Chapter 10

Advanced Topics in Signal Processing

Overview

This chapter presents three topics often not included in basic discrete-time signal processing textbooks; (1) basic of image processing; (2) discrete-time wavelets; and (3) basic compressed sensing. Yet these topics are accessible to students who have the background presented in Chapters 7 through 9.

Image processing is applying techniques such as filtering and denoising to images, which are two-dimensional signals. An image is an array of numbers, called *pixels*, as a signal is a vector of numbers. Linear time-invariance, frequency response, filtering, impulse response, convolution, deconvolution, DTFT and DFT all generalize directly from 1-D to 2-D. But causality, difference equations and poles and zeros do not generalize in a useful way. We apply filtering, denoising and deconvolution to images and introduce *Wiener* filters as *Tikhonov regularization* minimizing a cost functional. The concept of minimizing a cost functional will prove useful when compressed sensing is introduced in a later section.

The discrete-time *wavelet transform* is usually treated, if at all, using *filter banks*. While wavelets were historically developed from filter banks, that approach does not show why they perform denoising and compression so much better than Fourier-based methods. Here, we derive *Daubechies* discrete-time wavelets as an orthonormalization of *piecewise-polynomial* functions, which are useful for modelling signals that are mostly smooth but have occasional sharp changes. Wavelets are especially useful for compressing and denoising images, so we present them after presenting image processing. Wavelets became an important part of signal processing in the 1990's.

Compressed sensing makes use of the fact that many signals and images of interest can be represented using a sparse (mostly zero-values) linear combination of wavelet functions. This means that a signal or image can be reconstructed from a set of linear combinations of it much smaller in size than the signal or image itself. This is useful for deconvolution, reconstruction of a signal or image from a relatively small number of 2-D DFT values, and restoring missing pixels. We present an introduction to this topic, showing the reader how to make use of it without getting into the details of why it works so well (this is still an active area of research). Compressed sensing requires wavelets, so we present it after wavelets. Compressed sensing became an important part of signal processing in the 2000's.

Of course, these topics are all presented in more detail in graduate-level textbooks and courses.

10.1 Image Processing Basics

10.1.1 Preliminaries

Generalizing 1-D to 2-D

An image is an array of numbers, called *pixels*, as a signal is a vector of numbers. We denote an image as $\{x[m,n], 0 \leq m \leq M, 0 \leq n \leq N\}$, analogous to the

1-D representation $x[n]$. Color images consist of three separate images, each representing a different color. Usually these colors are red, green, and blue. In this section, we restrict attention to grayscale (black and white) images.

As we will see, the following concepts generalize in a straightforward manner from 1-D to 2-D:

- LTI; impulses; convolution; impulse response (now called *point-spread* function);
- Frequency (now *wavenumber*) response;
- Fourier transforms; sampling theorem;
- DTFT (now DSFT); DFT; FFT.

The following concepts do *not* generalize in a useful manner from 1-D to 2-D:

- Laplace transforms; difference equations;
- Transfer functions; poles and zeros;
- Partial fraction expansions; **z**-transforms.

2-D Sampling

Images are often obtained by sampling a 2-D continuous-*space* signal $z(x, y)$ at points $x = mT$ and $y = nT$ for some small discretization *length* T. Images are usually heavily oversampled. In cameras, images acquired using an array of charge-coupled-device (CCD) sensors are already discretized.

The 2-D continuous-space Fourier transform consists of two successive 1-D continuous-space Fourier transforms. One Fourier transform takes spatial variable x to *wavenumber* (spatial frequency) k_x, and the other takes spatial variable y to wavenumber k_y. Applying this to $z(x, y)$ gives

$$\mathbf{Z}(k_x, k_y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} z(x, y)e^{-(jk_x x + k_y y)}dx\,dy. \tag{10.1}$$

$z(x, y)$ is defined to be bandlimited to B radians per length if $\mathbf{Z}(k_x, k_y) = 0$ unless both $|k_x| < B$ and $|k_y| < B$. Applying the sampling theorem to x and then y shows that if 2-D continuous-space signal $z(x, y)$ is bandlimited to *wavenumber* $\frac{\pi}{T}$ radians per length, then $z(x, y)$ can be uniquely reconstructed from its samples $z[m, n] = z(mT, nT)$ for integers m, n. For the rest of this chapter, we will assume $z(x, y)$ has been so sampled, and consider only discrete-space image processing.

10.1.2 LSI Systems and 2-D Convolution

A 2-D system accepts as input an image $x[m, n]$ and outputs another image $y[m, n]$.

- $x[m, n] \rightarrow \boxed{\text{SYSTEM}} \rightarrow y[m, n]$.

A 2-D system is LSI (Linear *Shift* Invariant) if

- $x[m, n] \rightarrow \boxed{\text{LSI}} \rightarrow y[m, n]$ implies:
- $x[m - M, n - N] \rightarrow \boxed{\text{LSI}} \rightarrow y[m - M, n - N]$ for any integers M and N;
- $cx[m, n] \rightarrow \boxed{\text{LSI}} \rightarrow cy[m, n]$ for any constant c;
- The response to the sum of inputs is the sum of the responses to the inputs.

This is a direct generalization of 1-D LTI to 2-D LSI.

A 2-D impulse $\delta[m, n]$ is defined as

$$\delta[m, n] = \begin{cases} 1 & m = n = 0 \\ 0 & \text{otherwise} \end{cases} \tag{10.2}$$

The *Point-Spread* Function (PSF) $h[m, n]$ of a 2-D LSI system is its 2-D impulse response:

- $\delta[m, n] \rightarrow \boxed{\text{LSI}} \rightarrow h[m, n]$.

In medical imaging systems, the PSF $h[m, n]$ is sometimes measured physically by imaging a small bead, which acts as a 2-D impulse $\delta[m, n]$. $h[m, n]$ is the spread of the bead caused by the imaging system, its "point-spread function." In astronomy, a small star is assumed to be modelled as a 2-D impulse. $h[m, n]$ is then the image of the star in the telescope.

The response of a 2-D LSI system to any image $x[m, n]$ is the 2-D convolution with its PSF $h[m, n]$:

$$\begin{aligned} y[m, n] &= h[m, n] * *x[m, n] \qquad (10.3) \\ &= \sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} h[i, j]x[m - i, n - j]. \end{aligned}$$

The derivation of this result is analogous to the derivation of 1-D discrete-time convolution.

Example 10-1: 2-D Convolution.
Compute the 2-D convolution

$$\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} ** \begin{bmatrix}5 & 6\\ 7 & 8\end{bmatrix} = ? \tag{10.4}$$

Solution:
Using Eq. (10.3) shows that

$$\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} ** \begin{bmatrix}5 & 6\\ 7 & 8\end{bmatrix} = \begin{bmatrix}5 & 16 & 12\\ 22 & 60 & 40\\ 21 & 52 & 32\end{bmatrix} \tag{10.5}$$

$$= \begin{bmatrix}1\cdot 5 & 1\cdot 6+2\cdot 5 & 2\cdot 6\\ 1\cdot 7+3\cdot 5 & 4\cdot 5+3\cdot 6+2\cdot 7+1\cdot 8 & 2\cdot 8+4\cdot 6\\ 3\cdot 7 & 3\cdot 8+4\cdot 7 & 4\cdot 8\end{bmatrix}$$

Another way to compute this 2-D convolution is

$$\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} ** \begin{bmatrix}5 & 6\\ 7 & 8\end{bmatrix} = \begin{bmatrix}5 & 16 & 12\\ 22 & 60 & 40\\ 21 & 52 & 32\end{bmatrix} \tag{10.6}$$

$$= 1\begin{bmatrix}5 & 6 & 0\\ 7 & 8 & 0\\ 0 & 0 & 0\end{bmatrix} + 2\begin{bmatrix}0 & 5 & 6\\ 0 & 7 & 8\\ 0 & 0 & 0\end{bmatrix}$$

$$+3\begin{bmatrix}0 & 0 & 0\\ 5 & 6 & 0\\ 7 & 8 & 0\end{bmatrix} + 4\begin{bmatrix}0 & 0 & 0\\ 0 & 5 & 6\\ 0 & 7 & 8\end{bmatrix}.$$

That is, shift the image in space and multiply it by the corresponding element of the PSF. This illustrates the term "point-spread function." This can also be implemented in Matlab/Matscript as `conv2([1 2;3 4],[5 6;7 8])`.

