```
function dat=iwt1(wc,coarse,filter)
% Inverse Discrete Wavelet Transform
dat=wc(1:2^coarse);
n=length(wc); j=log2(n);
for i=coarse:j-1
 dat=ILoPass(dat,filter)+ ...
  IHiPass(wc((2^(i)+1):(2^(i+1))),filter);
end

function f=ILoPass(dt,filter)
f=iconv(filter,AltrntZro(dt));

function f=IHiPass(dt,filter)
f=aconv(mirror(filter),rshift(AltrntZro(dt)));

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

function f=AltrntZro(dt)
% returns a vector of length 2*n with zeros
% placed between consecutive values
n =length(dt)*2; f =zeros(1,n);
f(1:2:(n-1))=dt;
```

A simple test of iwt1 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)
rec=iwt1(wc,1,filt)
```

Figure 4.26: Code and Test for the One-Dimensional Inverse Discrete Wavelet Transform.

Figure 4.26 lists the Matlab code of the inverse one-dimensional discrete wavelet transform function `iwt1(wc,coarse,filter)` and includes a test.

Readers who take the trouble to read and understand functions `fwt1` and `iwt1` (Figures 4.25 and 4.27) may be interested in their two-dimensional equivalents, functions `fwt2` and `iwt2`, listed in Figures 4.28 and 4.29, respectively, with a simple test routine.

In addition to the Daubechies family of filters (by the way, the Haar wavelet can be considered the Daubechies filter of order 2) there are many other families of wavelets, each with its own properties. Some well-known filters are Beylkin, Coifman, Symmetric, and Vaidyanathan.

The Daubechies family of wavelets is a set of orthonormal, compactly supported functions where consecutive members are increasingly smoother. Section 4.8 discusses the Daubechies D4 wavelet and its building block. The term *compact support* means that these functions are zero (exactly zero, not just very small) outside a finite interval.

```
function dat=iwt1(wc,coarse,filter)
% Inverse Discrete Wavelet Transform
dat=wc(1:2^coarse);
n=length(wc); j=log2(n);
for i=coarse:j-1
 dat=ILoPass(dat,filter)+ ...
  IHiPass(wc((2^(i)+1):(2^(i+1))),filter);
end

function f=ILoPass(dt,filter)
f=iconv(filter,AltrntZro(dt));

function f=IHiPass(dt,filter)
f=aconv(mirror(filter),rshift(AltrntZro(dt)));

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

function f=AltrntZro(dt)
% returns a vector of length 2*n with zeros
% placed between consecutive values
n =length(dt)*2; f =zeros(1,n);
f(1:2:(n-1))=dt;
```

A simple test of iwt1 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)
rec=iwt1(wc,1,filt)
```

Figure 4.27: Code for the One-Dimensional Inverse Discrete Wavelet Transform.

```
function wc=fwt2(dat,coarse,filter)
% The 2D Forward Wavelet Transform
% dat must be a 2D matrix of size (2^n:2^n),
% "coarse" is the coarsest level of the transform
% (note that coarse should be <<n)
% filter is an orthonormal qmf of length<2^(coarse+1)
q=size(dat); n = q(1); j=log2(n);
if q(1)~=q(2), disp('Nonsquare image!'), end;
wc = dat; nc = n;
for i=j-1:-1:coarse,
 top = (nc/2+1):nc; bot = 1:(nc/2);
 for ic=1:nc,
  row = wc(ic,1:nc);
  wc(ic,bot)=LoPass(row,filter);
  wc(ic,top)=HiPass(row,filter);
 end
 for ir=1:nc,
  row = wc(1:nc,ir)';
  wc(top,ir)=HiPass(row,filter)';
  wc(bot,ir)=LoPass(row,filter)';
 end
nc = nc/2;
end

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 fwt2 and iwt2 is

```
filename='house128'; dim=128;
fid=fopen(filename,'r');
if fid==-1 disp('file not found')
 else img=fread(fid,[dim,dim])'; fclose(fid);
end
filt=[0.4830 0.8365 0.2241 -0.1294];
fwim=fwt2(img,4,filt);
figure(1), imagesc(fwim), axis off, axis square
rec=iwt2(fwim,4,filt);
figure(2), imagesc(rec), axis off, axis square
```

Figure 4.28: Code for the Two-Dimensional Forward Discrete Wavelet Transform.

