```
clear; % main program
filename='lena128'; dim=128;
fid=fopen(filename,'r');
if fid==-1 disp('file not found')
else img=fread(fid,[dim,dim])'; fclose(fid);
end
thresh=0.0;        % percent of transform coefficients deleted
figure(1), imagesc(img), colormap(gray), axis off, axis square
w=harmatt(dim);  % compute the Haar dim x dim transform matrix
timg=w*img*w';   % forward Haar transform
tsort=sort(abs(timg(:)));
tthresh=tsort(floor(max(thresh*dim*dim,1)));
cim=timg.*(abs(timg) > tthresh);
[i,j,s]=find(cim);
dimg=sparse(i,j,s,dim,dim);
% figure(2) displays the remaining transform coefficients
%figure(2), spy(dimg), colormap(gray), axis square
figure(2), image(dimg), colormap(gray), axis square
cimg=full(w'*sparse(dimg)*w);      % inverse Haar transform
density = nnz(dimg);
disp([num2str(100*thresh) '% of smallest coefficients deleted.'])
disp([num2str(density) ' coefficients remain out of ' ...
 num2str(dim) 'x' num2str(dim) '.'])
figure(3), imagesc(cimg), colormap(gray), axis off, axis square

File harmatt.m with two functions

function x = harmatt(dim)
num=log2(dim);
p = sparse(eye(dim)); q = p;
i=1;
while i<=dim/2;
 q(1:2*i,1:2*i) = sparse(individ(2*i));
 p=p*q; i=2*i;
end
x=sparse(p);

function f=individ(n)
x=[1, 1]/sqrt(2);
y=[1,-1]/sqrt(2);
while min(size(x)) < n/2
 x=[x, zeros(min(size(x)),max(size(x)));...
   zeros(min(size(x)),max(size(x))), x];
end
while min(size(y)) < n/2
 y=[y, zeros(min(size(y)),max(size(y)));...
   zeros(min(size(y)),max(size(y))), y];
end
f=[x;y];
```

Figure 4.12: Matlab Code for the Haar Transform of an Image.

and parameter "`dim`" should be a power of 2. The assignment "`thresh=`" specifies the percentage of transform coefficients to be deleted. This provides an easy way to experiment with lossy wavelet image compression.

File "`harmatt.m`" contains two functions that compute the Haar wavelet coefficients in a matrix form (Section 4.2.1).

(A technical note: A Matlab `m` file may include either commands or a function but not both. It may, however, contain more than one function, provided that only the top function is invoked from outside the file. All the other functions must be called from within the file. In our case, function `harmatt(dim)` calls function `individ(n)`.)

Example: The code of Figure 4.12 is used to compute the Haar transform of the well-known "Lena" image. The image is then reconstructed three times by discarding more and more detail coefficients. Figure 4.13 shows the results of reconstructing the original image from 3277, 1639, and 820 coefficients, respectively. Despite the heavy loss of wavelet coefficients, only a very small loss of image quality is noticeable. The number of wavelet coefficients is, of course, the same as the image resolution $128 \times 128 = 16,384$. Using 820 out of 16,384 coefficients corresponds to discarding 95% of the smallest of the transform coefficients (notice, however, that some of the coefficients were originally zero, so the actual loss may amount to less than 95%).

4.2 The Haar Transform

The Haar transform uses a scale function $\phi(t)$ and a wavelet $\psi(t)$, both shown in Figure 4.14a, to represent a large number of functions $f(t)$. The representation is the infinite sum

$$f(t) = \sum_{k=-\infty}^{\infty} c_k \phi(t-k) + \sum_{k=-\infty}^{\infty} \sum_{j=0}^{\infty} d_{j,k} \psi(2^j t - k),$$

where c_k and $d_{j,k}$ are coefficients to be calculated.