10.2 2-D Fourier Transforms

As in 1-D, the two important Fourier transforms of images are the DSFT and the 2-D DFT. Also as in 1-D, their applications are (1) computing wavenumber response and (2) computing 2-D spectra. We cover both of these in separate subsections.

10.2.1 Discrete-Space-Fourier-Transform (DSFT)

The Discrete-Space-Fourier-Transform (DSFT) is the 2-D DTFT computed by applying 1-D DTFT's to first the rows, and then the columns (or vice-versa), of the image:

$$\mathbf{X}(e^{j\Omega_1}, e^{j\Omega_2}) = \sum_{m=-\infty}^{\infty}\sum_{n=-\infty}^{\infty} x[m,n]e^{-j(\Omega_1 m+\Omega_2 n)}. \tag{10.7}$$

The DSFT is periodic in Ω_1 and Ω_2 with periods 2π each. The inverse DSFT is computed by applying inverse 1-D DTFT's to first the rows, and then the columns (or vice-versa), of the DSFT:

$$x[m,n] = \frac{1}{4\pi^2}\int_{-\pi}^{\pi}\int_{-\pi}^{\pi}\mathbf{X}(e^{j\Omega_1}, e^{j\Omega_2})e^{j(\Omega_1 m+\Omega_2 n)}d\Omega_1\, d\Omega_2. \tag{10.8}$$

$\boxed{\text{The DSFT is just a 2-D DTFT.}}$

The following two applications of the (1-D) DTFT generalize directly to the (2-D) DSFT:

- The DSFT of the point-spread function $h[m,n]$ is the *wavenumber response* $\mathbf{H}(e^{j\Omega_1}, e^{j\Omega_2})$
- The DSFT of an image $x[m,n]$ is its 2-D spectrum $\mathbf{X}(e^{j\Omega_1}, e^{j\Omega_2})$

We cover these applications next, in separate subsections.

Example 10-2: Computing a DSFT
Compute the DSFT of the 2×2 "image"

$$\begin{aligned} x[m,n] &= \frac{1}{4}\delta[m,n] + \frac{1}{4}\delta[m-1,n-1]\\ &+ \frac{1}{4}\delta[m-1,n] + \frac{1}{4}\delta[m,n-1]\\ &= \frac{1}{4}\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}. \end{aligned} \tag{10.9}$$

Solution:
From the definition of DSFT, we have

$$\mathbf{X}(e^{j\Omega_1}, e^{j\Omega_2}) = \sum_{m=0}^{1}\sum_{n=0}^{1}\frac{1}{4}e^{-j(\Omega_1 m+\Omega_2 n)} \tag{10.10}$$

$$\begin{aligned}
&= \frac{1}{4}[1+e^{-j\Omega_1}+e^{-j\Omega_2}+e^{-j(\Omega_1+\Omega_2)}] \\
&= \frac{1}{4}[1+e^{-j\Omega_1}][1+e^{-j\Omega_2}] \\
&= \frac{1}{4}e^{-j(\Omega_1+\Omega_2)/2}[e^{j\Omega_1/2}+e^{-j\Omega_1/2}] \\
&\times\ [e^{j\Omega_2/2}+e^{-j\Omega_2/2}] \\
&= \cos(\Omega_1/2)\cos(\Omega_2/2)e^{-j(\Omega_1+\Omega_2)/2}.
\end{aligned}$$

We will use this result in Example 10-3 below.

10.2.2 Wavenumber Response

The derivations of the following two results are analogous to those of Section 7-12, and will not be given here.

For 2-D complex exponential functions of space, we have (compare to Eq. (7.124))

$$e^{j(\Omega_1 m+\Omega_2 n)} \rightarrow \boxed{\mathbf{H}(e^{j\Omega_1},e^{j\Omega_2})} \rightarrow \mathbf{H}(e^{j\Omega_1},e^{j\Omega_2})e^{j(\Omega_1 m+\Omega_2 n)} \qquad (10.11)$$

For 2-D sinusoidal functions of space, we have (compare to Eq. (7.127))

$$\cos(\Omega_1 m+\Omega_2 n) \rightarrow \boxed{\mathbf{H}(e^{j\Omega_1},e^{j\Omega_2})} \rightarrow$$
$$|\mathbf{H}(e^{j\Omega_1},e^{j\Omega_2})|\cos(\Omega_1 m+\Omega_2 n+\theta) \text{ where}$$
$$\mathbf{H}(e^{j\Omega_1},e^{j\Omega_2}) = |\mathbf{H}(e^{j\Omega_1},e^{j\Omega_2})|e^{j\theta}. \qquad (10.12)$$

The *wavenumber response* of an LSI system with PSF $h[m,n]$ is the DSFT $\mathbf{H}(e^{j\Omega_1},e^{j\Omega_2})$ of the PSF $h[m,n]$, analogous to 1-D (see Section 7-12).

Example 10-3: Wavenumber Response.
Compute the point-spread function (PSF) and the 2-D wavenumber response of the 2-D LSI system

$$\begin{aligned}
y[m,n] &= \frac{1}{4}x[m,n]+\frac{1}{4}x[m-1,n-1] \\
&+ \frac{1}{4}x[m-1,n]+\frac{1}{4}x[m,n-1] \qquad (10.13)
\end{aligned}$$

This system averages the pixels in a 2×2 block of the image, so it is a 2-D version of the two-point averager of Example 7-23.

Solution:

We can read off the PSF $h[m,n]$ as in 1-D. The result is the same $x[m,n]$ used in Example 10-2. So the wavenumber response is given by Eq. (10.22). The 2-D gain $|\mathbf{H}(e^{j\Omega_1},e^{j\Omega_2})|$ is depicted in the 3-D plot Fig. 10-1. The origin (Ω_1,Ω_2)=$(0,0)$ is in the middle of the plot (the gain is one there), and the highest wavenumbers $\Omega_1=\pm\pi$ and $\Omega_2=\pm\pi$ are at the edges.

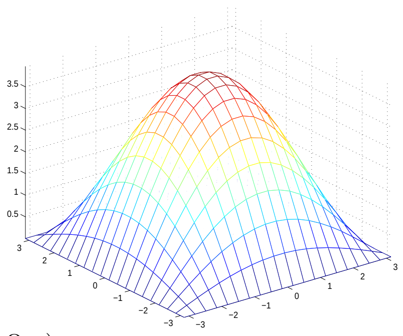


Figure 10.1: 2-D gain for 2×2 Averager.

This 3-D plot of the 2-D gain function shows that the system is a crude spatial low-pass filter. For low wavenumbers $(\Omega_1,\Omega_2)\approx 0$, the gain$\approx$1. For large wavenumbers $\Omega_1\approx\pi$ or $\Omega_2\approx\pi$, the gain is close to zero. So low-wavenumber components of the input image are passed, and high-wavenumber components of the input image rejected. Also,

$$\mathbf{H}(e^{j\Omega_1},e^{j\Omega_2}) = \mathbf{H_1}(e^{j\Omega_1})\mathbf{H_1}(e^{j\Omega_2}) \qquad (10.14)$$

where $\mathbf{H_1}(e^{j\Omega})$ is the frequency response of the 1-D two-point averager from Example 7-23. The 2-D gain is a *separable* function. A separable function is the product of separate functions of Ω_1 and Ω_2. This happens because $h[m,n]$ is also a separable function.