```
function dat=iwt2(wc,coarse,filter)
% Inverse Discrete 2D Wavelet Transform
n=length(wc); j=log2(n);
dat=wc;
nc=2^(coarse+1);
for i=coarse:j-1,
 top=(nc/2+1):nc; bot=1:(nc/2); all=1:nc;
 for ic=1:nc,
  dat(all,ic)=ILoPass(dat(bot,ic)',filter)'  ...
   +IHiPass(dat(top,ic)',filter)';
 end % ic
 for ir=1:nc,
  dat(ir,all)=ILoPass(dat(ir,bot),filter)  ...
   +IHiPass(dat(ir,top),filter);
 end % ir
nc=2*nc;
end % i

function f=ILoPass(dt,filter)
f=iconv(filter,AltrntZro(dt));

function f=IHiPass(dt,filter)
f=aconv(mirror(filter),rshift(AltrntZro(dt)));

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

function f=AltrntZro(dt)
% returns a vector of length 2*n with zeros
% placed between consecutive values
n =length(dt)*2; f =zeros(1,n);
f(1:2:(n-1))=dt;
```

A simple test of `fwt2` and `iwt2` is

```
filename='house128'; dim=128;
fid=fopen(filename,'r');
if fid==-1 disp('file not found')
 else img=fread(fid,[dim,dim])'; fclose(fid);
end
filt=[0.4830 0.8365 0.2241 -0.1294];
fwim=fwt2(img,4,filt);
figure(1), imagesc(fwim), axis off, axis square
rec=iwt2(fwim,4,filt);
figure(2), imagesc(rec), axis off, axis square
```

Figure 4.29: Code for the Two-Dimensional Inverse Discrete Wavelet Transform.

The Daubechies D4 wavelet is based on four coefficients, shown in Equation (4.12). The D6 wavelet is, similarly, based on six coefficients. They are calculated by solving six equations, three of which represent orthogonality requirements and the other three, the vanishing of the first three moments. The result is listed in Equation (4.13):

$$\begin{aligned}
c_0 &= (1+\sqrt{10}+\sqrt{5+2\sqrt{10}})/(16\sqrt{2}) \approx .3326,\\
c_1 &= (5+\sqrt{10}+3\sqrt{5+2\sqrt{10}})/(16\sqrt{2}) \approx .8068,\\
c_2 &= (10-2\sqrt{10}+2\sqrt{5+2\sqrt{10}})/(16\sqrt{2}) \approx .4598,\\
c_3 &= (10-2\sqrt{10}-2\sqrt{5+2\sqrt{10}})/(16\sqrt{2}) \approx -.1350,\\
c_4 &= (5+\sqrt{10}-3\sqrt{5+2\sqrt{10}})/(16\sqrt{2}) \approx -.0854,\\
c_5 &= (1+\sqrt{10}-\sqrt{5+2\sqrt{10}})/(16\sqrt{2}) \approx .0352.
\end{aligned} \tag{4.13}$$

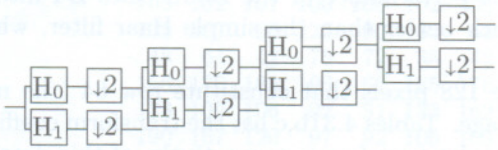
Each member of this family has two more coefficients than its predecessor and is smoother. The derivation of these functions is discussed in [Daubechies 88], [DeVore et al. 92], and [Vetterli and Kovacevic 95].

4.7 Examples

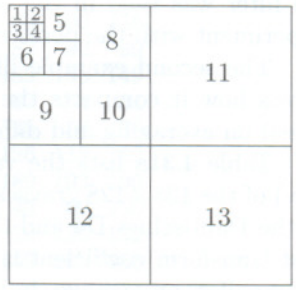
We already know that the discrete wavelet transform can reconstruct images from a small number of transform coefficients. The first example in this section illustrates an important property of the discrete wavelet transform, namely its ability to reconstruct images that degrade *gracefully*, without exhibiting any artifacts, when more and more transform coefficients are zeroed or are coarsely quantized. Other transforms, most notably the DCT, may introduce artifacts in the reconstructed image, but this property of the DWT makes it ideal for applications such as fingerprint compression [Salomon 00].

The example uses functions `fwt2` and `iwt2` of Figures 4.28 and 4.29 to blur an image. The idea is to compute the four-step subband transform of an image (thus ending up with 13 subbands), then set most of the transform coefficients to zero and heavily quantize some of the others. This, of course, results in a loss of image information and in a nonperfectly reconstructed image. The point is that the reconstructed image is *blurred* rather than being coarse or having artifacts.

Figure 4.30 shows the result of blurring the Lena image. Parts (a) and (b) show the logarithmic multiresolution tree and the subband structure, respectively. Part (c) shows the results of the quantization. The transform coefficients of subbands 5–7 have been divided by two, and all the coefficients of subbands 8–13 have been cleared. At first, most of the image in part (b) looks uniformly black (i.e., all zeros), but a careful examination shows many nonzero elements in subbands 5–10. We can say that the blurred image of part (d) has been reconstructed from the coefficients of subbands 1–4 (1/64th of the total number of transform coefficients) and half of the coefficients of subbands 5-7 (half of 3/64, or 3/128). On average, the image has been reconstructed from $5/128 \approx 0.039$ or 3.9% of the transform coefficients. Notice that the Daubechies



(a)



(b)

(c)

(d)