The basic scale function $\phi(t)$ is the unit pulse

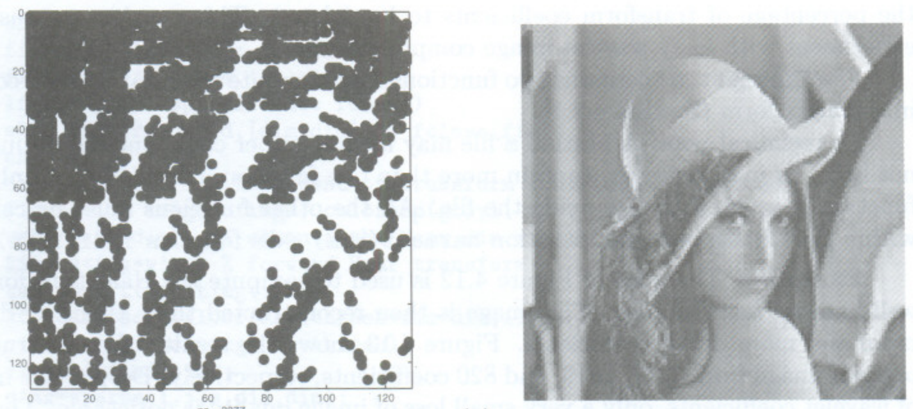
$$\phi(t) = \begin{cases} 1, & 0 \le t < 1, \\ 0, & \text{otherwise.} \end{cases}$$

The function $\phi(t-k)$ is a copy of $\phi(t)$, shifted k units to the right. Similarly, $\phi(2t-k)$ is a copy of $\phi(t-k)$ scaled to half the width of $\phi(t-k)$. The shifted copies are used to approximate $f(t)$ at different times t. The scaled copies are used to approximate $f(t)$ at higher resolutions. Figure 4.14b shows the functions $\phi(2^j t - k)$ for $j = 0, 1, 2,$ and 3 and for $k = 0, 1, \ldots, 7$.

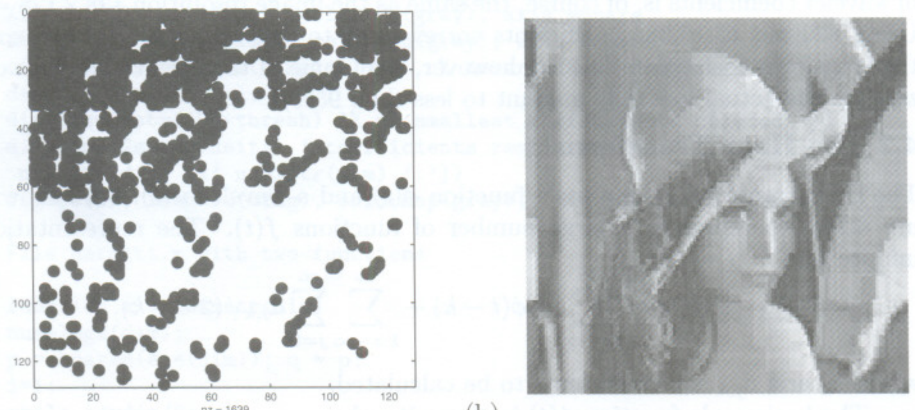
The basic Haar wavelet is the step function

$$\psi(t) = \begin{cases} 1, & 0 \le t < 0.5, \\ -1, & 0.5 \le t < 1. \end{cases}$$

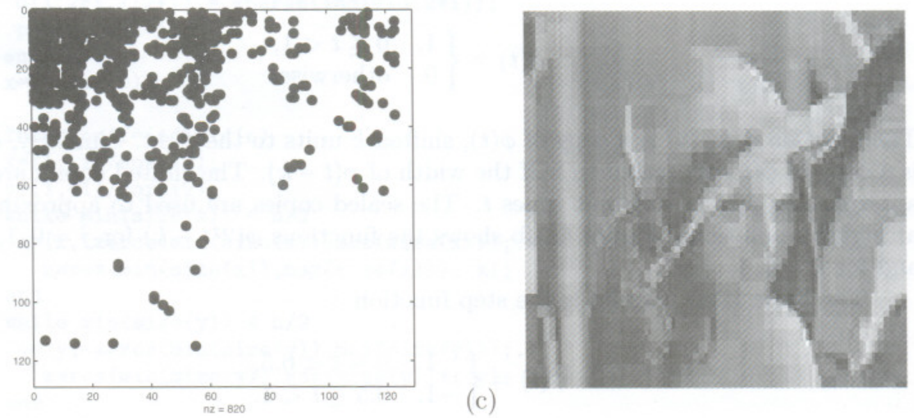
From this we can see that the general Haar wavelet $\psi(2^j t - k)$ is a copy of $\psi(t)$ shifted k units to the right and scaled such that its total width is $1/2^j$. The four Haar wavelets $\psi(2^2 t - k)$ for $k = 0, 1, 2,$ and 3 are shown in Figure 4.14c.



