_{0}} \cdot \chi_{\omega_{0}}(\omega)
\]

\[
j_{k}(\omega_{0} t) = \int_{-\infty}^{\infty} e^{-i\omega t} j_{k}(\omega) \frac{d\omega}{\omega} = \int_{-\infty}^{\infty} e^{-i\omega t} \frac{\pi i}{\omega_{0} \sqrt{\omega}} \frac{P_{k}\left(\frac{\omega_{0}}{\omega_{0}}\right)}{\omega_{0}} \frac{d\omega}{\w 2\pi}
\] (B.17)

In fact, the functions \( j_{k}(\omega) \), like \( \psi_{n}(t) \), form a complete and orthogonal basis of the subspace \( B_{\omega_{0}} \), see [1325,1336]. Their mutual orthogonality follows from Parseval’s identity and the orthogonality property (B.6) of the Legendre polynomials:

\[
\int_{-\infty}^{\infty} j_{k}(\omega_{0} t) \psi_{n}(\omega_{0} t) \frac{d\omega_{0}}{\omega_{0}} = \int_{-\infty}^{\infty} j_{k}(\omega) \psi_{n}(\omega) \frac{d\omega}{\w 2\pi} = \frac{\pi i}{\omega_{0} \sqrt{\omega}} \frac{P_{k}\left(\frac{\omega_{0}}{\omega_{0}}\right)}{\omega_{0}} \cdot \chi_{\omega_{0}}(\omega)
\] (B.18)

The bandlimited Fourier transform \( \psi_{n}(\omega) \) 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,

\[
\psi_{n}(\omega) = \sqrt{\frac{c}{2\pi \omega_{0}}} \sum_{k} \beta_{nk} \sqrt{k + \frac{1}{2}} 2^{k-n} j_{k}(\omega)
\]

\[
= \sqrt{\frac{c}{2\pi \omega_{0}}} \sum_{k} \beta_{nk} \sqrt{k + \frac{1}{2}} 2^{k-n} \frac{\pi c}{\omega_{0} \sqrt{\omega}} P_{k}\left(\frac{\omega_{0}}{\omega_{0}}\right) \cdot \chi_{\omega_{0}}(\omega)
\]

\[
= \frac{2\pi}{\omega_{0}} \frac{1}{\mu_{n}} \frac{\lambda_{n}}{\omega_{0}} \sum_{k} \beta_{nk} \sqrt{k + \frac{1}{2}} 2^{k-n} \frac{\pi c}{\omega_{0} \sqrt{\omega}} P_{k}\left(\frac{\omega_{0}}{\omega_{0}}\right) \cdot \chi_{\omega_{0}}(\omega) = \frac{2\pi}{\omega_{0}} \frac{1}{\mu_{n}} \frac{\lambda_{n}}{\omega_{0}} \chi_{\omega_{0}}(\omega)
\]

where \( \mu_{n} \) was defined in terms of \( \lambda_{n} \) as follows:

\[
\mu_{n} = \pi^{3} |\mu_{n}|, \quad |\mu_{n}| = \sqrt{\frac{2\pi \lambda_{n}}{c}} \Rightarrow \lambda_{n} = \frac{c}{2\pi} |\mu_{n}|^{2}
\] (B.19)

Thus, we find that \( \psi_{n}(\omega) \) is a scaled version of \( \psi_{n}(t) \) itself,

\[
\psi_{n}(\omega) = \int_{-\infty}^{\infty} e^{-i\omega t} \psi_{n}(t) \frac{d\omega}{\omega} = \frac{2\pi}{\omega_{0}} \frac{1}{\mu_{n}} \psi_{n}\left(\frac{\omega_{0}}{\omega_{0}}\right) \cdot \chi_{\omega_{0}}(\omega)
\] (B.20)

The inverse Fourier transform brings out more clearly the meaning of \( \mu_{n} \):

\[
\psi_{n}(t) = \int_{-\infty}^{\infty} e^{-i\omega t} \psi_{n}(\omega) \frac{d\omega}{\omega} = \int_{-\infty}^{\infty} e^{-i\omega t} \psi_{n}\left(\frac{\omega_{0}}{\omega_{0}}\right) \frac{d\omega}{\omega} = \mu_{n} \psi_{n}(t)
\] (B.21)

for all \( t \). By changing variables to \( \omega = t\omega_{0}/t_{0} \) and \( t = \omega_{0}/\omega_{0} \), we also have,

\[
\int_{-t_{0}}^{t_{0}} e^{-i\omega t} \psi_{n}(\omega) \frac{d\omega}{\omega} = \mu_{n} \psi_{n}\left(\frac{\omega_{0}}{\omega_{0}}\right) \text{ for all } \omega
\] (B.22)
B. PROLATE SPHEROIDAL WAVE FUNCTIONS

If (B.21) is written in terms of the scaled function $\phi_n(\eta)$ of (B.10), then, $\mu_n$ is the eigenvalue and $\phi_n(\eta)$ the eigenfunction of the following integral operator with exponential kernel:

$$\int_{-1}^{1} e^{j\eta\xi} \phi_n(\xi) d\xi = \mu_n \phi_n(\eta)$$

$n = 0, 1, 2, \ldots$, and all $\eta$ (B.23)

Similarly, (B.3) reads as follows with respect to $\phi_n(\eta)$,

$$\int_{-1}^{1} \sin(c(\eta - \xi)) \phi_n(\xi) d\xi = \lambda_n \phi_n(\eta) \quad \text{for all } \eta$$

(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 $\mu_n^*$ and taking the complex conjugate of (B.22), we have,

$$|\mu_n|^2 \psi_n(t) = \int_{-\omega_0}^{\omega_0} \mu_n^* \psi_n(\omega) \frac{e^{j\omega t}}{\omega} \frac{d\omega}{2\pi} = \int_{-\omega_0}^{\omega_0} \left[ \int_{-t_0}^{t_0} e^{j\omega t'} \psi_n(t') \frac{dt'}{t_0} \right] \frac{e^{j\omega t}}{\omega} \frac{d\omega}{2\pi}$$

$$= \frac{2\pi}{t_0} \int_{-\omega_0}^{\omega_0} \int_{-t_0}^{t_0} \frac{e^{j\omega(t-t')}}{2\pi} \psi_n(t') dt' = \frac{2\pi}{c} \int_{-t_0}^{t_0} \sin(\omega_0(t-t')) \psi_n(t') dt'$$

where we used the sinc-function transform,

$$\frac{\sin(\omega_0 t)}{\pi t} = \int_{-\infty}^{\infty} X_{\omega_0}(\omega) \frac{e^{j\omega t}}{2\pi} \frac{d\omega}{\omega} = \int_{-\omega_0}^{\omega_0} e^{j\omega t} \frac{d\omega}{2\pi}$$

(B.25)

Eq. (B.3) follows now by multiplying both sides by $c/2\pi t$ and using the definition (B.19). Any function $f(t)$ in $B_{\omega_0}$ 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,

$$f(t) = \int_{-\omega_0}^{\omega_0} \hat{f}(\omega) \frac{e^{j\omega t}}{2\pi} \frac{d\omega}{\omega} = \int_{-\infty}^{\infty} X_{\omega_0}(\omega) \hat{f}(\omega) \frac{d\omega}{2\pi} = \int_{-\infty}^{\infty} \sin(\omega_0(t-t')) \frac{1}{\pi(t-t')} f(t') dt'$$

that is, for $f(t) \in B_{\omega_0}$, and for all $t$,

$$f(t) = \int_{-\infty}^{\infty} \sin(\omega_0(t-t')) \frac{1}{\pi(t-t')} f(t') dt'$$

(B.26)

Thus, because they lie in $B_{\omega_0}$, all $\psi_n(t)$ satisfy a similar condition,

$$\int_{-\infty}^{\infty} \sin(\omega_0(t-t')) \frac{1}{\pi(t-t')} \psi_n(t') dt' = \psi_n(t), \quad \text{for all } t$$

(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 $L^2_{t_0}$ and $B_{\omega_0}$. With respect to the space $L^2_{t_0}$, we have,

$$\int_{-t_0}^{t_0} \psi_n(t) \psi_m(t) dt = \lambda_n \delta_{nm}$$

(orthogonality) (B.28)

$$\sum_{n=0}^{\infty} \frac{1}{\lambda_n} \psi_n(t) \psi_n(t') = \delta(t-t') \quad \text{(completeness)}$$