MATLAB code for this example:

```
W=linspace(-pi,pi,20);
[X Y]=meshgrid(W,W);
Z=4*cos(X/2).*cos(Y/2);
mesh(X,Y,Z),axis tight
```

10.2.3 2-D Spectrum

Since the DSFT is doubly periodic in Ω_1 and Ω_2, it is customary to depict the 2-D spectrum for the ranges

$-\pi < \Omega_1, \Omega_2 < \pi$, with the origin $(\Omega_1, \Omega_2) = (0,0)$ at the center. This is analogous to two-sided 1-D spectra.

An important difference from 1-D is that many images have all non-negative pixel values. The depiction of the 2-D spectrum of a non-negative image appears as a single dot at the origin $(\Omega_1, \Omega_2) = (0,0)$, since this is the dc value and

$$\mathbf{X}(e^{j0}, e^{j0}) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} x[m,n] \qquad (10.15)$$

is much larger than other values of $|\mathbf{X}(e^{j\Omega_1}, e^{j\Omega_2})|$.

To be able to see the 2-D spectrum at other locations, we depict not $|\mathbf{X}(e^{j\Omega_1}, e^{j\Omega_2})|$ but its *logarithm.* Taking logarithms of a set of numbers reduces the range of values. Stellar magnitudes, and the Richter scale for earthquakes, are examples of logarithmic scales which compress a huge range of numbers into a smaller range of numbers.

Often, features of the spectrum appear in $\log|\mathbf{X}(e^{j\Omega_1}, e^{j\Omega_2})|$ that do not appear in $|\mathbf{X}(e^{j\Omega_1}, e^{j\Omega_2})|$, which appears as just a single dot. An example of this follows.

Example 10-4: 2-D Spectrum of an Image
Compute the 2-D spectrum of the eyechart image depicted in Fig. 10-2.

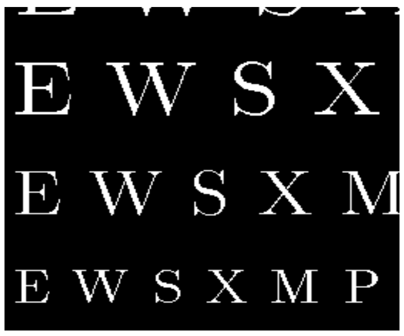


Figure 10.2: Eyechart Image.

Solution:
The logarithm $\log|\mathbf{X}(e^{j\Omega_1}, e^{j\Omega_2})|$ of the 2-D spectrum is depicted in Fig. 10-3.

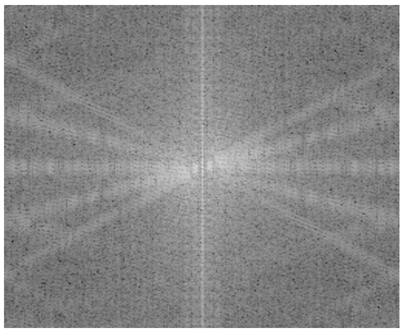


Figure 10.3: Log of Spectrum of Eyechart Image.

The lines in the spectrum appear because the letters in the original image consist mostly of line segments oriented in four directions.

MATLAB code for this example:
```
clear;load letters.mat;
imagesc(X),colormap(gray),axis off
FX=fftshift(log(abs(fft2(X))));
imagesc(FX),colormap(gray),axis off
```

10.2.4 2-D DFT

In practice the DSFT of an $(M \times N)$ image (to compute its spectrum) or PSF (to compute wavenumber response) is computed numerically by sampling in wavenumber. This results in the 2-D Discrete Fourier Transform (DFT), just as in 1-D.

Sampling the DSFT of an $(M \times N)$ image at

- $\Omega_1 = 2\pi\frac{k_1}{M}, k_1 = 0, 1 \ldots (M-1)$
- $\Omega_2 = 2\pi\frac{k_2}{N}, k_2 = 0, 1 \ldots (N-1)$

gives the $(M \times N)$ 2-D Discrete Fourier Transform

$$\mathbf{X}_{k_1,k_2} = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} x[m,n] e^{-j2\pi(\frac{k_1 m}{M}+\frac{k_2 n}{N})}. \qquad (10.16)$$

The inverse 2-D DFT consists of two separate inverse 1-D DFT's:

$$x[m,n] = \frac{1}{MN}\sum_{k_1=0}^{M-1}\sum_{k_2=0}^{N-1} \mathbf{X}_{k_1,k_2} e^{j2\pi(\frac{k_1 m}{M}+\frac{k_2 n}{N})}. \qquad (10.17)$$

The 2-D DFT and its inverse consist of 1-D DFT's or inverse DFT's of first the rows, then the columns, or vice-versa, just like the DSFT and its inverse.

The 2-D DFT of $x[m,n]$ can be computed rapidly using a 2-D version of the FFT. In Matlab, the 2-D DFT of the image stored in array `X` can be computed using `fft2(X)=fft((fft(X)).')`, which applies the 1-D FFT to each row, and then to each column.

The 2-D DFT can also be used to compute the 2-D *cyclic* convolution of two images. The 2-D cyclic convolution consists of 1-D cyclic convolutions (see Section 7-15.4) applied to each row, and then to each column.

Example 10-5: 2-D Cyclic Convolution

Compute the following 2-D Cyclic Convolution using (a) 2-D DFT's and (b) aliasing of the result of Example 10-1.

$$\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} \copyright\copyright \begin{bmatrix}5 & 6\\ 7 & 8\end{bmatrix} = ? \qquad (10.18)$$

Solution: The (2×2) 2-D DFTs are:

$$\begin{bmatrix}1+2+3+4 & 1+3-2-4\\ 1+2-3-4 & 1+4-2-3\end{bmatrix} = \begin{bmatrix}10 & -2\\ -4 & 0\end{bmatrix}$$

$$\begin{bmatrix}5+6+7+8 & 5+7-6-8\\ 5+6-7-8 & 5+8-6-7\end{bmatrix} = \begin{bmatrix}26 & -2\\ -4 & 0\end{bmatrix}$$

$$\begin{bmatrix}26\cdot 10 & -2\cdot -2\\ -4\cdot -4 & 0\cdot 0\end{bmatrix} = \begin{bmatrix}260 & 4\\ 16 & 0\end{bmatrix}$$

$$\frac{1}{2^2}\begin{bmatrix}260+4+16 & 260+16-4\\ 260+4-16 & 260-4-16\end{bmatrix} = \begin{bmatrix}70 & 68\\ 62 & 60\end{bmatrix}$$

This 2-D cyclic convolution can be implemented in Matlab/Mathscript as

```
FX=fft2([1 2;3 4]);FY=fft2([5 6;7 8]);
Y=real(ifft2(FX.*FY));
```

This 2-D cyclic convolution can also be computed by aliasing the 2-D linear convolution computed in Example 10-1:

$$\begin{bmatrix}5 & 16 & 12\\ 22 & 60 & 40\\ 21 & 52 & 32\end{bmatrix} \rightarrow \begin{bmatrix}5+12 & 16\\ 22+40 & 60\\ 21+32 & 52\end{bmatrix} \rightarrow \begin{bmatrix}17+53 & 16+52\\ 62 & 60\end{bmatrix} \qquad (10.19)$$

which gives the same answer.