(a)



(b)



(c)

Figure 4.13: Three Lossy Reconstructions of the 128×128 Lena Image.

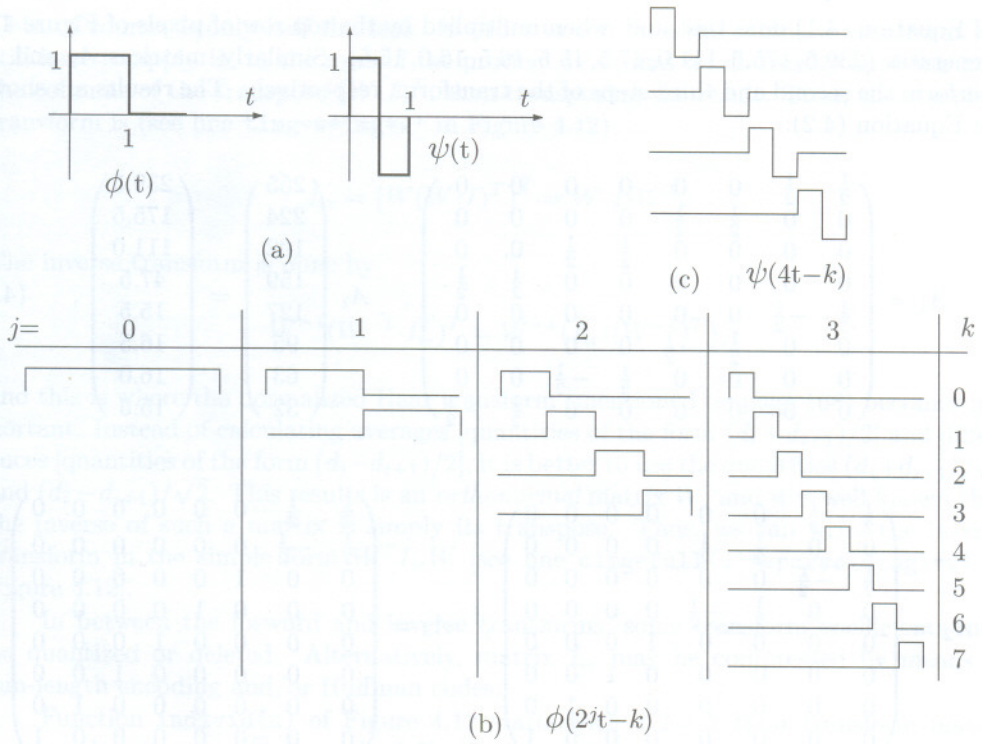


Figure 4.14: The Haar Basis Scale and Wavelet Functions.

Both $\phi(2^j t-k)$ and $\psi(2^j t-k)$ are nonzero in an interval of width $1/2^j$. This interval is their *support*. Since this interval tends to be short, we say that these functions have *compact support*.

We illustrate the basic transform on the simple step function

$$f(t) = \begin{cases} 5, & 0 \le t < 0.5, \\ 3, & 0.5 \le t < 1. \end{cases}$$

It is easy to see that $f(t) = 4\phi(t) + \psi(t)$. We say that the original steps $(5,3)$ have been transformed to the (low resolution) average 4 and the (high resolution) detail -1. Using matrix notation, this can be expressed as $(5,3)\mathbf{A}_2 = (4,-1)$, where $\mathbf{A}_2$ is the order-2 Haar transform matrix of Equation (3.16).