(B.29)

for $t, t' \in [-t_0, t_0]$. And, with respect to the subspace $B_{\omega_0}$, we have for the infinite time interval, $-\infty < t < \infty$,

$$\int_{-\infty}^{\infty} \psi_n(t) \psi_m(t) dt = \delta_{nm}$$

(B.30)

$$\sum_{n=0}^{\infty} \psi_n(t) \psi_n(t') = \frac{\sin(\omega_0(t-t'))}{\pi(t-t')}$$

for all $t, t'$ (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 $B_{\omega_0}$, 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 $t$,

$$\sum_{n=0}^{\infty} \psi_n^*(t) = \frac{\omega_0}{\pi}$$

(B.32)

Integrating this over $[-t_0, t_0]$ and using (B.28) for $n = m$, we obtain the sums,

$$\sum_{n=0}^{\infty} \lambda_n = \frac{2\pi}{\omega_0} N_c \Rightarrow \sum_{n=0}^{\infty} |\mu_n|^2 = 4$$

(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,

$$e^{-j\omega t} \cdot X_{\omega_0}(\omega) = \sum_{n=0}^{\infty} \psi_n(t) \psi_n(\omega) = \frac{2\pi}{\omega_0} X_{\omega_0}(\omega) \sum_{n=0}^{\infty} \frac{1}{\mu_n} \psi_n(t) \psi_n \left( \frac{\omega t_0}{\omega_0} \right)$$

or,

$$\frac{2\pi}{\omega_0} \sum_{n=0}^{\infty} \frac{1}{\mu_n} \psi_n(t) \psi_n \left( \frac{\omega t_0}{\omega_0} \right) = e^{-j\omega t} \quad \text{for all } t \text{ and } |\omega| < \omega_0$$

(B.34)

and in particular, setting $t_0 = 0$, we have for all $t$,

$$\frac{2\pi}{\omega_0} \sum_{n=0}^{\infty} \frac{1}{\mu_n} \psi_n(0) \psi_n(t) = 1$$

(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 $\omega'$, with $|\omega'| < \omega_0$,

$$\frac{2\pi}{\omega_0} \sum_{n=0}^{\infty} \frac{1}{\mu_n} \hat{\psi}_n(\omega') \hat{\psi}_n \left( \frac{\omega t_0}{\omega_0} \right) = 2\pi \delta(\omega - \omega')$$

or,
for \( \omega, \omega' \in [-\omega_0, \omega_0] \). This becomes equivalent to Eq. (B.29) after replacing, \( \omega t_0/\omega_0 \rightarrow t \) and \( \omega t_0/\omega_0 \rightarrow t' \), 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,

\[
g(t) = \int_{-t_0}^{t_0} \frac{\sin(\omega_0(t' - t'))}{\pi(t' - t')} f(t') \, dt' \Rightarrow f(t) = \int_{-t_0}^{t_0} K(t,t') \, g(t') \, dt'
\]

(B.37)

for \( |t| \leq t_0 \). It is easily verified that \( K(t,t') \) is given formally by [1365],

\[
K(t,t') = \sum_n \frac{1}{\lambda_n} \psi_n(t) \psi_n(t'), \quad \text{for } t,t' \in [-t_0, t_0]\]  

(B.38)

The operation of \( K(t,t') \) on the sinc-kernel generates the identity kernel, that is,

\[
\int_{-t_0}^{t_0} K(t,t') \frac{\sin(\omega_0(t'' - t'))}{\pi(t'' - t')} \, dt' = \delta(t - t'), \quad \text{for } t,t' \in [-t_0, t_0]
\]

(B.39)

Indeed, using (B.28), (B.29), (B.31), and (B.38), we have,

\[
\int_{-t_0}^{t_0} K(t,t') \frac{\sin(\omega_0(t'' - t'))}{\pi(t'' - t')} \, dt' = \sum_{n,m} \frac{1}{\lambda_n \lambda_m} \psi_n(t) \psi_n(t') \psi_m(t') \psi_m(t'') \int_{-t_0}^{t_0} \psi_n(t'') \, dt''
\]

(B.42)

where \( \hat{\delta}(t,t') \) approximates \( \delta(t - t') \), for \( t,t' \in [-t_0, t_0] \). With \( f(t) \) expanded as,

\[
f(t) = \sum_{n=0}^{\infty} f_n \psi_n(t), \quad |t| \leq t_0
\]

(B.43)

it follows that the restoration approximations correspond to the finite sums,

\[
f_{\text{rest}}(t) = \hat{f}(t) + \hat{u}(t) = \sum_{n=0}^{M} f_n \psi_n(t) + \sum_{n=0}^{M} \frac{\psi_n(t)}{\lambda_n} \psi_n(t)
\]

(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 \([-t_0, t_0]\),

\[
\hat{\delta}(t,t') = \sum_{n=0}^{\infty} \frac{\lambda_n}{\lambda_n^2 + \varepsilon^2} \psi_n(t) \, \psi_n(t')
\]

(B.45)

where \( \varepsilon \) small regularization parameter, \( \varepsilon^2 < 1 \). The summation in (B.45) effectively cuts off as soon as \( \lambda_n \approx \varepsilon \), while it behaves like (B.42) for \( 1 \geq \lambda_n \gg \varepsilon \).
B. PROLATE SPHEROIDAL WAVE FUNCTIONS

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 \([-t_0, t_0]\),

\[
\delta(t) = \sum_{n=0}^{\infty} \frac{1}{\lambda_n} \psi_n(0) \psi_n(t), \quad -t_0 \leq t \leq t_0
\]  

(B.46)

\[
\hat{\delta}(t) = \sum_{n=0}^{\infty} \frac{1}{\lambda_n} \psi_n(0) \psi_n(t), \quad -t_0 \leq t \leq t_0
\]  

(B.47)

\[
\hat{\delta}(t) = \sum_{n=0}^{\infty} \frac{\lambda_n}{\lambda_n^2 + \varepsilon^2} \psi_n(0) \psi_n(t), \quad -t_0 \leq t \leq t_0
\]  

(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, \(-t_0 \leq t \leq t_0\), 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 \( t \),

\[
f(t) = \sum_n c_n \psi_n(t), \quad \text{with} \quad c_n = \int_{-t_0}^{t_0} \psi_n(t) f(t) \, dt
\]  

(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 \([-t_0, t_0]\),

\[
c_n = \frac{1}{\lambda_n} \int_{-t_0}^{t_0} \psi_n(t) f(t) \, dt
\]  

(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 \( B_{\omega_0} \) and a segment of it is known over the interval \([-t_0, t_0]\).

If \( f(t) \) is an arbitrary function in \( L^2 \), but is not necessarily a segment of a bandlimited function in \( B_{\omega_0} \), then the orthogonality and completeness properties (B.28) and (B.29) still allow an expansion in the following form, but valid only for \( |t| \leq t_0 \),

\[
f(t) = \sum_n c_n \psi_n(t), \quad \text{with} \quad c_n = \frac{1}{\lambda_n} \int_{-t_0}^{t_0} \psi_n(t) f(t) \, dt
\]  

(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 \( B_{\omega_0} \) with a bandlimited Fourier transform \( \hat{f}(\omega) \), then, in the \( \psi_n(\omega) \) basis,

\[
f(t) = \sum_n c_n \psi_n(t) \]  

(PSWF basis) (B.52)

\[
\hat{f}(\omega) = \sum_n \frac{2\pi}{\omega_0} \frac{1}{\mu_n} \psi_n(\omega_0 t), \quad \chi_{\omega_0}(\omega)
\]  

(B.54)

\[
c_n = \int_{-\infty}^{\infty} \psi_n(t) f(t) \, dt = \frac{1}{\lambda_n} \int_{-t_0}^{t_0} \psi_n(t) f(t) \, dt = \frac{1}{\mu_n} \int_{-\omega_0}^{\omega_0} \psi_n(\omega_0 t) \hat{f}(\omega) \, d\omega \]  

(B.53)

Similarly, in the \( j_n(\omega_0 t) \) basis, with Legendre polynomial Fourier transform,

\[
f(t) = \sum_n c_n j_n(\omega_0 t) \]  

(spherical Bessel basis) (B.55)

\[
\hat{f}(\omega) = \sum_n \frac{\pi}{\omega_0} \frac{1}{\mu_n} j_n(\omega_0 t) \cdot \chi_{\omega_0}(\omega)
\]  

and in the sinc-function basis, with \( T_s = \pi/\omega_0 \) denoting the sampling time interval,

\[
f(t) = \sum_n c_n \sin(\omega_0 (t - nT_s)) \]  