10.2.5 Image Denoising

Image processing is usually performed using batch processing, often using the 2-D DFT, computed using the 2-D FFT. The FFT is even more important in 2-D than in 1-D, since 2-D images often contain more samples (pixels) than 1-D signals.

The simplest way to perform noise filtering of images is to set high wavenumber components of the noisy image to zero. This implements a 2-D brick-wall low-pass filter. However, there is a tradeoff: a small cutoff wavenumber eliminates more noise, but also eliminates some wavenumber components of the image. This has the effect of distorting edges in the image. A high cutoff wavenumber has less effect on the image, but also eliminates less noise.

The following example illustrates this tradeoff.

Example 10-6: Using the 2-D DFT to Denoise an Image.

High-wavenumber noise was added to the eyechart image shown in Fig. 10-2. Use the 2-D DFT to reduce the noise.

Solution:

The noisy eyechart image is shown in Fig. 10-4.

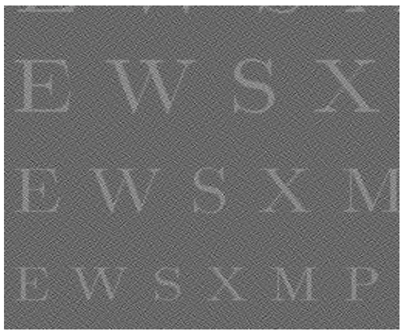


Figure 10.4: Noisy Eyechart image.

Two 2-D brick-wall low-pass filters with two different cutoffs are implemented using the 2-D DFT. The filtered images are shown in Fig. 10-5a and Fig. 10-5b for low and high cutoffs, respectively.

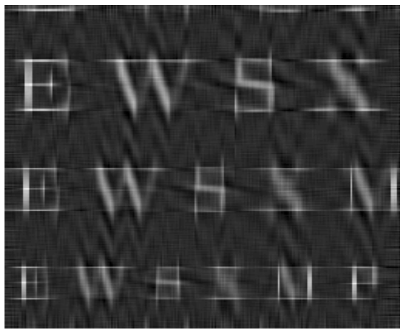
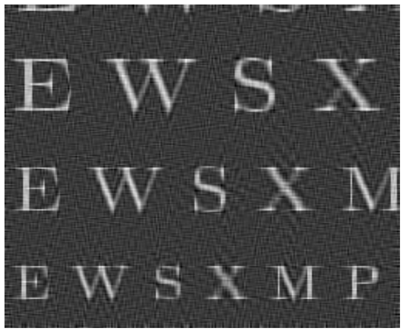



Figure 10.5: Image Filtered with Cutoff Index: (a) L=10; (b) L=20.

The image filtered with a lower cutoff has less noise, but the letters are distorted.

The image filtered with a higher cutoff has more noise, but the letters are less distorted.

This is a simple example of *bias-variance tradeoff*. The noise can be reduced (low variance) at the price of distorting the image (high bias). Or the image can be undistorted (low bias) at the price of high noise level (high variance). This tradeoff occurs throughout signal and image processing.

The cutoff is the 2-D DFT index above which all 2-D DFT values were set to zero. This preserved 2-D DFT values for which either 2-D DFT index was near zero. This greatly improves the filtered image. Note that the blurring shows up mostly along the "X's," which have components angled 60^o to the horizontal, not horizontal (0^o) or vertical (90^o) components.

MATLAB code for this example:

```
  clear;load letters.mat;
  N=randn(257,257);
  NN1=N(1:256,1:256)-N(1:256,2:257);
  NN2=N(2:257,1:256)-N(2:257,2:257);
  Y=X+100*(NN1-NN2);FY=fft2(Y);
  FY(20:238,20:238)=0;Z1=real(ifft2(FY));
  FY(10:248,10:248)=0;Z2=real(ifft2(FY));
  figure,imagesc(Y),colormap(gray),axis off
  figure,imagesc(Z2),colormap(gray),axis
off
  figure,imagesc(Z1),colormap(gray),axis
off
```

Another way to perform denoising is to filter the image using the system with the PSF from Example 10-3.

In Example 10-17 in Section 10-8 below we show how to use *wavelets* to perform a much better job of denoising images.

Exercise 10-1: Which type of filter has a wavenumber response very different from the other three: **(a)** Lowpass filter **(b)** Highpass filter **(c)** Differentiator **(d)** Edge detector

Answer:
Lowpass. The other three all reject dc and enhance large wavenumbers.

Exercise 10-2. The noise in Example 10-6 was generated by passing white Gaussian noise through the 2-D system

$$\begin{aligned} y[m,n] &= x[m,n] + x[m-1,n-1] \\ &- x[m-1,n] - x[m,n-1]. \end{aligned} \tag{10.20}$$

Show that the 2-D gain is

$$|\mathbf{H}(e^{j\Omega_1}, e^{j\Omega_2})| = 4|\sin(\Omega_1/2)\sin(\Omega_2/2)| \tag{10.21}$$

which is a crude high-pass filter emphasizing the high-wavenumber components of the noise.

Answer:
We can read off the PSF as $h[m,n] = \begin{bmatrix} \underline{1} & -1 \\ -1 & 1 \end{bmatrix}$. Following Example 10-2 and noting $(2j)^2 = -4$,

$$\begin{aligned} \mathbf{H}(e^{j\Omega_1}, e^{j\Omega_2}) &= 1 - e^{-j\Omega_1} - e^{-j\Omega_2} + e^{-j(\Omega_1+\Omega_2)} \\ &= [1 - e^{-j\Omega_1}][1 - e^{-j\Omega_2}] \end{aligned}$$

$$\begin{aligned} &= e^{-j(\Omega_1+\Omega_2)/2}[e^{j\Omega_1/2} - e^{-j\Omega_1/2}] \\ &\times\ [e^{j\Omega_2/2} - e^{-j\Omega_2/2}] \\ &= -4\sin(\Omega_1/2)\sin(\Omega_2/2)e^{-j(\Omega_1+\Omega_2)/2} \end{aligned}$$

The gain is the magnitude of the 2-D wavenumber response $\mathbf{H}(e^{j\Omega_1}, e^{j\Omega_2})$. This system could be used for edge detection in images, since flat areas in the image are zeroed out. But better methods are presented next.

The next section is an insert, so numbering for figures and examples are out of sequence.

10.3 Edge Detection

An *edge* in an image is a sharp boundary between two different regions of an image. Here "sharp" means a width of at most a few pixels, and a significant difference in the pixel values on either side of the edge. "Significant" is not clearly defined; its definition depends on characteristics of the image and the reason why edges are of interest. This is a nebulous definition, but there is no uniform definition of an edge.

Edge detection is the problem of determining the locations $[m, n]$ of edges in an image $x[m, n]$. Edge detection is used to *segment* an image into different regions, or to determine the boundaries of a region of interest. For example, a medical image may consist of different human organs; interpretation of the image is easier if (say) the region of the image corresponding to the pancreas is identified separately from the rest of the image. Identification of a face is easier if the eyes in an image of the face are identified as a region separate from the rest of the image of the face. Ideally, an edge is a contour that encloses a region of the image whose values differ significantly from the values around it. Edge detection is important in computer vision.

10.3.1 1-D Edge Detection

We first examine 1-D edge detection, since it inspires a commonly-used 2-D edge detection algorithm.

An obvious way to detect the locations of sharp changes in a 1-D signal $x(t)$ is to take its derivative, which will be large at times t_0 where $x(t)$ changes rapidly, and small at other times. The times t_0 at which the derivative is large are the edges of $x(t)$. The meaning of "large" will depend on $x(t)$ itself.