4.2.1 A Matrix Approach

The principle of the Haar transform is to calculate averages and differences. It turns out that this can be done by means of matrix multiplication ([Mulcahy 96] and [Mulcahy 97]). As an example, we select the top row of the simple 8×8 image of Figure 4.8. Anyone with a little experience with matrices can construct a matrix that when multiplied by this vector creates a vector with four averages and four differences. Matrix A_1

of Equation (4.1) does that and, when multiplied by the top row of pixels of Figure 4.8, generates $(239.5, 175.5, 111.0, 47.5, 15.5, 16.5, 16.0, 15.5)$. Similarly, matrices A_2 and A_3 perform the second and third steps of the transform, respectively. The results are shown in Equation (4.2):

$$A_1 = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & -\frac{1}{2} \end{pmatrix}, \quad A_1 \begin{pmatrix} 255 \\ 224 \\ 192 \\ 159 \\ 127 \\ 95 \\ 63 \\ 32 \end{pmatrix} = \begin{pmatrix} 239.5 \\ 175.5 \\ 111.0 \\ 47.5 \\ 15.5 \\ 16.5 \\ 16.0 \\ 15.5 \end{pmatrix}, \tag{4.1}$$

$$A_2 = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad A_3 = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

$$A_2 \begin{pmatrix} 239.5 \\ 175.5 \\ 111.0 \\ 47.5 \\ 15.5 \\ 16.5 \\ 16.0 \\ 15.5 \end{pmatrix} = \begin{pmatrix} 207.5 \\ 79.25 \\ 32.0 \\ 31.75 \\ 15.5 \\ 16.5 \\ 16.0 \\ 15.5 \end{pmatrix}, \quad A_3 \begin{pmatrix} 207.5 \\ 79.25 \\ 32.0 \\ 31.75 \\ 15.5 \\ 16.5 \\ 16.0 \\ 15.5 \end{pmatrix} = \begin{pmatrix} 143.375 \\ 64.125 \\ 32. \\ 31.75 \\ 15.5 \\ 16.5 \\ 16. \\ 15.5 \end{pmatrix}. \tag{4.2}$$

Instead of calculating averages and differences of image rows, all we have to do is construct matrices A_1, A_2, and A_3, multiply them to get $W = A_1A_2A_3$, and apply W to all the rows of image I by multiplying $W \cdot I$:

$$W \begin{pmatrix} 255 \\ 224 \\ 192 \\ 159 \\ 127 \\ 95 \\ 63 \\ 32 \end{pmatrix} = \begin{pmatrix} \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} \\ \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{-1}{8} & \frac{-1}{8} & \frac{-1}{8} & \frac{-1}{8} \\ \frac{1}{4} & \frac{1}{4} & \frac{-1}{4} & \frac{-1}{4} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{-1}{4} & \frac{-1}{4} \\ \frac{1}{2} & \frac{-1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{-1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{-1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{-1}{2} \end{pmatrix} \begin{pmatrix} 255 \\ 224 \\ 192 \\ 159 \\ 127 \\ 95 \\ 63 \\ 32 \end{pmatrix} = \begin{pmatrix} 143.375 \\ 64.125 \\ 32 \\ 31.75 \\ 15.5 \\ 16.5 \\ 16 \\ 15.5 \end{pmatrix}.$$

This, of course, is only half the task. In order to compute the complete transform, we still have to apply W to the rows of the product $W\cdot I$, and we do this by applying it to the columns of the transpose $(W\cdot I)^T$, then transposing the result. Thus, the complete transform is (see line `timg=w*img*w'` in Figure 4.12)

$$I_{\rm tr} = \left(W(W\cdot I)^T\right)^T = W\cdot I\cdot W^T.$$

The inverse transform is done by

$$W^{-1}(W^{-1}\cdot I_{\rm tr}^T)^T = W^{-1}\left(I_{\rm tr}\cdot (W^{-1})^T\right),$$

and this is where the normalized Haar transform (mentioned on page 168) becomes important. Instead of calculating averages [quantities of the form $(d_i+d_{i+1})/2$] and differences [quantities of the form $(d_i-d_{i+1})/2$], it is better to use the quantities $(d_i+d_{i+1})/\sqrt{2}$ and $(d_i-d_{i+1})/\sqrt{2}$. This results is an *orthonormal* matrix W, and it is well known that the inverse of such a matrix is simply its transpose. Thus, we can write the inverse transform in the simple form $W^T I_{\rm tr} W$ [see line `cimg=full(w'*sparse(dimg)*w)` in Figure 4.12].

