Notice that the limits of the sum above have not been stated explicitly. They depend on the sizes of vectors x and h. Since our independent variable is the time t, it is convenient to assume that the inputs (and, as a result, also the outputs) come at all times $t = \ldots, -2, -1, 0, 1, 2, \ldots$. Thus, we use the notation

$$x = (\ldots, a, b, c, d, e, \ldots),$$

where the central value c is the input at time zero [$c = x(0)$], values d and e are the inputs at times 1 and 2, respectively, and $b = x(-1)$ and $a = x(-2)$. In practice, the inputs are always finite, so the infinite vector x will have only a finite number of nonzero elements.

Deeper insight into the behavior of a linear filter can be gained by considering the simple input $x = (\ldots, 0, 0, 1, 0, 0, \ldots)$. This input is zero at all times except at $t = 0$. It is called a *unit pulse* or a *unit impulse*. Even though the limits of the sum in the convolution have not been specified, it is easy to see that for any n there is only one nonzero term in the sum, so $y(n) = h(n)x(0) = h(n)$. We say that the output $y(n) = h(n)$ at time n is the *response* at time n to the unit impulse $x(0) = 1$. Since the number of filter coefficients $h(i)$ is finite, the filter is a *finite impulse response*, or FIR.

Figure 4.18 shows the basic idea of a filter bank. It shows an *analysis bank* consisting of two filters, a lowpass filter H_0 and a highpass filter H_1. The lowpass filter employs convolution to remove the high frequencies from the input signal x and let the low frequencies go through. The highpass filter does the opposite. Together, they separate the input into *frequency bands*.

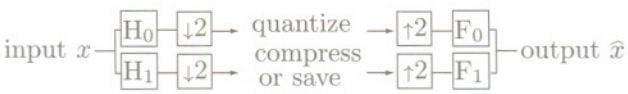

Figure 4.18: A Two-Channel Filter Bank.

The input x can be a one-dimensional signal (a vector of real numbers, which is what we assume in this section) or a two-dimensional signal, an image. The elements $x(n)$ of x are fed into the filters one by one, and each filter computes and outputs one number $y(n)$ in response to $x(n)$. The number of responses is therefore double the number of inputs (because there are two filters), an unfortunate result, since we are interested in compression. To correct this situation, each filter is followed by a *downsampling* process where the odd-numbered outputs are thrown away. This operation is also called *decimation* and is represented by the boxes marked "↓2". After decimation, the number of outputs from the two filters together equals the number of inputs.

Example: It is easy to construct a filter bank where the lowpass part produces averaging and the highpass part produces differences, essentially generating the Haar transform of the input. The filter coefficients for the lowpass filter are $h(0) = h(1) = 1/2$ and the ones for the highpass filter are $h(0) = -1/2$ and $h(1) = 1/2$. Applying these filters to the one-dimensional input sequence

$$\big(x(0), \ldots, x(7)\big) = (255, 224, 192, 159, 127, 95, 63, 32)$$

produces the sequence of averages $a(i)$

$$\begin{aligned}
a(0) &= \frac{1}{2}x(0) + x(-1) = \frac{1}{2}(255 + 0) = 127.5,\\
a(1) &= \frac{1}{2}x(1) + x(0) = \frac{1}{2}(224 + 255) = 239.5,\\
a(2) &= \frac{1}{2}x(2) + x(1) = \frac{1}{2}(192 + 224) = 208,\\
a(3) &= \frac{1}{2}x(3) + x(2) = \frac{1}{2}(159 + 192) = 175.5,\\
a(4) &= \frac{1}{2}x(4) + x(3) = \frac{1}{2}(127 + 159) = 143,\\
a(5) &= \frac{1}{2}x(5) + x(4) = \frac{1}{2}(95 + 127) = 111,\\
a(6) &= \frac{1}{2}x(6) + x(5) = \frac{1}{2}(63 + 95) = 79,\\
a(7) &= \frac{1}{2}x(7) + x(6) = \frac{1}{2}(32 + 63) = 47.5,\\
a(8) &= \frac{1}{2}x(8) + x(7) = \frac{1}{2}(0 + 32) = 16,
\end{aligned}$$

