```
figure; plot(k,HT);  figure; plot(k,HS);          % upper graphs
```

9.10 Problems

9.1 First prove Eq. (9.1.2) for all n. Then, using the DFT/IDFT pair in Eq. (9.1.1), show that a more general form of (9.1.2) is,

$$\sum_{m=0}^{D-1} s_{n-m} e^{j\omega_k m} = e^{j\omega_k n} S_k\,, \quad k = 0, 1, \ldots, D-1, \quad -\infty < n < \infty$$

9.2 Consider the analog signal $s(t) = \cos(2\pi f_1 t)$ and its sampled version $s_n = \cos(2\pi f_1 nT)$, where T is the sampling interval related to the sampling rate by $f_s = 1/T$. It is required that s_n be periodic in n with period of D samples, that is, $\cos\big(2\pi f_1 (n+D)T\big) = \cos(2\pi f_1 nT)$, for all n. How does this requirement constrain f_s and f_1?

9.3 Show that the IIR comb and notch filters defined in Eq. (9.1.9) are complementary and power complementary in the sense that they satisfy Eqs. (9.1.7).

Working with the magnitude response $|H_{\text{comb}}(\omega)|^2$ show that the 3-dB width of the comb peaks is given by Eq. (9.1.11).

9.4 Show that the solution of the system (9.9.7) can be written in the more symmetric, but computationally less efficient, form:

$$\mathbf{t} = (Q + P + QP)^{-1} Q\mathbf{y}$$

$$\mathbf{s} = (P + Q + PQ)^{-1} P\mathbf{y}$$

10

Wavelets

Over the past two decades, wavelets have become useful signal processing tools for signal representation, compression, and denoising [665–833]. There exist several books on the subject [665–686], and several tutorial reviews [687–708]. The theory of wavelets and multiresolution analysis is by now very mature [709–761] 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 [762–833].

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.

10.1 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 $L^2(\mathbb{R})$ of square integrable functions on the real line $\mathbb{R}$:

$$\cdots \subset V_{-2} \subset V_{-1} \subset V_0 \subset V_1 \subset V_2 \subset \cdots \subset L^2(\mathbb{R}) \tag{10.1.1}$$

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 of the V_j subspaces is the entire $L^2(\mathbb{R})$ space, and their intersection, the zero function:

$$\lim_{j\to\infty} V_j = \bigcup_{j=-\infty}^{\infty} V_j = L^2(\mathbb{R})\,, \qquad \lim_{j\to-\infty} V_j = \bigcap_{j=-\infty}^{\infty} V_j = \{0\} \tag{10.1.2}$$

The spaces V_j have a special structure, being defined as the *linear spans* of the scaled and translated replicas of a single function $\phi(t)$, called the *scaling function*, or the *father wavelet*, which can be of compact support. The scaled/translated replicas of $\phi(t)$ are defined for any integers j, n by:

$$\phi_{jn}(t) = 2^{j/2}\phi(2^j t - n) \tag{10.1.3}$$

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:†

$$(\phi_{jn}, \phi_{jm}) = \int_{-\infty}^{\infty} \phi_{jn}(t)\,\phi_{jm}(t)\,dt = \delta_{nm} \tag{10.1.4}$$

The factor $2^{j/2}$ in Eq. (10.1.3) serves to preserve the unit norm of ϕ_{jn} for each j. Conditions (10.1.1)–(10.1.4) are strong constraints and it is remarkable that such functions $\phi(t)$ exist other than the simple Haar function defined to be unity over $0 \le t \le 1$ and zero otherwise. Fig. 10.1.1 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*.

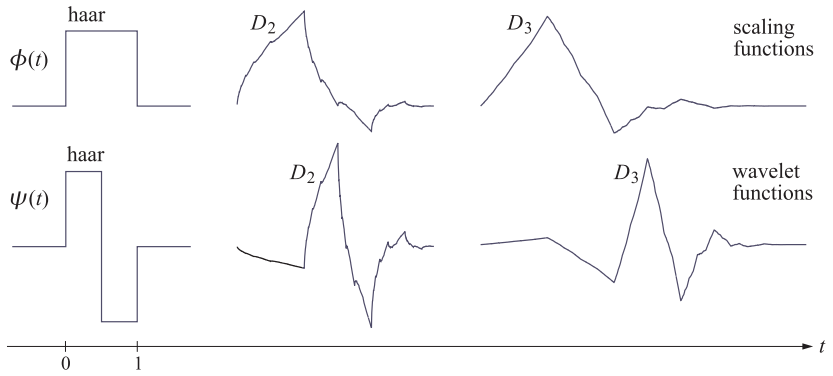


Fig. 10.1.1 Haar, Daubechies D_2 and D_3 scaling and wavelet functions $\phi(t), \psi(t)$.

Fig. 10.1.2 shows the scaled versions of the scaling and wavelet functions (for the D_2 case). Each successive copy $\phi(t), \phi(2t), \phi(2^2t), \phi(2^3t)$, etc., is compressed by a factor of two relative to the previous one, so that for higher and higher values of j, the basis function $\phi(2^j t)$ is capable of capturing smaller and smaller signal details.

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 ϕ_{jn} basis:

$$f_j(t) = \sum_n c_{jn}\phi_{jn}(t) = \sum_n c_{jn}\, 2^{j/2}\phi(2^j t - n) \tag{10.1.5}$$

†In this chapter, all time signals are assumed to be real-valued.

with coefficients following from the orthonormality of $\phi_{jn}(t)$:

$$c_{jn} = (f_j, \phi_{jn}) = (f, \phi_{jn}) = \int_{-\infty}^{\infty} f(t)\,\phi_{jn}(t)\,dt = \int_{-\infty}^{\infty} f(t)\,2^{j/2}\phi(2^j t - n)\,dt \tag{10.1.6}$$

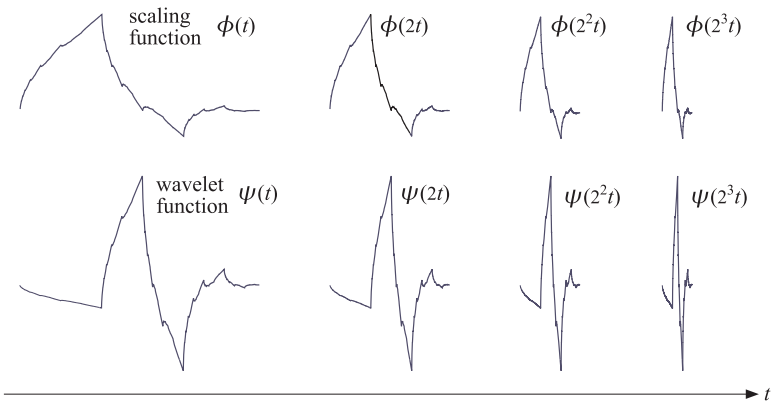


Fig. 10.1.2 Daubechies D_2 functions $\phi(t), \psi(t)$ and their compressed versions.

The projection $f_j(t)$ can be thought of as an approximation of $f(t)$ at scale j with time resolution of 2^{-j}. Because $V_i \subset V_j$ for $i \le j$, the signal $f_j(t)$ incorporates information about $f(t)$ from all coarser resolutions (cf. Eq. (10.2.8)).

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 ψ, that is, $\psi_{in}(t) = 2^{i/2}\psi(2^i t - n)$ for $i \ge j$, which are orthogonal to $\phi_{jn}(t)$, and are also mutually orthonormal,

$$(\phi_{jn}, \psi_{im}) = 0\,, \quad i \ge j\,, \qquad (\psi_{in}, \psi_{i'n'}) = \delta_{ii'}\delta_{nn'} \tag{10.1.7}$$

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:

$$f(t) = f_j(t) + w_j(t)\,, \quad f_j(t) \in V_j\,, \quad w_j(t) \in V_j^\perp\,, \quad f_j(t) \perp w_j(t) \tag{10.1.8}$$

The component $w_j(t)$ is referred to as the "detail," and incorporates the details of $f(t)$ at all the higher resolution levels $i \ge j$, or finer time scales $2^{-i} \le 2^{-j}$. It admits the ψ-basis expansion:

$$w_j(t) = \sum_{i \ge j}\sum_n d_{in}\psi_{in}(t) = \sum_{i \ge j}\sum_n d_{in}2^{i/2}\psi(2^i t - n) \tag{10.1.9}$$

with detail coefficients $d_{in} = (w_j, \psi_{in}) = (f, \psi_{in})$. In summary, one form of the multiresolution decomposition is,

$$\boxed{f(t) = f_j(t) + w_j(t) = \sum_n c_{jn}\phi_{jn}(t) + \sum_{i=j}^{\infty}\sum_n d_{in}\psi_{in}(t)} \tag{10.1.10}$$

Another form is obtained in the limit $j \to -\infty$. Since $V_{-\infty} = \{0\}$, we have $f_{-\infty}(t) = 0$, and we obtain the representation of $f(t)$ purely in terms of the wavelet basis $\psi_{in}(t)$:

$$\boxed{f(t) = \sum_{i=-\infty}^{\infty} \sum_n d_{in}\psi_{in}(t)}, \quad d_{in} = \int_{-\infty}^{\infty} f(t)\psi_{in}(t)dt \tag{10.1.11}$$

Yet another, and most practical, version of the multiresolution decomposition is obtained by noting that $V_\infty = L^2(\mathbb{R})$. We may assume then that our working signal $f(t)$ belongs to some V_J for a sufficiently large value of J, representing the highest desired resolution, or finest scale. Since $f(t) \in V_J$, it follows from the decomposition $f(t) = f_J(t) + w_J(t)$ that $w_J(t) = 0$, which implies that $d_{in} = 0$ for $i \geq J$, and therefore,

$$f(t) = f_J(t) = \sum_n c_{Jn}\phi_{Jn}(t) \tag{10.1.12}$$

Combining this with Eq. (10.1.10) applied at some lower resolution $j < J$, we obtain the two alternative forms (cf. Eq. (10.2.10)):

$$\boxed{f(t) = \sum_n c_{Jn}\phi_{Jn}(t) = \sum_n c_{jn}\phi_{jn}(t) + \sum_{i=j}^{J-1} \sum_n d_{in}\psi_{in}(t) = f_j(t) + w_j(t)} \tag{10.1.13}$$

The mapping of the expansion coefficients from level J to levels j through $J-1$,

$$c_{Jn} \to \{c_{jn};\ d_{in}, j \leq i \leq J-1\} \tag{10.1.14}$$

is essentially the discrete wavelet transform (DWT). For sufficiently large J, the coefficients c_{Jn} can be taken to be the time samples of $f(t)$, sampled at the rate $f_s = 2^J$, or sampling time interval $T_J = 2^{-J}$. To see this, we note that the function $2^J\phi(2^Jt - n)$ tends to a Dirac delta function for large J (see [665] for a proof), so that,

$$2^J\phi(2^Jt - n) \approx 2^J\delta(2^Jt - n) = \delta(t - n2^{-J}) \tag{10.1.15}$$

Therefore, $2^{J/2}\phi(2^Jt - n) \approx 2^{-J/2}\delta(t - n2^{-J})$, and Eq. (10.1.6) gives,

$$c_{Jn} \approx \int_{-\infty}^{\infty} f(t)2^{-J/2}\delta(t - n2^{-J})\Phi_0 dt = 2^{-J/2}f(n2^{-J}) \tag{10.1.16}$$

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 10.1.1: An example of the decomposition (10.1.13) is shown in Fig. 10.1.3 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 $0 \leq t \leq 1$,

$$f(t) = \begin{cases} \sin(4\pi t), & 0 \leq t < 0.25 \\ \sin(2\pi t), & 0.25 \leq t < 0.75 \\ \sin(4\pi t), & 0.75 \leq t < 1 \end{cases} \tag{10.1.17}$$

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.

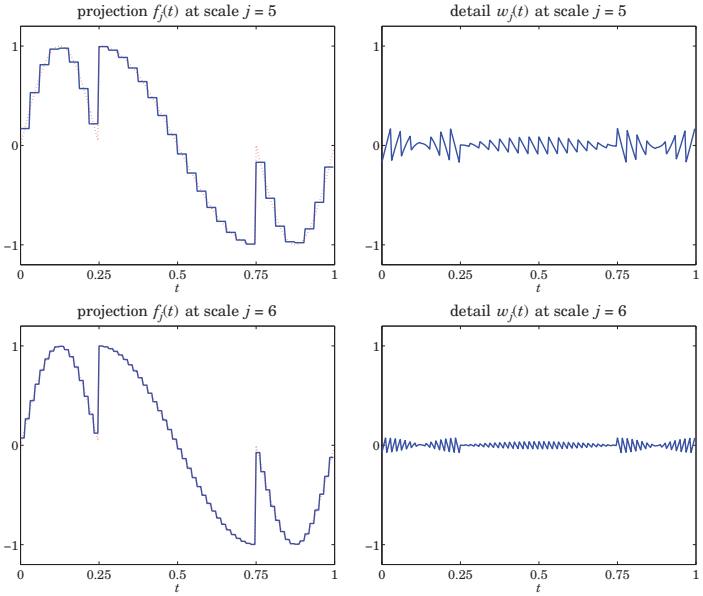


Fig. 10.1.3 Haar-basis projections $f_j(t), w_j(t)$ from scale $J = 8$ to scales $j = 5, 6$.

Fig. 10.1.4 shows the case of using the Daubechies D_3 wavelet basis for the same signal (10.1.17). The following MATLAB code generates the top graphs in the two figures:

```
J = 8; N = 2^J;
t1 = (0:N/4-1)'/N; t2 = (N/4:3*N/4-1)'/N; t3 = (3*N/4:N-1)'/N;
t = [t1; t2; t3];
y = [sin(4*pi*t1); sin(2*pi*t2); sin(4*pi*t3)];     % define signal

h = daub(1);                                        % use h=daub(3) for Fig. 10.1.4
j = 5; Y = dwtdec(y,h,j);                           % DWT decomposition to level j
fj = Y(:,1); wj = sum(Y(:,2:end),2);                % approximation f_j(t) and detail w_j(t)

figure; plot(t,fj, t,y,':'); figure; plot(t,wj);    % left, right graphs
```

The function `dwtdec` is explained in Sec. 10.5, but we mention here that its output Y is an $N \times (J-j+1)$ matrix whose first column holds the projection $f_j(t)$, and the sum of its other columns are the detail $w_j(t)$. □

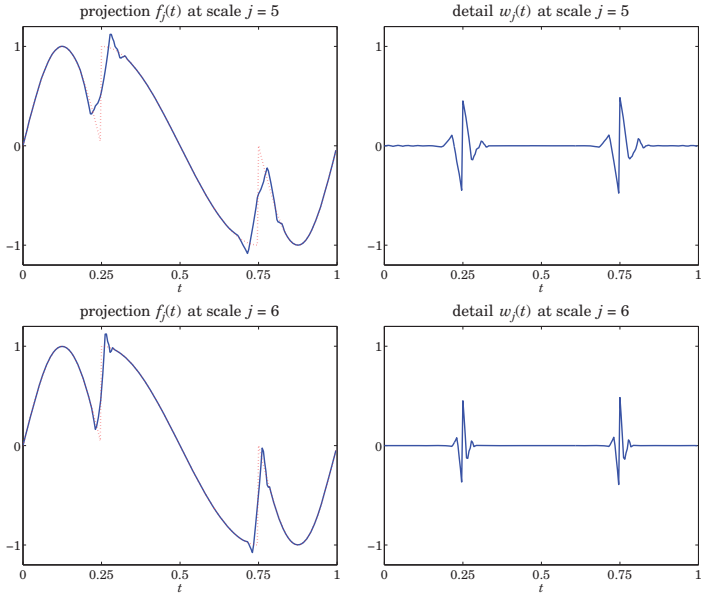


Fig. 10.1.4 Daubechies-D_3 projections $f_j(t), w_j(t)$ from scale $J = 8$ to scales $j = 5, 6$.

10.2 Dilation Equations

The subspaces V_j have even more interesting structure than described so far. Since $V_0 \subset V_1$, it follows that the scaling function $\phi(t) \in V_0$ can be expanded in the basis $\phi_{1n}(t) = 2^{1/2}\phi(2t-n)$ that spans V_1, that is, there must exist coefficients h_n such that

$$\boxed{\phi(t) = \sum_n h_n\, 2^{1/2}\phi(2t-n)} \qquad \text{(dilation equation)} \tag{10.2.1}$$

which is known as the dilation or refinement equation. The coefficients h_n are given by:

$$h_n = (\phi, \phi_{1n}) = 2^{1/2}\int_{-\infty}^{\infty} \phi(t)\,\phi(2t-n)\,dt \tag{10.2.2}$$

Moreover, the wavelet function $\psi(t)$ and its translates $\psi_{0n} = \psi(t-n)$ form an orthonormal basis for the orthogonal complement of V_0 relative to V_1, that is, the space $W_0 = V_1 \setminus V_0$, so that we have the direct-sum decomposition:

$$V_0 \oplus W_0 = V_1 \tag{10.2.3}$$

The space W_0 is referred to as the "detail" subspace. Because $\psi(t) \in W_0 \subset V_1$, it also can be expanded in the $\phi_{1n}(t)$ basis, as in Eq. (10.2.1),

$$\boxed{\psi(t) = \sum_n g_n\, 2^{1/2}\phi(2t-n)} \tag{10.2.4}$$

In a similar fashion, we have the decomposition $V_j \oplus W_j = V_{j+1}$, for all j, with W_j being spanned by the scaled/translated ψ-basis, $\psi_{jn}(t) = 2^{j/2}\psi(2^j t - n)$. The dilation equations can also be written with respect to the ϕ_{jn}, ψ_{jn} bases. For example,

$$\phi_{jk}(t) = 2^{j/2}\phi(2^j t - k) = \sum_m h_m 2^{(j+1)/2}\phi(2^{j+1}t - 2k - m) = \sum_m h_m \phi_{j+1,m+2k}(t)$$

and similarly for $\psi_{jk}(t)$. Thus, we obtain the alternative forms,

$$\begin{aligned} \phi_{jk}(t) &= \sum_m h_m \phi_{j+1,m+2k}(t) = \sum_n h_{n-2k}\phi_{j+1,n}(t) \\ \psi_{jk}(t) &= \sum_m g_m \phi_{j+1,m+2k}(t) = \sum_n g_{n-2k}\phi_{j+1,n}(t) \end{aligned} \tag{10.2.5}$$

Using the orthogonality property $(\phi_{j+1,n}, \phi_{j+1,m}) = \delta_{nm}$, we have the inner products,

$$\begin{aligned} h_{n-2k} &= (\phi_{jk}, \phi_{j+1,n}) \\ g_{n-2k} &= (\psi_{jk}, \phi_{j+1,n}) \end{aligned} \tag{10.2.6}$$

Also, because $\phi_{j+1,n}(t)$ is a basis for $V_{j+1} = V_j \oplus W_j$, it may be expanded into its two orthogonal parts belonging to the subspaces V_j and W_j, which are in turn spanned by ϕ_{jk} and ψ_{jk}, that is,

$$\phi_{j+1,n} = \sum_k (\phi_{j+1,n}, \phi_{jk})\,\phi_{jk} + \sum_k (\phi_{j+1,n}, \psi_{jk})\,\psi_{jk}$$

Using (10.2.6), we may rewrite this as,

$$\phi_{j+1,n}(t) = \sum_k h_{n-2k}\phi_{jk}(t) + \sum_k g_{n-2k}\psi_{jk}(t) \tag{10.2.7}$$

Eqs. (10.2.5)–(10.2.7) are the essential tools for deriving Mallat's pyramidal multiresolution algorithm for the discrete wavelet transform.

The various decompositions discussed in Sec. 10.1 can be understood in the geometric language of subspaces. For example, starting at level j and repeating the direct-sum decomposition, and using $V_{-\infty} = \{0\}$, we obtain the representation of the subspace V_j,

$$V_j = V_{j-1} \oplus W_{j-1} = V_{j-2} \oplus W_{j-2} \oplus W_{j-1} = \cdots = \bigoplus_{i=-\infty}^{j-1} W_i \tag{10.2.8}$$

which states that V_j incorporates the details of all coarser resolutions. Similarly, increasing j and using $V_\infty = L^2(\mathbb{R})$, we obtain the subspace interpretation of Eq. (10.1.10),

$$\begin{aligned} V_{j+1} &= V_j \oplus W_j \\ V_{j+2} &= V_{j+1} \oplus W_{j+1} = V_j \oplus W_j \oplus W_{j+1} \\ V_{j+3} &= V_{j+2} \oplus W_{j+2} = V_j \oplus W_j \oplus W_{j+1} \oplus W_{j+2} \\ &\cdots \\ L^2(\mathbb{R}) &= V_j \oplus (W_j \oplus W_{j+1} \oplus W_{j+2} \oplus \cdots) = V_j \oplus V_j^\perp \end{aligned} \tag{10.2.9}$$

which explains the remark that the term $w_j(t)$ in (10.1.10) incorporates all the higher-level details. Finally, going from level $j < J$ to level $J-1$, we obtain the geometric interpretation of Eq. (10.1.13),

$$V_J = V_j \oplus (W_j \oplus W_{j+1} \oplus \cdots \oplus W_{J-1}), \quad j < J \tag{10.2.10}$$

The coefficients h_n define a lowpass filter $H(z) = \sum_n h_n z^{-n}$ called the *scaling filter*. Similarly, g_n define a highpass filter $G(z)$, the *wavelet filter*. The coefficients h_n, g_n must satisfy certain orthogonality relations, discussed below, that follow from the dilation equations (10.2.5).