```
clear, colormap(gray);
filename='lena128'; dim=128;
fid=fopen(filename,'r');
img=fread(fid,[dim,dim])';
filt=[0.23037,0.71484,0.63088,-0.02798, ...
 -0.18703,0.03084,0.03288,-0.01059];
fwim=fwt2(img,3,filt);
figure(1), imagesc(fwim), axis square
 fwim(1:16,17:32)=fwim(1:16,17:32)/2;
 fwim(1:16,33:128)=0;
 fwim(17:32,1:32)=fwim(17:32,1:32)/2;
 fwim(17:32,33:128)=0;
 fwim(33:128,:)=0;
figure(2), colormap(gray), imagesc(fwim)
rec=iwt2(fwim,3,filt);
figure(3), colormap(gray), imagesc(rec)
```

Figure 4.30: Blurring as a Result of Coarse Quantization.

D8 filter was used in the calculations. Readers are encouraged to use this code and experiment with the performance of other filters.

The second example illustrates the performance of the Daubechies D4 filter and shows how it compacts the energy much better than the simple Haar filter, which is based on averaging and differencing.

Table 4.31a lists the values of the 128 pixels that constitute row 64 (the middle row) of the 128×128 grayscale Lena image. Tables 4.31b,c list the transform coefficients of the Daubechies D4 and the Haar wavelet transforms, respectively, of this data. The first transform coefficient is the same in both cases, but the remaining 127 coefficients are smaller, on average, in the Daubechies transform, which shows that this transform produces better energy compaction. The average of the absolute values of these 127 coefficients in the Daubechies D4 transform is 2.1790, whereas the corresponding average in the Haar transform is 9.8446, about 4.5 times greater.

Mathematica code for Table 4.31b and Matlab code for Table 4.31c are listed in Figure 4.32. Note that the former uses `WaveletTransform.m`, a *Mathematica* package by Alistair C. H. Rowe and Paul C. Abbott and available from [Alistair and Abbott 01].

4.8 The Daubechies Wavelets

Many useful mathematical functions are defined explicitly. A polynomial is perhaps the simplest such function. However, many other functions, not less useful, are defined recursively, in terms of themselves. Defining anything in terms of itself seems a contradiction, but the point is that a valid recursive definition must have several parts, and at least one part must be explicit. This part normally defines an initial value for whatever is defined. A simple example is the factorial function. It can be defined explicitly by

$$n! = n(n-1)(n-2)\cdots 3\cdot 2\cdot 1$$

but can also be defined recursively, by the 2-part definition

$$1! = 1, \quad n! = n\cdot(n-1)!.$$

Another interesting example is the exponential function e (or "exp"), which is defined by the differential recursive relation

$$e^0 = 1, \quad \frac{d\,e^x}{dx} = e^x.$$