(sinc) (B.56)

\[
\hat{f}(\omega) = \sum_n c_n e^{-j\omega T_s} \cdot \chi_{\omega_0}(\omega) = T_s \sum_n f(nT_s) e^{-j\omega T_s} \cdot \chi_{\omega_0}(\omega)
\]  

(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:

\[
\int_{-\infty}^{\infty} \frac{\sin(\omega_0 (t - nT_s))}{\pi(t - nT_s)} \cdot \frac{\sin(\omega_0 (t - mT_s))}{\pi(t - mT_s)} \, dt = \frac{1}{T_s} \delta_{nm}
\]  

(B.58)
The result, \( c_n = T_s f(nT_s) \), follows from (B.58) by applying (B.26) at \( t = nT_s \), indeed,

\[
f(nT_s) = \int_{-\infty}^{\infty} \frac{\sin(\omega_0(t - nT_s))}{\pi(t - nT_s)} f(t) dt
\]

\[
= \int_{-\infty}^{\infty} \frac{\sin(\omega_0(t - nT_s))}{\pi(t - nT_s)} \left[ \sum_m c_m \frac{\sin(\omega_0(t - mTs))}{\pi(t - mTs)} \right] dt
\]

\[
= \sum_m c_m \int_{-\infty}^{\infty} \frac{\sin(\omega_0(t - nT_s))}{\pi(t - nT_s)} \frac{\sin(\omega_0(t - mTs))}{\pi(t - mTs)} dt
\]

\[
= \sum_m c_m \frac{1}{T_s} \delta_{nm} = \frac{c_n}{T_s} \Rightarrow c_n = T_s f(nT_s)
\]

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 \( T_s \), say, \( 0 \leq nT_s \leq T \), then the maximum index, and hence the number of significant time samples will be, \( n_{\text{max}}T_s = T \), or, \( n_{\text{max}} = T/T_s = f_sT \), that is, the Shannon number.

### B.6 Energy Concentration Properties

Consider a bandlimited signal \( f(t) \) in \( B_{\omega_0} \) with an expansion of the form (B.52),

\[
f(t) = \sum_n c_n \psi_n(t), \quad \text{with} \quad c_n = \int_{-\infty}^{\infty} \psi_n(t) f(t) dt = \frac{1}{\lambda_n} \int_{-t_0}^{t_0} \psi_n(t) f(t) dt \quad (B.59)
\]

Then, its energy contained in \([-t_0, t_0]\) and in the infinite interval are given by the norms,

\[
\int_{-t_0}^{t_0} f^2(t) dt = \sum_{n=0}^{\infty} \lambda_n c_n^2, \quad \text{and} \quad \int_{-\infty}^{\infty} f^2(t) dt = \sum_{n=0}^{\infty} c_n^2 \quad (B.60)
\]

The proportion of the total energy contained in the interval \([-t_0, t_0]\) is the ratio,

\[
\mathcal{R}(f) = \frac{\int_{-t_0}^{t_0} f^2(t) dt}{\int_{-\infty}^{\infty} f^2(t) dt} = \frac{\sum_{n=0}^{\infty} \lambda_n c_n^2}{\sum_{n=0}^{\infty} c_n^2} \quad (B.61)
\]

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 \( \lambda_n \) 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 \( n \geq 1 \), that is, when \( f(t) = \psi_0(t) \), and the maximized value is \( \mathcal{R}(\psi_0) = \lambda_0 \). This follows from the inequality,

\[
\lambda_0 - \mathcal{R}(f) = \frac{\sum_{n=1}^{\infty} (\lambda_0 - \lambda_n) c_n^2}{\sum_{n=0}^{\infty} c_n^2} \geq 0
\]

with equality realized when \( c_n = 0 \) for all \( n \geq 1 \). Similarly, among all the functions in \( B_{\omega_0} \) 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 \( \psi_0 \), and \( \mathcal{R}(f) \) becomes,

\[
\mathcal{R}(f) = \frac{\sum_{n=1}^{\infty} \lambda_n c_n^2}{\sum_{n=0}^{\infty} c_n^2} \quad \Rightarrow \quad \lambda_1 - \mathcal{R}(f) = \frac{\sum_{n=1}^{\infty} (\lambda_1 - \lambda_n) c_n^2}{\sum_{n=1}^{\infty} c_n^2} \geq 0
\]

More generally, among all the functions in \( B_{\omega_0} \) that are simultaneously orthogonal to \( \psi_0(t), \psi_1(t), \ldots, \psi_{n-1}(t) \), 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 \( B_{\omega_0} \) 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 \( B_{\omega_0} \) 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 \( g(t) \in B_{\omega_0} \) that minimizes the following performance index subject to the energy ratio constraint,

\[
\mathcal{J} = \int_{-t_0}^{t_0} (f(t) - g(t))^2 dt = \min, \quad \text{subject to} \quad \mathcal{R}_0 = \int_{-t_0}^{t_0} g^2(t) dt \quad (B.62)
\]

or equivalently, we may introduce a Lagrange multiplier \( \mu \) for the constraint,

\[
\mathcal{J} = \int_{-t_0}^{t_0} (f(t) - g(t))^2 + \mu \left( \mathcal{R}_0 \int_{-\infty}^{\infty} g^2(t) dt - \int_{-t_0}^{t_0} g^2(t) dt \right) = \min \quad (B.63)
\]
Expand \( f(t) \) and \( g(t) \) in the \( \psi_n(t) \) basis,
\[
\begin{align*}
  f(t) &= \sum_n f_n \psi_n(t), \quad \text{for } |t| \leq t_0 \\
  g(t) &= \sum_n g_n \psi_n(t), \quad \text{for all } t
\end{align*}
\] (B.64)
where \( f_n \) may be assumed to be known since \( f(t) \) is given. Then, (B.63) becomes,
\[
  \mathcal{J} = \sum_n \lambda_n (g_n - f_n)^2 + \mu \left( R_0 \sum_n g_n^2 - \sum_n \lambda_n g_n^2 \right) = \min \tag{B.65}
\]
The minimization condition with respect to \( g_n \) gives,
\[
  \frac{\partial \mathcal{J}}{\partial g_n} = 0 \Rightarrow \lambda_n (g_n - f_n) + \mu (R_0 - \lambda_n) g_n = 0, \quad \text{or,}
\]
\[
  g_n = \frac{\lambda_n f_n}{\lambda_n + \mu (R_0 - \lambda_n)} \tag{B.66}
\]
with the Lagrange multiplier \( \mu \) determined by solving the constraint equation,
\[
  R_0 \sum_n g_n^2 - \sum_n \lambda_n g_n^2 = \sum_n (R_0 - \lambda_n) g_n^2 = 0, \quad \text{or,}
\]
\[
  \sum_n (R_0 - \lambda_n) \left( \frac{\lambda_n f_n}{\lambda_n + \mu (R_0 - \lambda_n)} \right)^2 = 0 \tag{B.67}
\]

### 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., \( 0 \leq k \leq K - 1 \),
\[
\psi_n(t) = \sum_{k=0}^{K-1} \beta_{nk} \sqrt{k + \frac{1}{2}} P_k \left( \frac{1}{t_0} \right) \sqrt{\frac{c}{2\pi \nu_0}} \sum_{k=0}^{K-1} \beta_{nk} \sqrt{k + \frac{1}{2}} 2^{k-n} j_k (\omega_0 t) \tag{B.68}
\]
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, \( \psi_n(t) \), \( n = 0, 1, 2, \ldots, M \).