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 [665] has constructed several families of such FIR filters: the minimum-phase family or daublets, the least-asymmetric family or symmlets, and coiflets. The MATLAB function `daub` incorporates these three families:

```
h = daub(K,type);          % Daubechies scaling filters - daublets, symmlets, coiflets

h = daub(K,1) = Daublets K = 1,2,3,4,5,6,7,8,9,10, denoted as D_K  (D_1 = Haar)
h = daub(K,2) = Symmlets K = 4,5,6,7,8,9,10, denoted as S_K
h = daub(K,3) = Coiflets K = 1,2,3,4,5
h = daub(K)   = equivalent to daub(K,1)

Daublets (minimum phase) have length = 2K and K vanishing moments for ψ(t).
Symmlets (least asymmetric) have length = 2K and K vanishing moments for ψ(t).
Coiflets have length = 6K and 2K vanishing moments for ψ(t), and 2K-1 for φ(t).
         for coiflets, h(n) is indexed over -2K <= n <= 4K-1

all filters have norm(h) = 1 and sum(h) = √2
```

For example, the scaling filters for the Haar, and daublet D_2 and D_3 cases, whose $\phi(t), \psi(t)$ functions were shown in Fig. 10.1.1, are obtained from the MATLAB calls:

```
h = daub(1) ⇒ h = [0.7071    0.7071]
h = daub(2) ⇒ h = [0.4830    0.8365    0.2241   -0.1294]
h = daub(3) ⇒ h = [0.3327    0.8069    0.4599   -0.1350   -0.0854    0.0352]
```

The filters h_n can be taken to be causal, i.e., h_n, $0 \le n \le M$, where M is the filter order, which is odd for the above three families, with $M = 2K-1$ for daublets and symmlets, and $M = 6K-1$ for coiflets (these are defined to be slightly anticausal, over $-2K \le n \le 4K-1$). The parameter K is related to certain flatness constraints or moment constraints for h_n at the Nyquist frequency $\omega = \pi$.

The filters g_n are defined to be the *conjugate* or *quadrature* mirror filters to h_n, that is, $g_n = (-1)^n h_n^R$, where $h_n^R = h_{M-n}$, $n = 0, 1, \ldots, M$ is the reversed version of h_n.

In the z-domain, we have $H^R(z) = z^{-M} H(z^{-1})$, while multiplication by $(-1)^n$ is equivalent to the substitution $z \to -z$, therefore, $G(z) = H^R(-z) = (-z)^{-M} H(-z^{-1})$. In the frequency domain, this reads:

$$G(\omega) = e^{-jM(\omega+\pi)} H^*(\omega+\pi) \quad \Leftrightarrow \quad g_n = (-1)^n h_{M-n}, \quad 0 \le n \le M \tag{10.2.11}$$

with the frequency-responses defined by:

$$G(\omega) = \sum_{n=0}^{M} g_n e^{-j\omega n}, \quad H(\omega) = \sum_{n=0}^{M} h_n e^{-j\omega n} \tag{10.2.12}$$

The function `cmf` implements this definition:

```
g = cmf(h);          % conjugate mirror filter
```

For example, if $\mathbf{h} = [h_0, h_1, h_2, h_3]$, then, $\mathbf{g} = [h_3, -h_2, h_1, -h_0]$, e.g., we have for the daublet D_2:

```
h = daub(2) = [ 0.4830    0.8365    0.2241   -0.1294]
g = cmf(h)  = [-0.1294   -0.2241    0.8365   -0.4830]
```

Fig. 10.2.1 shows the magnitude responses of the Haar, Daubechies D_2, and Symmlet S_6 scaling and wavelet filters.

For all scaling filters, the DC gain of $H(\omega)$, and the Nyquist gain of $G(\omega)$, are equal to $\sqrt{2}$ because of the conditions (which are a consequence of the dilation equation):

$$\begin{aligned} H(0) &= \sum_{n=0}^{M} h_n = \sqrt{2} \\ G(\pi) &= \sum_{n=0}^{M} (-1)^n g_n = \sum_{n=0}^{M} h_{M-n} = H(0) = \sqrt{2} \end{aligned} \tag{10.2.13}$$

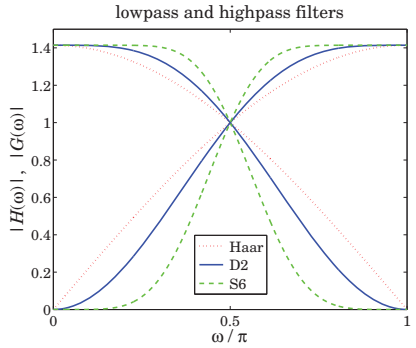


Fig. 10.2.1 Haar, Daubechies D_2, and Symmlet S_6 scaling and wavelet filters.

The dilation equations can be expressed in the frequency domain as follows:

$$\begin{aligned} \Phi(\omega) &= 2^{-1/2} H(2^{-1}\omega)\Phi(2^{-1}\omega) \\ \Psi(\omega) &= 2^{-1/2} G(2^{-1}\omega)\Phi(2^{-1}\omega) \end{aligned} \tag{10.2.14}$$

where $\Phi(\omega), \Psi(\omega)$ are the Fourier transforms:

$$\Phi(\omega) = \int_{-\infty}^{\infty} \phi(t) e^{-j\omega t}\, dt, \quad \Psi(\omega) = \int_{-\infty}^{\infty} \psi(t) e^{-j\omega t}\, dt \tag{10.2.15}$$

In fact, setting $\omega = 0$ in the first of (10.2.14) and assuming that $\Phi(0) \neq 0$, we immediately obtain the gain conditions (10.2.13). The iteration of Eqs. (10.2.14) leads to the infinite product expressions:

$$\begin{aligned}\Phi(\omega) &= \Phi(0)\prod_{j=1}^{\infty}\Big[2^{-1/2}H(2^{-j}\omega)\Big]\\ \Psi(\omega) &= \Phi(0)\Big[2^{-1/2}G(2^{-1}\omega)\Big]\prod_{j=2}^{\infty}\Big[2^{-1/2}H(2^{-j}\omega)\Big]\end{aligned} \tag{10.2.16}$$

We show later that $\Phi(0)$ can be chosen to be unity, $\Phi(0) = 1$. As an example, Fig. 10.2.2 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. 10.2.2, an accuracy of 0.001 percent was achieved with the values of $J = 7$ and $J = 5$ for the left and right graphs, respectively. The following MATLAB code illustrates the generation of the left graph:

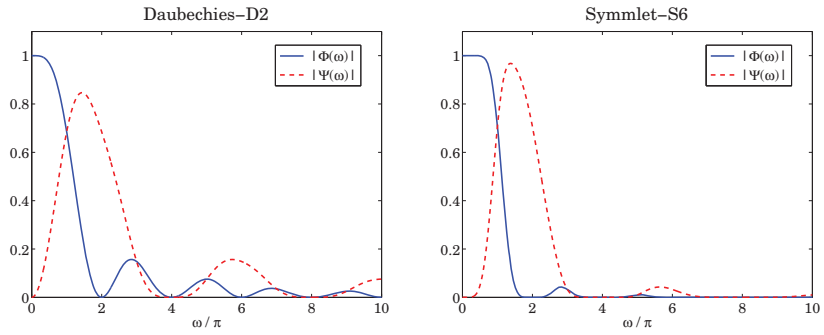


Fig. 10.2.2 Fourier transforms $\Phi(\omega), \Psi(\omega)$ of scaling and wavelet functions $\phi(t), \psi(t)$.

```
epsilon = 1e-5; Jmax = 30;                    % percent error = 100ϵ
f = linspace(0,10,513); w = pi*f;             % frequency range in units of π
h = daub(2)/sqrt(2); g = cmf(h);              % normalize h_n such that H(0) = 1.
                                              % use h=daub(6,2)/sqrt(2) for right graph
Phi0 = abs(freqz(h,1,w/2));                   % initialize recursions at |H(ω/2)|, |G(ω/2)|
Psi0 = abs(freqz(g,1,w/2));

for J = 2:Jmax,
   Phi = Phi0 .* abs(freqz(h,1,w/2^J));       % update by the factor |H(2^{-J}ω)|
   Psi = Psi0 .* abs(freqz(h,1,w/2^J));
   if norm(Phi-Phi0) < norm(Phi0) * epsilon, J, break; end      % stopping J
   Phi0 = Phi;
   Psi0 = Psi;
end

figure; plot(f,Phi, f,Psi,'--');
```


We observe from the graphs that $\Phi(\omega)$ has zeros at $\omega = 2\pi m$ for non-zero integers m. This is justified in the next section. Similarly, $\Psi(\omega)$ vanishes at $\omega = 2\pi m$ for even m, including zero.

Given the filters h_n, g_n, the dilation equations (10.2.1) and (10.2.4) can be solved iteratively for the functions $\phi(t), \psi(t)$ by the so-called *cascade algorithm*, which amounts to the iterations,

$$\begin{aligned}\phi^{(r+1)}(t) &= \sum_n h_n\, 2^{1/2}\phi^{(r)}(2t-n)\\ \psi^{(r+1)}(t) &= \sum_n g_n\, 2^{1/2}\phi^{(r)}(2t-n)\end{aligned} \tag{10.2.17}$$

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 `daub`. The algorithm can be cast as a convolutional operation with the so-called à trous[†] filters generated from the scaling filter. First, we note that if h_n, g_n have order M, and are defined over $0 \leq n \leq M$, 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 rth 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 $\phi^{(r)}(t)$:

$$f^{(r)}(n) = 2^{-r/2}\phi^{(r)}(2^{-r}n) \tag{10.2.18}$$

where $2^{-r/2}$ is a convenient normalization factor. It follows then from Eq. (10.2.17) that

$$2^{-(r+1)/2}\phi^{(r+1)}(2^{-(r+1)}n) = \sum_m h_m\, 2^{-r/2}\phi^{(r)}(2^{-r}n-m)\,, \quad \text{or,}$$

$$f^{(r+1)}(n) = \sum_m h_m f^{(r)}(n-2^r m) = \sum_k h^{[r]}(k) f^{(r)}(n-k) \tag{10.2.19}$$

where we defined the à trous filter corresponding to the interpolation factor 2^r by

$$h^{[r]}(k) = \sum_m h_m \delta(k-2^r m) \tag{10.2.20}$$

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. (10.2.19) can be interpreted as the convolution of the rth iterate with the rth à trous filter. The recursion can be iterated for $r = 0, 1, 2, \ldots, J$, for sufficiently large J (typically, $J = 10$ works well.) The MATLAB function `casc` implements this algorithm:

```
[phi,psi,t] = casc(h,J,phi0);          % cascade algorithm
```

where t is the vector of final evaluation points $t = 2^{-J}n$, $0 \leq n \leq M2^J$. For example, the Daubechies D_2 functions $\phi(t), \psi(t)$ shown in Fig. 10.1.1 can be computed and plotted by the following code:

```
h = daub(2); J = 10; phi0 = 1;

[phi,psi,t] = casc(h,J,phi0);

figure;  plot(t,phi, t,psi,'--');
```

[†] "a trous" means "with holes" in French. The filters are similar to the comb "seasonal" filters of Chap. 9.

The scaling function output `phi` is normalized to unit L_2-norm, and the wavelet output `psi` is commensurately normalized. The following MATLAB code fragment from `casc` illustrates the construction method:

```
phi0=1;
for r=0:J-1,
   phi = conv(phi, upr(h,r));
end
```

where the function `upr` constructs the à trous filter $h^{[r]}(k)$ by upsampling h_n by a factor of 2^r. This function can also be implemented using the MATLAB's built-in function `upsample`. For example, if $\mathbf{h} = [h_0, h_1, h_2, h_3]$, then for $r = 2$, the à trous filter will be,

$$\mathbf{h}^{[r]} = \text{upr}(\mathbf{h}, r) = \text{upsample}(\mathbf{h}, 2^r) = [h_0, 0, 0, 0, h_1, 0, 0, 0, h_2, 0, 0, 0, h_3, 0, 0, 0]$$

10.3 Wavelet Filter Properties

The scaling and wavelet filters h_n, g_n must satisfy certain necessary constraints which are a consequence of the orthogonality of the scaling and wavelet basis functions. Using the property $(\phi_{j+1,n}, \phi_{j+1,m}) = \delta_{nm}$, it follows from Eq. (10.2.5) that,

$$(\phi_{j0}, \phi_{jk}) = \Big(\sum_n h_n \phi_{j+1,n}, \sum_m h_{m-2k} \phi_{j+1,m}\Big) = \sum_{n,m} h_n h_{m-2k} (\phi_{j+1,n}, \phi_{j+1,m}) = \sum_n h_n h_{n-2k}$$

Similarly, we find,

$$(\psi_{j0}, \psi_{jk}) = \Big(\sum_n g_n \phi_{j+1,n}, \sum_m g_{m-2k} \phi_{j+1,m}\Big) = \sum_n g_n g_{n-2k}$$

$$(\phi_{j0}, \psi_{jk}) = \Big(\sum_n h_n \phi_{j+1,n}, \sum_m g_{m-2k} \phi_{j+1,m}\Big) = \sum_n h_n g_{n-2k}$$

But $(\phi_{j0}, \phi_{jk}) = (\psi_{j0}, \psi_{jk}) = \delta_k$ and $(\phi_{j0}, \psi_{jk}) = 0$, therefore h_n, g_n must satisfy the orthogonality properties:

$$\boxed{\begin{aligned} \sum_n h_n h_{n-2k} &= \delta_k \\ \sum_n g_n g_{n-2k} &= \delta_k \\ \sum_n h_n g_{n-2k} &= 0 \end{aligned}} \tag{10.3.1}$$

These may also be expressed in the frequency domain. We will make use of the following cross-correlation identities that are valid for any two filters h_n, g_n and their frequency responses $H(\omega), G(\omega)$:

$$\begin{aligned} \sum_n h_n g_{n-k} &\quad\Leftrightarrow\quad H(\omega) G^*(\omega) \\ (-1)^k \sum_n h_n g_{n-k} &\quad\Leftrightarrow\quad H(\omega+\pi) G^*(\omega+\pi) \\ \Big[1 + (-1)^k\Big] \sum_n h_n g_{n-k} &\quad\Leftrightarrow\quad H(\omega) G^*(\omega) + H(\omega+\pi) G^*(\omega+\pi) \end{aligned} \tag{10.3.2}$$

where the second follows from the "modulation" property of Fourier transforms, and the third, by adding the first two. We note next that Eqs. (10.3.1) can be written in the following equivalent manner obtained by replacing $2k$ by any k, even or odd:

$$\begin{aligned} \big[1 + (-1)^k\big] \sum_n h_n h_{n-k} &= 2\delta_k \\ \big[1 + (-1)^k\big] \sum_n g_n g_{n-k} &= 2\delta_k \\ \big[1 + (-1)^k\big] \sum_n h_n g_{n-k} &= 0 \end{aligned} \tag{10.3.3}$$

Taking the Fourier transforms of both sides of (10.3.3) and using the transform properties (10.3.2), we obtain the frequency-domain equivalent conditions to Eqs. (10.3.1):

$$\boxed{\begin{aligned} &|H(\omega)|^2 + |H(\omega+\pi)|^2 = 2 \\ &|G(\omega)|^2 + |G(\omega+\pi)|^2 = 2 \\ &H(\omega) G^*(\omega) + H(\omega+\pi) G^*(\omega+\pi) = 0 \end{aligned}} \tag{10.3.4}$$

The conjugate mirror filter choice (10.2.11) for $G(\omega)$ automatically satisfies the third of Eqs. (10.3.4). Indeed, using the 2π-periodicity of $H(\omega)$, we have,

$$\begin{aligned} G^*(\omega) &= e^{jM(\omega+\pi)} H(\omega+\pi) \\ G^*(\omega+\pi) &= e^{jM(\omega+2\pi)} H(\omega+2\pi) = e^{jM\omega} H(\omega) \end{aligned}$$

so that,

$$H(\omega) G^*(\omega) + H(\omega+\pi) G^*(\omega+\pi) = e^{jM\omega} H(\omega) H(\omega+\pi) \Big[e^{jM\pi} + 1\Big] = 0$$

where $e^{jM\pi} = -1$, because M was assumed to be odd. With this choice of $G(\omega)$, the first of (10.3.4) can be written in the following form, which will be used later on to derive the undecimated wavelet transform:

$$\boxed{\frac{1}{2}\big[H^*(\omega) H(\omega) + G^*(\omega) G(\omega)\big] = 1} \tag{10.3.5}$$

Setting $\omega = 0$ in the first of Eqs. (10.3.4), and using the DC gain constraint $H(0) = \sqrt{2}$, we find immediately that $H(\pi) = 0$, that is, the scaling filter must have a zero at the Nyquist frequency $\omega = \pi$. Since

$$H(\pi) = \sum_n (-1)^n h_n = \sum_{n=\text{even}} h_n - \sum_{n=\text{odd}} h_n \,,$$

it follows in conjunction with the DC condition that:

$$\sum_{n=\text{even}} h_n = \sum_{n=\text{odd}} h_n = \frac{1}{\sqrt{2}} \tag{10.3.6}$$

The correlation constraints and the DC gain condition,

$$\sum_n h_n h_{n-2k} = \delta_k\,, \quad \sum_n h_n = \sqrt{2}, \tag{10.3.7}$$

provide only $N/2+1$ equations, where N is the (even) length of the filter h_n. Therefore, one has $N/2-1$ additional degrees of freedom to specify the scaling filters uniquely. For example, Daubechies' minimum-phase D_K filters have length $N=2K$ and K zeros at Nyquist. These zeros translate into K equivalent moment constraints, or derivative flatness constraints at Nyquist:

$$\sum_{n=0}^{N-1}(-1)^n n^i h_n = 0 \quad \Leftrightarrow \quad \left.\frac{d^i H(\omega)}{d\omega^i}\right|_{\omega=\pi} = 0\,, \quad i=0,1,\ldots,K-1 \tag{10.3.8}$$

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 (10.3.7), determines the $N=2K$ coefficients h_n uniquely. The construction method may be found in [665]. As an example, we work out the three cases D_1, D_2, D_3 explicitly. The Haar D_1 case corresponds to $K=1$ or $N=2K=2$, so that $\mathbf{h}=[h_0,h_1]$ must satisfy:

$$h_0^2+h_1^2=1\,, \quad h_0+h_1=\sqrt{2} \tag{10.3.9}$$

with (lowpass) solution $h_0=h_1=1/\sqrt{2}$. For the Daubechies D_2 case, we have $K=2$ and $N=2K=4$, so that $\mathbf{h}=[h_0,h_1,h_2,h_3]$ must satisfy,

$$\begin{aligned} &h_0^2+h_1^2+h_2^2+h_3^2=1\\ &h_0+h_2=\frac{1}{\sqrt{2}}\,, \quad h_1+h_3=\frac{1}{\sqrt{2}}\\ &-h_1+2h_2-3h_3=0 \end{aligned} \tag{10.3.10}$$

where the third is the Nyquist moment constraint with $i=1$, and the middle two are equivalent to the DC gain and the $h_0h_2+h_1h_3=0$ correlation constraint; indeed, this follows from the identity:

$$\left(h_0+h_2-\frac{1}{\sqrt{2}}\right)^2+\left(h_1+h_3-\frac{1}{\sqrt{2}}\right)^2 =$$
$$= 1+(h_0^2+h_1^2+h_2^2+h_3^2)-\sqrt{2}(h_0+h_1+h_2+h_3)+2(h_0h_2+h_1h_3)$$

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:

$$4h_0^2-\sqrt{2}\,h_0-\frac{1}{4}=0 \quad \Rightarrow \quad h_0=\frac{1\pm\sqrt{3}}{4\sqrt{2}}$$