and the sequence of differences $d(i)$

$$\begin{aligned}
d(0) &= -\frac{1}{2}x(0) + x(-1) = \frac{1}{2}(-255 + 0) = -127.5,\\
d(1) &= -\frac{1}{2}x(1) + x(0) = \frac{1}{2}(-224 + 255) = 15.5,\\
d(2) &= -\frac{1}{2}x(2) + x(1) = \frac{1}{2}(-192 + 224) = 16,\\
d(3) &= -\frac{1}{2}x(3) + x(2) = \frac{1}{2}(-159 + 192) = 16.5,\\
d(4) &= -\frac{1}{2}x(4) + x(3) = \frac{1}{2}(-127 + 159) = 16,\\
d(5) &= -\frac{1}{2}x(5) + x(4) = \frac{1}{2}(-95 + 127) = 16,\\
d(6) &= -\frac{1}{2}x(6) + x(5) = \frac{1}{2}(-63 + 95) = 16,\\
d(7) &= -\frac{1}{2}x(7) + x(6) = \frac{1}{2}(-32 + 63) = 15.5,\\
d(8) &= -\frac{1}{2}x(8) + x(7) = \frac{1}{2}(-0 + 32) = 16.
\end{aligned}$$

After decimation, the first sequence is reduced to $(239.5, 175.5, 111, 47.5)$ and the second sequence is reduced to $(15.5, 16.5, 16, 15.5)$. These values can be concatenated to produce the combined sequence $(239.5, 175.5, 111, 47.5, 15.5, 16.5, 16, 15.5)$, which is identical to that produced by Equation (4.1).

Reconstructing the original sequence is done by upsampling (inserting zeros), followed by the synthesis filter bank, which is different from the analysis filter bank. The lowpass synthesis filter uses the two filter coefficients 1 and 1 to produce the four reconstructed values $y(0)$, $y(2)$, $y(4)$, and $y(6)$, while the highpass synthesis filter uses the two filter coefficients 1 and -1 to produce the four reconstructed values $y(1)$, $y(3)$, $y(5)$, and $y(7)$. These eight values are then interleaved to reconstruct the original sequence.

$$\begin{aligned}
y(0) &= a(1) + d(1) = (239.5 + 15.5) = 255,\\
y(2) &= a(3) + d(3) = (175.5 + 16.5) = 192,\\
y(4) &= a(5) + d(5) = (111 + 16) = 127,\\
y(6) &= a(7) + d(7) = (47.5 + 15.5) = 63,\\
\hline
y(1) &= a(1) - d(1) = (239.5 - 15.5) = 224,\\
y(3) &= a(3) - d(3) = (175.5 - 16.5) = 159,\\
y(5) &= a(5) - d(5) = (111 - 16) = 95,\\
y(7) &= a(7) - d(7) = (47.5 - 15.5) = 32.
\end{aligned}$$

Filter banks are a general way of looking at the Haar transform, but they are the key to designing other, more sophisticated discrete wavelet transforms. This technique is discussed in Section 4.5.

The reason for having a bank of filters as opposed to just one filter is that several filters working together, with downsampling, can exhibit behavior that is impossible to obtain with just a single filter. The most important feature of a filter bank is its ability to reconstruct the input from the outputs H_0x and H_1x, even though each has been decimated.

Downsampling is not time invariant. After downsampling, the output is the even-numbered values $y(0)$, $y(2)$, $y(4)$,..., but if we delay the inputs by one time unit, the new outputs will be $y(-1)$, $y(1)$, $y(3)$,..., and these are different from and independent of the original outputs. These two sequences of signals are two phases of vector y.