In between the forward and inverse transforms, some transform coefficients may be quantized or deleted. Alternatively, matrix $I_{\rm tr}$ may be compressed by means of run-length encoding and/or Huffman codes.

Function `individ(n)` of Figure 4.12 starts with a 2×2 Haar transform matrix (notice that it uses $\sqrt{2}$ instead of 2), then uses it to construct as many individual matrices A_i as necessary. Function `harmatt(dim)` combines those individual matrices to form the final Haar matrix for an image of `dim` rows and `dim` columns.

Example: The Matlab code of Figure 4.15 calculates W as the product of the three matrices A_1, A_2, and A_3, then transforms the 8×8 image of Figure 4.8 by computing the product $W\cdot I\cdot W^T$. The result is an 8×8 matrix of transform coefficients whose top left value, 131.375, is the average of all 64 image pixels.

4.3 Subband Transforms

The transforms discussed in Section 3.5 are *orthogonal* because each is based on an orthogonal matrix. An orthogonal transform can also be expressed as an *inner product* of the data (pixel values or audio samples) with a set of *basis functions*. The result of an orthogonal transform is a set of transform coefficients that can be compressed with RLE, Huffman coding, or other methods. Lossy compression is obtained when the transform coefficients are quantized before being compressed.

The discrete inner product of the two vectors f_i and g_i is defined by

$$\langle f, g\rangle = \sum_i f_i\, g_i,$$

and Section 3.5.1 starts with a transform of the form $c_i = \sum_j d_j w_{ij}$, where d_j are the data items and w_{ij} are certain weights.

```
a1=[1/2 1/2 0 0 0 0 0 0; 0 0 1/2 1/2 0 0 0 0;
 0 0 0 0 1/2 1/2 0 0; 0 0 0 0 0 0 1/2 1/2;
 1/2 -1/2 0 0 0 0 0 0; 0 0 1/2 -1/2 0 0 0 0;
 0 0 0 0 1/2 -1/2 0 0; 0 0 0 0 0 0 1/2 -1/2];
% a1*[255; 224; 192; 159; 127; 95; 63; 32];
a2=[1/2 1/2 0 0 0 0 0 0; 0 0 1/2 1/2 0 0 0 0;
 1/2 -1/2 0 0 0 0 0 0; 0 0 1/2 -1/2 0 0 0 0;
 0 0 0 0 1 0 0 0; 0 0 0 0 0 1 0 0;
 0 0 0 0 0 0 1 0; 0 0 0 0 0 0 0 1];
a3=[1/2 1/2 0 0 0 0 0 0; 1/2 -1/2 0 0 0 0 0 0;
 0 0 1 0 0 0 0 0; 0 0 0 1 0 0 0 0;
 0 0 0 0 1 0 0 0; 0 0 0 0 0 1 0 0;
 0 0 0 0 0 0 1 0; 0 0 0 0 0 0 0 1];
w=a3*a2*a1;
dim=8; fid=fopen('8x8','r');
img=fread(fid,[dim,dim])'; fclose(fid);
w*img*w' % Result of the transform
```

```
131.375   4.250   −7.875   −0.125   −0.25   −15.5    0   −0.25
      0       0        0        0       0       0    0       0
      0       0        0        0       0       0    0       0
      0       0        0        0       0       0    0       0
 12.000  59.875   39.875   31.875   15.75    32.0   16   15.75
 12.000  59.875   39.875   31.875   15.75    32.0   16   15.75
 12.000  59.875   39.875   31.875   15.75    32.0   16   15.75
 12.000  59.875   39.875   31.875   15.75    32.0   16   15.75
```

Figure 4.15: Code and Results for the Matrix Wavelet Transform $W\cdot I\cdot W^T$.