The "+" choice leads to the following minimum-phase filter (the "−" choice leads to the reverse of that, which has maximum phase):

$$\begin{aligned} \mathbf{h}=[h_0,h_1,h_2,h_3] &= \frac{1}{4\sqrt{2}}\big[1+\sqrt{3}\,,\ 3+\sqrt{3}\,,\ 3-\sqrt{3}\,,\ 1-\sqrt{3}\big]\\ &= [0.4830\,,\ 0.8365\,,\ 0.2241\,,\ -0.1294] \end{aligned} \tag{10.3.11}$$

The corresponding transfer function $H(z)$ has a double zero at Nyquist $z=-1$ and one inside the unit circle at $z=2-\sqrt{3}$. In fact, $H(z)$ factors as follows:

$$H(z)=h_0(1+z^{-1})^2(1-(2-\sqrt{3})z^{-1})$$

For the D_3 case corresponding to $K=3$, we have the following two quadratic equations and four linear ones that must be satisfied by the filter $\mathbf{h}=[h_0,h_1,h_2,h_3,h_4,h_5]$:

$$\begin{aligned} &h_0^2+h_1^2+h_2^2+h_3^2+h_4^2+h_5^2=1\,, \quad h_0h_4+h_1h_5=0\\ &h_0+h_2+h_4=1/\sqrt{2}\,, \quad h_1+h_3+h_5=1/\sqrt{2}\\ &-h_1+2h_2-3h_3+4h_4-5h_5=0\\ &-h_1+2^2h_2-3^2h_3+4^2h_4-5^2h_5=0 \end{aligned} \tag{10.3.12}$$

where the last two correspond to the values $i=1,2$ in (10.3.8), 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. (10.3.12). Solving the linear ones for h_2,h_3,h_4,h_5 in terms of h_0,h_1, we find,

$$\begin{aligned} h_2&=-4h_0+2h_1+\sqrt{2}/8\\ h_3&=-2h_0+3\sqrt{2}/8\\ h_4&=3h_0-2h_1+3\sqrt{2}/8\\ h_5&=2h_0-h_1+\sqrt{2}/8 \end{aligned} \tag{10.3.13}$$

Inserting these into the first two of Eqs. (10.3.12), we obtain the quadratic system:

$$\begin{aligned} &34h_0^2-(32h_1-\sqrt{2}/4)h_0+10h_1^2-5\sqrt{2}h_1/4-3/8=0\\ &h_1^2-\sqrt{2}h_1/8-3h_0^2-3\sqrt{2}h_0/8=0 \end{aligned} \tag{10.3.14}$$

Solving the second for h_1 in terms of h_0, we find:

$$h_1=\frac{1}{16}\big[\sqrt{2}+\sqrt{768h_0^2+96\sqrt{2}h_0+2}\big] \tag{10.3.15}$$

and inserting this into the first of (10.3.14), we obtain:

$$64h_0^2+2\sqrt{2}h_0-2h_0\sqrt{768h_0^2+96\sqrt{2}h_0+2}-\frac{3}{8}=0$$

or, the equivalent quartic equation:

$$1024h_0^4-128\sqrt{2}h_0^3-48h_0^2-\frac{3\sqrt{2}}{2}h_0+\frac{9}{64}=0 \tag{10.3.16}$$

which has two real solutions and two complex-conjugate ones. Of the real solutions, the one that leads to a minimum-phase filter $\mathbf{h}$ is

$$h_0=\frac{\sqrt{2}}{32}\left[1+\sqrt{10}+\sqrt{5+2\sqrt{10}}\right] \tag{10.3.17}$$

With this solution for h_0, Eqs. (10.3.15) and (10.3.13) lead to the desired minimum-phase filter. Its transfer function $H(z)$ factors as:

$$H(z) = h_0 + h_1 z^{-1} + h_2 z^{-2} + h_3 z^{-3} + h_4 z^{-4} + h_5 z^{-5} = h_0 (1+z^{-1})^3 (1 - z_1 z^{-1}) (1 - z_1^* z^{-1})$$

where z_1 is the following zero lying inside the unit circle:

$$z_1 = \frac{\sqrt{10} - 1 + j\sqrt{2\sqrt{10} - 5}}{1 + \sqrt{10} + \sqrt{5 + 2\sqrt{10}}} \quad \Rightarrow \quad |z_1| = \frac{\sqrt{6}}{1 + \sqrt{10} + \sqrt{5 + 2\sqrt{10}}} = 0.3254 \tag{10.3.18}$$

Finally, we mention that the K flatness constraints (10.3.8) at $\omega = \pi$ for $H(\omega)$ are equivalent to K flatness constraints for the wavelet filter $G(\omega)$ at DC, that is,

$$\left. \frac{d^i G(\omega)}{d\omega^i} \right|_{\omega=0} = 0, \quad i = 0, 1, \ldots, K-1 \tag{10.3.19}$$

In turn, these are equivalent to the K vanishing moment constraints for the wavelet function $\psi(t)$, that is,

$$\int_{-\infty}^{\infty} t^i \psi(t)\, dt = 0, \quad i = 0, 1, \ldots, K-1 \tag{10.3.20}$$

The equivalence between (10.3.19) and (10.3.20) is easily established by differentiating the dilation equation (10.2.14) for $\Psi(\omega)$ with respect to ω and setting $\omega = 0$.

Because the D_K filters have minimum phase by construction, their energy is concentrated at earlier times and their shape is very asymmetric. Daubechies' other two families of scaling and wavelet filters, the "least asymmetric" symmlets, and the coiflets, have a more symmetric shape. They are discussed in detail in [665].

Another consequence of the orthonormality of the ϕ and ψ bases can be stated in terms of the Fourier transforms $\Phi(\omega)$ and $\Psi(\omega)$ as identities in ω:

$$\begin{aligned} &\sum_{m=-\infty}^{\infty} |\Phi(\omega + 2\pi m)|^2 = \sum_{m=-\infty}^{\infty} |\Psi(\omega + 2\pi m)|^2 = 1 \\ &\sum_{m=-\infty}^{\infty} \Phi(\omega + 2\pi m) \Psi^*(\omega + 2\pi m) = 0 \end{aligned} \tag{10.3.21}$$

These follow by applying Parseval's identity to the cross-correlation inner products of the ϕ and ψ bases. For example, we have,

$$\begin{aligned} \delta_k &= (\phi_{j0}, \phi_{jk}) = (\phi_{00}, \phi_{0k}) = \int_{-\infty}^{\infty} \phi(t)\phi(t-k)\, dt = \frac{1}{2\pi} \int_{-\infty}^{\infty} |\Phi(\omega)|^2 e^{j\omega k}\, d\omega \\ &= \frac{1}{2\pi} \int_{-\pi}^{\pi} \left[\sum_{m=-\infty}^{\infty} |\Phi(\omega + 2\pi m)|^2 \right] e^{j\omega k}\, d\omega \end{aligned}$$

where the last expression was obtained by noting that because k is an integer, the exponential $e^{j\omega k}$ is periodic in ω with period 2π, which allowed us to fold the infinite integration range into the $[-\pi, \pi]$ range. But this result is simply the inverse DTFT of

the first of Eqs. (10.3.21). The other results are shown in a similar fashion using the inner products $(\psi_{j0}, \psi_{jk}) = \delta_k$ and $(\phi_{j0}, \psi_{jk}) = 0$.

It can be easily argued from Eqs. (10.2.16) that $\Phi(2\pi m) = 0$ for all non-zero integers m. Indeed, setting $m = 2^p(2q+1)$ for some integers $p \geq 0, q \geq 0$, it follows that after p iterations, an H-factor will appear such that $H\big((2q+1)\pi\big) = H(\pi) = 0$. Setting $\omega = 0$ in the first of Eqs. (10.3.21) and using this property, it follows that $|\Phi(0)|^2 = 1$. Thus, up to a sign, we may set:

$$\Phi(0) = \int_{-\infty}^{\infty} \phi(t)\, dt = 1 \tag{10.3.22}$$

10.4 Multiresolution and Filter Banks

We saw in Eq. (10.1.12) that a signal belonging to a higher-resolution subspace can be expanded in terms of its lower-resolution components. If J and J_0 are the highest and lowest resolutions of interest, then for a signal $f(t) \in V_J$, the multiresolution expansion will have the form:

$$f(t) = \sum_n c_{Jn} \phi_{Jn}(t) = \sum_k c_{J_0 k} \phi_{J_0 k}(t) + \sum_{j=J_0}^{J-1} \sum_k d_{jk} \psi_{jk}(t) \tag{10.4.1}$$

with the various terms corresponding to the direct-sum decomposition:

$$V_J = V_{J_0} \oplus (W_{J_0} \oplus W_{J_0+1} \oplus \cdots \oplus W_{J-1}) \tag{10.4.2}$$

The choice of J, J_0 is dictated by the application at hand. Typically, we start with a signal $f(t)$ sampled at $N = 2^J$ samples that are equally-spaced over the signal's duration. The duration interval can always be normalized to be $0 \leq t \leq 1$ so that the sample spacing is 2^{-J}. The lowest level is $J_0 = 0$ corresponding to sample spacing $2^{-J_0} = 1$, that is, one sample in the interval $0 \leq t \leq 1$. One does not need to choose $J_0 = 0$; any value $0 \leq J_0 \leq J-1$ could be used.

The lower-level expansion coefficients $\{c_{J_0 k}; d_{jk}, J_0 \leq j \leq J-1\}$ can be computed from those of the highest level c_{Jn} by *Mallat's multiresolution algorithm* [721], which establishes a connection between multiresolution analysis and filter banks.

The algorithm successively computes the coefficients at each level from those of the level just above. It is based on establishing the relationship between the expansion coefficients for the decomposition $V_{j+1} = V_j \oplus W_j$ and iterating it over $J_0 \leq j \leq J-1$. An arbitrary element $f(t)$ of V_{j+1} can be expanded in two ways:

$$f(t) = \underbrace{\sum_n c_{j+1,n} \phi_{j+1,n}(t)}_{V_{j+1}} = \underbrace{\sum_k c_{jk} \phi_{jk}(t)}_{V_j} + \underbrace{\sum_k d_{jk} \psi_{jk}(t)}_{W_j} \tag{10.4.3}$$

The right-hand side coefficients are:

$$c_{jk} = (f, \phi_{jk}) = \left(\sum_n c_{j+1,n} \phi_{j+1,n}, \phi_{jk} \right) = \sum_n c_{j+1,n} (\phi_{j+1,n}, \phi_{jk})$$

$$d_{jk} = (f, \psi_{jk}) = \left(\sum_n c_{j+1,n} \phi_{j+1,n}, \psi_{jk} \right) = \sum_n c_{j+1,n} (\phi_{j+1,n}, \psi_{jk})$$

which become, using Eq. (10.2.6),

$$\boxed{\begin{aligned} c_{jk} &= \sum_n h_{n-2k}\, c_{j+1,n} \\ d_{jk} &= \sum_n g_{n-2k}\, c_{j+1,n} \end{aligned}} \qquad \text{(analysis)} \tag{10.4.4}$$

for $j = J-1, J-2, \ldots, J_0$, initialized at $c_{Jn} = f(t_n)$, $n = 0, 1, \ldots, 2^J-1$, with $t_n = n2^{-J}$, that is, the 2^J samples of $f(t)$ in the interval $0 \le t \le 1$. Conversely, the coefficients $c_{j+1,n}$ can be reconstructed from c_{jk}, d_{jk}:

$$\begin{aligned} c_{j+1,n} &= (f, \phi_{j+1,n}) = \left(\sum_k c_{jk}\phi_{jk} + \sum_k d_{jk}\psi_{jk},\ \phi_{j+1,n} \right) \\ &= \sum_k c_{jk}(\phi_{jk}, \phi_{j+1,n}) + \sum_k d_{jk}(\psi_{jk}, \phi_{j+1,n}) \end{aligned}$$

or,

$$\boxed{c_{j+1,n} = \sum_k h_{n-2k}\, c_{jk} + \sum_k g_{n-2k}\, d_{jk}} \qquad \text{(synthesis)} \tag{10.4.5}$$

for $j = J_0, J_0+1, \ldots, J-1$. To see the filter bank interpretation of these results, let us define the *time-reversed* filters $\bar{h}_n = h_{-n}$ and $\bar{g}_n = g_{-n}$, and the downsampling and upsampling operations by a factor of two [670]:

$$\begin{aligned} y_{\text{down}}(n) &= x(2n) \\ y_{\text{up}}(n) &= \sum_k x(k)\,\delta(n-2k) = \begin{cases} x(k), & \text{if } n = 2k \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{10.4.6}$$

and pictorially,

The downsampling operation decreases the sampling rate by a factor of two by keeping only the even-index samples of the input. The upsampling operation increases the sampling rate by a factor of two by inserting a zero between successive input samples. It is the same as the "à trous" operation for filters that we encountered earlier.

With these definitions, the analysis algorithm (10.4.4) is seen to be equivalent to convolving with the time-reversed filters, followed by downsampling. Symbolically,

$$\begin{aligned} c_{jk} &= \sum_n \bar{h}_{2k-n}\, c_{j+1,n} = (\bar{h} * c_{j+1})(2k) \\ d_{jk} &= \sum_n \bar{g}_{2k-n}\, c_{j+1,n} = (\bar{g} * c_{j+1})(2k) \end{aligned} \quad \Rightarrow \quad \boxed{\begin{aligned} \mathbf{c}_j &= (\bar{\mathbf{h}} * \mathbf{c}_{j+1})_{\text{down}} \\ \mathbf{d}_j &= (\bar{\mathbf{g}} * \mathbf{c}_{j+1})_{\text{down}} \end{aligned}} \tag{10.4.7}$$

Similarly, the synthesis algorithm (10.4.5) is equivalent to upsampling the signals c_{jk} and d_{jk} by two and then filtering them through h_n, g_n,

$$c_{j+1,n} = \sum_k h_{n-2k}\, c_{jk} + \sum_k g_{n-2k}\, d_{jk} = \sum_m h_{n-m}\, c^{\text{up}}_{jm} + \sum_m g_{n-m}\, d^{\text{up}}_{jm} \tag{10.4.8}$$

where $c^{\text{up}}_{jm} = \sum_k c_{jk}\,\delta(m-2k)$. Symbolically,

$$\boxed{\mathbf{c}_{j+1} = \mathbf{h} * \mathbf{c}^{\text{up}}_j + \mathbf{g} * \mathbf{d}^{\text{up}}_j} \tag{10.4.9}$$

Fig. 10.4.1 shows a block diagram realization of the analysis and synthesis equations (10.4.7) and (10.4.9) in terms of a so-called *tree-structured iterated filter bank*.

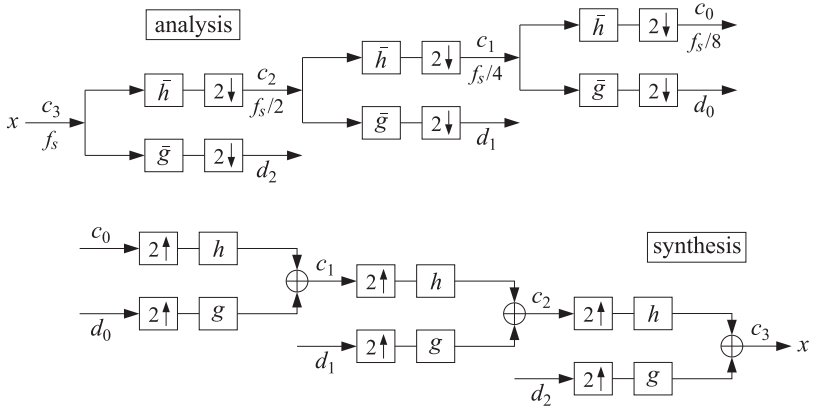


Fig. 10.4.1 Analysis and synthesis filter bank.

In the figure, we used $J = 3$ and $J_0 = 0$. Each stage of the analysis bank produces the coefficients at the next coarser level. Similarly, the synthesis bank starts with the coarsest level and successively reconstructs the higher levels.

The time-reversed filters $\bar{h}_n, \bar{g}_n$ are still lowpass and highpass, indeed, their frequency responses are $\bar{H}(\omega) = H^*(\omega)$ and $\bar{G}(\omega) = G^*(\omega)$. Therefore, at the first analysis stage, the input signal c_3 is split into the low- and high-frequency parts c_2, d_2 representing, respectively, a smoother trend and a more irregular detail. At the second stage, the smooth trend c_2 is split again into a low and high frequency part, c_1, d_1, and so on. The subband frequency operation of the filter bank can be understood by looking at the spectra of the signals at the successive output stages.

Because successive stages operate at different sampling rates, it is best to characterize the spectra using a common frequency axis, for example, the physical frequency f. The spectrum of a discrete-time signal $x(n)$ sampled at a rate f_s is defined by,

$$X(f) = \sum_n x(n)\, e^{-j\omega n} = \sum_n x(n)\, e^{-2\pi j f n/f_s} \tag{10.4.10}$$

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 [30] that for a downsampling ratio by a factor L, and input and output

rates of f_s and $f_s^{\text{down}} = f_s/L$, the downsampled signal $y_{\text{down}}(n) = x(nL)$ will have the following replicated spectrum at multiples of f_s^{down}:

$$Y_{\text{down}}(f) = \frac{1}{L}\sum_{m=0}^{L-1} X(f - mf_s^{\text{down}}) \tag{10.4.11}$$

where according to (10.4.10),

$$Y_{\text{down}}(f) = \sum_n y_{\text{down}}(n)\, e^{-2\pi jfn/f_s^{\text{down}}} = \sum_n x(nL)\, e^{-2\pi jfnL/f_s} \tag{10.4.12}$$

In particular, for downsampling by $L = 2$, we have $f_s^{\text{down}} = f_s/2$ and

$$Y_{\text{down}}(f) = \frac{1}{2}\Big[X(f) + X(f - f_s^{\text{down}})\Big] = \sum_n x(2n)\, e^{-2\pi jf2n/f_s} \tag{10.4.13}$$

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. 10.4.2 shows the corresponding spectra, including the effect of filtering and downsampling.

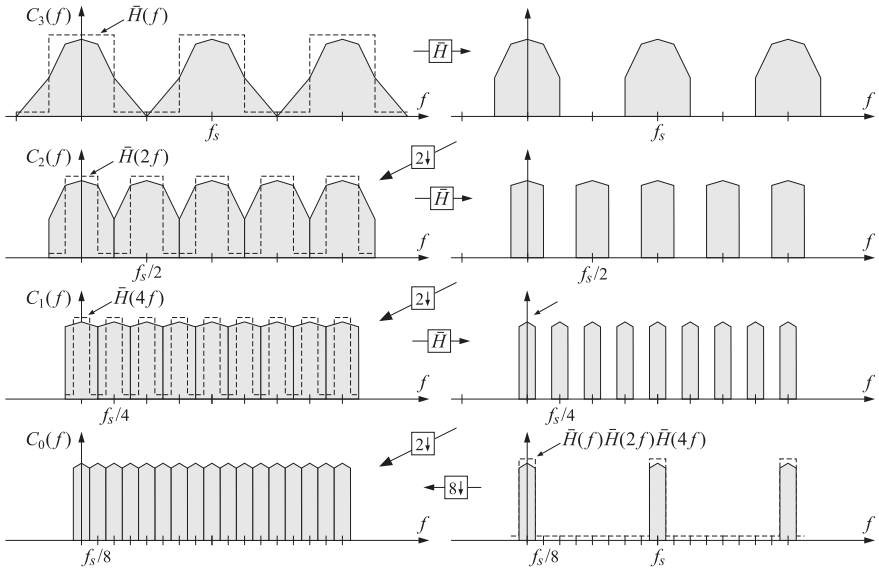


Fig. 10.4.2 Spectra of the signals c_3, c_2, c_1, c_0 at successive stages of the analysis bank of Fig. 10.4.1.

For clarity, we took $\bar{H}(f)$ to be an ideal lowpass filter with cutoff frequency equal to half the Nyquist frequency, that is, $f_s/4$. Starting at the top left with the input spectrum $C_3(f)$, which is replicated at multiples of f_s, the first lowpass filtering operation produces the spectrum at the upper right. According to Eq. (10.4.13), downsampling

will replicate this spectrum at multiples of $f_s^{\text{down}} = f_s/2$, thereby filling the gaps created by the ideal filter, and resulting in the spectrum $C_2(f)$ shown on the left graph of the second row. The sampling rate at that stage is now $f_s/2$.