The outputs of the analysis bank are called *subband coefficients*. They can be quantized (if lossy compression is acceptable), and they can be compressed by means of RLE, Huffman, arithmetic coding, or any other method. Eventually, they are fed into the *synthesis bank*, where they are first upsampled (by inserting zeros for each odd-numbered coefficient that was thrown away), then passed through the inverse filters F_0 and F_1, and finally combined to form a single output vector $\hat{x}$. The output of each analysis filter (after decimation) is

$$(\downarrow y) = (\ldots, y(-4), y(-2), y(0), y(2), y(4), \ldots).$$

Upsampling inserts zeros for the decimated values, so it converts the output vector above to

$$(\uparrow y) = (\ldots, y(-4), 0, y(-2), 0, y(0), 0, y(2), 0, y(4), 0, \ldots).$$

Downsampling causes loss of data. Upsampling alone cannot compensate for it, because it simply inserts zeros for the missing data. In order to achieve lossless re-

construction of the original signal x, the filters have to be designed such that they compensate for this loss of data. One feature that is commonly used in the design of good filters is *orthogonality*. The Haar analysis filter bank uses the two coefficients $(1/2, 1/2)$ in the lowpass filter and the two coefficients $(-1/2, 1/2)$ in the highpass filter. The dot product of these two 2-element vectors is $(1/2, 1/2)\cdot(-1/2, 1/2) = 0$. Similarly, the Haar synthesis filter bank uses the two sets $(1, 1)$ and $(1, -1)$ of orthogonal filter coefficients.

Figure 4.19 shows a set of orthogonal filters of size 4. The filters of the set are orthogonal because their dot product is zero:

$$(a, b, c, d) \cdot (d, -c, b, -a) = 0.$$

Notice how similar H_0 and F_0 are (and also H_1 and F_1). It still remains, of course, to choose actual values for the four filter coefficients a, b, c, and d. A full discussion of this topic is outside the scope of this book, but Section 4.5 illustrates some of the methods and rules used in practice to determine the values of various filter coefficients. An example is the Daubechies D4 filter, whose values are listed in Equation (4.12).

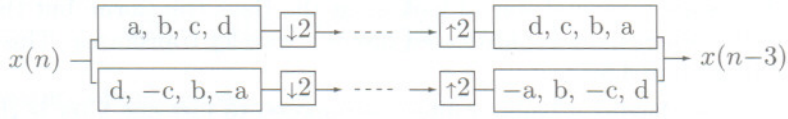

Figure 4.19: An Orthogonal Filter Bank with Four Filter Coefficients.

Simulating the operation of this filter manually shows that the reconstructed input is identical to the original input but lags three time units behind it.

A filter bank can also be *biorthogonal*, a less restricted type of filter. Figure 4.20 shows an example of such a set of filters that can reconstruct a signal exactly. Notice the similarity of H_0 and F_1 and also of H_1 and F_0.

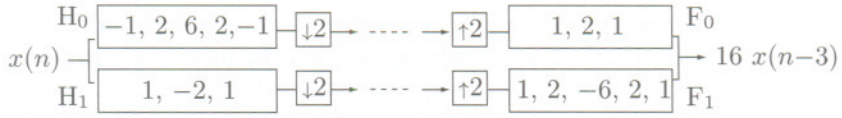

Figure 4.20: A Biorthogonal Filter Bank with Perfect Reconstruction.

We already know, from the discussion in Section 4.2, that the outputs of the lowpass filter H_0 are normally passed through the analysis filter several times, creating shorter and shorter outputs. This recursive process can be illustrated as a tree (Figure 4.21). Since each node of this tree produces half the number of outputs as its predecessor, the tree is called a *logarithmic tree*. Figure 4.21 shows how the scaling function $\phi(t)$ and the wavelet $\psi(t)$ are obtained at the limit of the logarithmic tree. This is the connection

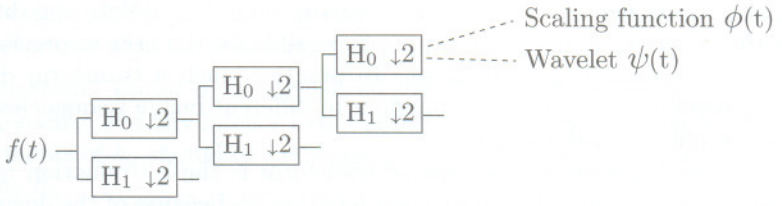

Figure 4.21: Scaling Function and Wavelet as Limits of a Logarithmic Tree.

between the discrete wavelet transform (using filter banks) and the continuous wavelet transform (CWT, [Salomon 00]).