The computation of \( \beta_{nk} \) 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,
\[
\left( t^2 - t_0^2 \frac{d^2}{dt^2} + 2t \frac{d}{dt} + \omega_0^2 t^2 \right) \psi_n(t) = \chi_n \psi_n(t), \quad n = 0, 1, 2, \ldots \tag{B.69}
\]
The eigenvalues \( \chi_n \) are real and positive and when listed in increasing order, they match the eigenvalues \( \lambda_n \) of Eq. (B.3) listed in decreasing order, that is, we have the correspondence,
\[
0 < \chi_0 < \chi_1 < \chi_2 < \ldots, \\
1 > \lambda_0 > \lambda_1 > \lambda_2 > \ldots
\]

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 tridiagonaleigenvalue problem for calculating the coefficients \( \beta_{nk} \), 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 \( \lambda_n \) and eigenfunctions \( \psi_n(t) \) is very demanding for large values of \( c \) and large values of \( n \) because the \( \lambda_n \) 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
\[
\frac{\partial}{\partial g_n} \left( \lambda_n f_n \right) = 0 \Rightarrow \lambda_n (g_n - f_n) + \mu (R_0 - \lambda_n) g_n = 0, \quad \text{or,}
\]
\[
  g_n = \frac{\lambda_n f_n}{\lambda_n + \mu (R_0 - \lambda_n)} \tag{B.66}
\]
with the Lagrange multiplier \( \mu \) determined by solving the constraint equation,
\[
  R_0 \sum_n g_n^2 - \sum_n \lambda_n g_n^2 = \sum_n (R_0 - \lambda_n) g_n^2 = 0, \quad \text{or,}
\]
\[
  \sum_n (R_0 - \lambda_n) \left( \frac{\lambda_n f_n}{\lambda_n + \mu (R_0 - \lambda_n)} \right)^2 = 0 \tag{B.67}
\]

The difference equation (B.69) is equivalent then to the following eigenvalue equation,
\[
A \beta_n = \chi_n \beta_n, \quad \beta_n = \begin{bmatrix} \beta_{n0} \\ \beta_{n1} \\ \vdots \\ \beta_{nK-1} \end{bmatrix} \tag{B.72}
\]
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,

$$
\begin{bmatrix}
\beta_{n,0} \\
\beta_{n,1} \\
\beta_{n,2} \\
\beta_{n,3} \\
\beta_{n,4} \\
\beta_{n,5} \\
\beta_{n,6} \\
\beta_{n,7}
\end{bmatrix}, \quad \beta_{n,odd} = \begin{bmatrix}
0 \\
\beta_{n,1} \\
0 \\
\beta_{n,3} \\
0 \\
\beta_{n,5} \\
0 \\
\beta_{n,7}
\end{bmatrix}
$$

MATLAB can very efficiently and accurately solve the above tridiagonal eigenvalue problem using the built-in function eig, and obtain all $K$ eigenvectors $\beta_n$ and eigenvalues $\lambda_n$ (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 $\beta_n$, for $n = 0, 1, \ldots, M$, that are needed in the computation of $\psi_n(t)$. One still has the task of determining the corresponding eigenvalues $\lambda_n$ of the eigenproblem (B.3). Since $\lambda_n$ appears in (B.2), we may solve for it by evaluating both sides of (B.68) at a particular value of $t$. It’s simpler to solve for $\mu_n$ and then calculate $\lambda_n$ from (B.19). Multiplying both sides of (B.68) by $\Gamma(\pi t_0/\sqrt{c})$, we obtain the relationship:

$$
\rho_n \sum_{k=0}^{K-1} P_k \left( \frac{t}{t_0} \right) = \mu_n \sum_{k=0}^{K-1} \beta_{nk} \sqrt{k + \frac{1}{2}} \ J_k \left( \omega_0 t \right)
$$

(B.73)

One obvious choice is to set $t = t_0$, and since $P_n(1) = 1$, we may solve for $\mu_n$,

$$
\mu_n = \frac{\sum_{k=0}^{K-1} \beta_{nk} \sqrt{k + \frac{1}{2}} \ J_k (c)}{\sum_{k=0}^{K-1} \beta_{nk} \sqrt{k + \frac{1}{2}}} \quad n = 0, 1, \ldots, M
$$

(B.74)

Another choice is to set $t = 0$, resulting in

$$
\mu_n \sum_{k=0}^{K-1} \beta_{nk} \sqrt{k + \frac{1}{2}} \ P_k(0) = \sum_{k=0}^{K-1} \beta_{nk} \sqrt{k + \frac{1}{2}} \ J_k (0)
$$

(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),

$$
\mu_n \sum_{k=0}^{K-1} \beta_{nk} \sqrt{k + \frac{1}{2}} \ P_k'(t_0) = \sum_{k=0}^{K-1} \beta_{nk} \sqrt{k + \frac{1}{2}} \ J_{k'}(\omega_0 t), \quad \text{or,}
$$

$$
\mu_n \sum_{k=0}^{K-1} \beta_{nk} \sqrt{k + \frac{1}{2}} \ P_k(t_0) = \sum_{k=0}^{K-1} \beta_{nk} \sqrt{k + \frac{1}{2}} \ J_k(\omega_0 t)
$$

(B.76)

Evaluating this at $t = 0$ will work for $n$ odd, but not for $n$ even,

$$
\mu_n \sum_{k=0}^{K-1} \beta_{nk} \sqrt{k + \frac{1}{2}} \ P_k'(0) = \sum_{k=0}^{K-1} \beta_{nk} \sqrt{k + \frac{1}{2}} \ J_k(0)
$$

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 $\mu_n$,

$$
\mu_n = \frac{\sum_{k=0}^{K-1} \beta_{nk} \sqrt{k + \frac{1}{2}} \ 2 \ J_k [J_k(0) + c J_k'(0)]}{\sum_{k=0}^{K-1} \beta_{nk} \sqrt{k + \frac{1}{2}} \ [P_k(0) + P_k'(0)]}
$$

(B.77)

The spherical Bessel functions $J_k(x)$ for $k \geq 0$ have the following limiting behavior,

$$
J_k(x) \xrightarrow{x \to \infty} \frac{x^k}{1 \cdot 3 \cdot 5 \cdot \ldots \cdot (2k + 1)} \Rightarrow J_k(0) = \delta_k = \begin{cases} 1, & k = 0 \\
0, & k \neq 0
\end{cases}
$$

(B.78)

The derivative, $J_k'(x)$, can be computed from the recursion:

$$
J_k'(x) = \frac{k J_{k-1}(x) - (k + 1) J_k(x)}{2k + 1}
$$

(B.79)

and using the result (B.78),

$$
J_k'(0) = \frac{k \delta_{k-1} - (k + 1) \delta_{k+1}}{2k + 1} = \frac{1}{3} \delta_{k-1}
$$

(B.80)

Using (B.78) and (B.80), the numerator in (B.77) simplifies to the $k = 0$ and $k = 1$ terms,

$$
\mu_n = \left[ \frac{\sqrt{2} \beta_{n0} + \sqrt{2} \Gamma \beta_{n1}}{\sum_{k=0}^{K-1} \beta_{nk} \sqrt{k + \frac{1}{2}} [P_k(0) + P_k'(0)]} \right]_{n = 0, 1, \ldots, M}
$$

(B.81)

The quantities, $P_k(0), P_k'(0)$, can be calculated in terms of gamma-functions,

$$
P_k(0) = \frac{\sqrt{\pi}}{\Gamma \left( \frac{k + 1}{2} \right) \Gamma \left( \frac{k}{2} \right)}, \quad P_k'(0) = k P_k(-1)(0) = \frac{-2 \sqrt{\pi}}{\Gamma \left( \frac{k + 1}{2} \right) \Gamma \left( \frac{k}{2} \right)}
$$

(B.82)

These are valid for all $k \geq 0$ 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:
B. PROlate SPHERoidal WAVE FUNCTIONS

\[ [\Psi, \La, d\Psi, \Chi, B] = \text{pswf}(t_0, w_0, M, t); \]  % prolate spheroidal wave functions

\[ [\Psi, \La, d\Psi, \Chi, B] = \text{pswf}(t_0, w_0, M, t, K); \]  % prolate spheroidal wave functions

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
\( t = \text{length-L vector of time instants (sec)}, \ e.g., t = [t_1, t_2, ..., t_L] \)

\( \Psi = (M+1) \times L\) matrix of prolate function values
\( \La = (M+1) \times 1\) vector of eigenvalues
\( d\Psi = (M+1) \times L\) matrix of derivatives of prolate functions
\( \Chi = (M+1) \times 1\) vector of eigenvalues of \( \text{spherical differential operator} \)
\( B = (M+1) \times K\) matrix of expansion coefficients
K = number of expansion terms (default, \( K = 2N + 30, N = M + 1 \))

For a given maximum order \( M \) and time vector \( t = [t_1, t_2, ..., t_L] \), the output matrix \( \Psi \) contains the computed values \( \psi_n(t_i) \), arranged into an \( (M+1) \times L \) matrix,

\[
\Psi = \begin{bmatrix}
\psi_0(t_1) & \psi_0(t_2) & \psi_0(t_3) & \cdots & \psi_0(t_L) \\
\psi_1(t_1) & \psi_1(t_2) & \psi_1(t_3) & \cdots & \psi_1(t_L) \\
\psi_2(t_1) & \psi_2(t_2) & \psi_2(t_3) & \cdots & \psi_2(t_L) \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\psi_M(t_1) & \psi_M(t_2) & \psi_M(t_3) & \cdots & \psi_M(t_L)
\end{bmatrix}
\]