The second lowpass filtering operation of the signal c_2 indicated on Fig. 10.4.1 will be by the filter $\bar{H}(f, f_s/2)$ which is equal to $\bar{H}(2f, f_s)$ if referred to the original sampling rate f_s; indeed, we have,

$$\bar{H}(f, f_s/2) = \sum_n \bar{h}_n e^{-2\pi jfn/(f_s/2)} = \sum_n \bar{h}_n e^{-2\pi jf2n/f_s} = \bar{H}(2f, f_s) \tag{10.4.14}$$

The filter $\bar{H}(2f, f_s)$ is the twice-compressed version of $\bar{H}(f, f_s)$, and still has an ideal shape but with cutoff frequency $f_s/8$. The result of the second filtering operation is shown on the right graph of the second row. The lowpass-filtered replicas are at multiples of $f_s/2$, and after the next downsampling operation, they will be replicated at multiples of $f_s/4$ resulting in the spectrum $C_1(f)$ of the signal c_1 shown on the left of the third row. At the new sampling rate $f_s/4$, the third-stage lowpass filter will be:

$$\bar{H}(f, f_s/4) = \bar{H}(2f, f_s/2) = \bar{H}(4f, f_s) \tag{10.4.15}$$

which is the four-times compressed version of that at rate f_s, or twice-compressed of that of the previous stage. Its cutoff is now at $f_s/16$. The result of filtering by $\bar{H}(4f, f_s)$ is shown on the right of the third row, and its downsampled version replicated at multiples of $f_s/8$ is shown on the bottom left as the spectrum $C_0(f)$.

Thus, the output spectra $C_2(f), C_1(f), C_0(f)$ capture the frequency content of the original signal in the corresponding successive subbands, each subband having half the passband of the previous one (often referred to as an octave filter bank.)

The bottom-right graph shows an equivalent way of obtaining the same final output $C_0(f)$, namely, by first filtering by the combined filter,

$$\bar{H}(f, f_s)\, \bar{H}(2f, f_s)\, \bar{H}(4f, f_s) = \bar{H}(f, f_s)\, \bar{H}(f, f_s/2)\, \bar{H}(f, f_s/4)$$

running at the original rate f_s, and then dropping the rate all at once by a factor of $2^3 = 8$, which will cause a replication at multiples of $f_s/8$. This point of view is justified by applying the standard multirate identity depicted below [670]:

Fig. 10.4.3 shows the successive application of this identity to the three stages of Fig. 10.4.1 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.

For non-ideal filters $\bar{H}(f), \bar{G}(f)$, such as the scaling and wavelet filters, the downsampling replication property (10.4.13) will cause aliasing. However, because of Eq. (10.3.4), the filter bank satisfies the so-called *pefect reconstruction* property, which allows the aliasing to be canceled at the reconstruction, synthesis, stage.

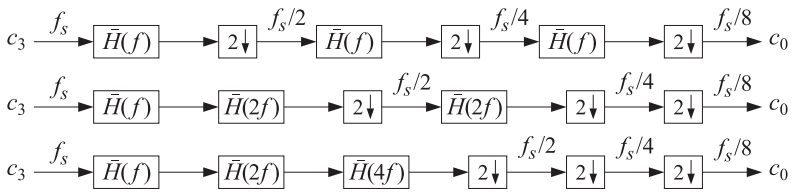


Fig. 10.4.3 Equivalent realizations of the lowpass portion of the analysis bank of Fig. 10.4.1.

10.5 Discrete Wavelet Transform

We summarize the analysis and synthesis algorithms:

$$\boxed{\begin{aligned}\mathbf{c}_{j-1} &= (\bar{\mathbf{h}} * \mathbf{c}_j)_{\text{down}} \\ \mathbf{d}_{j-1} &= (\bar{\mathbf{g}} * \mathbf{c}_j)_{\text{down}}\end{aligned}} \qquad j = J, J-1, \ldots, J_0+1 \tag{10.5.1}$$

$$\boxed{\mathbf{c}_j = \mathbf{h} * \mathbf{c}_{j-1}^{\text{up}} + \mathbf{g} * \mathbf{d}_{j-1}^{\text{up}}} \qquad j = J_0+1, J_0+2, \ldots, J \tag{10.5.2}$$

The discrete wavelet transform (DWT) consists of the coefficients generated by the analysis algorithm. The DWT can be defined for each resolution level. Starting with an input signal vector $\mathbf{x} = [x_0, x_1, \ldots, x_{N-1}]^T$, where $N = 2^J$, the DWTs at successive stages are defined as the following sets of coefficients:

$$\begin{aligned}\mathbf{x} = \mathbf{c}_J &\to [\mathbf{c}_{J-1}, \mathbf{d}_{J-1}], && \text{(level } J-1\text{)} \\ &\to [\mathbf{c}_{J-2}, \mathbf{d}_{J-2}, \mathbf{d}_{J-1}], && \text{(level } J-2\text{)} \\ &\to [\mathbf{c}_{J-3}, \mathbf{d}_{J-3}, \mathbf{d}_{J-2}, \mathbf{d}_{J-1}], && \text{(level } J-3\text{)} \\ &\quad\ \vdots \\ &\to [\mathbf{c}_{J_0}, \mathbf{d}_{J_0}, \mathbf{d}_{J_0+1}, \ldots, \mathbf{d}_{J-1}], && \text{(level } J_0\text{)}\end{aligned} \tag{10.5.3}$$

Starting with the coefficients at any level, the inverse discrete wavelet transform (IDWT) applies the synthesis algorithm to reconstruct the original signal $\mathbf{x}$.

In practice, there are as many variants of the DWT as there are ways to implement the filtering operations in (10.5.1)–(10.5.2), 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 [834–848].

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" [29] as its periodic extension with period N:

$$\tilde{x}(n) = \sum_{p=-\infty}^{\infty} x(n+pN) \tag{10.5.4}$$

The signal $\tilde{x}(n)$ is periodic with period N, and therefore, we only need to know it over one period, $0 \le n \le N-1$. It is characterized by the property that it has the same N-point DFT as the signal $x(n)$, that is,

$$X(\omega_k) = \sum_{n=-\infty}^{\infty} x(n) e^{-j\omega_k n} = \sum_{n=0}^{N-1} \tilde{x}(n) e^{-j\omega_k n} \tag{10.5.5}$$

where ω_k are the DFT frequencies $\omega_k = 2\pi k/N$, $k = 0, 1, \ldots, N-1$. The signal $\tilde{x}(n)$ can be visualized as dividing the original signal $x(n)$ into contiguous blocks of length N, then aligning them in time, and adding them up. This operation is referred to as "mod-N wrapping" and is depicted in Fig. 10.5.1 for a signal $x(n)$ of length $4N$.

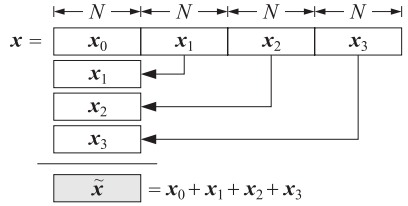
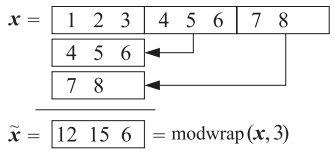


Fig. 10.5.1 Modulo-N reduction or wrapping.

The MATLAB function `modwrap` implements this operation. Its argument can be a row or column vector, or a matrix. For the matrix case, it wraps each column modulo N:

```
Y = modwrap(X,N);      % mod-N reduction of a matrix
```

For example, we have for the signal $\mathbf{x} = [1, 2, 3, 4, 5, 6, 7, 8]$ and $N = 3$,



Circular convolution is defined as the modulo-N reduction of ordinary linear convolution, that is,

$$\mathbf{y} = \mathbf{h} * \mathbf{x} \quad \Rightarrow \quad \mathbf{y}_{\text{circ}} = \tilde{\mathbf{y}} = \widetilde{\mathbf{h} * \mathbf{x}} \tag{10.5.6}$$

or more explicitly,

$$y(n) = \sum_m h(m) x(n-m) \quad \Rightarrow \quad \tilde{y}(n) = \sum_p y(n+pN)$$

Its MATLAB implementation is straightforward with the help of the function `modwrap`, for example,

```
y = modwrap(conv(h,x), N);
```

This code has been incorporated into the function `circonv`, with usage:

```
y = circonv(h,x,N);      % mod-N circular convolution
```

For example, we have the outputs for $N = 8$:

$\boldsymbol{h} =$ 1 2 3 2 1

$\boldsymbol{x} =$ 1 2 3 4 5 6 7 8

$\boldsymbol{y} =$ 1 4 10 18 27 36 45 54 | 54 44 23 8 $= \text{conv}(\boldsymbol{h},\boldsymbol{x})$

54 44 23 8

$\tilde{\boldsymbol{y}} =$ 55 48 33 26 27 36 45 54 $= \text{circonv}(\boldsymbol{h},\boldsymbol{x},8)$

Circular convolution can also be implemented in the frequency domain by computing the N-point DFTs of the signals $\mathbf{h},\mathbf{x}$, multiplying them pointwise together, and performing an inverse N-point DFT. Symbolically,

$$\mathbf{y}_{\text{circ}} = \tilde{\mathbf{y}} = \widetilde{\mathbf{h} * \mathbf{x}} = \text{IDFT}\big[\text{DFT}(\mathbf{h}) \cdot \text{DFT}(\mathbf{x})\big] \tag{10.5.7}$$

or, explicitly,

$$\tilde{y}(n) = \frac{1}{N}\sum_{k=0}^{N-1} H(\omega_k)\,X(\omega_k)\,e^{j\omega_k n} \tag{10.5.8}$$

where $H(\omega_k), X(\omega_k)$ are N-point DFTs as in Eq. (10.5.5). The following MATLAB code illustrates the implementation of the above example in the frequency and time domains:

```
h = [1 2 3 2 1];
x = [1 2 3 4 5 6 7 8];
H = fft(h,8); X = fft(x,8);          % calculate 8-point DFTs
Y = H.*X;                            % point-wise multiplication of the DFTs
ytilde = ifft(Y,8);                  % inverse DFT generates ỹ = [55,48,33,26,27,36,45,54]
ytilde = circonv(h,x,8);             % time-domain calculation
```

The frequency method (10.5.7) becomes efficient if FFTs are used in the right-hand side. However, for our DWT functions, we have used the time-domain implementations, which are equally efficient because the typical wavelet filter lengths are fairly short. The convolutional operations in Eqs. (10.5.1) and (10.5.2) can now be replaced by their circular versions, denoted symbolically,

$$\boxed{\begin{aligned}\mathbf{c}_{j-1} &= \big(\text{circonv}(\bar{\mathbf{h}},\mathbf{c}_j)\big)_{\text{down}}\\ \mathbf{d}_{j-1} &= \big(\text{circonv}(\bar{\mathbf{g}},\mathbf{c}_j)\big)_{\text{down}}\end{aligned}} \quad j = J, J-1, \ldots, J_0+1$$

$$\boxed{\mathbf{c}_j = \text{circonv}(\mathbf{h},\mathbf{c}^{\text{up}}_{j-1}) + \text{circonv}(\mathbf{g},\mathbf{c}^{\text{up}}_{j-1})} \quad j = J_0+1, J_0+2, \ldots, J \tag{10.5.9}$$

DWT in Matrix Form

The convolutional operation $\mathbf{y} = \mathbf{h} * \mathbf{x}$ can be represented in matrix form:

$$\mathbf{y} = H\mathbf{x}$$

where H is the convolution matrix of the filter h_n, defined by its matrix elements:

$$H_{nm} = h_{n-m}$$

The convolution matrix corresponding to the time-reversed filter $\tilde{h}_n = h_{-n}$ 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:

$$\mathbf{y} = (H^T\mathbf{x})_{\text{down}}, \qquad \mathbf{y} = H\mathbf{x}^{\text{up}} \tag{10.5.10}$$

Moreover, replacing the linear convolutions by circular ones amounts to replacing the convolutional matrices by their mod-N wrapped versions obtained by reducing their columns modulo-N, where N is the length of the input vector $\mathbf{x}$. Denoting $\tilde{H} = \text{modwrap}(H,N)$, then the circular version of (10.5.10) would read:

$$\tilde{\mathbf{y}} = (\tilde{H}^T\mathbf{x})_{\text{down}}, \qquad \tilde{\mathbf{y}} = \tilde{H}\mathbf{x}^{\text{up}} \tag{10.5.11}$$

The reduced matrix $\tilde{H}$ will have size $N\times N$, and after downsampling, the output $\tilde{\mathbf{y}} = (\tilde{H}^T\mathbf{x})_{\text{down}}$ will have size $N/2$. Similarly, in the operation $\tilde{\mathbf{y}} = \tilde{H}\mathbf{x}^{\text{up}}$, the upsampled vector $\mathbf{x}^{\text{up}}$ will have length N, as will the output $\tilde{\mathbf{y}}$. Before upsampling, the input $\mathbf{x}$ had length $N/2$. Because every other entry of $\mathbf{x}^{\text{up}}$ is zero, the matrix operation $\tilde{H}\mathbf{x}^{\text{up}}$ can be simplified by replacing $\tilde{H}$ by its "downsampled" version $\tilde{H}_{\text{down}}$ obtained by keeping every other column, and acting on the original vector $\mathbf{x}$, that is, $\tilde{H}\mathbf{x}^{\text{up}} = \tilde{H}_{\text{down}}\mathbf{x}$. The matrix elements of H_{down}, before they are wrapped modulo-N, are $(H_{\text{down}})_{nk} = h_{n-2k}$.

To clarify these remarks, we look at some examples. Consider a length-6 filter, such as D_3 or a Coiflet-1 filter, $\mathbf{h} = [h_0,h_1,h_2,h_3,h_4,h_5]^T$ and take $N = 8$. If the length-4 signal vector $\mathbf{x} = [x_0,x_2,x_4,x_6]^T$ is upsampled by a factor of two, it will become the length-8 vector $\mathbf{x}^{\text{up}} = [x_0,0,x_2,0,x_4,0,x_6,0]^T$. Before wrapping them modulo-8, the convolution matrices H, H_{down} generate the following equivalent outputs:

$$\mathbf{y} = \begin{bmatrix} h_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ h_1 & h_0 & 0 & 0 & 0 & 0 & 0 & 0\\ h_2 & h_1 & h_0 & 0 & 0 & 0 & 0 & 0\\ h_3 & h_2 & h_1 & h_0 & 0 & 0 & 0 & 0\\ h_4 & h_3 & h_2 & h_1 & h_0 & 0 & 0 & 0\\ h_5 & h_4 & h_3 & h_2 & h_1 & h_0 & 0 & 0\\ 0 & h_5 & h_4 & h_3 & h_2 & h_1 & h_0 & 0\\ 0 & 0 & h_5 & h_4 & h_3 & h_2 & h_1 & h_0\\ 0 & 0 & 0 & h_5 & h_4 & h_3 & h_2 & h_1\\ 0 & 0 & 0 & 0 & h_5 & h_4 & h_3 & h_2\\ 0 & 0 & 0 & 0 & 0 & h_5 & h_4 & h_3\\ 0 & 0 & 0 & 0 & 0 & 0 & h_5 & h_4\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & h_5 \end{bmatrix} \begin{bmatrix} x_0\\0\\x_2\\0\\x_4\\0\\x_6\\0 \end{bmatrix} = \begin{bmatrix} h_0 & 0 & 0 & 0\\ h_1 & 0 & 0 & 0\\ h_2 & h_0 & 0 & 0\\ h_3 & h_1 & 0 & 0\\ h_4 & h_2 & h_0 & 0\\ h_5 & h_3 & h_1 & 0\\ 0 & h_4 & h_2 & h_0\\ 0 & h_5 & h_3 & h_1\\ 0 & 0 & h_4 & h_2\\ 0 & 0 & h_5 & h_3\\ 0 & 0 & 0 & h_4\\ 0 & 0 & 0 & h_5\\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_0\\x_2\\x_4\\x_6 \end{bmatrix}$$

or, $\mathbf{y} = H\mathbf{x}^{\text{up}} = H_{\text{down}}\mathbf{x}$. The circular convolution output can be obtained by either wrapping $\mathbf{y}$ modulo-8 or by wrapping H, H_{down} columnwise:

$$\tilde{\mathbf{y}} = \tilde{H}\mathbf{x}^{\text{up}} = \tilde{H}_{\text{down}}\mathbf{x} = \begin{bmatrix} h_0 & 0 & h_4 & h_2 \\ h_1 & 0 & h_5 & h_3 \\ h_2 & h_0 & 0 & h_4 \\ h_3 & h_1 & 0 & h_5 \\ h_4 & h_2 & h_0 & 0 \\ h_5 & h_3 & h_1 & 0 \\ 0 & h_4 & h_2 & h_0 \\ 0 & h_5 & h_3 & h_1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_2 \\ x_4 \\ x_6 \end{bmatrix} \tag{10.5.12}$$

Similarly, in the analysis operation $\tilde{\mathbf{y}} = (\tilde{H}^T\mathbf{x})_{\text{down}}$, downsampling amounts to keeping every other row of the matrix $\tilde{H}^T$, which is $\tilde{H}^T_{\text{down}}$. For example, for the length-8 signal $\mathbf{x} = [x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7]^T$, the corresponding operation will be:

$$\tilde{\mathbf{y}} = (\tilde{H}^T\mathbf{x})_{\text{down}} = \tilde{H}^T_{\text{down}}\mathbf{x} = \begin{bmatrix} h_0 & h_1 & h_2 & h_3 & h_4 & h_5 & 0 & 0 \\ 0 & 0 & h_0 & h_1 & h_2 & h_3 & h_4 & h_5 \\ h_4 & h_5 & 0 & 0 & h_0 & h_1 & h_2 & h_3 \\ h_2 & h_3 & h_4 & h_5 & 0 & 0 & h_0 & h_1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} \tag{10.5.13}$$

The wrapped/downsampled convolution matrix $\tilde{H}_{\text{down}}$ can be calculated very simply in MATLAB, using, for example, the built-in convolution matrix function `convmtx` and the function `modwrap`:

```
H = convmtx(h(:), N);      % ordinary convolution matrix with N columns, h entered as column
H = H(:, 1:2:N);           % downsampled convolution matrix
H = modwrap(H, N);         % wrapped column-wise modulo-N
```

Because $\mathbf{h}$ is fairly short and N typically large, the convolution matrix H can be defined as sparse. This can be accomplished by replacing `convmtx` by the function `convmat`, which we encountered before in Sec. 3.9. Similar convolution matrices $\tilde{G}_{\text{down}}$ can be constructed for the conjugate mirror filter g_n. The function `dwtmat` constructs both matrices for any scaling filter $\mathbf{h}$ and signal length N using `convmat`:

```
[H,G] = dwtmat(h,N);       % sparse DWT matrices
```

The output matrices H, G are defined as sparse and have dimension $N\times(N/2)$. They represent the matrices $\tilde{H}_{\text{down}}, \tilde{G}_{\text{down}}$.

We can now state the precise form of the matrix version of the periodized DWT algorithm. Given a signal (column) vector $\mathbf{x}$ of length $N = 2^J$, we define the DWT matrices H_j, G_j at level j with dimension $N_j\times(N_j/2)$, where $N_j = 2^j$, by

$$[H_j, G_j] = \text{dwtmat}(\mathbf{h}, N_j)\,, \quad J_0 + 1 \le j \le J \tag{10.5.14}$$

Then, the analysis and synthesis algorithms are as follows, initialized with $\mathbf{c}_J = \mathbf{x}$,

$$\begin{aligned} &\text{(DWT)} \quad \boxed{\begin{aligned} \mathbf{c}_{j-1} &= H_j^T\mathbf{c}_j \\ \mathbf{d}_{j-1} &= G_j^T\mathbf{c}_j \end{aligned}} \quad j = J, J-1, \ldots, J_0+1 \\ &\text{(IDWT)} \quad \boxed{\mathbf{c}_j = H_j\mathbf{c}_{j-1} + G_j\mathbf{d}_{j-1}} \quad j = J_0+1, J_0+2, \ldots, J \end{aligned} \tag{10.5.15}$$

The column vector $\mathbf{c}_j$ has dimension $N_j = 2^j$, while the vectors $\mathbf{c}_{j-1}, \mathbf{d}_{j-1}$ have dimension half of that, $N_{j-1} = N_j/2 = 2^{j-1}$. The computations for the forward and inverse transforms are illustrated in Fig. 10.5.2. The *discrete wavelet transform* of $\mathbf{x}$ to level J_0 is the concatenation of the coefficient vectors:

$$\mathbf{w} = \begin{bmatrix} \mathbf{c}_{J_0} \\ \mathbf{d}_{J_0} \\ \mathbf{d}_{J_0+1} \\ \vdots \\ \mathbf{d}_{J-1} \end{bmatrix} \quad \text{(DWT)} \tag{10.5.16}$$

Its total dimension is $N = 2^J$, as can be verified easily,

$$2^{J_0} + 2^{J_0} + \left(2^{J_0+1} + \cdots + 2^{J-1}\right) = 2^J$$

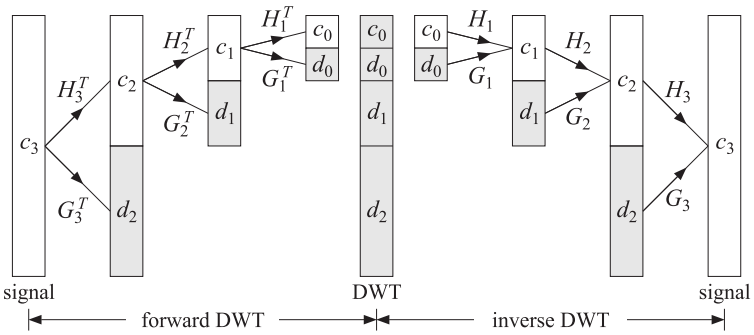


Fig. 10.5.2 Forward and inverse DWT in matrix form.