As we "climb" up the logarithmic tree from level i to the next finer level $i+1$, we compute the new averages from the new, higher-resolution scaling functions $\phi(2^i t - k)$ and the new details from the new wavelets $\psi(2^i t - k)$:

$$\begin{array}{rl} \text{signal at level } i \text{ (averages)} & \searrow \\ & + \quad \text{signal at level } i+1. \\ \text{details at level } i \text{ (differences)} & \nearrow \end{array}$$

Each level of the tree corresponds to twice the frequency (or twice the resolution) of the preceding level, which is why the logarithmic tree is also called a *multiresolution tree*. Successive filtering through the tree separates lower and lower frequencies.

Those who do quantitative work with sound and music know that two tones at frequencies ω and 2ω sound like the same note and differ only in pitch. The frequency interval between ω and 2ω is divided into 12 subintervals (the so-called *chromatic scale*), but Western music has a tradition of favoring just eight of the twelve tones that result from this division (a *diatonic scale*, made up of seven notes, with the eighth note as the "octave"). This is why the basic frequency interval used in music is traditionally called an *octave*. We therefore say that adjacent levels of the multiresolution tree differ in an octave of frequencies.

Summary: The discussion of filter banks in this section should be compared to the discussion of image transforms in Section 3.5. Even though both sections describe transforms, they differ in their approach, since they describe different classes of transforms. Each of the transforms described in Section 3.5 is based on a set of *orthogonal* basis functions (or orthogonal basis images) and is computed as an inner product of the input signal with the basis functions. The result is a set of transform coefficients that are subsequently compressed either losslessly (by RLE or some entropy encoder) or lossily (by quantization followed by entropy coding).

This section deals with *subband transforms*, a different type of transform that is computed by taking the *convolution* of the input signal with a set of bandpass filters and decimating the results. Each decimated set of transform coefficients is a subband signal that encodes a specific range of the frequencies of the input. Reconstruction is done by upsampling, followed by computing the inverse transforms, and adding the resulting sets of outputs from the inverse filters.

The main advantage of subband transforms is that they isolate the different frequencies of the input signal, thereby making it possible for the user to precisely control the loss of data in each frequency range. In practice, such a transform decomposes an image into several subbands, corresponding to different image frequencies, and each subband can be quantized differently.

The main disadvantage of this type of transform is the introduction of artifacts, such as aliasing and ringing, into the reconstructed image because of the downsampling. This is why the Haar transform is not satisfactory, and most of the research in this field has been aimed at finding better sets of filters.

Figure 4.22 shows a general subband filter bank with N bandpass filters and three stages. Notice how the output of the highpass filter H_0 of each stage is sent to the next stage for further decomposition and how the combined output of the synthesis bank of a stage is sent to the top inverse filter of the synthesis bank of the preceding stage.

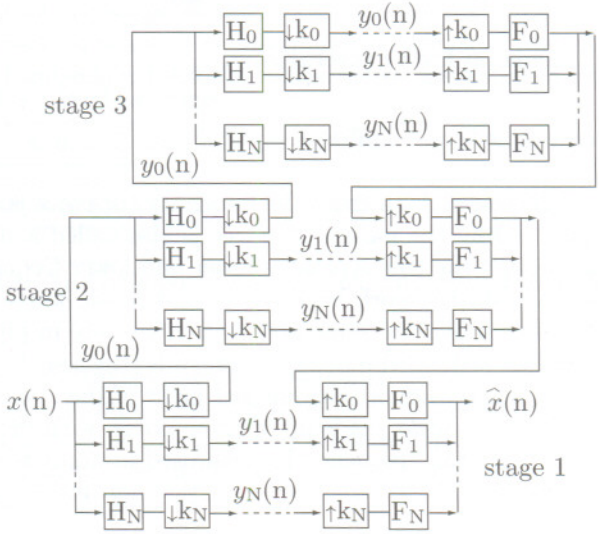

Figure 4.22: A General Filter Bank.