Ingrid Daubechies has introduced a wavelet ψ and a scaling function (or building block) φ. One requirement was that the scaling function have finite support. It had to be zero outside a finite range. Daubechies selected the range $(0,3)$ to be the support of the function and has proved that this function cannot be expressed in terms of elementary functions such as polynomials, trigonometric, or exponential. She also showed that φ can be defined recursively, in terms of several initial values and a recursion relation. The initial values selected by her are

$$\varphi(0)=0, \quad \varphi(1)=\frac{1+\sqrt{3}}{2}, \quad \varphi(2)=\frac{1-\sqrt{3}}{2}, \quad \text{and} \quad \varphi(3)=0,$$

148	141	137	124	101	104	105	103	98	89	100	136
156	173	175	176	179	171	152	116	80	82	92	99
103	102	101	100	100	102	106	104	112	139	155	149
139	107	90	126	90	65	65	93	62	87	61	84
48	64	42	75	72	35	42	53	73	45	58	130
156	176	185	196	167	185	178	121	113	126	113	122
133	109	106	92	91	133	162	165	174	189	193	190
190	167	120	97	92	106	103	81	55	43	60	150
126	55	61	65	61	50	52	53	52	79	135	132
147	163	161	158	157	157	156	156	156	158	159	156
155	154	155	155	157	157	154	150				

(a)

117.95	−10.38	−5.99	−0.19	−11.64	12.6	−5.95	4.15
−2.57	6.61	−17.08	−0.50	7.88	−15.53	4.10	−10.80
−5.29	2.94	−0.63	5.42	−2.39	0.53	−5.96	2.67
−6.4	9.71	−5.43	0.56	−0.13	0.83	−0.02	1.17
−1.38	−2.68	1.92	3.14	−3.71	0.62	−0.02	−0.04
−1.41	−2.37	0.08	−1.62	−1.03	−3.50	2.52	2.81
−1.68	1.41	−1.79	1.11	3.55	−0.24	−7.44	0.28
−0.49	−2.56	1.98	−0.00	0.10	−0.17	0.42	0.65
0.35	−1.00	0.15	0.21	−1.30	0.31	0.21	0.45
0.85	−1.62	0.04	0.25	0	−0.10	0.23	−0.93
1.06	0.98	−2.43	0.35	−1.48	−1.72	−1.51	−1.54
−1.91	1.86	−0.67	1.95	−2.99	0.78	0.04	−1.55
2.42	−1.46	−0.64	1.47	0.23	−1.98	1.26	−0.32
0.42	0.95	−0.75	−1.02	1.01	−0.55	−3.45	3.31
−0.80	0.39	−0.11	−1.17	2.19	−0.25	0.25	−0.07
−0.03	−0.09	0.18	−0.02	0.02	0.06	0.08	0.19

(b)

117.95	−9.68	−16.44	1.31	−20.81	3.31	14.38	−29.44
−6.63	8.38	−20.56	39.38	10.44	−31.50	−14.25	1.13
7.75	22.13	4.25	−13.88	−24.38	21.50	24.00	9.25
0.13	11.38	−22.75	−28.88	0.38	−0.38	1.25	0.13
7.25	13.25	15.00	1.00	−1.75	11.00	−6.50	−25.75
−9.00	−5.00	6.50	35.00	4.75	3.50	21.50	−28.00
7.25	13.75	3.75	1.50	−6.50	−34.00	−10.75	−2.25
1.25	0.50	−0.50	−0.25	1.50	−0.25	−1.00	2.50
14.50	−9.00	3.50	28.50	4.00	−6.50	6.50	−4.50
−5.50	12.00	1.50	7.00	0.50	−21.00	−14.50	−1.50
−4.50	−7.50	−2.00	1.50	0	11.50	23.50	11.50
2.50	−7.00	1.50	11.00	13.00	6.00	−8.50	−45.00
12.00	35.50	−3.00	−2.00	2.00	5.50	−1.00	−0.50
0.50	−13.50	−28.00	1.50	−7.50	−8.00	1.00	1.50
0.50	0	0.50	0	0	−1.00	−0.50	1.50
0.50	0.50	−0.50	0	−1.00	0	1.50	2.00

(c)

Table 4.31: Daubechies and Haar Transforms of Middle Row in Lena Image.