At each level j, the $N_j\times N_j$ matrix $U_j = [H_j, G_j]$ is an orthogonal matrix, as required by the consistency of the analysis and synthesis steps:

$$\begin{bmatrix} \mathbf{c}_{j-1} \\ \mathbf{d}_{j-1} \end{bmatrix} = \begin{bmatrix} H_j^T \\ G_j^T \end{bmatrix}\mathbf{c}_j \quad \Leftrightarrow \quad \mathbf{c}_j = H_j\mathbf{c}_{j-1} + G_j\mathbf{d}_{j-1} = [H_j, G_j]\begin{bmatrix} \mathbf{c}_{j-1} \\ \mathbf{d}_{j-1} \end{bmatrix}$$

implying the conditions $U_j^TU_j = U_jU_j^T = I_{N_j}$, or,

$$\begin{bmatrix} H_j^T \\ G_j^T \end{bmatrix}[H_j, G_j] = [H_j, G_j]\begin{bmatrix} H_j^T \\ G_j^T \end{bmatrix} = I_{N_j}$$

which are equivalent to the orthogonality conditions:

$$H_j^T H_j = G_j^T G_j = I_{N_j/2}\,, \quad H_j^T G_j = 0\,, \quad H_j H_j^T + G_j G_j^T = I_{N_j} \tag{10.5.17}$$

These follow from the scaling filter orthogonality properties (10.3.1). To see the mechanics by which this happens, consider again our length-6 filter h_n and the corresponding CMF filter g_n defined by $[g_0, g_1, g_2, g_3, g_4, g_5] = [h_5, -h_4, h_3, -h_2, h_1, -h_0]$. Let us also define the cross-correlation quantities:

$$R_k = \sum_n h_n h_{n-2k} \quad \Rightarrow \quad \begin{cases} R_0 = h_0^2 + h_1^2 + h_2^2 + h_3^2 + h_4^2 + h_5^2 \\ R_1 = h_5 h_3 + h_4 h_2 + h_3 h_1 + h_2 h_0 \\ R_2 = h_5 h_1 + h_4 h_0 \end{cases} \tag{10.5.18}$$

From Eq. (10.3.1), we have $R_k = \delta_k$, but let us not assume this just yet, but rather treat h_n as an arbitrary filter and g_n as the corresponding CMF filter. Then, starting with level $J = 3$, the wavelet matrices H_j at $j = 3, 2, 1$, will be:

$$H_3 = \begin{bmatrix} h_0 & 0 & h_4 & h_2 \\ h_1 & 0 & h_5 & h_3 \\ h_2 & h_0 & 0 & h_4 \\ h_3 & h_1 & 0 & h_5 \\ h_4 & h_2 & h_0 & 0 \\ h_5 & h_3 & h_1 & 0 \\ 0 & h_4 & h_2 & h_0 \\ 0 & h_5 & h_3 & h_1 \end{bmatrix}, \qquad H_2 = \begin{bmatrix} h_0 + h_4 & h_2 \\ h_1 + h_5 & h_3 \\ h_2 & h_0 + h_4 \\ h_3 & h_1 + h_5 \end{bmatrix}, \qquad H_1 = \begin{bmatrix} h_0 + h_2 + h_4 \\ h_1 + h_3 + h_5 \end{bmatrix} \tag{10.5.19}$$

with similar definitions for G_j, $j = 3, 2, 1$. By explicit multiplication, we can verify:

$$H_3^T H_3 = G_3^T G_3 = \begin{bmatrix} R_0 & R_1 & 2R_2 & R_1 \\ R_1 & R_0 & R_1 & 2R_2 \\ 2R_2 & R_1 & R_0 & R_1 \\ R_1 & 2R_2 & R_1 & R_0 \end{bmatrix}, \quad H_3^T G_3 = 0$$

$$H_3 H_3^T + G_3 G_3^T = \begin{bmatrix} R_0 & 0 & R_1 & 0 & 2R_2 & 0 & R_1 & 0 \\ 0 & R_0 & 0 & R_1 & 0 & 2R_2 & 0 & R_1 \\ R_1 & 0 & R_0 & 0 & R_1 & 0 & 2R_2 & 0 \\ 0 & R_1 & 0 & R_0 & 0 & R_1 & 0 & 2R_2 \\ 2R_2 & 0 & R_1 & 0 & R_0 & 0 & R_1 & 0 \\ 0 & 2R_2 & 0 & R_1 & 0 & R_0 & 0 & R_1 \\ R_1 & 0 & 2R_2 & 0 & R_1 & 0 & R_0 & 0 \\ 0 & R_1 & 0 & 2R_2 & 0 & R_1 & 0 & R_0 \end{bmatrix}$$

Similarly, we have,

$$H_2^T H_2 = G_2^T G_2 = \begin{bmatrix} R_0 + 2R_2 & 2R_1 \\ 2R_1 & R_0 + 2R_2 \end{bmatrix}, \quad H_2^T G_2 = 0$$

$$H_2 H_2^T + G_2 G_2^T = \begin{bmatrix} R_0 + 2R_2 & 0 & 2R_1 & 0 \\ 0 & R_0 + 2R_2 & 0 & 2R_1 \\ 2R_1 & 0 & R_0 + 2R_2 & 0 \\ 0 & 2R_1 & 0 & R_0 + 2R_2 \end{bmatrix}$$

$$H_1^T H_1 = G_1^T G_1 = R_0 + 2R_1 + 2R_2\,, \quad H_1^T G_1 = 0$$

$$H_1 H_1^T + G_1 G_1^T = \begin{bmatrix} R_0 + 2R_1 + 2R_2 & 0 \\ 0 & R_0 + 2R_1 + 2R_2 \end{bmatrix}$$

Setting $R_0 = 1$ and $R_1 = R_2 = 0$ in all of the above, we verify the orthogonality properties (10.5.17) at all levels $j = 3, 2, 1$. We note that the matrix H_{j-1} can be derived very simply from H_j by keeping only the first $N_{j-1}/2 = N_j/4$ columns and wrapping them modulo-N_{j-1}, that is, in MATLAB notation:

$$H_{j-1} = \text{modwrap}\big(H_j(:, 1 : N_{j-1}/2)\,, N_{j-1}\big)\,, \quad j = J, J-1, \ldots, J_0 + 1 \tag{10.5.20}$$

and similarly for G_{j-1}. This simple operation has been incorporated into the function `dwtwrap`, with usage:

```
H_lower = dwtwrap(H);      % wrap a DWT matrix into a lower one
```

This is evident in Eq. (10.5.19), where H_2 is derivable from H_3, and H_1 from H_2. Because the successive DWT matrices H_j, G_j have different dimensions, $2^j \times 2^{j-1}$, it is convenient to use a cell array to store them in MATLAB. The function `dwtcell` constructs and stores them in sparse form:

```
F = dwtcell(h,N);      % cell array of sparse DWT matrices
```

with the conventions $H_j = F\{1, j\}$ and $G_j = F\{2, j\}$, for $J_0 + 1 \le j \le J$, where N is the highest dimension. The function `fwtm` implements the analysis algorithm in (10.5.15). Its inputs are the signal vector $\mathbf{x}$, the cell array F, and the lowest desired level J_0,

```
w = fwtm(x,F,J0);      % fast wavelet transform in matrix form
```

The vector $\mathbf{w}$ is as in Eq. (10.5.16). If J_0 is omitted, it defaults to $J_0 = 0$. Once the cell array F is created, the function `fwtm` is extremely fast, even faster than the convolution-based function `fwt` discussed below. The function `ifwtm` implements the inverse DWT synthesis algorithm in (10.5.15),

```
x = ifwtm(w,F,J0);      % inverse fast wavelet transform in matrix form
```

An example is the following MATLAB code using the D_3 scaling filter:

```
x = [1 2 3 4 5 6 7 8];
h = daub(3);              % h = [0.3327, 0.8069, 0.4599, -0.1350, -0.0854, 0.0352]
F = dwtcell(h,8);         % construct cell array of DWT matrices

w = fwtm(x,F,0);          % w = [12.7279, -1.4794, -4.4090, 2.2467, 0, 0, -3.7938, 0.9653]
x = ifwtm(w,F,0);         % returns x = [1, 2, 3, 4, 5, 6, 7, 8]
                          % similarly, for J0 = 1,2,3, we find,
w = fwtm(x,F,1);          % w = [7.9539, 10.0461, -4.409, 2.2467, 0, 0, -3.7938, 0.9653]
w = fwtm(x,F,2);          % w = [2.5702, 5.3986, 8.6288, 8.8583, 0, 0, -3.7938, 0.9653]
w = fwtm(x,F,3);          % w = [1, 2, 3, 4, 5, 6, 7, 8] = x, as expected since J = 3
```

These outputs can be understood by looking at the individual matrix operations. Defining, $\mathbf{c}_3 = \mathbf{x} = [1,2,3,4,5,6,7,8]^T$, and the level-3 matrices H_3, G_3, obtained from the call, $[H_3, G_3] = \text{dwtmat}(\mathbf{h}, 8)$,

$$H_3 = \begin{bmatrix} 0.3327 & 0 & -0.0854 & 0.4599 \\ 0.8069 & 0 & 0.0352 & -0.1350 \\ 0.4599 & 0.3327 & 0 & -0.0854 \\ -0.1350 & 0.8069 & 0 & 0.0352 \\ -0.0854 & 0.4599 & 0.3327 & 0 \\ 0.0352 & -0.1350 & 0.8069 & 0 \\ 0 & -0.0854 & 0.4599 & 0.3327 \\ 0 & 0.0352 & -0.1350 & 0.8069 \end{bmatrix}, \; G_3 = \begin{bmatrix} 0.0352 & 0 & 0.8069 & -0.1350 \\ 0.0854 & 0 & -0.3327 & -0.4599 \\ -0.1350 & 0.0352 & 0 & 0.8069 \\ -0.4599 & 0.0854 & 0 & -0.3327 \\ 0.8069 & -0.1350 & 0.0352 & 0 \\ -0.3327 & -0.4599 & 0.0854 & 0 \\ 0 & 0.8069 & -0.1350 & 0.0352 \\ 0 & -0.3327 & -0.4599 & 0.0854 \end{bmatrix}$$

we calculate the level-2 coefficient vectors $\mathbf{c}_2, \mathbf{d}_2$, and the level-2 DWT,

$$\mathbf{c}_2 = H_3^T \mathbf{c}_3 = \begin{bmatrix} 2.5702 \\ 5.3986 \\ 8.6288 \\ 8.8583 \end{bmatrix}, \quad \mathbf{d}_2 = G_3^T \mathbf{c}_3 = \begin{bmatrix} 0 \\ 0 \\ -3.7938 \\ 0.9653 \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} \mathbf{c}_2 \\ \mathbf{d}_2 \end{bmatrix}$$

which agrees with the above MATLAB output of `fwtm(x,F,2)`. Then, from the matrices H_2, G_2, obtained from $[H_2, G_2] = \text{dwtmat}(\mathbf{h}, 4)$, or, from $H_2 = \text{dwtwrap}(H_3)$,

$$H_2 = \begin{bmatrix} 0.2472 & 0.4599 \\ 0.8421 & -0.1350 \\ 0.4599 & 0.2472 \\ -0.1350 & 0.8421 \end{bmatrix}, \; G_2 = \begin{bmatrix} 0.8421 & -0.1350 \\ -0.2472 & -0.4599 \\ -0.1350 & 0.8421 \\ -0.4599 & -0.2472 \end{bmatrix}$$

we calculate the level-1 coefficient vectors $\mathbf{c}_1, \mathbf{d}_1$, and the level-1 DWT,

$$\mathbf{c}_1 = H_2^T \mathbf{c}_2 = \begin{bmatrix} 7.9539 \\ 10.0461 \end{bmatrix}, \quad \mathbf{d}_1 = G_2^T \mathbf{c}_2 = \begin{bmatrix} -4.4090 \\ 2.2467 \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{d}_1 \\ \mathbf{d}_2 \end{bmatrix}$$

which agrees with the above MATLAB output of `fwtm(x,F,1)`. Finally, from the matrices H_1, G_1, obtained from $[H_1, G_1] = \text{dwtmat}(\mathbf{h}, 2)$, or, from $H_1 = \text{dwtwrap}(H_2)$,

$$H_1 = \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix}, \; G_1 = \begin{bmatrix} 0.7071 \\ -0.7071 \end{bmatrix}$$

we find the level-0 coefficient vectors $\mathbf{c}_0, \mathbf{d}_0$, and the level-0 DWT,

$$\mathbf{c}_0 = H_1^T \mathbf{c}_1 = 12.7279, \quad \mathbf{d}_0 = G_1^T \mathbf{c}_1 = -1.4794, \quad \mathbf{w} = \begin{bmatrix} \mathbf{c}_0 \\ \mathbf{d}_0 \\ \mathbf{d}_1 \\ \mathbf{d}_2 \end{bmatrix}$$

Orthogonal DWT Transformation

The mapping of a length-N signal vector $\mathbf{x}$ to the length-N vector $\mathbf{w}$ of wavelet coefficients given in Eq. (10.5.16) is equivalent to an orthogonal matrix transformation, say, $\mathbf{w} = W^T \mathbf{x}$, with inverse $\mathbf{x} = W\mathbf{w}$, such that $W^T W = W W^T = I_N$. The overall $N \times N$ matrix W depends on the stopping level J_0 and can be constructed in terms of the matrices H_j, G_j of the successive stages of the analysis or synthesis algorithms. For example, we have for $N = 2^3$, and $J_0 = 2, 1, 0$,

$$W = [H_3,\, G_3]$$

$$W = [H_3[H_2,\, G_2],\, G_3] = [H_3 H_2,\, H_3 G_2,\, G_3]$$

$$W = [H_3 H_2[H_1,\, G_1],\, H_3 G_2,\, G_3] = [H_3 H_2 H_1,\, H_3 H_2 G_1,\, H_3 G_2,\, G_3]$$

We verify the reconstruction of $\mathbf{x}$ from $\mathbf{w}$ starting at $J_0 = 0$,

$$[H_3 H_2 H_1,\, H_3 H_2 G_1,\, H_3 G_2,\, G_3] \begin{bmatrix} \mathbf{c}_0 \\ \mathbf{d}_0 \\ \mathbf{d}_1 \\ \mathbf{d}_2 \end{bmatrix} = \begin{cases} H_3 H_2 (H_1 \mathbf{c}_0 + G_1 \mathbf{d}_0) + H_3 G_2 \mathbf{d}_1 + G_3 \mathbf{d}_2 = \\ H_3 (H_2 \mathbf{c}_1 + G_2 \mathbf{d}_1) + G_3 \mathbf{d}_2 = \\ H_3 \mathbf{c}_2 + G_3 \mathbf{d}_2 = \mathbf{c}_3 = \mathbf{x} \end{cases}$$

The construction of W can be carried out with the following very simple recursive algorithm, stated in MATLAB notation,

$$\begin{aligned} &W = I_N, \quad (N = 2^J) \\ &\text{for } j = J, J-1, \ldots, J_0 + 1, \\ &\quad W(:,\, 1:2^j) = W(:,\, 1:2^j)\,[H_j, G_j] \end{aligned} \tag{10.5.21}$$

The algorithm updates the first 2^j columns of W at each level j. The MATLAB function `fwtmat` implements (10.5.21) and constructs W as a sparse matrix:

```
W = fwtmat(h,N,J0);      % overall DWT orthogonal matrix
```

As an example, for the D_3 scaling filter and $N = 8$ and lowest level $J_0 = 0$, we find:

$$W = \begin{bmatrix} 0.3536 & -0.3806 & 0.0802 & -0.2306 & 0.0352 & 0 & 0.8069 & -0.1350 \\ 0.3536 & -0.0227 & 0.7368 & -0.0459 & 0.0854 & 0 & -0.3327 & -0.4599 \\ 0.3536 & 0.2197 & 0.3443 & -0.1940 & -0.1350 & 0.0352 & 0 & 0.8069 \\ 0.3536 & 0.5535 & -0.3294 & -0.3616 & -0.4599 & 0.0854 & 0 & -0.3327 \\ 0.3536 & 0.3806 & -0.2306 & 0.0802 & 0.8069 & -0.1350 & 0.0352 & 0 \\ 0.3536 & 0.0227 & -0.0459 & 0.7368 & -0.3327 & -0.4599 & 0.0854 & 0 \\ 0.3536 & -0.2197 & -0.1940 & 0.3443 & 0 & 0.8069 & -0.1350 & 0.0352 \\ 0.3536 & -0.5535 & -0.3616 & -0.3294 & 0 & -0.3327 & -0.4599 & 0.0854 \end{bmatrix}$$

which generates the same DWT as the example above:

```
x = [1 2 3 4 5 6 7 8]'; h = daub(3); W = fwtmat(h,8,0);
w = W'*x;     % gives w = [12.7279, -1.4794, -4.4090, 2.2467, 0, 0, -3.7938, 0.9653]^T
```

The matrix W becomes more and more sparse as N increases. Its sparsity pattern is illustrated in Fig. 10.5.3, for the case of the D_3 scaling filter and dimensions $N = 64$ and $N = 512$. The graphs were generated by the MATLAB code:

```
h = daub(3); N = 64; W = fwtmat(h,N,0); spy(W); percent_nonzero = 100*nnz(W)/N^2
```

The percentages of nonzero entries were 30.5% for $N = 64$, and 6.7% for $N = 512$.

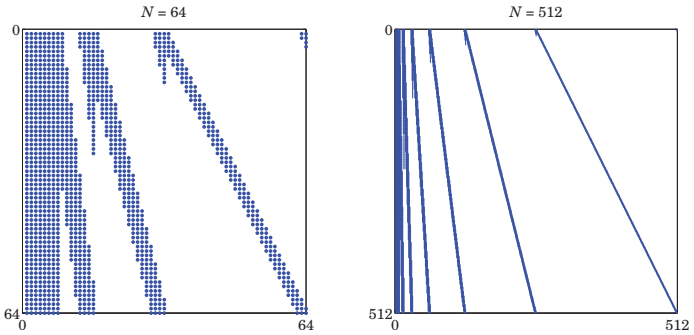


Fig. 10.5.3 Sparsity patterns of DWT matrices.

DWT in Convolutional Form

Next, we look at the detailed implementation of Eq. (10.5.9) using filtering by circular convolution. For practical implementation, we must replace the time-reversed filters $\bar{h}_n, \bar{g}_n$ of the analysis algorithm by their reversed versions, which are delayed by the filter order M to make them causal, that is, $h_n^R = \bar{h}_{n-M} = h_{M-n}$, or in the z-domain $H^R(z) = z^{-M}\bar{H}(z) = z^{-M}H(z^{-1})$.