4.5 Deriving the Filter Coefficients

After presenting the basic operation of filter banks, the natural question is "How are the filter coefficients derived?" A full answer is outside the scope of this book (see, for example, [Akansu and Haddad 92]), but this section provides a glimpse at the rules and methods used to figure out the values of various filter banks.

Given a set of two forward and two inverse N-tap filters H_0 and H_1 and F_0 and F_1 (where N is even), we denote their coefficients by

$$h_0 = \big(h_0(0), h_0(1), \ldots, h_0(N-1)\big), \quad h_1 = \big(h_1(0), h_1(1), \ldots, h_1(N-1)\big),$$
$$f_0 = \big(f_0(0), f_0(1), \ldots, f_0(N-1)\big), \quad f_1 = \big(f_1(0), f_1(1), \ldots, f_1(N-1)\big).$$

The four vectors h_0, h_1, f_0, and f_1 are the *impulse responses* of the four filters. Following is the simplest set of conditions that these quantities have to satisfy:

1. Normalization: Vector h_0 is normalized (i.e., its length is one unit).
2. Orthogonality: For any integer i that satisfies $1 \le i < N/2$, the vector formed by the first $2i$ elements of h_0 should be orthogonal to the vector formed by the last $2i$ elements of the same h_0.
3. Vector f_0 is the reverse of h_0.
4. Vector h_1 is a copy of f_0 where the signs of the odd-numbered elements (the first, third, etc.) are reversed. We can express this by saying that h_1 is computed by coordinate multiplication of h_1 and $(-1, 1, -1, 1, \ldots, -1, 1)$.
5. Vector f_1 is a copy of h_0 where the signs of the even-numbered elements (the second, fourth, and so on) are reversed. We can express this by saying that f_1 is computed by coordinate multiplication of h_0 and $(1, -1, 1, -1, \ldots, 1, -1)$.

For two-tap filters, rule 1 implies

$$h_0^2(0) + h_0^2(1) = 1. \tag{4.10}$$

Rule 2 is not applicable because $N = 2$, so $i < N/2$ implies $i < 1$. Rules 3–5 yield

$$f_0 = \big(h_0(1), h_0(0)\big), \quad h_1 = \big(-h_0(1), h_0(0)\big), \quad f_1 = \big(h_0(0), -h_0(1)\big).$$

It all depends on the values of $h_0(0)$ and $h_0(1)$, but the single Equation (4.10) is not enough to determine them. However, it is not hard to see that the choice $h_0(0) = h_0(1) = 1/\sqrt{2}$ satisfies Equation (4.10).

For four-tap filters, rules 1 and 2 imply

$$h_0^2(0) + h_0^2(1) + h_0^2(2) + h_0^2(3) = 1, \quad h_0(0)h_0(2) + h_0(1)h_0(3) = 0, \tag{4.11}$$

and rules 3–5 yield

$$\begin{aligned} f_0 &= \big(h_0(3), h_0(2), h_0(1), h_0(0)\big), \\ h_1 &= \big(-h_0(3), h_0(2), -h_0(1), h_0(0)\big), \\ f_1 &= \big(h_0(0), -h_0(1), h_0(2), -h_0(3)\big). \end{aligned}$$

Again, Equation (4.11) is not enough to determine four unknowns, and other considerations (aided by mathematical intuition) are needed to derive the four values. These values are listed in Equation (4.12)—the Daubechies D4 filter.