The output \( d\Psi \) contains the derivatives \( \psi_n'(t_i) \) arranged in a similar fashion. The outputs \( \La, \Chi \) are column vectors containing the eigenvalues \( \lambda_n, \chi_n, n = 0, 1, ..., M \). 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,

\[
B = \begin{bmatrix}
\beta_{0,0} & 0 & \beta_{0,2} & 0 & \beta_{0,4} & 0 & \cdots & 0 & \beta_{0,k-1} \\
0 & \beta_{1,1} & 0 & \beta_{1,3} & 0 & \beta_{1,5} & \cdots & 0 & \beta_{1,k-1} \\
\beta_{2,0} & 0 & \beta_{2,2} & 0 & \beta_{2,4} & 0 & \cdots & 0 & \beta_{2,k-1} \\
0 & \beta_{3,1} & 0 & \beta_{3,3} & 0 & \beta_{3,5} & \cdots & 0 & \beta_{3,k-1} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & \beta_{M,1} & 0 & \beta_{M,3} & 0 & \beta_{M,5} & \cdots & 0 & \beta_{M,k-1}
\end{bmatrix}
\]

The eigenvector matrix is actually the transposed, \( B^T = [\beta_0, \beta_1, \ldots, \beta_M] \), which satisfies the eigenvalue equation, \( A \beta^T = \chi \beta^T \), where \( \chi = \text{diag}(\chi_0, \chi_1, \ldots, \chi_M) \), and the orthonormality property,

\[
BB^T = I_{M+1} \Rightarrow \sum_{k=0}^{K-1} \beta_{nk} \beta_{mk} = \delta_{nm}, \quad n, m = 0, 1, ..., M
\]

The required spherical Bessel function computations are implemented by the function, \( \text{spherj.m} \), which also computes the derivatives using the recursion (B.79). It arranges its output similarly to \( \text{pswf} \), and has usage:

\[
[J, dJ] = \text{spherj}(k, x); \]  % spherical Bessel functions and derivatives

The derivatives, \( \psi_n'(t) \), are computed using the expansion,

\[
\psi_n'(t) = \sqrt{\frac{c}{2 \pi t_0}} \sum_{k=0}^{K-1} \beta_{nk} \sqrt{k + \frac{1}{2}} 2^{k-n} \omega_0 f_k(\omega_0 t)
\]  \hspace{1cm} (B.83)

The functions \( \text{pswf} \) and \( \text{spherj} \) 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 \( \text{legpol} \) with usage:

\[
P = \text{legpol}(N, x); \]  % evaluates Legendre polynomials of orders n=0:N

Example B.1: The following MATLAB code illustrates the usage of the \( \text{pswf} \) function and reproduces the data tables and plots from Rhodes [1305] for the case \( c = 6 \).

\[
t_{0}=1; \quad w_{0}=6; \quad c=t_{0} \cdot w_{0}; \quad M=13; \\
t = \text{linspace}(-5,5,501); \\
[\Psi, \La, d\Psi, \Chi] = \text{pswf}(t_0, w_0, M, t); \]  % calculate the first 14 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 \( \chi_n, \mu_n, \lambda_n \) for \( n = 0, 1, \ldots, 13 \), as well the values of the sum, \( \psi_n(0) + \psi_n'(0) \), which represents either \( \psi_n(0) \) if \( n \) is even, or \( \psi_n'(0) \) if \( n \) is odd. They agree with those in [1305].
The table was generated by the MATLAB code,

\[
\begin{align*}
\text{n} & = (0:M)', \\
\text{Mu} & = \sqrt{2\pi/c*La}; \quad \text{\% calculate } |\mu_n| \text{ from la_n} \\
\text{P} & = \text{Psi(:,1)} + \text{dPsi(:,1)}; \quad \text{\% interlace psi_n(0) and its derivative psi_n'(0)} \\
\end{align*}
\]

\[
\text{fprintf(' n chi_n |mu_n| psi_n(0)+psi_n''(0) lambda_n
')}
\]

\[
\text{fprintf('----------------------------------------------------------------------------
')}
\]

\[
\text{fprintf('%3d %1.9e %1.9e % 1.9e %1.9e
', [n,Chi,Mu,P,La'])}
\]

The Shannon number is

\[N_c = \frac{2c}{\pi} = 3.8197\]. We observe in the above table how quickly the eigenvalues \(\lambda_n\) decay towards zero for \(n > N_c\). The following figure plots \(\lambda_n\) 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,

\[
F_M(t) = \sum_{n=0}^{M} \frac{\pi}{\omega_0} |\psi_n(0)|^2
\]

which should tend to unity as \(M\) increases and more terms are included in the sum.
Because $\psi_n(t) = (-1)^n \psi_n(-t)$, it follows that $\psi_n(0) = 0$ for odd $n$ and the summation in (8.85) 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 $-2t_0 \leq t \leq 2t_0$.

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 (8.85) 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),

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 $M = 30$.

```matlab
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-', 'linestyle','-', 'linewidth',2);
    axis([-2,2,-10,20]); xlabel('\itt');
end
```

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),
\[
\Delta_M(\omega) = \sum_{n=0}^{M} \frac{2\pi}{\omega_0} \lambda_n \mu_n \psi_n(0) \psi_n(\frac{\omega t_0}{\omega_0}) \cdot X_{\omega_0}(\omega)
\]
\[
= \frac{2\pi \omega_0}{\omega_0} \sum_{n=0}^{M} \frac{1}{\omega_0^2} \lambda_n^2 \psi_n(0) \psi_n(\frac{\omega t_0}{\omega_0}) \cdot X_{\omega_0}(\omega)
\]
(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 \(M = 20, 30.\)

```matlab
% normalised 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./[1:n'/La.
end

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,

\[
y(t) = a_0 + a_1 \cos(2\pi t) + a_2 \cos(3\pi t) + a_3 \cos(4\pi t) + a_4 \cos(5\pi t) + a_5 \cos(10\pi t)
\]

where,

\[
\begin{align*}
a_0 &= 1 \\
1 &= +13295000 \\
a_2 &= -30802818 \\
a_3 &= +26581909 \\
a_4 &= -10836909 \\
a_5 &= +1762818
\end{align*}
\]

The maximum frequency present in the signal is, \(f_z = 5,\) and one would naively expect that \(y(t)\) would not exhibit oscillations that are faster than \(f_z.\) However, as can be seen in the left graph below, plotted over the narrow time interval, \(|t| \leq 0.04,\) 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 \(f_z,\) is plotted with the green dotted curve,

\[
y_{3}(t) = \cos(10\pi t)
\]
The left graph below shows the same signals plotted over the wider time range, $|t| \leq 0.1$, while the right graph plots the signals of the range, $|t| \leq 2$.

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,

```matlab
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

REFERENCES


**Sampling**


**A/D & D/A Conversion, Quantization, Dithering, and Noise Shaping**


REFERENCES


DSP Hardware


[97] TMS320C2x, TMS320C3x, TMS320C4x, TMS320C5x User Guides, Texas Instruments, Dallas, TX, 1989-93.


Computer Music


REFERENCES


Digital Audio Effects

[132] IEEE ASSP Mag., 2, no. 4, October 1985, Special Issue on Digital Audio.


Dynamic Range Control


REFERENCES


Biomedical Signal Processing


Digital TV


REFERENCES

[209] K. B. Benson and D. G. Fink, HDTV, Advanced Television for the 1990s, Inter-


Signal Averaging

[217] P. Horowitz and W. Hill, The Art of Electronics, 2nd ed., Cambridge University Press, Cam-
bridge, 1989.