In order to get the same output as the matrix implementation, we must compensate for such a delay by advancing the input by the same amount. In other words, filtering by $\bar{H}(z)$ is equivalent to advancing the input and then filtering by $H^R(z)$. In the z-domain,

$$Y(z) = \bar{H}(z)X(z) = z^M H^R(z)X(z) = H^R(z)\left[z^M X(z)\right]$$

With these changes, Eq. (10.5.9) now reads,

$$\boxed{\begin{aligned} &\text{advance}(\mathbf{c}_j, M)\\ &\mathbf{c}_{j-1} = \big(\text{circonv}(\mathbf{h}^R, \mathbf{c}_j)\big)_{\text{down}}\\ &\mathbf{d}_{j-1} = \big(\text{circonv}(\mathbf{g}^R, \mathbf{c}_j)\big)_{\text{down}}\end{aligned}} \qquad j = J, J-1, \ldots, J_0+1 \tag{10.5.22}$$

$$\boxed{\mathbf{c}_j = \text{circonv}(\mathbf{h}, \mathbf{c}_{j-1}^{\text{up}}) + \text{circonv}(\mathbf{g}, \mathbf{c}_{j-1}^{\text{up}})} \qquad j = J_0+1, J_0+2, \ldots, J$$

The concrete MATLAB implementation for computing the forward DWT is:

```
g = cmf(h);                          % conjugate mirror of h
hR = flip(h);                        % reversed h
gR = flip(g);
M = length(h) - 1;                   % filter order

c = x(:);                            % initial smooth, x has length 2^J
w = [];                              % DWT coefficient vector

for j=J:-1:J0+1,                     % loop from finest down to coarsest level
   c = advance(c, M);                % length(c) = 2^j
   d = dn2(circonv(gR, c, 2^j));     % convolve circularly and downsample
   c = dn2(circonv(hR, c, 2^j));
   w = [d; w];                       % prepend detail d to previous details
end

w = [c; w];                          % prepend last smooth
```


The function `advance` actually performs a *circular* time-advance modulo the length of its argument vector. The function `dn2` performs downsampling by a factor of two. The results of each loop calculation are appended into the DWT vector **w**. Similarly, the inverse DWT can be calculated by the loop:

```
w = w(:);                                                % work columnwise
c = wcoeff(w,J0);                                        % coarsest smooth at level J0
for j=J0+1:J,
  d = wcoeff(w,J0,j-1);                                  % get detail at level j-1
  c = circonv(h, up2(c), 2^j) + circonv(g, up2(d), 2^j); % output c is 2^j-dimensional
end
x = c;                                                   % reconstructed x
```

Here, the function `wcoeff(w,J0,j-1)` extracts the subvector $\mathbf{d}_{j-1}$ from the wavelet transform vector **w**, and the function `up2` upsamples by a factor of two.

The MATLAB functions `fwt` and `ifwt` incorporate the above code segments to realize the convolutional forms of the DWT and IDWT:

```
w = fwt(x,h,J0);      % fast wavelet transform
x = ifwt(w,h,J0);     % inverse fast wavelet transform
```

Some examples are,

```
x = [1 2 3 4 5 6 7 8];
h = daub(3);
w = fwt(x,h,0);       % w = [12.7279, -1.4794, -4.4090, 2.2467, 0, 0, -3.7938, 0.9653]
x = ifwt(w,h,0);      % returns x = [1, 2, 3, 4, 5, 6, 7, 8]
w = fwt(x,h,1);       % w = [7.9539, 10.0461, -4.409, 2.2467, 0, 0, -3.7938, 0.9653]
w = fwt(x,h,2);       % w = [2.5702, 5.3986, 8.6288, 8.8583, 0, 0, -3.7938, 0.9653]
w = fwt(x,h,3);       % w = [1, 2, 3, 4, 5, 6, 7, 8] = x, as expected
```

A second optional output of `fwt` (and `fwtm`) is the $N\times(J-J_0+1)$ matrix V whose columns are the sub-blocks of **w** according to their resolution,

$$\mathbf{w} = \begin{bmatrix} \mathbf{c}_{J_0} \\ \mathbf{d}_{J_0} \\ \mathbf{d}_{J_0+1} \\ \vdots \\ \mathbf{d}_{J-1} \end{bmatrix} \quad\Rightarrow\quad V = \begin{bmatrix} \mathbf{c}_{J_0} & 0 & 0 & \cdots & 0 \\ 0 & \mathbf{d}_{J_0} & 0 & \cdots & 0 \\ 0 & 0 & \mathbf{d}_{J_0+1} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \mathbf{d}_{J-1} \end{bmatrix} \tag{10.5.23}$$

It is obtained by the calls,

```
[w,V] = fwt(x,h,J0);
[w,V] = fwtm(x,F,J0);
```

The computations in `fwt` are very efficient, resulting in an $O(N)$ algorithm, or more precisely, $O(MN)$, where M is the filter order. By contrast, the FFT is an $O(N\log_2 N)$ algorithm. However, because of their sparsity, the matrix versions are just as efficient if the sparse wavelet matrices are precomputed.

We mentioned earlier that there are several different implementations of the DWT. Different packages may produce different answers, sometimes only differing by a sign or a cyclic permutation within each level. For example, we obtained the following answers for the above example ($\mathbf{x} = [1,2,3,4,5,6,7,8]$ with D_3 and $J_0 = 0$) from the packages:

$\mathbf{w} = [12.7279, -1.4794, -4.4090, \ 2.2467, \ 0.0000, \ 0.0000, -3.7938, \ 0.9653]$ = fwt - ours
$\mathbf{w} = [12.7279, \ 1.4794, \ 4.4090, -2.2467, \ 3.7938, -0.9653, \ 0.0000, \ 0.0000]$ = Wavelab850, Ref. [836]
$\mathbf{w} = [12.7279, -1.4794, -4.4090, \ 2.2467, -3.7938, \ 0.9653, \ 0.0000, \ 0.0000]$ = Wavethresh, Ref. [840]
$\mathbf{w} = [12.7279, \ 1.4794, -2.2467, \ 4.4090, -0.9653, \ 3.7938, \ 0.0000, \ 0.0000]$ = WMTSA, Ref. [844]
$\mathbf{w} = [12.7279, -1.4794, \ 2.2467, -4.4090, \ 0.9653, \ 0.0000, \ 0.0000, -3.7938]$ = Uvi-Wave, Ref. [845]
$\mathbf{w} = [12.7279, \ 1.4794, \ 4.4090, -2.2467, \ 0.0000, \ 0.0000, \ 3.7938, -0.9653]$ = Getz, Ref. [846]
$\mathbf{w} = [12.7279, -1.4794, -4.4090, \ 2.2467, \ 0.0000, \ 0.0000, -3.7938, \ 0.9653]$ = Wavekit, Ref. [847]

10.6 Multiresolution Decomposition

The multiresolution decomposition defined in Eq. (10.1.13), with coarsest level J_0, which was illustrated by Example 10.1.1, and implemented by the function `dwtdec`,

$$f(t) = \sum_n c_{Jn}\phi_{Jn}(t) = \sum_n c_{J_0 n}\phi_{J_0 n}(t) + \sum_{j=J_0}^{J-1}\sum_n d_{jn}\psi_{jn}(t), \tag{10.6.1}$$

can be given a vectorial interpretation. Let $\mathbf{x}$ be the $N = 2^J$ dimensional vector of time samples of the function $f(t)$ at the finest level J, and let W be the orthogonal DWT matrix down to level J_0, with corresponding DWT, $\mathbf{w} = W^T\mathbf{x}$, and inverse $\mathbf{x} = W\mathbf{w}$.

Writing the DWT $\mathbf{w}$ in the partitioned form of Eq. (10.5.23), we may write $\mathbf{x}$ as the sum of multiresolution components, corresponding to the terms of (10.6.1), with each term representing the part of $\mathbf{x}$ arising from a particular level j with all the other levels having zero coefficients:

$$\begin{aligned}\mathbf{x} = W\begin{bmatrix}\mathbf{c}_{J_0}\\ \mathbf{d}_{J_0}\\ \mathbf{d}_{J_0+1}\\ \vdots\\ \mathbf{d}_{J-1}\end{bmatrix} &= W\begin{bmatrix}\mathbf{c}_{J_0}\\ 0\\ 0\\ \vdots\\ 0\end{bmatrix} + W\begin{bmatrix}0\\ \mathbf{d}_{J_0}\\ 0\\ \vdots\\ 0\end{bmatrix} + W\begin{bmatrix}0\\ 0\\ \mathbf{d}_{J_0+1}\\ \vdots\\ 0\end{bmatrix} + \cdots + W\begin{bmatrix}0\\ 0\\ 0\\ \vdots\\ \mathbf{d}_{J-1}\end{bmatrix} \\ &= \mathbf{x}_{J_0} + \underbrace{\left(\tilde{\mathbf{x}}_{J_0} + \tilde{\mathbf{x}}_{J_0+1} + \cdots + \tilde{\mathbf{x}}_{J-1}\right)}_{\mathbf{x}^{\perp}_{J_0}} \\ &= \mathbf{x}_{J_0} + \mathbf{x}^{\perp}_{J_0}\end{aligned} \tag{10.6.2}$$

The terms $\mathbf{x}_{J_0}, \mathbf{x}^{\perp}_{J_0}$ represent the two parts of $\mathbf{x}$ lying in the subspaces V_{J_0} and $V^{\perp}_{J_0}$. The individual terms of $\mathbf{x}^{\perp}_{J_0} = \tilde{\mathbf{x}}_{J_0} + \tilde{\mathbf{x}}_{J_0+1} + \cdots + \tilde{\mathbf{x}}_{J-1}$ contain all the details for levels $J_0 \le j \le J-1$. The various components are mutually orthogonal, as follows from the property $WW^T = I$, and the non-overlapping of the sub-blocks of $\mathbf{w}$,

$$\begin{aligned}\mathbf{x}^T_{J_0}\tilde{\mathbf{x}}_j = 0, \quad & J_0 \le j \le J-1 \\ \tilde{\mathbf{x}}^T_i\tilde{\mathbf{x}}_j = 0, \quad & J_0 \le i,j \le J-1\,, \quad i \ne j\end{aligned} \tag{10.6.3}$$

For the "diagonal" terms, we obtain the norms, again following from $WW^T = I$,

$$\|\mathbf{x}_{J_0}\|^2 = \|\mathbf{c}_{J_0}\|^2\,, \quad \|\tilde{\mathbf{x}}_j\|^2 = \|\mathbf{d}_j\|^2\,, \quad J_0 \le j \le J-1 \tag{10.6.4}$$

where $\|\mathbf{x}\|^2 = \mathbf{x}^T\mathbf{x}$, which lead to the sum,

$$\|\mathbf{x}\|^2 = \|\mathbf{x}_{J_0}\|^2 + \sum_{j=J_0}^{J-1}\|\tilde{\mathbf{x}}_j\|^2 = \|\mathbf{c}_{J_0}\|^2 + \sum_{j=J_0}^{J-1}\|\mathbf{d}_j\|^2 = \|\mathbf{w}\|^2 \tag{10.6.5}$$

The $N\times(J-J_0+1)$ matrix $X = [\mathbf{x}_{J_0}, \tilde{\mathbf{x}}_{J_0}, \tilde{\mathbf{x}}_{J_0+1}\cdots\tilde{\mathbf{x}}_{J-1}]$ incorporates the individual orthogonal columns and is produced as the output of the MATLAB function `dwtdec`,

```
X = dwtdec(x,h,J0);
```
% DWT decomposition into orthogonal multiresolution components

In fact, X is the product $X = WV$, where V is the DWT-component matrix given in (10.5.23). As an example of `dwtdec`, we have for $\mathbf{x} = [1,2,3,4,5,6,7,8]^T$ and D_3,

$$\mathbf{x} = \begin{bmatrix}1\\2\\3\\4\\5\\6\\7\\8\end{bmatrix} \quad\Rightarrow\quad \mathbf{w} = \begin{bmatrix}12.7279\\-1.4794\\-4.4090\\2.2467\\0\\0\\-3.7938\\0.9653\end{bmatrix} \quad\Rightarrow\quad X = \begin{bmatrix}4.5000 & 0.5631 & -0.8716 & -3.1915\\ 4.5000 & 0.0337 & -3.3518 & 0.8181\\ 4.5000 & -0.3251 & -1.9538 & 0.7789\\ 4.5000 & -0.8188 & 0.6399 & -0.3211\\ 4.5000 & -0.5631 & 1.1967 & -0.1336\\ 4.5000 & -0.0337 & 1.8578 & -0.3241\\ 4.5000 & 0.3251 & 1.6287 & 0.5462\\ 4.5000 & 0.8188 & 0.8541 & 1.8271\end{bmatrix}$$

generated with the MATLAB code and test,

```
h = daub(3); x = [1 2 3 4 5 6 7 8]'; X = dwtdec(x,h,0);
[w,V] = fwt(x,h,0); W = fwtmat(h,8,0); norm(X-W*V)
```


10.7 Wavelet Denoising

Figure 10.7.1 shows some wavelet denoising examples consisting of the same four signals (bumps, blocks, heavisine, doppler) that we discussed in Sec. 5.4 under local polynomial modeling with adaptive variable bandwidth. These examples have served as benchmarks in the wavelet denoising literature [821–824].

Fig. 10.7.1 should be compared with Figs. 5.4.1–5.4.4. 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. 5.4. The following code segment illustrates the generation of the upper row of graphs and demonstrates the use of the denoising function `wdenoise`:

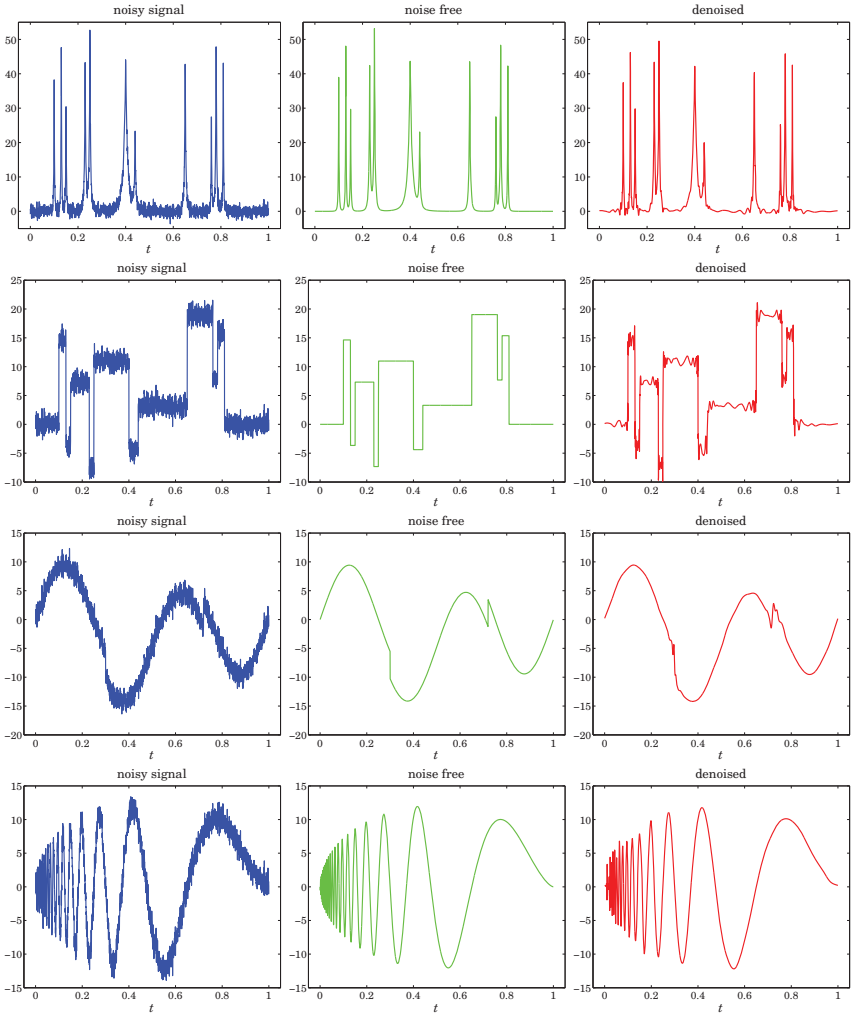


Fig. 10.7.1 Wavelet denoising.

```
F = inline('1./(1 + abs(x)).^4');                          % bumps function

N = 2048; t = (0:N-1)'/N; x = zeros(size(t));              % normalize time to 0 ≤ t ≤ 1

t0 = [10 13 15 23 25 40 44 65 76 78 81]/100;               % signal parameters
a = [40,50,30,40,50,42,21,43,31,51,42] * 1.0523;
w = [5,5,6,10,10,30,10,10,5,8,5]/1000;

for i=1:length(a),                                         % construct noise-free signal
```



```
    x = x + a(i) * F((t-t0(i))/w(i));
end

seed=2009; randn('state',seed); v = randn(size(t));        % generate noise

y = x + v;                                                 % noisy signal with SNR, σx/σv = 7

h = daub(8,2); J0=5; type=1;                               % use Symmlet-8 and soft thresholding
xd = wdenoise(y,h,J0,type);                                % wavelet denoising

figure; plot(t,y,'-'); figure; plot(t,x,'-'); figure; plot(t,xd,'r-');    % top row
```

The main idea in wavelet denoising is to (a) perform a DWT on the noisy signal down to some lower resolution level , (b) modify the wavelet detail coefficients by reducing them to zero if they fall below a certain threshold, and (c) perform an inverse DWT to obtain the denoised signal. The procedure is depicted below:

$$\mathbf{x} \overset{\text{DWT}}{\longrightarrow} \mathbf{w} \overset{\text{thresh}}{\longrightarrow} \mathbf{w}_{\text{thr}} \overset{\text{IDWT}}{\longrightarrow} \mathbf{x}_{\text{thr}}$$

Given a wavelet coefficient d, we denote the thresholding operation by $d_{\text{thr}} = f(d,\lambda)$, where λ is threshold. There are various thresholding functions, but the two simplest ones are the so-called hard and soft thresholding, defined with the help of the unit-step function $u(x)$ as follows:

$$\begin{aligned} d_{\text{thr}} &= f(d,\lambda) = d\,u(|d|-\lambda) && \text{(hard)} \\ d_{\text{thr}} &= f(d,\lambda) = \operatorname{sign}(d)\,(|d|-\lambda)\,u(|d|-\lambda) && \text{(soft)} \end{aligned} \tag{10.7.1}$$

or, equivalently,

$$d_{\text{thr}}^{\text{hard}} = \begin{cases} d\,, & |d| \geq \lambda \\ 0\,, & |d| < \lambda \end{cases}, \qquad d_{\text{thr}}^{\text{soft}} = \begin{cases} d - \operatorname{sign}(d)\,\lambda\,, & |d| \geq \lambda \\ 0\,, & |d| < \lambda \end{cases}$$

If the wavelet transform starts at level J (input length $N = 2^J$) and proceeds down to level J_0, the wavelet transform coefficients will be $\mathbf{w} = \{c_{J_0 n}\,;\, d_{jn}\,,\ J_0 \leq j \leq J-1\}$. The thresholding operation is applied only to the detail coefficients d_{jn}, replacing them by their thresholded values, with a possibly level-dependent threshold λ_j, that is,

$$d_{jn}^{\text{thr}} = f(d_{jn}, \lambda_j) \tag{10.7.2}$$

The simplest possibility is to use the same threshold for all levels. Donoho & Johnstone [821] suggest the following "universal" threshold,

$$\lambda = \sigma\sqrt{2\log_2 N} \qquad \text{(universal threshold)} \tag{10.7.3}$$

where σ^2 is the variance of the additive noise in the data. Since σ is not known, it can be estimated from the wavelet detail coefficients $\mathbf{d}_{J-1}$ at level $J-1$, which for a smooth desired signal are presumably dominated mostly by the noise component. The vector $\mathbf{d} \equiv \mathbf{d}_{J-1}$ has length $N/2 = 2^{J-1}$ and one may estimate σ by using either the standard deviation of $\mathbf{d}$, or its mean-absolute-deviation (MAD), that is,

$$\hat{\sigma} = \operatorname{std}(\mathbf{d})\,, \qquad \hat{\sigma} = \frac{\operatorname{mad}(\mathbf{d})}{0.6745} = \frac{\operatorname{median}(\,|\,\mathbf{d} - \operatorname{median}(\mathbf{d})\,|\,)}{0.6745} \tag{10.7.4}$$

where the factor 0.6745 arises from the implicit assumption that $\mathbf{d}$ is a vector of zero-mean independent normally-distributed components (for a zero-mean, unit-variance, gaussian random variable x, one has the relationship, median$(|x|) = 0.6745$).