For eight-tap filters, rules 1 and 2 imply

$$\begin{gathered} h_0^2(0) + h_0^2(1) + h_0^2(2) + h_0^2(3) + h_0^2(4) + h_0^2(5) + h_0^2(6) + h_0^2(7) = 1, \\ h_0(0)h_0(2) + h_0(1)h_0(3) + h_0(2)h_0(4) + h_0(3)h_0(5) + h_0(4)h_0(6) + h_0(5)h_0(7) = 0, \\ h_0(0)h_0(4) + h_0(1)h_0(5) + h_0(2)h_0(6) + h_0(3)h_0(7) = 0, \\ h_0(0)h_0(6) + h_0(1)h_0(7) = 0, \end{gathered}$$

and rules 3–5 yield

$$f_0 = \big(h_0(7), h_0(6), h_0(5), h_0(4), h_0(3), h_0(2), h_0(1), h_0(0)\big),$$
$$h_1 = \big(-h_0(7), h_0(6), -h_0(5), h_0(4), -h_0(3), h_0(2), -h_0(1), h_0(0)\big),$$
$$f_1 = \big(h_0(0), -h_0(1), h_0(2), -h_0(3), h_0(4), -h_0(5), h_0(6), -h_0(7)\big).$$

The eight coefficients are listed in Table 4.23 (this is the Daubechies D8 filter).

.230377813309	.714846570553	.630880767930	−.027983769417
−.187034811719	.030841381836	.032883011667	−.010597401785

Table 4.23: Coefficients for the Daubechies 8-Tap Filter.

Determining the N filter coefficients for each of the four filters H_0, H_1, F_0, and F_1 depends on $h_0(0)$ through $h_0(N-1)$, so it requires N equations. However, in each of the cases, rules 1 and 2 supply only $N/2$ equations. Other conditions have to be imposed and satisfied before the N quantities $h_0(0)$ through $h_0(N-1)$ can be determined. Here are some examples:

Lowpass H_0 filter: We want H_0 to be a lowpass filter, so it makes sense to require that the frequency response $H_0(\omega)$ be zero for the highest frequency $\omega = \pi$.

Minimum phase filter: This condition requires the zeros of the complex function $H_0(z)$ to lie on or inside the unit circle in the complex plane.

Controlled collinearity: The linearity of the phase response can be controlled by requiring that the sum

$$\sum_i \big(h_0(i) - h_0(N-1-i)\big)^2$$

be a minimum.

Other conditions are discussed in [Akansu and Haddad 92].

4.6 The DWT

Information that is produced and analyzed in real-life situations is discrete. It comes in the form of numbers, rather than as a continuous function. This is why the discrete wavelet transform (DWT) is used in practical calculations. There is also a continuous wavelet transform (CWT, [Lewalle 95] and [Rao and Bopardikar 98]) and studying the CWT may help in understanding the operation of the DWT.

The DWT involves a convolution, but experience shows that the quality of this type of transform depends heavily on two things, the choice of scale factors and time shifts and the choice of wavelet.

In practice, the DWT is computed with scale factors that are negative powers of 2 and with time shifts that are nonnegative powers of 2. Figure 4.24 shows the so-called *dyadic lattice* that illustrates this particular choice. The wavelets used are those that generate orthonormal (or biorthogonal) wavelet bases.

The main thrust in wavelet research has therefore been the search for wavelet families that form orthogonal bases. Of those wavelets, the preferred ones are those that

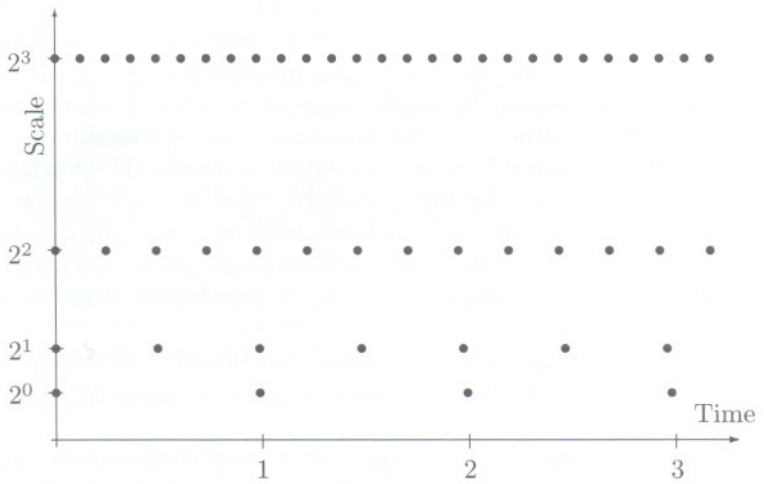

Figure 4.24: The Dyadic Lattice Showing the Relation between Scale Factors and Time.

have compact support, because they allow for DWT computations with *finite impulse response* (FIR) filters.