For 1-D discrete time signals $x[n]$, the discrete-time counterpart to the derivative is the *difference*

$$y[n] = x[n] - x[n-1] \qquad (10.22)$$

which is large when $x[n]$ is changing rapidly with n, yet is localized to pinpoint the time n_0 of the edge. As simple as it is, computing the difference $y[n]$ from $x[n]$ and thresholding $|y[n]|$ is very effective at detecting 1-D edges. Here, thresholding $|y[n]|$ with threshold t_1 means replacing $|y[n]|$ with $z[n]$, where

$$z[n] = \begin{cases} 1 & \text{for } |y[n]| > t_1 \\ 0 & \text{for } |y[n]| < t_1 \end{cases} \qquad (10.23)$$

The threshold t_1 depends on $x[n]$; in practice, many values of t_1 are tried until the result seems reasonable. The times n_i where $z[n_i] = 1$ are the edges of $x[n]$.

Example 10-9: 1-D Edge Detection

Identify the edges of the 1-D signal $x[n]$ shown in Fig. 10-12a by thresholding the absolute values of differences $y[n]$.

Solution:

Figure 10-12a plots $x[n]$; Figure 10-12b plots $|y[n]|$; Figure 10-12c plots the thresholded $|y[n]|$ using $t_1 = 1$. Different values of t_1 were tried. The times of the edges are clearly indicated in Fig. 10-12c.

MATLAB code for this figure:

```
T=1;X(1:5)=3;X(6:14)=2;X(15:21)=5;X(22:30)=4;Y=X+rand
subplot(311),stem(Y),Z=Y(2:30)-Y(1:29);subplot(312),s
Z(abs(Z)<T)=0;Z(abs(Z)>T)=1;subplot(313),stem(Z)
```

10.3.2 2-D Edge Detection

The above approach can be generalized to edge detection in images as follows.

Define a vertical edge VE as a vertical line at $n = n_0$ extending from $m = m_1$ to $m = m_2$, so it has length $m_2 - m_1 + 1$:

$$VE = \{[m, n] : m_1 \leq m \leq m_2; n = n_0\}$$

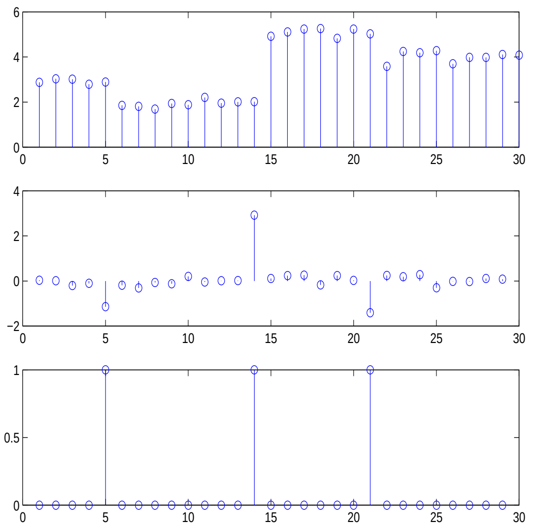


Figure 10.6: Edge Detection by Thresholding Absolute Values of Differences. (a) Original signal $x[n]$. (b) $y[n] = x[n] - x[n-1]$. (c) $|y[n]|$ thresholded using $t_1 = 1$.

An evident way to detect a vertical edge is to apply the difference operator Eq. (10.22) to each row m of the image. But this is not enough; a large difference at location $[m_0, n_0]$ could be due to an isolated jump in $x[m,n]$. A vertical edge at $n = n_0$ requires a large difference not only at $[m_0, n_0]$, but at the points $[m_0+1, n_0]$ and $[m_0-1, n_0]$ just above and below $[m_0, n_0]$.

This suggests that a horizontal edge detector should not only take horizontal differences, but vertical sums of the differences. The latter will be large only if the horizontal differences are large at consecutive vertical locations. This is likely to be a true vertical edge.

In practice the following scheme is used for vertical edge detection.

Define *forward* and *backward* differences

$$\begin{aligned} d_f[n] &= x[n+1] - x[n] && \text{forward} \\ d_b[n] &= x[n] - x[n-1] && \text{backward} \end{aligned} \quad (10.24)$$

and forward and backwards sums

$$\begin{aligned} s_f[n] &= x[n+1] + x[n] && \text{forward} \\ s_b[n] &= x[n] + x[n-1] && \text{backward} \end{aligned} \quad (10.25)$$

Now define the *central* difference as the sum of the forward and backwards differences

$$\begin{aligned} d_c[n] &= d_f[n] + d_b[n] \\ &= x[n+1] - x[n-1] \end{aligned} \quad (10.26)$$

and the *central* sum as the sum of the forward and backwards sums

$$\begin{aligned} s_c[n] &= s_f[n] + s_b[n] \\ &= x[n+1] + 2x[n] + x[n-1] \end{aligned} \quad (10.27)$$

Finally, define the horizontal edge detector as the vertical central sum of three horizontal central differences:

$$\begin{aligned} y_H[m,n] &= (x[m+1,n+1] - x[m+1,n-1]) \\ &+ 2(x[m,n+1] - x[m,n-1]) \\ &+ (x[m-1,n+1] - x[m-1,n-1]) \end{aligned} \quad (10.28)$$

At a vertical edge, all three terms in Eq. (10.28) are large. This makes $y_H[m,n]$ much larger at locations $[m,n] \in VE$ than at other locations, even locations where a single term of Eq. (10.28) is large. Thresholding $|y_H[m,n]|$ can be expected to be a good indicator of vertical edges.

A horizontal edge HE at row $m = m_0$ extending from $n = n_1$ to $n = n_2$ and having length $n_2 - n_1 + 1$ is defined as

$$HE = \{[m,n] : m = m_0; n_1 \leq m \leq n_2\}$$

Exchanging m and n in Eq. (10.28) gives the vertical edge detector

$$\begin{aligned} y_V[m,n] &= (x[m+1,n+1] - x[m-1,n+1]) \\ &+ 2(x[m+1,n] - x[m-1,n]) \\ &+ (x[m+1,n-1] - x[m-1,n-1]) \end{aligned} \quad (10.29)$$

Thresholding $|y_V[m,n]|$ can be expected to be a good indicator of horizontal edges.

Horizontal edge detection can be implemented using the following PSF, which can be read off from Eq.

(10.28):

$$h_H[m,n] = \begin{bmatrix} 1 & 0 & -1 \\ 2 & \underline{0} & -2 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} [1 \quad 0 \quad -1] \tag{10.30}$$

Vertical edge detection can be implemented using the following PSF, which can be read off from Eq. (10.29):

$$h_V[m,n] = \begin{bmatrix} 1 & 2 & 1 \\ 0 & \underline{0} & 0 \\ -1 & -2 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} [1 \quad 2 \quad 1] \tag{10.31}$$

Both of these PSF's are separable, so that they can be implemented as a central difference in one direction, followed by a central sum in the other direction. Both PSF's are also 3×3 and centered at the origin.

Of course, most edges are neither purely horizontal nor purely vertical. But a edge at an angle different from 0^o or 90^o can be viewed as having horizontal and vertical components. This suggests the use of the following edge detection formula.

$$\begin{aligned} y[m,n] &= \sqrt{(y_H[m,n])^2 + (y_V[m,n])^2} \\ z[m,n] &= \begin{cases} 1 & \text{for } y[m,n] > t_1 \\ 0 & \text{for } y[m,n] < t_1 \end{cases} \end{aligned} \tag{10.32}$$

$y[m,n]$ is called the *gradient*, even though this is discrete space, since it is the discrete space version of the continuous space gradient magnitude.

The threshold t_1 can be chosen by examining a histogram of the values of $y[m,n]$, looking for a wide gap in the range of values of $y[m,n]$, and choosing t_1 to be in the middle of the gap. The edges of $x[m,n]$ are the locations $[m_i, n_i]$ where $z[m_i, n_i] = 1$.