The wavelet transform, on the other hand, is a *subband transform*. It is done by computing a *convolution* of the data items (pixel values or audio samples) with a set of *bandpass filters*. Each resulting subband encodes a particular portion of the frequency content of the data.

The word "convolution" means coiling together. The discrete convolution of the two vectors f_i and g_i is denoted by $f \star g$. Each element $(f \star g)_i$ of the convolution is defined by

$$(f \star g)_i = \sum_j f_j \, g_{i-j}. \tag{4.3}$$

(Convolution is also defined for functions, but the field of data compression deals with discrete quantities, so only the discrete convolution is discussed here.) Notice that the limits of the sum above have not been stated explicitly. They depend on the sizes of vectors f and g, and Equation (4.9) is an example.

The remainder of this section discusses linear systems and why convolution is defined in this peculiar way. This material can be skipped by nonmathematical readers.

We start with the simple, intuitive concept of a *system*. This is anything that receives input and generates output in response. The input and output can be one-dimensional (a function of the time) two-dimensional (a function of two spatial variables), or it can have any number of dimensions. We will be concerned with the relation of the output to the input, not with the internal operation of the system. We will also concentrate on *linear systems*, since they are both simple and important. A linear system is defined as follows: If input $x_1(t)$ produces output $y_1(t)$ [we denote this by $x_1(t) \to y_1(t)$] and if $x_2(t) \to y_2(t)$, then $x_1(t)+x_2(t) \to y_1(t)+y_2(t)$. Any system that does not satisfy this condition is considered nonlinear.

This definition implies that $2x_1(t) = x_1(t) + x_1(t) \to y_1(t) + y_1(t) = 2y_1(t)$ or, in general, that $a\,x_1(t) \to a\,y_1(t)$ for any real a.

Some linear systems are *shift invariant*. If such a linear system satisfies $x(t) \to y(t)$, then $x(t-T) \to y(t-T)$; i.e., shifting the input by an amount T shifts the output by the same amount but does not otherwise affect the output. In the discussion of convolution, we assume that the systems in question are linear and shift invariant. This is true (or true to a very good approximation) for electrical networks and optical systems, the main pieces of hardware used in image processing and compression.

It is useful to have a general relation between the input and output of a linear, shift-invariant system. It turns out that the expression

$$y(t) = \int_{-\infty}^{+\infty} f(t,\tau)x(\tau)\,d\tau \tag{4.4}$$

is general enough for this purpose. In other words, there is always a two-parameter function $f(t,\tau)$ that can be used to predict the output $y(t)$ if the input $x(\tau)$ is known. However, we want to express this relation with a one-parameter function, and we use the shift invariance of the system for this purpose. For a linear, shift-invariant system we can write

$$y(t-T) = \int_{-\infty}^{+\infty} f(t,\tau)x(\tau - T)\,d\tau.$$

If we change variables by adding T to both t and τ, we get

$$y(t) = \int_{-\infty}^{+\infty} f(t+T,\tau+T)x(\tau)\,d\tau. \tag{4.5}$$

Comparing Equations (4.4) and (4.5) shows that $f(t,\tau) = f(t+T,\tau+T)$. Thus, function f has the property that if we add T to both its parameters, it does not change. The function is constant as long as the difference between its parameters is constant. Function f depends only on the difference of its parameters, so it is essentially a single parameter function. We can therefore write $g(t-\tau) = f(t,\tau)$, which changes Equation (4.4) to

$$y(t) = \int_{-\infty}^{+\infty} g(t-\tau)x(\tau)\,d\tau. \tag{4.6}$$

This is the *convolution integral*, an important relation between $x(t)$ and $y(t)$ or between $x(t)$ and $g(t)$. This relation is denoted by $y = g \star x$ and it says that the output y of a linear, shift-invariant system is given by the convolution of its input x with a certain function $g(t)$ (or by *convolving* x with g). Function g, which is characteristic of the system, is called the *impulse response* of the system. Figure 4.16 shows a graphical description of a convolution, where the final result (the integral) is the gray area under the curve.