The simplest way to describe the discrete wavelet transform is by means of matrix multiplication, along the lines developed in Section 4.2.1. The Haar transform depends on two *filter coefficients* c_0 and c_1, both with a value of $1/\sqrt{2} \approx 0.7071$. The smallest transform matrix that can be constructed in this case is $\binom{1\ \ 1}{1\ -1}/\sqrt{2}$. This is a 2×2 matrix, and it generates two transform coefficients, an average and a difference. (Notice that these are not exactly an average and a difference, because $\sqrt{2}$ is used instead of 2. Better names for them are *coarse detail* and *fine detail*, respectively.) In general, the DWT can use any set of wavelet filters, but it is computed in the same way regardless of the particular filter used.

We start with one of the most popular wavelets, the Daubechies D4. As its name implies, it is based on four filter coefficients c_0, c_1, c_2, and c_3, whose values are listed in Equation (4.12). The transform matrix W is [compare with matrix A_1, Equation (4.1)]

$$W = \begin{pmatrix} c_0 & c_1 & c_2 & c_3 & 0 & 0 & \dots & 0 \\ c_3 & -c_2 & c_1 & -c_0 & 0 & 0 & \dots & 0 \\ 0 & 0 & c_0 & c_1 & c_2 & c_3 & \dots & 0 \\ 0 & 0 & c_3 & -c_2 & c_1 & -c_0 & \dots & 0 \\ \vdots & \vdots & & & & \ddots & & \\ 0 & 0 & \dots & 0 & c_0 & c_1 & c_2 & c_3 \\ 0 & 0 & \dots & 0 & c_3 & -c_2 & c_1 & -c_0 \\ c_2 & c_3 & 0 & \dots & 0 & 0 & c_0 & c_1 \\ c_1 & -c_0 & 0 & \dots & 0 & 0 & c_3 & -c_2 \end{pmatrix}.$$

When this matrix is applied to a column vector of data items $(x_1, x_2, \dots, x_n)$, its top

row generates the weighted sum $s_1 = c_0x_1 + c_1x_2 + c_2x_3 + c_3x_4$, its third row generates the weighted sum $s_2 = c_0x_3 + c_1x_4 + c_2x_5 + c_3x_6$, and the other odd-numbered rows generate similar weighted sums s_i. Such sums are *convolutions* of the data vector x_i with the four filter coefficients. In the language of wavelets, each of them is called a *smooth coefficient*, and together they are called an H smoothing filter.

In a similar way, the second row of the matrix generates the quantity $d_1 = c_3x_1 - c_2x_2 + c_1x_3 - c_0x_4$, and the other even-numbered rows generate similar convolutions. Each d_i is called a *detail coefficient*, and together they are called a G filter. G is not a smoothing filter. In fact, the filter coefficients are chosen such that the G filter generates small values when the data items x_i are correlated. Together, H and G are called *quadrature mirror filters* (QMF).

The discrete wavelet transform of an image can therefore be viewed as passing the original image through a QMF that consists of a pair of lowpass (H) and highpass (G) filters.