Eq. (10.32) is called the *Sobel* edge detector, named after Irwin Sobel, who developed it in 1968, when computer-based image processing was in its infancy and only simple algorithms could be used.

The following example applies Sobel edge detection to the letters and clown images:

Example 10-10: Sobel Edge Detection.
Apply Eq. (10.32) to the letters and clown images. See how well it determines the edges of the images.

Solution: The results are shown in Fig. 10-13 and Fig. 10-14. In the letters image, the obvious edges have been identified, even the diagonal and curved edges. Using a threshold t_1 anywhere in the range $100 < t_1 < 350$ did not alter $z[m,n]$, so $t_1 = 200$ was used.

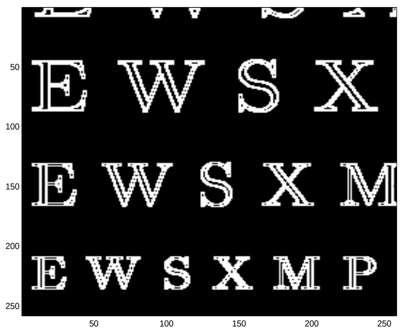


Figure 10.7: Result of Sobel Edge Detection Applied to Letters Image.

In the clown image, a threshold of $t_1 = 1.3$ seemed to give the best result. Edges have been identified, but sporadically.

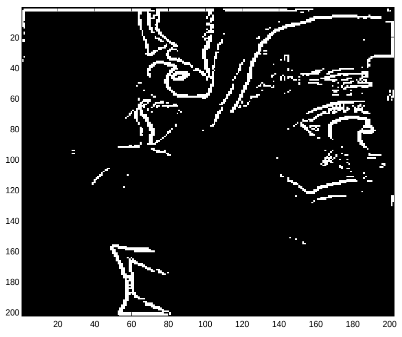


Figure 10.8: Result of Sobel Edge Detection Applied to Clown Image.

MATLAB code for this example:

```
clear;load letters;
H=conv2(X,[1 0 -1;2 0 -2;1 0 -1]);
V=conv2(X,[1 2 1;0 0 0;-1 -2 -1]);
W=sqrt(H.*H+V.*V);W(abs(W)<200)=0;W(abs(W)>200)=1;
figure,imagesc(W),colormap(gray)
clear;load clown;
H=conv2(X,[1 0 -1;2 0 -2;1 0 -1]);
V=conv2(X,[1 2 1;0 0 0;-1 -2 -1]);
W=sqrt(H.*H+V.*H);W(abs(W)<1.3)=0;W(abs(W)>1.3)=1;
figure,imagesc(W),colormap(gray)
```

The result of applying the Sobel edge detector to the clown image suggests that an improvement to the Sobel edge detector that tracks edges and completes edge contours is needed. This leads to the Canny edge detector.

10.3.3 Canny Edge Detection

A very commonly used edge detection algorithm is the *Canny* algorithm, named after John F. Canny. The Canny edge detection algorithm is an extension of the Sobel edge detection algorithm. Prior to computing the horizontal and vertical edges, a preprocessing step of filtering the image with a truncated 2-D Gaussian PSF is used to reduce noise and isolated features that are not edges. After computing $y[m,n]$ in Eq. (10.32), two additional steps are performed. The first is an *edge thinning* step that localizes the edges better than $y[m,n]$ does. The second is a double threshold that separates the candidate edges into two categories: "Strong" edges that are definitely edges, and "weak" edges that are only considered to be edges if they adjoin another weak or strong edge. This connects the candidate edges into contours.

The Canny edge detection can be summarized as:

Step #1: Blur the image by convolving it with a truncated Gaussian function. In practice, the following 5×5 PSF function, which is a truncated 2-D Gaussian with standard deviation 1.4, is used:

$$h_G[m,n]=\frac{1}{159}\begin{bmatrix}2&4&5&4&2\\4&9&12&9&4\\5&12&15&12&5\\4&9&12&9&4\\2&4&5&4&2\end{bmatrix} \qquad (10.33)$$

The result is $x_1[m,n]=h_G[m,n]**x[m,n]$.

Step #2: Compute the horizontal and vertical differences using Eq. (10.28) and Eq. (10.29) on the filtered image $x_1[m,n]$.

Step #3: Compute gradient size and orientation

$$\begin{aligned} y[m,n] &= \sqrt{(x_H[m,n])^2+(x_V[m,n])^2}\\ \theta[m,n] &= \arctan(x_V[m,n]/x_H[m,n]) \end{aligned} \qquad (10.34)$$

This is actually a rectangular-to-polar conversion of $(x_H[m,n],x_V[m,n])$ to $(y[m,n],\theta[m,n])$ for each location $[m,n]$. The gradient $y[m,n]$ is again computed using Eq. (10.32). So far, the Canny edge detection algorithm is the Sobel edge detection algorithm with prefiltering by the truncated Gaussian PDF.

Step #4: At each $[m,n]$, round $\theta[m,n]$ to the nearest of $\{0^o,45^o,90^o,135^o\}$. Examine three values:

- $\theta[m,n]=0^o$: $\{y[m,n-1],y[m,n],y[m,n+1]\}$
- $\theta[m,n]=90^o$: $\{y[m-1,n],y[m,n],y[m+1,n]\}$
- $\theta[m,n]=45^o$: $\{y[m+1,n-1],y[m,n],y[m-1,n+1]\}$
- $\theta[m,n]=135^o$: $\{y[m-1,n-1],y[m,n],y[m+1,n+1]\}$

If $y[m,n]$ is the largest of these three values, keep it; otherwise set it to zero. This is called *edge thinning.* The idea is to avoid making an edge wider than necessary to indicate its presence.

Step #5: Replace the threshold in Eq. (10.32) with the double threshold

$$z[m,n]=\begin{cases}2 & \text{for } y[m,n]>t_1\\ 1 & \text{for } t_2<y[m,n]<t_1\\ 0 & \text{for } y[m,n]<t_1\end{cases} \qquad (10.35)$$

- If $y[m,n]<t_2$, there is no edge at location $[m,n]$.
- If $y[m,n]>t_1$, there is an edge at location $[m,n]$.
- If $t_2<y[m,n]<t_1$, there is an edge at location $[m,n]$ only if one of the following is also an edge: $\{y[m-1,n-1],y[m-1,n],y[m-1,n+1]$ $y[m,n-1],y[m,n+1]$ and $y[m+1,n-1],y[m+1,n],y[m+1,n+1]\}$.

The idea is that $[m,n]$ is an edge location only if it connected (adjoins) another edge location. As before, the numbers 0,1,2 are merely indicators for presence/possible presence/no presence of an edge.

Example 10-11: Canny Edge Detection.
Apply the Canny edge detection algorithm to the clown image. See how well it determines the edges of the images.

Solution: The results are shown in Fig. 10-15. Thresholds $t_1 = 0.125$ and $t_2 = 0.050$ seemed to give the best result. The edges are now connected into contours that segment the clown image.

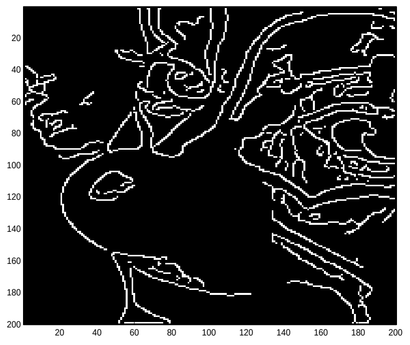


Figure 10.9: Result of Canny Edge Detection Applied to Clown Image.