Windows


[231] A. D. Bresler, “A New Algorithm for Calculating the Current Distributions of Dolph-
ical Memorandum no.324, Hughes Aircraft Company, Sept. 1953. Available online from:
[237] A. T. Walden, “Accurate Approximation of a 0th Order Discrete Prolate Spheroidal Se-

Spectral Analysis and DFT/FFT Algorithms

[245] C. Van Loan, Computational Frameworks for the Fast Fourier Transform, SIAM, Philadel-
[246] W. L. Briggs and V. E. Henson, The DFT: An Owner’s Manual for the Discrete Fourier Trans-


[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


Gibbs Phenomenon


Second-Order IIR Filter Design


**Analog Filter Design**


**Elliptic Filter Design**


REFERENCES


High-Order Digital Parametric Equalizer Design


Quantization Effects in Digital Filter Structures


Interpolation, Decimation, Oversampling, and Noise Shaping


REFERENCES


STFT and Phase Vocoder


DCT, MDCT, Data Compression


REFERENCES

1453

[399] https://en.wikipedia.org/wiki/JPEG
https://jpeg.org


[410] https://en.wikipedia.org/wiki/Modified_discrete_cosine_transform
https://en.wikipedia.org/wiki/Advanced_Audio_Coding


Wavelets and Applications


REFERENCES


REFERENCES


REFERENCES


REFERENCES


Henderson Filters

REFERENCES


### Predictive and Fractional-Delay Filters


REFERENCES


Maximally Flat Filters


Local Polynomial Modeling and Loess


REFERENCES


Exponential Moving Average Filters


REFERENCES


[873] International Federation of Technical Analysts, *www.ifta.org*


REFERENCES

https://www.oanda.com/forex-trading/learn/technical-analysis-for-traders

Markowitz Portfolios


Spline Smoothing

REFERENCES

1479


Whittaker-Henderson Smoothing

REFERENCES


Hodrick-Prescott and Bandpass Filters


REFERENCES


\[ L_1 \text{ Trend Filtering} \]


Regularization

1486


L1 Regularization and Sparsity


REFERENCES


1147 D. L. Donoho, “For most large underdetermined systems of linear equations the minimal $\ell_1$-norm solution is also the sparsest solution,” Comm. Pure Appl. Math., 59, 797 (2006).


REFERENCES


http://dsp.rice.edu/cs, Compressive Sensing Resources.

MATLAB packages for solving the $L_1$ regularization and related problems:

- ADM http://stanford.edu/~boyd/admm.html
- CVX http://cvxr.com/cvx/
- FISTA http://ie.electro.ucl.ac.be/becka/papers/rstls_package.zip
- Homotopy http://www.ece.ucr.edu/~safiti/homotopy/
- L1-MAGIC https://statweb.stanford.edu/~candes/l1magic/
- LARS https://sourceforge.net/projects/sparsemodels/files/LARS/
- NESTA https://statweb.stanford.edu/~candes/nesta/
- REGTOOLS https://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 http://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**


REFERENCES


Model-Based Seasonal Adjustment


REFERENCES


Random Number Generators

[1264] S. Kirkpatrick and E. Stoll, “A Very Fast Shift-Register Sequence Random Number Genera-

[1272] G. Marsaglia and T. A. Bray, “On-line Random Number Generators and their Use in Com-
[1276] Suggested by Dr. N. Moayeri, private communication, 1993.

1/f Noise

REFERENCES


Prolate Spheroidal Wave Functions


REFERENCES


Superoscillations


Superresolution, Restoration, Degrees of Freedom


REFERENCES


REFERENCES


Neural Networks

[1418] M. Nielsen, Neural Networks and Deep Learning, 2019, free online book, neuralnetworksanddeeplearning.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
MNIST handwritten digit database, yann.lecun.com/exdb/mnist
Neural network software, Wikipedia, en.wikipedia.org/wiki/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 quantization, 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, 1233  
activation functions, 1378  
adaptive signal processing, 95, 99, 813, 832  
ADC, see A/D converter  
aliased frequency, 11  
aliasing, 5, 8, 10, 34, 432  
equivalence, 11–25  
in rotational motion, 26  
alternating-step response, 249  
amplitude modulation, 796  
analogue  
Bessel filters, 667  
Butterworth filters, 58, 538, 665, 675  
Chebyshev filters, 559  
dither, 84  
frequency response, 3  
impulse response, 3  
linear filtering, 3  
linear system, 2  
postfilter, 45  
reconstructor, 10, 42, 623  
antenna tracking system, 1037  
anti-image postfilter, see postfilter  
anti-aliasing prefilter, see prefilter  
atenuation, 21, 39, 47  
3-dB width, 39  
dB per octave, 21, 40  
of window sidelobes, 364  
audio effects processor, 266, 801, 829  
auto-correlation, 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  
biasing, 416  

bilinear transformation, 504, 996  
for comb filters, 334, 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, 278  
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  
BPON, basis pursuit denoising, 1307  
Butterworth moving average filters, 1228  
C routines:  
I0, modified Bessel function, 494  
adc, A/D conversion, 80  
1-pass, allpass reverse, 816  
btree, bit reversal, 444  
blockconv, overlap-add, 143  
can2, canonical form, 287  
can3, canonical form, 311  
can, canonical form, 287  
casc2can, cascade to canonical, 308  
casc, cascade form, 297  
casc2, circular canonical, 314  
casc2, circular cascade, 321  
casc, circular cascade, 319  
cdelay2, circular delay line, 183  
cdel, circular delay line, 318  
cfird2, circular FIR filtering, 178  
cfird2, circular FIR filtering, 182  
cfird, circular FIR filtering, 176  
conv, convolution, 137  
cos2, circular SOS, 321  
cos, circular SOS, 318  
dac, D/A conversion, 75  
delay, delay operation, 158  
delta, delta function, 140  
dftmerge, DFT merge, 445  
dir2d, 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 reverber, 819  
modwrap, modulo-N reduction, 426  
plain, plain reverber, 816  
rani1, 1/f noise generator, 1400  
rart, random generator, 1395  
rart, linearly interpolated, 1396  
ran, uniform random generator, 1392  
shuffle, shuffling in FFT, 443  
sine, sinusoidal waveable, 793  
sos, second-order section, 294  
square, square waveable, 794  
swap, swapping in FFT, 444  
tap2, circular tap outputs, 183  
tap1, interpolated delay, 807  
tap, circular tap outputs, 181  
tap1, interpolated delay, 807  
tap, circular tap outputs, 181  
trapez, trapezoidal waveable, 794  
u, unit-step function, 81  
waven1, waveable generator, 792  
waven, waveable generator, 792  
waven, waveable generator, 792  
waven, waveable generator, 791  
wrap2, circular index wrapping, 182  
wrap, circular pointer wrapping, 176  
canonical form, 320, 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  
faracter matrices, 296  
internal states, 296  
pipelining of, 313  
processing time, 312  
sample processing algorithm, 297  
cascade to canonical, 301, 307  
causality, 111, 402  
and finite 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  
Chakrin money flow, 1255  
Chakrin oscillator, 1255  
Chakrin volatility, 1255  
Chande momentum oscillator, CMO, 1252  
channels, 1242  
Chobyshev window, 378, 390  
chorusing, 809  
chrominance signals, 751  
circular  
addressing, 121, 169, 72, 314  
buffer, 172  
canonical form, 315  
cascade form, 319  
convolution, 449  
delay, 127, 180, 183  
direct form, 176  
pointer, 173  
pointer index, 174  
state vector, 174, 175, 181, 183  
wrapping, 176, 177, 180, 182  
classical seasonal analysis, 1344  
classical spectral analysis, 399  
coefficient quantization, 353  
COLA, see constant-overlap-add property  
comb filters, 259, 263, 324, 746  
3-dB width, 746  

commodity property, 738  
design, 534, 746  
for digital reverb, 805  
for noise reduction, 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 on computer music  
amplitude modulation, 796  
frequency modulation, 783  
musical instrument models, 783, 824  
physical modeling, 783, 824  
waveable generators, 783  
constant-overlap-add property, 886
INDEX

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
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
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, 460
of IIR filters, 504
of Savitzy-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, 107
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, 1272, 1279
hold interpolators, 647
Holt’s exponential smoothing, 1203
Hörmann orthogonal polynomials, 1118
Hamming window, 51, 630
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
polynomial smoothing, 1203
Hull moving average, 1231
Hörmann’s rule, 1239
Kaiser window, 366, 387
KBD windows, 918
kernel machines, 1283
Krawtchouk polynomials, 1126
L0 trend indicator, 1239
L1 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
if, 4-fold, 626, 651
computational rate, 631
cutoff frequency, 629
DAC equalization, 661
filter design, 626
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 DTT, 30, 203, 241, 242, 364, 429, 1417
and Fourier series, 30, 204
inverse FFT, 446
inverse filters, 264
stable and causal, 267
in direct form, 240
in polyphase form, 622, 630, 632, 655
Kaiser window, 366, 387
for filter design, 481
for interpolator designs, 638
for spectral analysis, 366, 495
parameters, 484, 496
resolution, 495
Karpplus-Strong string algorithm, 824
KBD windows, 918
Keltner bands, 1242
kernel machines, 1283
Krawtchouk polynomials, 1126
L0 trend indicator, 1239
luminance signal, 751
MAC, 169, 171, 310, 313, 355
magnitude response, 207, 240
marginal stability, 202, 205, 250, 402
Market indicators:

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accdist</td>
<td>accumulation/distribution line, 1252</td>
</tr>
<tr>
<td>atr</td>
<td>average true range, 1246</td>
</tr>
<tr>
<td>bbands</td>
<td>Bollinger bands, 1246</td>
</tr>
<tr>
<td>bma</td>
<td>Butterworth moving average, 1230</td>
</tr>
<tr>
<td>cci</td>
<td>commodity channel index, 1252</td>
</tr>
<tr>
<td>chosc</td>
<td>Chauvenet oscillator, 1252</td>
</tr>
<tr>
<td>chvol</td>
<td>Chauvenet volatility, 1252</td>
</tr>
<tr>
<td>cmflow</td>
<td>Chauvenet money flow, 1252</td>
</tr>
<tr>
<td>cmo</td>
<td>Chande momentum oscillator, 1252</td>
</tr>
<tr>
<td>delay</td>
<td>d-fold delay, 1235</td>
</tr>
<tr>
<td>dema</td>
<td>double EMA, 1217</td>
</tr>
<tr>
<td>dirmov</td>
<td>directional movement system, 1252</td>
</tr>
<tr>
<td>dmi</td>
<td>dynamic momentum index, 1252</td>
</tr>
<tr>
<td>donch</td>
<td>Donchian channels, 1246</td>
</tr>
<tr>
<td>dpo</td>
<td>detrended price oscillator, 1252</td>
</tr>
<tr>
<td>ehma</td>
<td>exponential Hull moving average, 1235</td>
</tr>
<tr>
<td>fbands</td>
<td>fixed-width bands, 1246</td>
</tr>
<tr>
<td>forosc</td>
<td>forecast oscillator, 1252</td>
</tr>
<tr>
<td>gema</td>
<td>generalized EMA, 1235</td>
</tr>
<tr>
<td>hma</td>
<td>Hull moving average, 1235</td>
</tr>
<tr>
<td>ilrs</td>
<td>integrated linear regression slope, 1212</td>
</tr>
<tr>
<td>kbands</td>
<td>Keltner bands, 1246</td>
</tr>
<tr>
<td>ltrend</td>
<td>Lo trend indicator, 1240</td>
</tr>
<tr>
<td>mreg</td>
<td>linear regression indicators, level, slope, R-square, standard-errors, 1220</td>
</tr>
<tr>
<td>mom</td>
<td>momentum, price rate of change, 1252</td>
</tr>
<tr>
<td>ohc</td>
<td>open-high-close-low bar chart, 1221</td>
</tr>
<tr>
<td>pbands</td>
<td>projection bands &amp; oscillator, 1246</td>
</tr>
<tr>
<td>pma2</td>
<td>quadratic PMA, 1214</td>
</tr>
<tr>
<td>pmaimp2</td>
<td>PMA2 impulse response, 1214</td>
</tr>
<tr>
<td>pmaimp</td>
<td>PMA impulse response, 1214</td>
</tr>
<tr>
<td>pma</td>
<td>predictive moving average, 1214</td>
</tr>
<tr>
<td>pnv4</td>
<td>positive/negative volume indices, 1252</td>
</tr>
<tr>
<td>prosc</td>
<td>price oscillator and MACD, 1252</td>
</tr>
<tr>
<td>psar</td>
<td>parabolic SAR, 1250</td>
</tr>
<tr>
<td>rcrit</td>
<td>R-square critical values, 1219</td>
</tr>
<tr>
<td>sema</td>
<td>simple EMA, 1217</td>
</tr>
<tr>
<td>shma</td>
<td>simple Hull moving average, 1235</td>
</tr>
<tr>
<td>sma</td>
<td>simple moving average, 1212</td>
</tr>
<tr>
<td>srbands</td>
<td>Starc bands, 1246</td>
</tr>
<tr>
<td>stdev</td>
<td>length-N standard deviation, 1242</td>
</tr>
<tr>
<td>stoch</td>
<td>stochastic, percent-K, percent-D, 1252</td>
</tr>
<tr>
<td>tcrit</td>
<td>t-distribution critical values, 1219</td>
</tr>
<tr>
<td>tema</td>
<td>triple EMA, 1217</td>
</tr>
<tr>
<td>tma</td>
<td>triangular moving average, 1212</td>
</tr>
<tr>
<td>trix</td>
<td>TRIX oscillator, 1252</td>
</tr>
<tr>
<td>vema</td>
<td>variable-length EMA, 1252</td>
</tr>
<tr>
<td>vhlfit</td>
<td>Vertical horizontal filter, 1252</td>
</tr>
<tr>
<td>wema</td>
<td>Wilder’s EMA, 1228</td>
</tr>
<tr>
<td>wema</td>
<td>weighted moving average, 1212</td>
</tr>
<tr>
<td>yylin</td>
<td>adjust left/right y-axes limits, 1221</td>
</tr>
<tr>
<td>zema</td>
<td>zero-lag EMA, 1235</td>
</tr>
<tr>
<td>zigzag</td>
<td>zigzag indicator, 1238</td>
</tr>
<tr>
<td>market</td>
<td>portfolio, 1208</td>
</tr>
<tr>
<td>Markowitz portfolio theory, 1259</td>
<td></td>
</tr>
<tr>
<td>MDCT</td>
<td>905, 913</td>
</tr>
<tr>
<td>data compression system, 906</td>
<td></td>
</tr>
<tr>
<td>inverse MDCT, 914</td>
<td></td>
</tr>
<tr>
<td>KBD window, 918</td>
<td></td>
</tr>
<tr>
<td>Princen-Bradley windows, 916</td>
<td></td>
</tr>
<tr>
<td>window construction, 918</td>
<td></td>
</tr>
<tr>
<td>MFLOPS</td>
<td>179, 180</td>
</tr>
<tr>
<td>minimum roughness filters, 110</td>
<td></td>
</tr>
<tr>
<td>minimum variance filters, 1082</td>
<td></td>
</tr>
<tr>
<td>MIFS</td>
<td>203, 207</td>
</tr>
<tr>
<td>notch filters, 402</td>
<td></td>
</tr>
<tr>
<td>noise reduction ratio, 402</td>
<td></td>
</tr>
<tr>
<td>noise shaping, 66, 684</td>
<td></td>
</tr>
<tr>
<td>dithered, 688</td>
<td></td>
</tr>
<tr>
<td>error spectrum shaping, 355, 688, 771</td>
<td></td>
</tr>
<tr>
<td>quantizer, 69, 688</td>
<td></td>
</tr>
<tr>
<td>quantizer comparisons, 70</td>
<td></td>
</tr>
<tr>
<td>requantizer, 688, 694</td>
<td></td>
</tr>
<tr>
<td>non-rectangular windows, 366</td>
<td></td>
</tr>
<tr>
<td>nonlinear signal processing, 99</td>
<td></td>
</tr>
<tr>
<td>notch and comb filters, 1320</td>
<td></td>
</tr>
<tr>
<td>notch and comb filters with fractional delay, 1326</td>
<td></td>
</tr>
<tr>
<td>notch filters, 258, 259, 514, 811</td>
<td></td>
</tr>
<tr>
<td>3-dB width, 811</td>
<td></td>
</tr>
<tr>
<td>for ECG processing, 738, 740, 742</td>
<td></td>
</tr>
<tr>
<td>phasing audio effects, 810</td>
<td></td>
</tr>
<tr>
<td>Q-factor, 812</td>
<td></td>
</tr>
<tr>
<td>notch polynomial, 260</td>
<td></td>
</tr>
<tr>
<td>NRR, see noise reduction ratio</td>
<td></td>
</tr>
<tr>
<td>NRR, noise reduction ratio, 402</td>
<td></td>
</tr>
<tr>
<td>Nyquist frequency, 6</td>
<td></td>
</tr>
</tbody>
</table>