Donoho & Johnstone's [821] so-called VisuShrink method uses the universal threshold with the MAD estimate of σ and soft thresholding. The MATLAB function `wdenoise` implements the VisuShrink procedure, but also allows the use of hard thresholding:

```
y = wdenoise(x,h,J0,type);      % wavelet denoising
```

It is possible to derive the soft thresholding rule, as well as some of the other rules, from a regularized optimization point of view. Let $\mathbf{y}$ and $\mathbf{w} = W^T\mathbf{y}$ be the noisy data vector and its DWT, and let $\hat{\mathbf{x}}$ and $\hat{\mathbf{w}} = W^T\hat{\mathbf{y}}$ be the sought estimate and its DWT of the desired signal component $\mathbf{x}$ in the noisy signal model $\mathbf{y} = \mathbf{x} + \mathbf{v}$. An estimation criterion similar to the smoothing spline and reproducing kernel criteria that we considered earlier is the following performance index,

$$\mathcal{J} = \|\mathbf{y} - \hat{\mathbf{x}}\|^2 + P(\hat{\mathbf{x}}) = \min$$

where the first term is the L_2-norm and the second, a positive penalty term. Since the DWT matrix W is orthogonal the first term can be written in terms of the DWTs $\|\mathbf{y} - \hat{\mathbf{x}}\|^2 = \|\mathbf{w} - \hat{\mathbf{w}}\|^2$. Therefore, with a redefinition of P, we may replace the above criterion with one that is formulated in the wavelet domain:

$$\begin{aligned}\mathcal{J} &= \|\mathbf{w} - \hat{\mathbf{w}}\|^2 + P(\hat{\mathbf{w}}) \\ &= \|\mathbf{c}_{J_0} - \hat{\mathbf{c}}_{J_0}\|^2 + \sum_{j=J_0}^{J-1} \|\mathbf{d}_j - \hat{\mathbf{d}}_j\|^2 + P(\hat{\mathbf{d}}_{J_0}, \ldots, \hat{\mathbf{d}}_{J-1}) = \min\end{aligned} \tag{10.7.5}$$

where in the second expression, we used the component representation (10.5.23), and we assumed that P depends only on the wavelet detail coefficients. The following particular choice of P using the L_1 norm leads to the soft thresholding rule:

$$\mathcal{J} = \|\mathbf{c}_{J_0} - \hat{\mathbf{c}}_{J_0}\|^2 + \sum_{j=J_0}^{J-1} \sum_{n=0}^{N_j-1} \|d_{jn} - \hat{d}_{jn}\|^2 + 2\lambda \sum_{j=J_0}^{J-1} \sum_{n=0}^{N_j-1} |\hat{d}_{jn}| = \min \tag{10.7.6}$$

where $N_j = 2^j$ is the dimension of the vector $\mathbf{d}_j$. The minimization with respect to $\mathbf{c}_{J_0}$ gives $\hat{\mathbf{c}}_{J_0} = \mathbf{c}_{J_0}$. Since the d_{jn} terms are decoupled, their minimization can be carried on a typical such term, that is, with the simple scalar criterion:

$$\mathcal{J} = |d - \hat{d}|^2 + 2\lambda|\hat{d}| = \min \tag{10.7.7}$$

whose solution is the soft-thresholding rule,

$$\hat{d} = \begin{cases} d - \operatorname{sign}(d)\lambda\,, & |d| \geq \lambda \\ 0\,, & |d| < \lambda \end{cases} \tag{10.7.8}$$

Other variants of wavelet thresholding and other applications and uses of wavelets in statistics can be found in Refs. [819–833].

10.8 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 [748–761]. 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. 10.4.1 probe the frequency content of the input signal at successively lower frequency bands.

This property was depicted in Fig. 10.4.2 in which the output spectrum after three stages, shown at the bottom two graphs, was the result of filtering by the cascaded filter $\bar{H}(\omega)\bar{H}(2\omega)\bar{H}(4\omega)$, where $\omega = 2\pi f/f_s$, with f_s being the sampling rate at the finest scale. This frequency property is preserved whether the output is undecimated, as in the bottom right graph of Fig. 10.4.2, 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. 10.8.1 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.

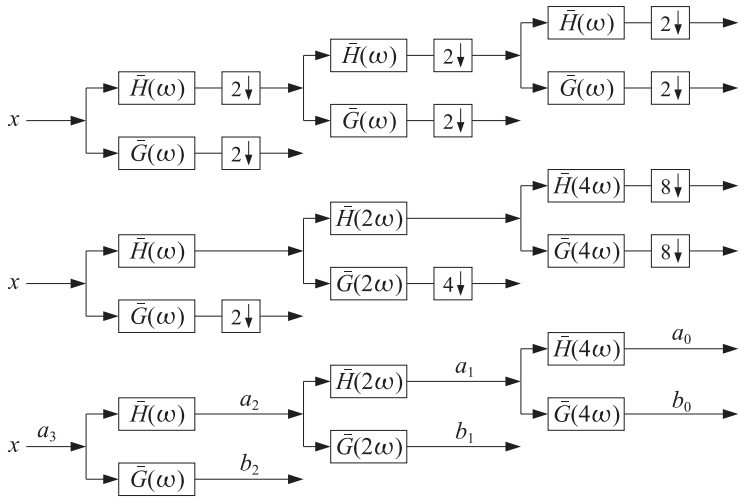


Fig. 10.8.1 Decimated and undecimated filter banks.

The bottom graph effectively implements the undecimated wavelet transform. The individual stages no longer have the orthogonality properties of the usual DWT, such as Eqs. (10.5.17). However, perfect reconstruction can still be achieved by using the

property (10.3.5) for the scaling and wavelet filters:

$$\frac{1}{2}\left[\bar{H}(\omega)H(\omega)+\bar{G}(\omega)G(\omega)\right]=1 \tag{10.8.1}$$

where $\bar{H}(\omega)=H^*(\omega)$ denotes the frequency response of the time-reversed filter $\bar{h}_n$. This relationship admits a block diagram realization as shown in Fig. 10.8.2.

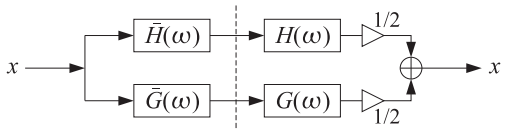


Fig. 10.8.2 Analysis and synthesis of single stage.

Because it is an identity in ω, the same relationship and block diagram will still be valid for the filter pairs $H(2\omega), G(2\omega)$ and $H(4\omega), G(4\omega)$ leading to an overall analysis and synthesis filter bank with perfect reconstruction as shown in Fig. 10.8.3.

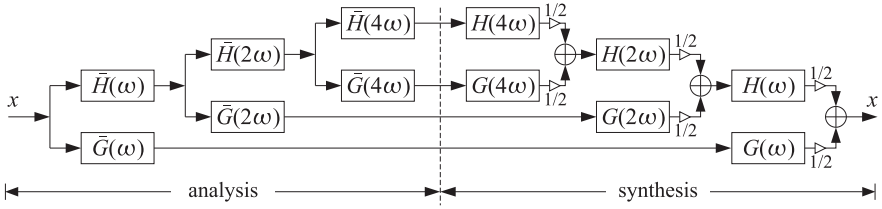


Fig. 10.8.3 Analysis and synthesis filter banks for the UWT.

Thus, it is possible with undecimated filtering operations to achieve (a) the desirable subband filter characteristics of the DWT, and (b) perfect reconstruction. To make the algorithm more concrete, first we recall that the filter with frequency response $H(2^r\omega)$ is the à trous filter defined in Eq. (10.2.20), that is,

$$h^{[r]}(k)=\sum_n h(n)\,\delta(k-2^r n) \quad\Leftrightarrow\quad H^{[r]}(\omega)=H(2^r\omega) \tag{10.8.2}$$

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:

$$\boxed{\begin{aligned}\mathbf{a}_{j-1}&=\bar{\mathbf{h}}^{[J-j]}*\mathbf{a}_j\\ \mathbf{b}_{j-1}&=\bar{\mathbf{g}}^{[J-j]}*\mathbf{a}_j\end{aligned}} \quad J\geq j\geq J_0+1, \qquad \text{(analysis)}$$

$$\boxed{\mathbf{a}_j=\frac{1}{2}\left[\mathbf{h}^{[J-j]}*\mathbf{a}_{j-1}+\mathbf{g}^{[J-j]}*\mathbf{b}_{j-1}\right]} \quad J_0+1\leq j\leq J, \qquad \text{(synthesis)} \tag{10.8.3}$$

where J, J_0 are the finest and coarsest desired resolution levels, and we must initialize the analysis algorithm by the overall input $a_J(n)=x(n)$. Different à trous filters are

used in each stage, unlike the DWT that uses the same filters $\mathbf{h}, \mathbf{g}$. The correctness of the algorithm can be verified by writing Eqs. (10.8.3) in the frequency domain and applying the identity (10.8.1).

To make the algorithm practical we may use mod-N circular convolutions, where $N=2^J$ is the length of the input signal block $\mathbf{x}$. The à trous filters $\mathbf{h}^{[r]}, \mathbf{g}^{[r]}$ can be represented by $N\times N$ matrices H_r, G_r, which are the ordinary convolution matrices of $\mathbf{h}^{[r]}, \mathbf{g}^{[r]}$ reduced modulo-N column-wise. Similarly, the time-reversed filters $\bar{\mathbf{h}}^{[r]}, \bar{\mathbf{g}}^{[r]}$ will be represented by the transposed matrices H_r^T, G_r^T. The construction of these matrices is straightforward, for example,

```
h = h(:);  g = cmf(h);                          % h, g filters

hr = upr(h,r);       gr = upr(g,r);             % upsample by 2^r
Hr = convmtx(hr, N); Gr = convmtx(gr, N);       % ordinary convolution matrices
Hr = modwrap(Hr, N); Gr = modwrap(Gr, N);       % wrapped mod-N column-wise
```

These steps have been incorporated into the function `uwtmat`, except the function `convmat` is used in place of `convmtx` to make the matrices sparse:

```
[Hr,Gr] = uwtmat(h,N,r);        % undecimated wavelet transform matrices
```

The matrices H_r, G_r satisfy the matrix version of Eq. (10.8.1):

$$\frac{1}{2}\left[H_r H_r^T+G_r G_r^T\right]=I_N \tag{10.8.4}$$

The concrete matrix realization of the UWT can be stated then as follows:

$$\boxed{\begin{aligned}\mathbf{a}_{j-1}&=H_{J-j}^T\mathbf{a}_j\\ \mathbf{b}_{j-1}&=G_{J-j}^T\mathbf{a}_j\end{aligned}} \quad J\geq j\geq J_0+1, \qquad \text{(analysis)}$$

$$\boxed{\mathbf{a}_j=\frac{1}{2}\left[H_{J-j}\mathbf{a}_{j-1}+G_{J-j}\mathbf{b}_{j-1}\right]} \quad J_0+1\leq j\leq J, \qquad \text{(synthesis)} \tag{10.8.5}$$

where all the vectors are N-dimensional, initialized at $\mathbf{a}_J=\mathbf{x}$. The algorithm is illustrated in Fig. 10.8.4. The UWT is the $N\times(J-J_0+1)$ matrix U defined column-wise by:

$$U=[\mathbf{a}_{J_0}, \mathbf{b}_{J_0}, \mathbf{b}_{J_0+1}, \ldots, \mathbf{b}_{J-1}] \quad \text{(UWT)} \tag{10.8.6}$$

The MATLAB functions `uwtm` and `iuwtm` implement the algorithms in Eq. (10.8.5):

```
U = uwtm(x,h,J0);       % UWT in matrix form
x = iuwtm(U,h);         % inverse UWT in matrix form
```

An example is as follows:

```
h = daub(3); x = [1 2 3 4 5 6 7 8]';
J0=0; U = uwtm(x,h,J0);                 % or, set J0 = 1 and J0 = 2
xinv = iuwtm(U,h); norm(x-xinv)
```

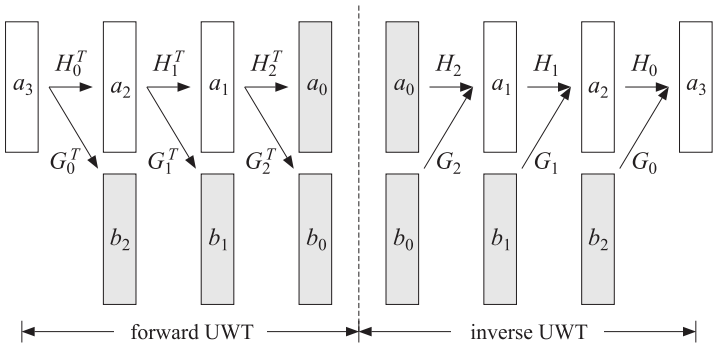


Fig. 10.8.4 Undecimated wavelet transform.

which generates, for $J_0 = 2, 1, 0$,

$$\texttt{U = uwtm(x,h,2)} = \begin{bmatrix} 2.5702 & 0 \\ 3.9844 & 0 \\ 5.3986 & 0 \\ 6.5310 & 2.6614 \\ 8.6288 & -3.7938 \\ 11.1231 & -0.1147 \\ 8.8583 & 0.9653 \\ 3.8173 & 0.2818 \end{bmatrix} = [\mathbf{a}_2, \mathbf{b}_2]$$

$$\texttt{U = uwtm(x,h,1)} = \begin{bmatrix} 7.9539 & -4.4090 & 0 \\ 11.0848 & -1.5166 & 0 \\ 12.3278 & 0.0351 & 0 \\ 12.1992 & 0.4022 & 2.6614 \\ 10.0461 & 2.2467 & -3.7938 \\ 6.9152 & 4.8818 & -0.1147 \\ 5.6722 & 2.1272 & 0.9653 \\ 5.8008 & -3.7674 & 0.2818 \end{bmatrix} = [\mathbf{a}_1, \mathbf{b}_1, \mathbf{b}_2] \tag{10.8.7}$$

$$\texttt{U = uwtm(x,h,0)} = \begin{bmatrix} 12.7279 & -1.4794 & -4.4090 & 0 \\ 12.7279 & 2.9484 & -1.5166 & 0 \\ 12.7279 & 4.7063 & 0.0351 & 0 \\ 12.7279 & 4.5243 & 0.4022 & 2.6614 \\ 12.7279 & 1.4794 & 2.2467 & -3.7938 \\ 12.7279 & -2.9484 & 4.8818 & -0.1147 \\ 12.7279 & -4.7063 & 2.1272 & 0.9653 \\ 12.7279 & -4.5243 & -3.7674 & 0.2818 \end{bmatrix} = [\mathbf{a}_0, \mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2]$$

The functions `uwtm` and `iuwtm` are somewhat slow because they generate the required matrices H_r, G_r on the fly at each stage. Of course, it would be possible to precompute the matrices and save them in a cell or a three-dimensional array as was done in the function `fwtm`. The functions `uwt` and `iuwt` are much faster versions that produce the same results and are implemented using circular convolutions:


```
U = uwt(x,h,J0);      % UWT in convolutional form
x = iuwt(U,h);        % inverse UWT
```

The following MATLAB code shows a possible implementation of the analysis part:

```
M=length(h)-1;
g=cmf(h); hR=flip(h); gR=flip(g);        % construct reversed filters
a = x;                                   % x is N = 2^J dimensional column vector
for r=0:J-J0-1,                          % à trous interpolation factor is 2^r, level j = J - r
   a = advance(a, 2^r*M);                % establishes equivalence with matrix form
   hRr = upr(hR,r);                      % reversed filters upsampled by 2^r
   gRr = upr(gR,r);
   b = circonv(gRr,a,N);                 % modulo-N circular convolution
   a = circonv(hRr,a,N);
   U = [b,U];                            % accumulate the columns of U
end
U = [a,U];
```

The time-advancing operation is necessary to compensate for the use of the reversed filters rather than the time-reversed ones. Although this algorithm works, it is wasteful because the à trous filters $\mathbf{h}^{[r]}$ have length $2^r(M+1)$ consisting mostly of zeros and only $(M+1)$ nonzero coefficients, where M is the filter order of $\mathbf{h}$. The computational cost of the indicated circular convolution operations is of the order of $2^r(M+1)N$. It is possible to restructure these operations so that only the nonzero filter coefficients are used, thereby reducing the computational cost to $(M+1)N$. Ordinary and mod-N circular convolution by the à trous filter (10.8.2) can be written as follows:

$$y(n) = \sum_k h^{[r]}(k)\,x(n-k) = \sum_m h_m\,x(n-2^r m)$$

$$\tilde{y}(n)_{\text{mod-}N} = \sum_p y(n+pN) = \sum_{p,m} h_m\,x(n+pN-2^r m)$$

We assume that $N = 2^J$ and that the à trous factor is such that $r \le J$, so that we may write $N = 2^r L$, where $L = 2^{J-r}$. Thus, a length-N block can be divided into 2^r sub-blocks of length L. We show below that the mod-N circular convolution can be replaced by 2^r mod-L circular convolutions. The total computational cost reduces then to $2^r L(M+1) = N(M+1)$. Setting $n = 2^r i + k$, with $0 \le k \le 2^r - 1$, we may define the k-th sub-block input and output signals:

$$x_k(i) = x(2^r i + k)\,, \quad y_k(i) = y(2^r i + k)\,, \quad 0 \le k \le 2^r - 1$$

It follows then that $y_k(i)$ is the convolution of $x_k(i)$ with the original filter h_m, and that the mod-N circular convolution output can be obtained by mod-L reduction:

$$y_k(i) = y(2^r i + k) = \sum_m h_m\,x(2^r i + k - 2^r m) = \sum_m h_m\,x_k(i-m)$$

$$\tilde{y}(2^r i + k)_{\text{mod-}N} = \sum_{p,m} h_m\,x(2^r i + k + 2^r pL - 2^r m) = \sum_{p,m} h_m\,x_k(i + pL - m)$$

$$= \sum_p y_k(i+pL) = \tilde{y}_k(i)_{\text{mod-}L}$$

These operations have been incorporated into the MATLAB function `convat`,

```
y = convat(h,x,r);        % convolution à trous
```

which is equivalent to the mod-N operation, where N is the length of x:

```
y = circonv(upr(h,r),x,N);
```

The essential part of the function uwt is then,

```
M=length(h)-1;
g=cmf(h); hR=flip(h); gR=flip(g);        % construct reversed filters
a = x;                                   % x is N = 2^J dimensional column vector
for r=0:J-J0-1,                          % à trous interpolation factor is 2^r, level j = J - r
   a = advance(a, 2^r*M);                % establishes equivalence with matrix form
   b = convat(gR, a, r);                 % convolution à trous
   a = convat(hR, a, r);
   U = [b,U];                            % accumulate the columns of U
end
U = [a,U];
```

Because all stages of the analysis and synthesis filter banks in Fig. 10.8.3 operate at the same sampling rate, the UWT satisfies a time-invariance property, in the sense that a time-delay in the input will cause the same delay in the outputs from all stages. Hence, the alternative name "stationary" or "translation-invariant" wavelet transform. Such time-invariance property is not shared by the ordinary DWT.

As an example, the DWTs and UWTs of a signal and its circularly-delayed version by three time units are as follows, using the D_3 scaling filter and coarsest level $J_0 = 0$:

$$\mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{bmatrix} \Rightarrow \mathbf{w} = \begin{bmatrix} 12.7279 \\ -1.4794 \\ -4.4090 \\ 2.2467 \\ 0 \\ 0 \\ -3.7938 \\ 0.9653 \end{bmatrix} \Rightarrow U = \begin{bmatrix} 12.7279 & -1.4794 & -4.4090 & 0 \\ 12.7279 & 2.9484 & -1.5166 & 0 \\ 12.7279 & 4.7063 & 0.0351 & 0 \\ 12.7279 & 4.5243 & 0.4022 & 2.6614 \\ 12.7279 & 1.4794 & 2.2467 & -3.7938 \\ 12.7279 & -2.9484 & 4.8818 & -0.1147 \\ 12.7279 & -4.7063 & 2.1272 & 0.9653 \\ 12.7279 & -4.5243 & -3.7674 & 0.2818 \end{bmatrix}$$

$$\mathbf{x} = \begin{bmatrix} 6 \\ 7 \\ 8 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix} \Rightarrow \mathbf{w} = \begin{bmatrix} 12.7279 \\ -2.9484 \\ 4.8818 \\ -1.5166 \\ -0.1147 \\ 0.2818 \\ 0 \\ 2.6614 \end{bmatrix} \Rightarrow U = \begin{bmatrix} 12.7279 & -2.9484 & 4.8818 & -0.1147 \\ 12.7279 & -4.7063 & 2.1272 & 0.9653 \\ 12.7279 & -4.5243 & -3.7674 & 0.2818 \\ 12.7279 & -1.4794 & -4.4090 & 0 \\ 12.7279 & 2.9484 & -1.5166 & 0 \\ 12.7279 & 4.7063 & 0.0351 & 0 \\ 12.7279 & 4.5243 & 0.4022 & 2.6614 \\ 12.7279 & 1.4794 & 2.2467 & -3.7938 \end{bmatrix}$$

We note that every column of U gets delayed circularly by three time units.