```
<<WaveletTransform.m
(* Middle row of 128x128 Lena image *)
data={148,141,137,124,101,104,105,103, 98, 89,100,136
...
,155,154,155,155,157,157,154,150};
forward = Wavelet[data, Daubechies[4]]
NumberForm[forward,{6,2}]
inverse = InverseWavelet[forward,Daubechies[4]]
data == inverse
```

(a)

```
% Haar transform (averages & differences)
data=[148 141 137 124 101 104 105 103  98  89 100 136 ...
...
155 154 155 155 157 157 154 150];
n=128; ln=7; %log_2 n=7
for k=1:ln,
 for i=1:n/2,
  i1=2*i; j=n/2+i;
  newdat(i)=(data(i1-1)+data(i1))/2;
  newdat(j)=(data(j-1)-data(j))/2;
 end
 data=newdat; n=n/2;
end
round(100*data)/100
```

(b)

Figure 4.32: (a) *Mathematica* and (b) Matlab Codes for Table 4.31.

and the recursion relation is

$$\begin{aligned}\varphi(r) &= \frac{1+\sqrt{3}}{4}\varphi(2r) + \frac{3+\sqrt{3}}{4}\varphi(2r-1) + \frac{3-\sqrt{3}}{4}\varphi(2r-2) + \frac{1-\sqrt{3}}{4}\varphi(2r-3)\\ &= h_0\varphi(2r) + h_1\varphi(2r-1) + h_2\varphi(2r-2) + h_3\varphi(2r-3) \qquad (4.14)\\ &= (h_0,h_1,h_2,h_3)\cdot\big(\varphi(2r),\varphi(2r-1),\varphi(2r-2),\varphi(2r-3)\big).\end{aligned}$$

Notice that the initial values add up to 1:

$$\varphi(0)+\varphi(1)+\varphi(2)+\varphi(3) = 0 + \frac{1+\sqrt{3}}{2} + \frac{1-\sqrt{3}}{2} + 0 = 1.$$

Further computations of φ must be performed in steps. In step 1, the finite support requirement, the four initial values of φ and the recurrence relation [Equation (4.14)] are applied to compute the values of $\varphi(r)$ at the three points $r = 0.5$, 1.5, and 2.5.

$$\begin{aligned}
\varphi(1/2) &= h_0\varphi(2/2) + h_1\varphi(2/2-1) + h_2\varphi(2/2-2) + h_3\varphi(2/2-3)\\
&= \frac{1+\sqrt{3}}{4}\cdot\frac{1+\sqrt{3}}{2} + h_1\cdot 0 + h_2\cdot 0 + h_3\cdot 0\\
&= \frac{2+\sqrt{3}}{4},\\
\varphi(3/2) &= h_0\varphi(6/2) + h_1\varphi(6/2-1) + h_2\varphi(6/2-2) + h_3\varphi(6/2-3)\\