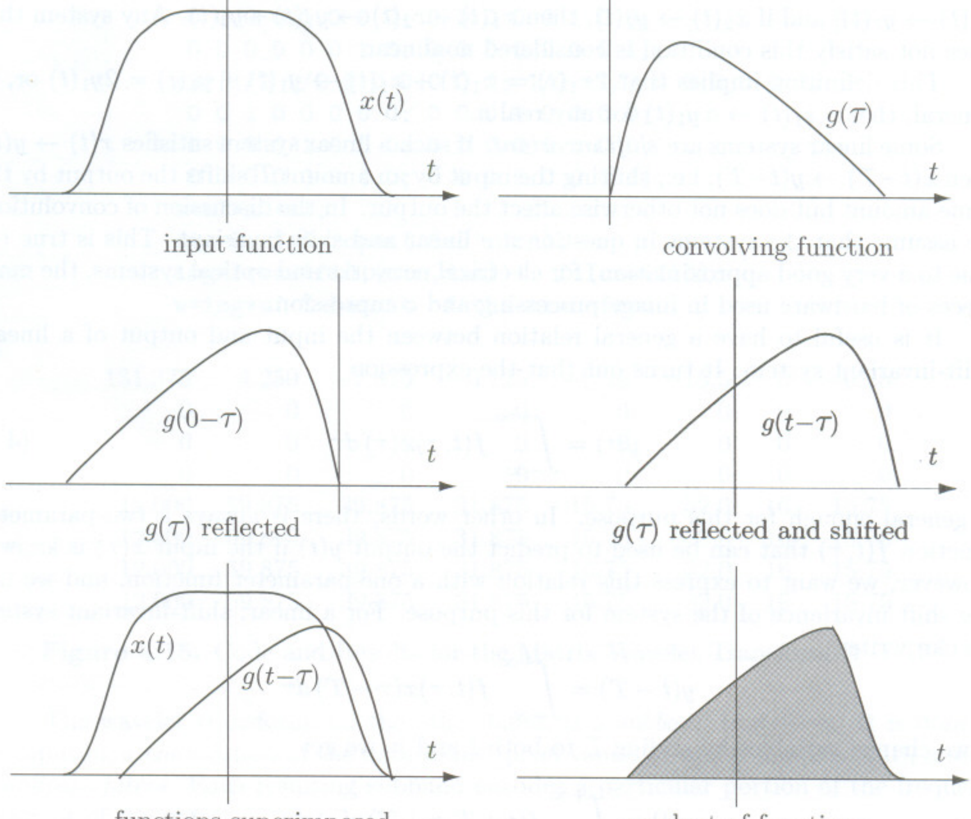


Figure 4.16: The Convolution of $x(t)$ and $g(t)$.

The convolution has a number of important properties. It is commutative, associative, and distributive over addition. These properties are listed in Equation (4.7)

$$\begin{aligned} f\star g &= g\star f,\\ f\star(g\star h) &= (f\star g)\star h, \qquad (4.7)\\ f\star(g+h) &= f\star g+f\star h. \end{aligned}$$

Practical problems normally involve discrete sequences of numbers, rather than continuous functions, so the *discrete convolution* is useful. The discrete convolution of the two sequences $f(i)$ and $g(i)$ is defined as

$$h(i) = f(i) \star g(i) = \sum_j f(j)\, g(i-j). \tag{4.8}$$

If the lengths of $f(i)$ and $g(i)$ are m and n, respectively, then $h(i)$ has length $m+n-1$.

Example: Given the two sequences $f = \bigl(f(0), f(1), \ldots, f(5)\bigr)$ (six elements) and $g = \bigl(g(0), g(1), \ldots, g(4)\bigr)$ (five elements), Equation (4.8) yields the ten elements of the convolution $h = f \star g$:

$$\begin{aligned}
h(0) &= \sum_{j=0}^{0} f(j)g(0-j) = f(0)g(0)\\
h(1) &= \sum_{j=0}^{1} f(j)g(1-j) = f(0)g(1) + f(1)g(0)\\
h(2) &= \sum_{j=0}^{2} f(j)g(2-j) = f(0)g(2) + f(1)g(1) + f(2)g(0)\\
h(3) &= \sum_{j=0}^{3} f(j)g(3-j) = f(0)g(3) + f(1)g(2) + f(2)g(1) + f(3)g(0)\\
h(4) &= \sum_{j=0}^{4} f(j)g(4-j) = f(0)g(4) + f(1)g(3) + f(2)g(2) + f(3)g(1) + f(4)g(0)\\
h(5) &= \sum_{j=1}^{5} f(j)g(5-j) = f(1)g(4) + f(2)g(3) + f(3)g(2) + f(4)g(1) + f(5)g(0)\\
h(6) &= \sum_{j=2}^{5} f(j)g(6-j) = f(2)g(4) + f(3)g(3) + f(4)g(2) + f(5)g(1)\\
h(7) &= \sum_{j=3}^{5} f(j)g(7-j) = f(3)g(4) + f(4)g(3) + f(5)g(2)\\
h(8) &= \sum_{j=4}^{5} f(j)g(8-j) = f(4)g(4) + f(5)g(3)\\
h(9) &= \sum_{j=5}^{5} f(j)g(9-j) = f(5)g(4)
\end{aligned} \tag{4.9}$$

A simple example of the use of a convolution is smoothing (or denoising). This shows how convolution can be used as a filter. Given a noisy function $f(t)$ (Figure 4.17), we select a rectangular pulse as the convolving function $g(t)$. It is defined as

$$g(t) = \begin{cases} 1, & -a/2 < t < a/2, \\ \frac{1}{2}, & t = \pm a/2, \\ 0, & \text{elsewhere}, \end{cases}$$

where a is a suitably small value (typically 1, but it could be anything). As the convolution proceeds, the pulse is moved from left to right and is multiplied by $f(t)$. The result of the product is a local average of $f(t)$ over an interval of width a. This has the effect of suppressing the high frequency fluctuations of $f(t)$.

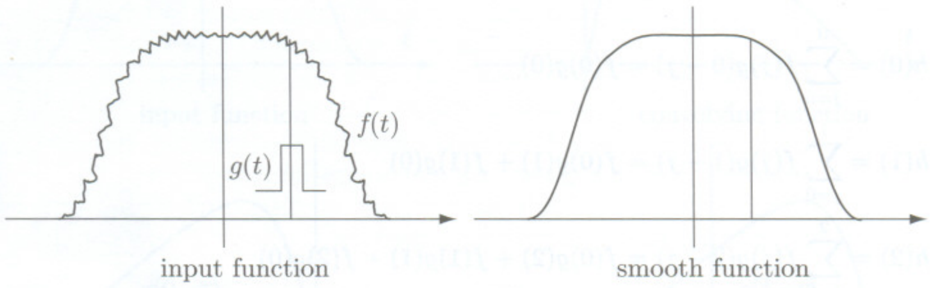


Figure 4.17: Applying Convolution to Denoising a Function.

"Oh no," George said. "It was more than money."

He leaned his forehead in his hand and tried to remember what else more than money. The darkness inside his head was full of convolutions. His eardrums were too tight. Only the higher registers of sound were getting through.

—Paul Scott, *The Bender*

4.4 Filter Banks

The matrix approach to the Haar transform is used in this section to introduce the idea of *filter banks* [Strang and Nguyen 96]. We show how the Haar transform can be interpreted as a bank of two filters, a lowpass and a highpass. We explain the terms "filter," "lowpass," and "highpass" and show how the idea of filter banks leads naturally to the concept of subband transform [Simoncelli et al. 90] The Haar transform, of course, is the simplest wavelet transform, so it is used here to illustrate the new concepts. However, using it as a filter bank may not be very efficient. Most practical applications of wavelet filters use more sophisticated sets of filter coefficients, but they are all based on the concept of filters and filter banks.

A *filter* is a linear operator defined in terms of its *filter coefficients* $h(0)$, $h(1)$, $h(2), \ldots$. The filter coefficients can be applied to an input vector x to produce an output vector y according to

$$y(n) = \sum_k h(k)x(n-k) = h \star x.$$