INDEX

L1 trend filtering, 1289
Landen transformations, 591
Laplace transform, 2
lma, 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
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
linear 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 transform, 499
LPSM filters, 1058
LTI system, 95
anticausal, 111
causal, 111
double-sided, 111
equal descriptions of, 111, 224
FIR, 95
frequency response, 2
IR, 95
impulse response, 2, 102
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, 684
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, 1253
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
<table>
<thead>
<tr>
<th>INDEX</th>
<th>1513</th>
<th>1514</th>
</tr>
</thead>
<tbody>
<tr>
<td>high-order analog designs, 879</td>
<td></td>
<td>risk premium, 1208</td>
</tr>
<tr>
<td>high-order Butterworth, 857</td>
<td></td>
<td>RLS algorithm, 1163</td>
</tr>
<tr>
<td>high-order Chebyshev type-1, 858</td>
<td></td>
<td>ROC, see region of convergence</td>
</tr>
<tr>
<td>high-order Chebyshev type-2, 859</td>
<td></td>
<td>roots of unity, 222</td>
</tr>
<tr>
<td>high-order elliptic, 861</td>
<td></td>
<td>412</td>
</tr>
<tr>
<td>MATLAB functions, 880</td>
<td>order determination, 863</td>
<td>roundoff error, 302, 353, 355</td>
</tr>
<tr>
<td>order determination, 863</td>
<td>realizations, 867</td>
<td>sample processing algorithm, 111, 120, 154</td>
</tr>
<tr>
<td>parametric resonators, 254, 519</td>
<td>practical, 18, 38</td>
<td>for allpass reverber, 815</td>
</tr>
<tr>
<td>parametric spectral analysis, 368</td>
<td>specifications of, 38, 680</td>
<td>for canonical form, 231, 285</td>
</tr>
<tr>
<td>Parks-McClellan method, 499</td>
<td>practical, 18, 38</td>
<td>for cascade form, 297</td>
</tr>
<tr>
<td>Parseval’s equation, 204</td>
<td>specifications of, 38, 680</td>
<td>for circular canonical form, 315</td>
</tr>
<tr>
<td>partial fraction expansion, 208, 243</td>
<td>remove/restore method, 209</td>
<td>for circular cascade form, 319</td>
</tr>
<tr>
<td>of z-transforms, 208</td>
<td>remove/restore method, 209</td>
<td>for circular FIR filter, 176, 184</td>
</tr>
<tr>
<td>parametric resonators, 254, 519</td>
<td>remove/restore method, 209</td>
<td>for decimation, 679</td>
</tr>
<tr>
<td>Parks-McClellan method, 499</td>
<td>remove/restore method, 209</td>
<td>for direct form, 227, 278</td>
</tr>
<tr>
<td>partial fraction expansion, 208, 243</td>
<td>remove/restore method, 209</td>
<td>for FIR Filtering, 159, 161, 163, 167, 168</td>
</tr>
<tr>
<td>of z-transforms, 208</td>
<td>remove/restore method, 209</td>
<td>for interpolation, 635</td>
</tr>
<tr>
<td>Parks-McClellan method, 499</td>
<td>remove/restore method, 209</td>
<td>for lowpass reverber, 819</td>
</tr>
<tr>
<td>partial fraction expansion, 208, 243</td>
<td>remove/restore method, 209</td>
<td>for multi-delay, 826</td>
</tr>
<tr>
<td>of z-transforms, 208</td>
<td>remove/restore method, 209</td>
<td>for multiband delay, 828</td>
</tr>
<tr>
<td>Parks-McClellan method, 499</td>
<td>remove/restore method, 209</td>
<td>for parallel form, 230</td>
</tr>
<tr>
<td>partial fraction expansion, 208, 243</td>
<td>remove/restore method, 209</td>
<td>for ringing delay, 826</td>
</tr>
<tr>
<td>of z-transforms, 208</td>
<td>remove/restore method, 209</td>
<td>for sample rate conversion, 685</td>
</tr>
<tr>
<td>Parks-McClellan method, 499</td>
<td>remove/restore method, 209</td>
<td>for Schroeder’s reverber processor, 817</td>
</tr>
<tr>
<td>partial fraction expansion, 208, 243</td>
<td>remove/restore method, 209</td>
<td>for transposed form, 232</td>
</tr>
<tr>
<td>of z-transforms, 208</td>
<td>remove/restore method, 209</td>
<td>sampled signal, 29</td>
</tr>
<tr>
<td>Parks-McClellan method, 499</td>
<td>remove/restore method, 209</td>
<td>flat-top sampled, 29</td>
</tr>
<tr>
<td>partial fraction expansion, 208, 243</td>
<td>remove/restore method, 209</td>
<td>ideally sampled, 29</td>
</tr>
<tr>
<td>of z-transforms, 208</td>
<td>remove/restore method, 209</td>
<td>sampling</td>
</tr>
<tr>
<td>Parks-McClellan method, 499</td>
<td>remove/restore method, 209</td>
<td>aliasing due to, 5</td>
</tr>
<tr>
<td>partial fraction expansion, 208, 243</td>
<td>remove/restore method, 209</td>
<td>and Poisson summation formula, 34</td>
</tr>
<tr>
<td>of z-transforms, 208</td>
<td>remove/restore method, 209</td>
<td>as modulation, 33</td>
</tr>
<tr>
<td>Parks-McClellan method, 499</td>
<td>remove/restore method, 209</td>
<td>frequencies introduced by, 5</td>
</tr>
<tr>
<td>partial fraction expansion, 208, 243</td>
<td>remove/restore method, 209</td>
<td>function, 33, 699</td>
</tr>
<tr>
<td>of z-transforms, 208</td>
<td>remove/restore method, 209</td>
<td>guard band, 34</td>
</tr>
<tr>
<td>Parks-McClellan method, 499</td>
<td>remove/restore method, 209</td>
<td>hardware limits in, 8</td>
</tr>
<tr>
<td>partial fraction expansion, 208, 243</td>
<td>remove/restore method, 209</td>
<td>ideal and practical, 29</td>
</tr>
<tr>
<td>of z-transforms, 208</td>
<td>remove/restore method, 209</td>
<td>Nyquist frequency, 6</td>
</tr>
<tr>
<td>Parks-McClellan method, 499</td>
<td>remove/restore method, 209</td>
<td>Nyquist rate, 6</td>
</tr>
<tr>
<td>partial fraction expansion, 208, 243</td>
<td>remove/restore method, 209</td>
<td>of periodic signals and DFT, 432</td>
</tr>
<tr>
<td>of z-transforms, 208</td>
<td>remove/restore method, 209</td>
<td>of sinuoids, 9, 35, 50</td>
</tr>
<tr>
<td>Parks-McClellan method, 499</td>
<td>remove/restore method, 209</td>
<td>process, 4</td>
</tr>
<tr>
<td>partial fraction expansion, 208, 243</td>
<td>remove/restore method, 209</td>
<td>rate, 5, 7</td>
</tr>
<tr>
<td>of z-transforms, 208</td>
<td>remove/restore method, 209</td>
<td>spectrum replication due to, 5</td>
</tr>
<tr>
<td>Parks-McClellan method, 499</td>
<td>remove/restore method, 209</td>
<td>theorem, 5, 9, 44</td>
</tr>
<tr>
<td>partial fraction expansion, 208, 243</td>
<td>remove/restore method, 209</td>
<td>time interval, 4</td>
</tr>
<tr>
<td>of z-transforms, 208</td>
<td>remove/restore method, 209</td>
<td>sampling rate conversion, 622, 681</td>
</tr>
<tr>
<td>Parks-McClellan method, 499</td>
<td>remove/restore method, 209</td>
<td>filter design for, 681</td>
</tr>
<tr>
<td>partial fraction expansion, 208, 243</td>
<td>remove/restore method, 209</td>
<td>sample processing algorithm, 685</td>
</tr>
<tr>
<td>of z-transforms, 208</td>
<td>remove/restore method, 209</td>
<td>sampling theorem, 1414</td>
</tr>
<tr>
<td>Parks-McClellan method, 499</td>
<td>remove/restore method, 209</td>
<td>Saramäki windows, 483</td>
</tr>
<tr>
<td>partial fraction expansion, 208, 243</td>
<td>remove/restore method, 209</td>
<td>Savitzky-Golay smoothing filters, 1058</td>
</tr>
<tr>
<td>of z-transforms, 208</td>
<td>remove/restore method, 209</td>
<td>scaling, 355</td>
</tr>
<tr>
<td>Parks-McClellan method, 499</td>
<td>remove/restore method, 209</td>
<td>Schroeder’s reverberator, 817</td>
</tr>
<tr>
<td>partial fraction expansion, 208, 243</td>
<td>remove/restore method, 209</td>
<td>Schur-Cohn Stability Test, 346</td>
</tr>
<tr>
<td>of z-transforms, 208</td>
<td>remove/restore method, 209</td>
<td>seasonal decomposition filters, 1342</td>
</tr>
</tbody>
</table>
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