&= h_0\cdot 0 + \frac{1+\sqrt{3}}{4}\cdot\frac{1-\sqrt{3}}{2} + \frac{1-\sqrt{3}}{4}\cdot\frac{1+\sqrt{3}}{2} + h_3\cdot 0\\
&= 0,\\
\varphi(5/2) &= h_0\varphi(10/2) + h_1\varphi(10/2-1) + h_2\varphi(10/2-2) + h_3\varphi(10/2-3)\\
&= h_0\cdot 0 + h_1\cdot 0 + h_2\cdot 0 + \frac{1-\sqrt{3}}{2}\cdot\frac{1-\sqrt{3}}{2}\\
&= \frac{2-\sqrt{3}}{4}.
\end{aligned}$$

The values of φ are now known at the four initial points 0, 1, 2, and 3 and at the three additional points 0.5, 1.5, and 2.5 midway between them, a total of seven points. In step 2, six more values are computed at the six points 1/4, 3/4, 5/4, 7/4, 9/4, and 11/4. The computations are similar and the results are

$$\frac{5+3\sqrt{3}}{16},\quad \frac{9+5\sqrt{3}}{16},\quad \frac{1+\sqrt{3}}{8},\quad \frac{1-\sqrt{3}}{8},\quad \frac{9-5\sqrt{3}}{16},\quad \frac{5-3\sqrt{3}}{16}.$$

The values of φ are now known at $4+3+6=13$ points (Figure 4.33).

Step 3 computes the 12 values midway between these 13 points, resulting in $12 + 13 = 25$ values. Further steps compute 24, 48, 96, and so on, values. After n steps, the values of φ are known at $4+3+6+12+24+\cdots+3\cdot 2^n = 4+3(2^{n+1}-1)$ points. After nine steps, $4+3(2^{10}-1) = 3073$ values are known (Figure 4.34).

Function φ serves as a building block for the construction of the Daubechies wavelet ψ, which is defined recursively by

$$\begin{aligned}
\psi(r) &= -\frac{1+\sqrt{3}}{4}\varphi(2r-1) + \frac{3+\sqrt{3}}{4}\varphi(2r) - \frac{3-\sqrt{3}}{4}\varphi(2r+1) + \frac{1-\sqrt{3}}{4}\varphi(2r+2)\\
&= -h_0\varphi(2r-1) + h_1\varphi(2r) - h_2\varphi(2r+1) + h_3\varphi(2r+2).
\end{aligned}$$

Recall that φ is nonzero only in the interval $(0,3)$. The definition above implies that $\psi(r)$ is nonzero in the interval $(-1,2)$. This definition is also the basis for the recursive calculation of ψ, similar to that of φ. Figure 4.35 shows the values of the wavelet at 3073 points. A glance at Figures 4.34 and 4.35 also explains (albeit very late in this chapter) the reason for the term "wavelet."