Edge detection can be implemented in MATLAB's Image Processing Toolbox using the commands
`E=edge(X,'sobel',T1)` for Sobel and
`E=edge(X,'canny',T1,T2)` for Canny.
The image is stored in array `X`, the edge image is stored in array `E` and `T1` and `T2` are the thresholds. MATLAB assigns default values to the thresholds, computed from the image, if they are not specified.

End of insert.

10.4 Image Deconvolution

10.4.1 Problem Formulation

The image deconvolution problem is to reconstruct an image $x[m,n]$ from its convolution $y[m,n]$ with a known point-spread-function (PSF) $h[m,n]$. The goal is to solve for $x[m,n]$ in the equation

$$y[m,n] = h[m,n] * * x[m,n] \qquad (10.36)$$

where

- $x[m,n]$ is the unknown $M \times M$ image;
- $h[m,n]$ is the known $L \times L$ PSF;
- $y[m,n]$ is the known $(L+M-1) \times (L+M-1)$ data.

Many problems in medical and optical imaging can be formulated as image deconvolution problems. The PSF represents the effects of any of the following:

- A medical or astronomical imaging system;
- Blurring in a camera caused by defocusing;
- An antenna aperture function;
- Distortion caused by the atmosphere.

Taking the DSFT of Eq. (10.36) gives

$$\mathbf{Y}(e^{j\Omega_1}, e^{j\Omega_2}) = \mathbf{H}(e^{j\Omega_1}, e^{j\Omega_2})\mathbf{X}(e^{j\Omega_1}, e^{j\Omega_2}). \qquad (10.37)$$

Recall that $y[m,n]$ is $(L+M-1) \times (L+M-1)$. Round up $(L+M-1)$ to N, the smallest power of two greater than $(L+M-1)$. The fast radix-2 2-D FFT can be used to compute $N \times N$ 2-D DFT's. Sampling the DSFT at $\omega_1 = \frac{2\pi}{N}k_1$ and $\omega_2 = \frac{2\pi}{N}k_2$ gives

$$\mathbf{Y}_{k_1,k_2} = \mathbf{H}_{k_1,k_2}\mathbf{X}_{k_1,k_2}. \qquad (10.38)$$

$\mathbf{Y}_{k_1,k_2}, \mathbf{H}_{k_1,k_2}, \mathbf{X}_{k_1,k_2}$ are the $N \times N$ 2-D DFT's of $y[m,n], h[m,n], x[m,n]$ zero-padded so that they all have size $N \times N$. Assume for the moment that $\mathbf{H}_{k_1,k_2} \neq 0$ for all (k_1,k_2). Then dividing by $\mathbf{H}_{k_1,k_2}$ gives

$$\mathbf{X}_{k_1,k_2} = \mathbf{Y}_{k_1,k_2}/\mathbf{H}_{k_1,k_2}. \qquad (10.39)$$

An $N \times N$ inverse 2-D DFT of $\mathbf{X}_{k_1,k_2}$ yields zero-padded $x[m,n]$; the zeros are easily discarded to obtain $x[m,n]$. This procedure is identical to Subsection (8-5.1) for 1-D deconvolution, except that now everything is 2-D.

Example 10-7: Noiseless 2-D deconvolution of Eyechart Image.
The eyechart image depicted in Fig. 10-2 is convolved with a 2-D Gaussian PSF (a common test PSF)

$$h[m,n] = e^{-(m^2+n^2)/20}, -10 \leq m,n \leq 10. \quad (10.40)$$

The result is the blurred image depicted in Fig. 10-6. Reconstruct the original image from the blurred image.

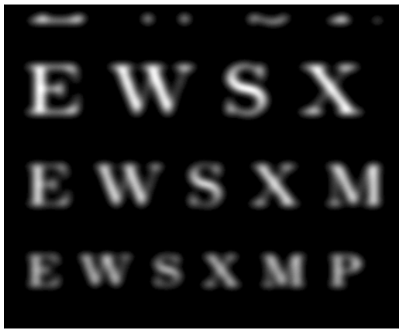


Figure 10.10: Eyechart Image blurred with Gaussian PSF.

Solution:
The image sizes are as follows:

- Eyechart image: 256×256;
- Gaussian PSF: 21×21;
- Blurred image: 276×276, since $256 + 21 - 1 = 276$.

All three images are zero-padded to 280×280, since 280=(2)(2)(2)(5)(7) has many factors. The Cooley-Tukey 2-D FFT will be fast, although not as fast as a radix-2 2-D FFT.

The deconvolved image computed using Eq. (10.39) is shown in Fig. 10-7 and matches the original eyechart image. But use of Eq. (10.39) is very sensitive to noise in $y[m,n]$, as Example 10-8 below shows.

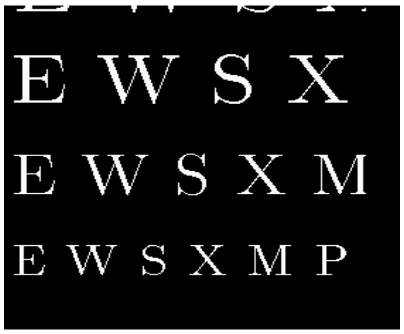


Figure 10.11: Eyechart Image Deconvolved from Noiseless Data.

MATLAB code for this example:
```
clear;load letters.mat;
H=exp(-[-10:10].*[-10:10]/20);
FH=fft2(H'*H,280,280);
FX=fft2(X,280,280);FY=FH.*FX;
Y=real(ifft2(FY));figure
imagesc(Y),colormap(gray),axis off
Z=real(ifft2(FY./FH));figure
imagesc(Z),colormap(gray),axis off
```

10.4.2 Tikhonov Regularization

The problem with using Eq. (10.39) to perform image deconvolution is that in many practical applications $\mathbf{H}_{k_1,k_2} \approx 0$ for large k_1 and k_2. Then the gain $\frac{1}{\mathbf{H}_{k_1,k_2}}$ multiplying $\mathbf{Y}_{k_1,k_2}$ will be very large, and any noise present in $\mathbf{Y}_{k_1,k_2}$ will be amplified by this large gain.

To avoid this difficulty, deconvolution must be *regularized.* This means that we solve not Eq. (10.36), but a variation of it. Usually this requires minimizing a *cost functional* that trades off accuracy of the reconstruction (bias) with the noise in it (variance). Bias-variance tradeoff was encountered previously in the image denoising problem. Now it appears again in the context of deconvolution.

A common type of regularization is *Tikhonov regularization.* In Tikhonov regularization, instead of simply solving Eq. (10.36) for $x[m,n]$, we minimize over $x[m,n]$ the *Tikhonov cost functional*

$$\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}[(y[m,n]-h[m,n]**x[m,n])^2+\lambda^2(x[m,n])^2], \tag{10.41}$$

where zero-padding to size $N \times N$ is used as before, and λ is a known constant. λ specifies the tradeoff between accuracy of the reconstruction (bias), as specified by the first term, and the size of the reconstructed image (variance), as specified by the second term. Specifically,

- Small λ leads to accurate but noisy reconstructed images $x[m,n]$
- Large λ leads to less accurate but less noisy reconstructed images.

Note that this is the same tradeoff encountered in the image denoising problem, except that the tradeoff is now specified by λ, instead of the cutoff wavenumber in the brick-wall low-pass filter.