Multiresolution Decomposition with the UWT

The synthesis filter bank or the synthesis algorithm for the UWT can be viewed as a system with $J-J_0+1$ inputs, i.e., the columns of $U = [\mathbf{a}_{J_0}, \mathbf{b}_{J_0}, \mathbf{b}_{J_0+1}, \ldots, \mathbf{b}_{J-1}]$, and

one output, the signal $\mathbf{x}$. The UWT multiresolution decomposition resolves $\mathbf{x}$ into components arising from the individual inputs when the other inputs are zero:

$$\begin{aligned} U = [\mathbf{a}_{J_0}, \mathbf{b}_{J_0}, \mathbf{b}_{J_0+1}, \ldots, \mathbf{b}_{J-1}] &\xrightarrow{\text{IUWT}} \mathbf{x} \\ = [\mathbf{a}_{J_0},\ 0,\ 0,\ \ldots,\ 0\ ] &\xrightarrow{\text{IUWT}} \mathbf{x}_{J_0} \\ + [\ 0,\ \mathbf{b}_{J_0},\ 0,\ \ldots,\ 0\ ] &\xrightarrow{\text{IUWT}} \tilde{\mathbf{x}}_{J_0} \\ + [\ 0,\ 0,\ \mathbf{b}_{J_0+1},\ \ldots,\ 0\ ] &\xrightarrow{\text{IUWT}} \tilde{\mathbf{x}}_{J_0+1} \\ \cdots \qquad\qquad &\qquad \cdots \\ + [\ 0,\ 0,\ 0,\ \ldots,\ \mathbf{b}_{J-1}] &\xrightarrow{\text{IUWT}} \tilde{\mathbf{x}}_{J-1} \end{aligned}$$

so that we have the sum,

$$\boxed{\mathbf{x} = \mathbf{x}_{J_0} + \tilde{\mathbf{x}}_{J_0} + \tilde{\mathbf{x}}_{J_0+1} + \cdots + \tilde{\mathbf{x}}_{J-1}} \tag{10.8.8}$$

This is similar to the DWT decomposition (10.6.2), with each term reflecting a different resolution level, except that the terms are not mutually orthogonal. The MATLAB function uwtdec implements this decomposition:

```
X = uwtdec(x,h,J0);       % UWT multiresolution decomposition
```

where X consists of the columns, $X = [\mathbf{x}_{J_0}, \tilde{\mathbf{x}}_{J_0}, \tilde{\mathbf{x}}_{J_0+1}, \ldots, \tilde{\mathbf{x}}_{J-1}]$.

Fig. 10.8.5 shows an application to the monthly housing starts from January 1988 to April 2009 (i.e., 256 months), using the symmlet S_8 scaling filter and going down to resolution level $J_0 = 5$. This a subset of the dataset that we used repeatedly in Chap. 9.

The upper left graph shows the smooth component arising from the UWT coefficients $\mathbf{a}_{J_0}$. The remaining graphs arise from the detail coefficients. $\mathbf{b}_j$, $J_0 \le j \le J-1$. The sum of the four components is equal to the original data (dotted line in the upper-left graph.) The following MATLAB code generates the four graphs:

```
Y = loadfile('newhouse.dat');              % data file in OSP toolbox

y = Y(349:end,1);                          % selects Jan.88 - Apr.09 = 256 months
t = taxis(y,12,1988)';                     % adjust time axis

h=daub(8,2); J0=5;                         % symmlet S8, note N = 256 = 2^8  ⇒  J = 8
X = uwtdec(y,h,J0);                        % UWT decomposition, try also X = dwtdec(y,h,J0)

figure; plot(t,y,':', t,X(:,1), '-');      % upper left graph
figure; plot(t,X(:,2));                    % upper right
figure; plot(t,X(:,3)); figure; plot(t,X(:,4));   % lower graphs
```

See Fig. 10.9.1 for an alternative way of plotting the UWT (or DWT) decomposition and the UWT (or DWT) wavelet coefficients using the function plotdec.

Wavelet Denoising with the UWT

The application of the UWT to denoising applications follows the same approach as the DWT. The detail columns $\mathbf{b}_j$ of U get thresholded by a possibly level-dependent threshold and the inverse UWT is constructed. The procedure is depicted below:

$$\mathbf{x} \xrightarrow{\text{UWT}} U = [\mathbf{a}_{J_0}, \mathbf{b}_{J_0}, \ldots, \mathbf{b}_{J-1}] \xrightarrow{\text{thresh}} U^{\text{thr}} = [\mathbf{a}_{J_0}, \mathbf{b}^{\text{thr}}_{J_0}, \ldots, \mathbf{b}^{\text{thr}}_{J-1}] \xrightarrow{\text{IUWT}} \mathbf{x}_{\text{thr}}$$

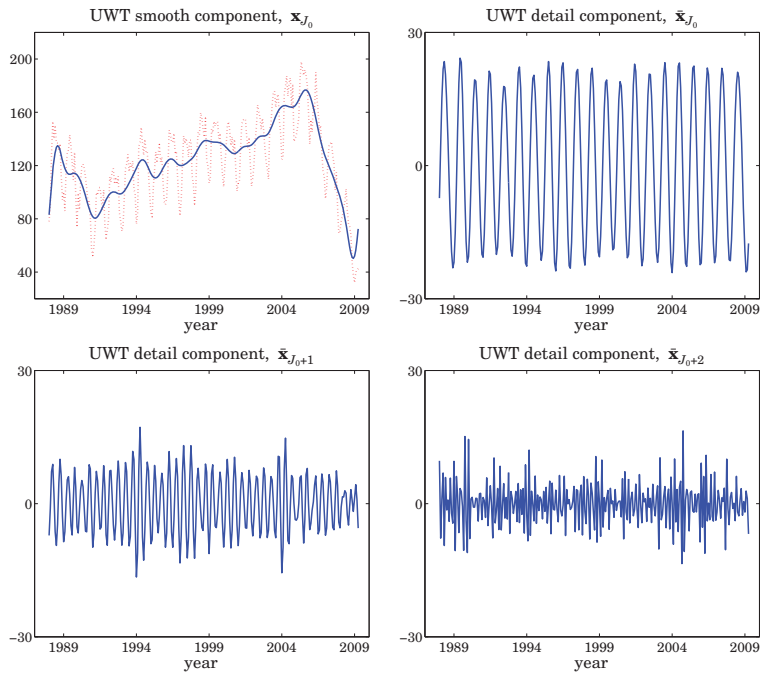


Fig. 10.8.5 UWT decomposition of monthly housing data, using S_8 with $J = 8$ and $J_0 = 5$.

The MATLAB function `wduwt` implements this denoising procedure using the universal threshold (10.7.3) and soft or hard thresholding:

```
y = wduwt(x,h,J0,type);      % wavelet denoising with UWT
```

Fig. 10.8.6 shows the same denoising example as that in Fig. 10.7.1, but denoised using the UWT. The following MATLAB code generates the top-row graphs:

```
F = inline('1./(1 + abs(x)).^4');                    % bumps function

N = 2048; t = (0:N-1)'/N; x = zeros(size(t));        % normalize time to 0 ≤ t ≤ 1

t0 = [10 13 15 23 25 40 44 65 76 78 81]/100;         % signal parameters
a = [40,50,30,40,50,42,21,43,31,51,42] * 1.0523;
w = [5,5,6,10,10,30,10,10,5,8,5]/1000;

for i=1:length(a),                                   % construct noise-free signal
   x = x + a(i) * F((t-t0(i))/w(i));
end

seed=2009; randn('state',seed); v = randn(size(t));  % generate noise

y = x + v;                                           % noisy signal with SNR, σx/σv = 7
```

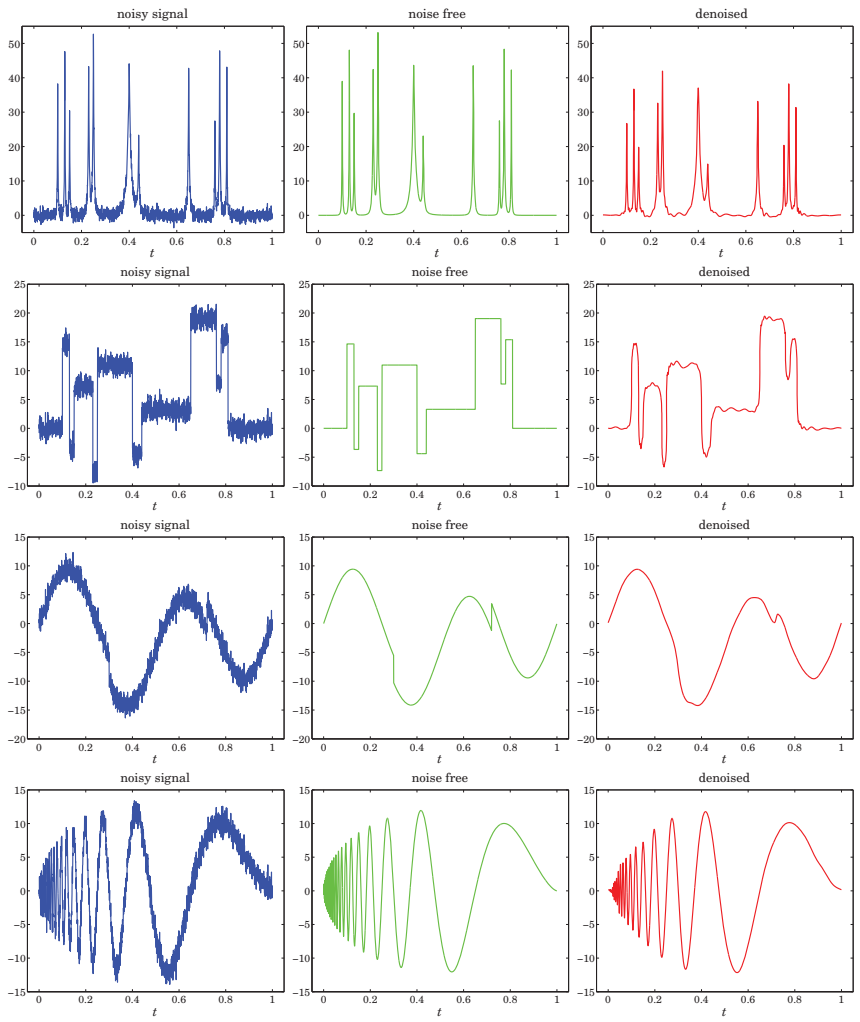



Fig. 10.8.6 Wavelet denoising with the UWT.

```
h = daub(8,2); J0=5; type=1;                         % use Symmlet-8 and soft thresholding
xd = wduwt(y,h,J0,type);                             % wavelet denoising using the UWT

figure; plot(t,y,'-'); figure; plot(t,x,'-'); figure; plot(t,xd,'r-');     % top row
```

Comparing Figs. 10.7.1 and 10.8.6, we note that the UWT outperforms the DWT, an observation that has been made repeatedly in the denoising literature.

10.9 MATLAB Functions

We summarize the MATLAB functions that we discussed in this chapter, and give few more details about `plotdec` that can be used to plot wavelet decompositions and wavelet coefficients.

```
Wavelet functions
-------------------------------------------------
advance   - circular time-advance (left-shift) of a vector
casc      - cascade algorithm for φ and ψ wavelet functions
circonv   - circular convolution
cmf       - conjugate mirror of a filter
convat    - convolution à trous
convmat   - sparse convolution matrix
daub      - Daubechies scaling filters (daublets, symmlets, coiflets)
dn2       - downsample by a factor of 2
dwtcell   - create cell array of sparse discrete wavelet transform matrices
dwtdec    - DWT decomposition into orthogonal multiresolution components
dwtmat    - discrete wavelet transform matrices - sparse
dwtmat2   - discrete wavelet transform matrices - nonsparse
dwtwrap   - wrap a DWT matrix into a lower DWT matrix
flip      - flip a column, a row, or both
fwt       - fast wavelet transform using convolution and downsampling
fwtm      - fast wavelet transform in matrix form
fwtmat    - overall DWT orthogonal matrix
ifwt      - inverse fast wavelet transform - convolutional form
ifwtm     - inverse fast wavelet transform - matrix form
iuwt      - inverse undecimated wavelet transform - convolutional form
iuwtm     - inverse undecimated wavelet transform - matrix form
modwrap   - wrap matrix column-wise mod-N
phinit    - eigenvector initialization of scaling function φ
plotdec   - plot DWT/UWT decomposition or DWT/UWT coefficients
up2       - upsample a vector by factor of 2
upr       - upsample a vector by factor of 2^r
uwt       - undecimated wavelet transform - convolutional form
uwtdec    - UWT multiresolution decomposition
uwtm      - undecimated wavelet transform - matrix form
uwtmat    - undecimated wavelet transform matrices - sparse
uwtmat2   - undecimated wavelet transform matrices - nonsparse
w2V       - wavelet vector to wavelet matrix
wcoeff    - extract wavelet coefficients from DWT at given level
wdenoise  - Donoho & Johnstone's VisuShrink denoising procedure
wduwt     - wavelet denoising with undecimated wavelet transform
wthr      - soft/hard level-dependent wavelet thresholding
-------------------------------------------------
```

The function `plotdec` allows a compact display of a DWT or UWT decomposition, or the display of the DWT/UWT wavelet smooth and detail coefficients:

```
plotdec(X,type,lin,Jmax);
```
% plot DWT/UWT decomposition or DWT/UWT coefficients

with inputs:

`X` = $N\times(J-J_0+1)$ matrix of DWT/UWT decomposition signals or DWT/UWT coefficients, $N = 2^J$
`type` = `'xs','xd','ws','wd'` (x=decomposition, w=wavelet coeffs, s=include smooth, d=details only)
`Jmax` = highest resolution level to plot, $J_{\max} \le J-1$, minimum is determined from $J_0 = J+1-\text{size}(X,2)$
`lin` = `'l','s'` for line or stem plot

See the help for this function for several usage examples. Fig. 10.9.1 shows an alternative plot of the UWT decomposition of Fig. 10.8.5, showing only the detail components, including a plot of the UWT wavelet coefficients. The MATLAB code used to generate the four graphs was as follows:

```
h=daub(8,2); J0=5;
Y = loadfile('newhouse.dat');        % load housing data - file in OSP toolbox
y = Y(349:end,1);                    % selects Jan.88 - Apr.09 = 256 months

Xdwt = dwtdec(y,h,J0);               % DWT decomposition
[w,V] = fwt(y,h,J0);                 % DWT coefficients
Xuwt = uwtdec(y,h,J0);               % UWT decomposition
U = uwt(y,h,J0);                     % UWT coefficients

figure; plotdec(Xdwt,'xd');  figure; plotdec(V,'wd');      % upper graphs
figure; plotdec(Xuwt,'xd');  figure; plotdec(U,'wd');      % lower graphs
```

10.10 Problems

10.1 An alternative way of determining the Daubechies D_2 scaling filter is to assume that its transfer function has the form (with $K = 2$ zeros at Nyquist):

$$H(z) = h_0(1+z^{-1})^2(1-z_1z^{-1})$$

Show that z_1 must the be a solution of the quadratic equation $z^2 - 4z + 1 = 0$. Pick that solution that has $|z_1| < 1$ and verify that the resulting filter $H(z)$ meets all the design constraints (10.3.10).

10.2 To determining the Daubechies D_3 scaling filter assume that its its transfer function has the following form with $K = 3$ zeros at Nyquist:

$$H(z) = h_0(1+z^{-1})^3(1-(a+jb)z^{-1})(1-(a-jb)z^{-1})$$

including a complex zero $z_1 = a + jb$, constrained such that $|z_1| < 1$. Show that the design constraints (10.3.12) imply that b is given by

$$b = \sqrt{\frac{a+a^3-3a^2}{3-a}}$$

and that a is obtained as the following solution of the quartic equation:

$$12a^4 - 72a^3 + 152a^2 - 132a + 27 = 0 \quad \Rightarrow \quad a = \frac{3}{2} - \frac{1}{6}\sqrt{15+12\sqrt{10}}$$

Verify that the zero $z_1 = a + jb$ coincides with that in Eq. (10.3.18). By expanding $H(z)$, express the filter coefficients h_n in terms of a, b, and normalize them to add up to $\sqrt{2}$.

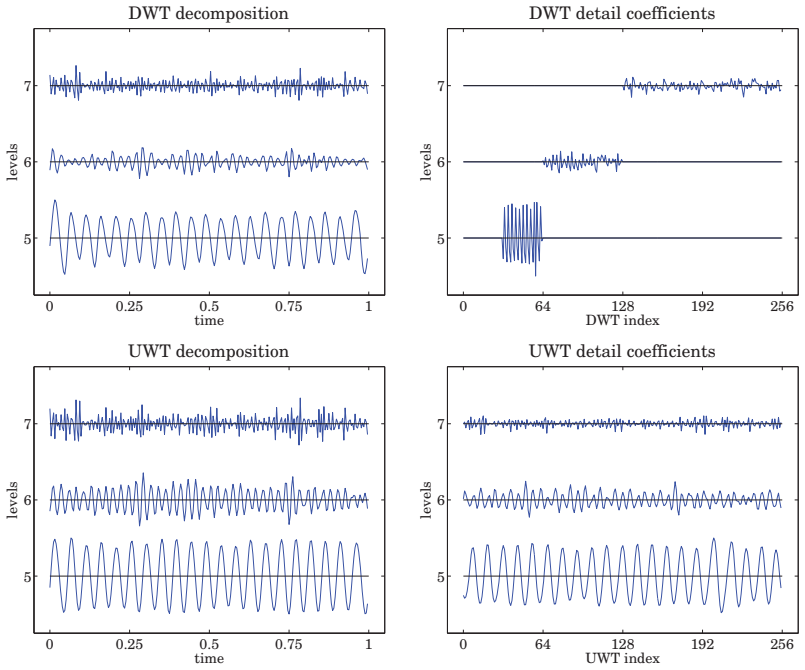


Fig. 10.9.1 UWT/DWT decompositions and wavelet coefficients of housing data.

10.3 Prove the downsampling replication property (10.4.11) by working backwards, that is, start from the Fourier transform expression and show that

$$\frac{1}{L}\sum_{m=0}^{L-1} X(f - mf_s^{\text{down}}) = \sum_k s(k)x(k)e^{-2\pi jfk/f_s} = \sum_n x(nL)e^{-2\pi jfnL/f_s} = Y_{\text{down}}(f)$$

where $s(k)$ is the periodic "sampling function" with the following representations:

$$s(k) = \frac{1}{L}\sum_{m=0}^{L-1} e^{-2\pi jkm/L} = \frac{1}{L}\frac{1-e^{-2\pi jk}}{1-e^{-2\pi jk/L}} = \sum_n \delta(k-nL)$$

Moreover, show that the above representations are nothing but the inverse L-point DFT of the DFT of one period of the periodic pulse train:

$$s(k) = [\ldots, 1, \underbrace{0,0,\ldots,0}_{L-1 \text{ zeros}}, 1, \underbrace{0,0,\ldots,0}_{L-1 \text{ zeros}}, 1, \underbrace{0,0,\ldots,0}_{L-1 \text{ zeros}}, \ldots\ ] = \sum_n \delta(k-nL)$$

10.4 Show that the solution to the optimization problem (10.7.7) is the soft-thresholding rule of Eq. (10.7.8).

10.5 Study the "Tikhonov regularizer" wavelet thresholding function:

$$d_{\text{thr}} = f(d,\lambda,a) = d\,\frac{|d|^a}{|d|^a + \lambda^a}, \quad a > 0,\ \lambda > 0$$

11

Wiener Filtering

The problem of estimating one signal from another is one of the most important in signal processing. In many applications, the desired signal is not available or observable directly. Instead, the observable signal is a degraded or distorted version of the original signal. The signal estimation problem is to recover, in the best way possible, the desired signal from its degraded replica.

We mention some typical examples: (1) The desired signal may be corrupted by strong additive noise, such as weak evoked brain potentials measured against the strong background of ongoing EEGs; or weak radar returns from a target in the presence of strong clutter. (2) An antenna array designed to be sensitive towards a particular "look" direction may be vulnerable to strong jammers from other directions due to sidelobe leakage; the signal processing task here is to null the jammers while at the same time maintaining the sensitivity of the array towards the desired look direction. (3) A signal transmitted over a communications channel can suffer phase and amplitude distortions and can be subject to additive channel noise; the problem is to recover the transmitted signal from the distorted received signal. (4) A Doppler radar processor tracking a moving target must take into account dynamical noise—such as small purely random accelerations—affecting the dynamics of the target, as well as measurement errors. (5) An image recorded by an imaging system is subject to distortions such as blurring due to motion or to the finite aperture of the system, or other geometric distortions; the problem here is to undo the distortions introduced by the imaging system and restore the original image. A related problem, of interest in medical image processing, is that of reconstructing an image from its projections. (6) In remote sensing and inverse scattering applications, the basic problem is, again, to infer one signal from another; for example, to infer the temperature profile of the atmosphere from measurements of the spectral distribution of infrared energy; or to deduce the structure of a dielectric medium, such as the ionosphere, by studying its response to electromagnetic wave scattering; or, in oil exploration to infer the layered structure of the earth by measuring its response to an impulsive input near its surface.

In this chapter, we pose the signal estimation problem and discuss some of the criteria used in the design of signal estimation algorithms.

We do not present a complete discussion of all methods of signal recovery and estimation that have been invented for applications as diverse as those mentioned above.