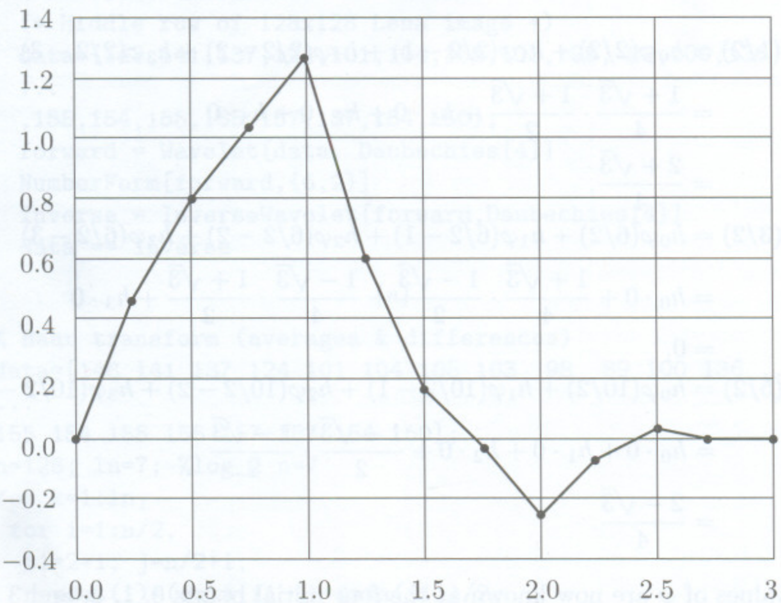


Figure 4.33: The Daubechies Scaling Function φ at 13 Points.

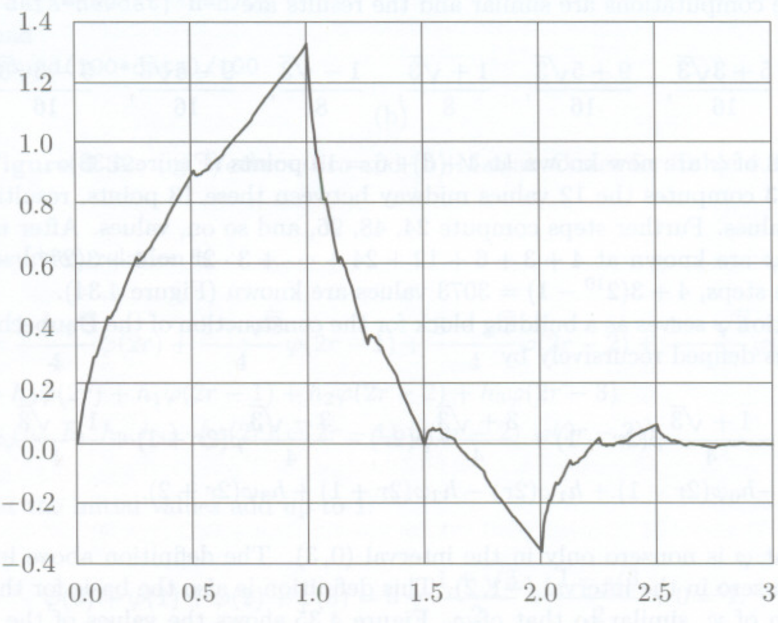


Figure 4.34: The Daubechies Scaling Function φ at 3073 Points.

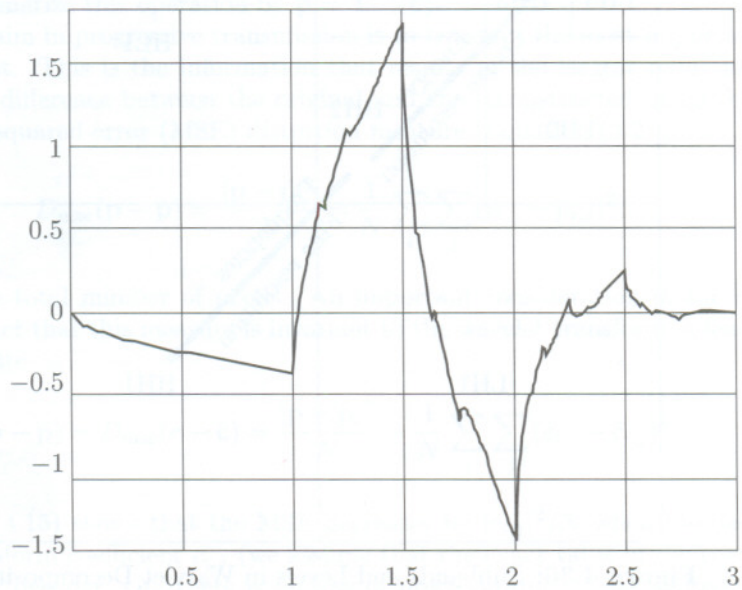


Figure 4.35: The Daubechies Wavelet ψ at 3073 Points.

4.9 SPIHT

SPIHT is an image compression method, but it is included in this chapter because it uses the wavelet transform as one of its compression steps and because its main data structure, the spatial orientation tree, uses the fact (mentioned on page 172) that the various subbands reflect the geometrical artifacts of the image.

Section 4.2 shows how the Haar transform can be applied several times to an image, creating regions (or subbands) of averages and details. The Haar transform is simple, and better compression can be achieved by other wavelet filters that produce better energy compaction. It seems that different wavelet filters produce different results depending on the image type, but it is currently not clear what filter is the best for any given image type. Regardless of the particular filter used, the image is decomposed into subbands such that lower subbands correspond to higher image frequencies and higher subbands correspond to lower image frequencies, where most of the image energy is concentrated (Figure 4.36). This is why we can expect the detail coefficients to get smaller as we move from high to low levels. Also, there are spatial similarities among the subbands (Figure 4.7b). An image part, such as an edge, occupies the same spatial position in each subband. These features of the wavelet decomposition are exploited by the SPIHT (set partitioning in hierarchical trees) method [Said and Pearlman 96].