If W is an $n \times n$ matrix, it generates $n/2$ smooth coefficients s_i and $n/2$ detail coefficients d_i. The transposed matrix is

$$W^T = \begin{pmatrix}
c_0 & c_3 & 0 & 0 & \dots & & & & c_2 & c_1 \\
c_1 & -c_2 & 0 & 0 & \dots & & & & c_3 & -c_0 \\
c_2 & c_1 & c_0 & c_3 & \dots & & & & 0 & 0 \\
c_3 & -c_0 & c_1 & -c_2 & \dots & & & & 0 & 0 \\
 & & & & \ddots & & & & & \\
 & & & & & c_2 & c_1 & c_0 & c_3 & 0 & 0 \\
 & & & & & c_3 & -c_0 & c_1 & -c_2 & 0 & 0 \\
 & & & & & & & c_2 & c_1 & c_0 & c_3 \\
 & & & & & & & c_3 & -c_0 & c_1 & -c_2
\end{pmatrix}.$$

It can be shown that in order for W to be orthonormal, the four coefficients have to satisfy the two relations $c_0^2 + c_1^2 + c_2^2 + c_3^2 = 1$ and $c_2c_0 + c_3c_1 = 0$. The other two equations used to calculate the four filter coefficients are $c_3 - c_2 + c_1 - c_0 = 0$ and $0c_3 - 1c_2 + 2c_1 - 3c_0 = 0$. They represent the vanishing of the first two moments of the sequence $(c_3, -c_2, c_1, -c_0)$. The solutions of these four equations are

$$\begin{aligned}
c_0 &= (1+\sqrt{3})/(4\sqrt{2}) \approx 0.48296, \quad & c_1 &= (3+\sqrt{3})/(4\sqrt{2}) \approx 0.8365, \\
c_2 &= (3-\sqrt{3})/(4\sqrt{2}) \approx 0.2241, \quad & c_3 &= (1-\sqrt{3})/(4\sqrt{2}) \approx -0.1294.
\end{aligned} \tag{4.12}$$

Using a transform matrix W is conceptually simple but not very practical, since W should be of the same size as the image, which can be large. However, a look at W shows that it is very regular, so there is really no need to construct the full matrix. It is enough to have just the top row of W. In fact, it is enough to have just an array with the filter coefficients. Figure 4.25 is Matlab code that performs this calculation. Function `fwt1(dat,coarse,filter)` takes a row vector `dat` of 2^n data items, and another array, `filter`, with filter coefficients. It then calculates the first `coarse` levels of the discrete wavelet transform.

```
function wc1=fwt1(dat,coarse,filter)
%  The 1D Forward Wavelet Transform
%  dat must be a 1D row vector of size 2^n,
%  coarse is the coarsest level of the transform
%  (note that coarse should be <<n)
%  filter is an orthonormal quadrature mirror filter
%  whose length should be <2^(coarse+1)
n=length(dat); j=log2(n); wc1=zeros(1,n);
beta=dat;
for i=j-1:-1:coarse
  alfa=HiPass(beta,filter);
  wc1((2^(i)+1):(2^(i+1)))=alfa;
  beta=LoPass(beta,filter) ;
end
wc1(1:(2^coarse))=beta;

function d=HiPass(dt,filter) % highpass downsampling
d=iconv(mirror(filter),lshift(dt));
% iconv is matlab convolution tool
n=length(d);
d=d(1:2:(n-1));

function d=LoPass(dt,filter) % lowpass downsampling
d=aconv(filter,dt);
% aconv is matlab convolution tool with time-
% reversal of filter
n=length(d);
d=d(1:2:(n-1));

function sgn=mirror(filt)
% return filter coefficients with alternating signs
sgn=-((-1).^(1:length(filt))).*filt;
```

A simple test of `fwt1` is

```
n=16; t=(1:n)./n;
dat=sin(2*pi*t)
filt=[0.4830 0.8365 0.2241 -0.1294];
wc=fwt1(dat,1,filt)
```

which outputs

```
dat=
0.3827  0.7071  0.9239 1.0000 0.9239 0.7071 0.3827 0
-0.3827 -0.7071 -0.9239 -1.0000 -0.9239 -0.7071 -0.3827 0
wc=
1.1365 -1.1365 -1.5685 1.5685 -0.2271 -0.4239 0.2271 0.4239
-0.0281 -0.0818 -0.0876 -0.0421 0.0281 0.0818 0.0876 0.0421
```

Figure 4.25: Code for the One-Dimensional Forward Discrete Wavelet Transform.