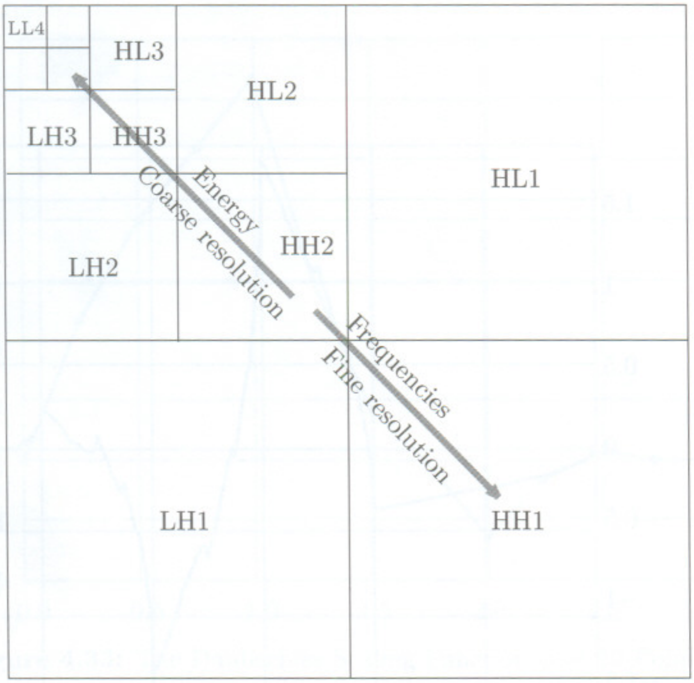


Figure 4.36: Subbands and Levels in Wavelet Decomposition.

SPIHT was designed for optimal progressive transmission, as well as for compression. One of the important features of SPIHT (perhaps a unique feature) is that at any point during the decoding of an image, the quality of the displayed image is the best that can be achieved for the number of bits input by the decoder up to that moment.

Another important SPIHT feature is its use of embedded coding. This feature is defined as follows: If an (embedded coding) encoder produces two files, a large one of size M and a small one of size m, then the smaller file is identical to the first m bits of the larger file.

The following example aptly illustrates the meaning of this definition. Suppose that three users wait for a certain compressed image to be sent them, but they need different image qualities. The first one needs the quality contained in a 10 KB file. The image qualities required by the second and third users are contained in files of sizes 20 KB and 50 KB, respectively. Most lossy image compression methods would have to compress the same image three times, at different qualities, to generate three files with the right sizes. SPIHT, on the other hand, produces one file and then three chunks—of lengths 10 KB, 20 KB, and 50 KB, all starting at the beginning of the file—that can be sent to the three users, thereby satisfying their needs.

We start with a general description of SPIHT. We denote the pixels of the original image $\mathbf{p}$ by $p_{i,j}$. Any set $\mathbf{T}$ of wavelet filters can be used to transform the pixels to wavelet coefficients (or transform coefficients) $c_{i,j}$. These coefficients constitute the