10.4.3 Wiener Filter

Using Parseval's theorem for the 2-D DFT, the Tikhonov cost functional Eq. (10.41) equals

$$\frac{1}{MN}\sum_{k_1=0}^{N-1}\sum_{k_2=0}^{N-1}[|\mathbf{Y}_{k_1,k_2}-\mathbf{H}_{k_1,k_2}\mathbf{X}_{k_1,k_2}|^2+\lambda^2|\mathbf{X}_{k_1,k_2}|^2]. \tag{10.42}$$

This can be minimized separately for each (k_1,k_2). The $\mathbf{X}_{k_1,k_2}$ minimizing this for each (k_1,k_2) is derived in Problem 10-4. It is

$$\mathbf{X}_{k_1,k_2} = \mathbf{Y}_{k_1,k_2}\frac{\mathbf{H}^*_{k_1,k_2}}{|\mathbf{H}_{k_1,k_2}|^2+\lambda^2}. \tag{10.43}$$

The inverse filter $\mathbf{G}_{k_1,k_2}$ defined as

$$\mathbf{G}_{k_1,k_2} = \frac{\mathbf{H}^*_{k_1,k_2}}{|\mathbf{H}_{k_1,k_2}|^2+\lambda^2} \tag{10.44}$$

is called a *Wiener* filter. Its effect can be summarized as follows.

(1) For values of (k_1,k_2) for which $\mathbf{H}_{k_1,k_2} >> \lambda$, the Wiener filter implements

$$\begin{aligned}\mathbf{X}_{k_1,k_2} &\approx \mathbf{Y}_{k_1,k_2}\frac{\mathbf{H}^*_{k_1,k_2}}{|\mathbf{H}_{k_1,k_2}|^2} \\ &= \frac{\mathbf{Y}_{k_1,k_2}}{\mathbf{H}_{k_1,k_2}}.\end{aligned} \tag{10.45}$$

which is just Eq. (10.39).

(2) For values of (k_1,k_2) for which $\mathbf{H}_{k_1,k_2} << \lambda$, the Wiener filter implements

$$\mathbf{X}_{k_1,k_2} \approx \mathbf{Y}_{k_1,k_2}\frac{\mathbf{H}^*_{k_1,k_2}}{\lambda^2}. \tag{10.46}$$

This is very different from Eq. (10.39), so it is incorrect (this is bias). But it does keep $\mathbf{X}_{k_1,k_2}$ from getting large (reduced variance). This is sensible: Since $\mathbf{H}_{k_1,k_2} \approx 0$, there is no way to recover $\mathbf{X}_{k_1,k_2}$ from $\mathbf{Y}_{k_1,k_2}=\mathbf{H}_{k_1,k_2}\mathbf{X}_{k_1,k_2}$, so the best thing to do is to at least keep the unknown values of $\mathbf{X}_{k_1,k_2}$ small, so they don't overwhelm the correct values of $\mathbf{X}_{k_1,k_2}$.

Example 10-8: Noisy 2-D Deconvolution of Eyechart Image.

Redo Example 10-7 with noise added to the blurred image. Reconstruct the image from the noisy blurred image depicted in Fig. 10-8.

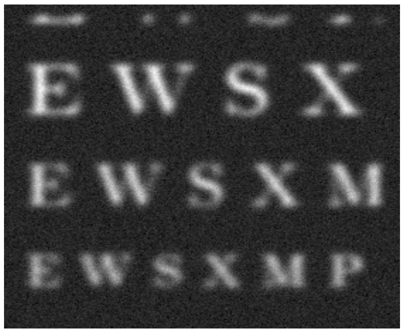
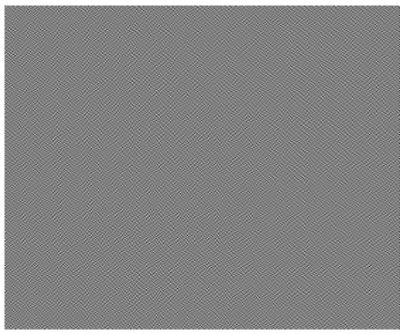


Figure 10.12: Eyechart Image Blurred with Gaussian PSF. Noise is Added Afterwards.

Solution:

First, we deconvolve the image using Eq. (10.39), which is identical to (10.43) with $\lambda = 0$. The result, shown in Fig. 10-9a, is swamped with noise. The eyechart image does not appear at all!

Second, we use (10.43) with $\lambda^2 = 5$. The result, shown in Fig. 10-9b, is a much better reconstruction.



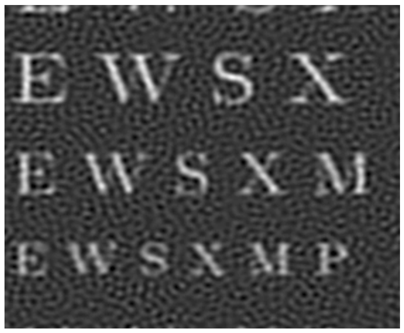


Figure 10.13: Eyechart Image Deconvolved from Noisy Data using: (a) $\lambda = 0$; (b) $\lambda^2 = 5$.

MATLAB code for this example:

```
clear;load letters.mat;
H=exp(-[-10:10].*[-10:10]/20);
FH=fft2(H'*H,280,280);
FX=fft2(X,280,280);FY=FH.*FX;
Y=real(ifft2(FY));
Y=Y+500*randn(280,280);
figure,imagesc(Y),colormap(gray),axis off
FY=fft2(Y);Z=real(ifft2(FY./FH));
figure,imagesc(Z),colormap(gray),axis off
FW=conj(FH)./(abs(FH).*abs(FH)+5);
W=real(ifft2(FY.*FW));
figure,imagesc(W),colormap(gray),axis
off
```

10.4.4 Median Filtering

While not LSI, median filtering is very useful in eliminating noise that consists of isolated points. This is called *salt-and-pepper* or *shot* noise. Such noise arises from bit errors or Poisson noise.

Median filtering of order L simply replaces each pixel with the median of the L^2 pixels in the $L \times L$ block centered on the pixel. For example, a median filter of order $L = 3$ replaces each pixel $x[m,n]$ with the median of the nine pixels neighboring it:

- $\{x[m-1,n-1], x[m-1,n], x[m-1,n+1]$
  $x[m,n-1], x[m,n], x[m,n+1]$ and
  $x[m+1,n-1], x[m+1,n], x[m+1,n+1]\}$.

Example 10-9: Median Filtering

The left image of Fig. 10-10 has been corrupted with salt-and-pepper noise. Use median filtering to reduce this noise.

Solution:

A median filter of order $L = 5$ was applied to the noisy image. The result is the right image of Fig. 10-10. The noise is almost completely eliminated.

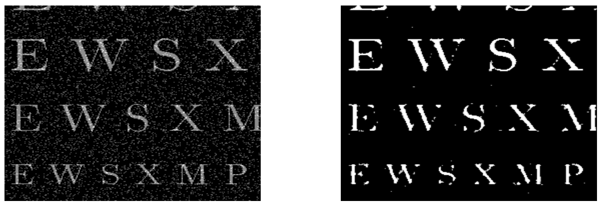


Figure 10.14: Median Filtering Example. Left: Image Corrupted by Salt-and-Pepper Noise. Right: Result of Median Filtering.

MATLAB code for this example:

```
clear;load letters.mat;
Y=X+200*floor(rand(256,256)+0.1);
Z=medfilt2(Y);
subplot(221),imagesc(Y),colormap(gray)
subplot(222),imagesc(Z),colormap(gray)
```

10.5 Haar Wavelet Transform

10.5.1 Overview of the Discrete-Time Wavelet Transform

The *wavelet transform* is an important signal processing tool for representing signals or images that consist mostly of slowly-varying segments, with a few fast-varying segments. The wavelet transform, like the discrete Fourier transform, is an *orthogonal* transformation–it is not only invertible, but the inverse transform is almost identical to the forward transform. This makes it useful for analyzing, processing, and synthesizing signals. But the discrete Fourier transform represents signals as linear combinations of complex exponential functions, which require both low and high frequencies to represent signals or images with even a few fast-varying segments, such as edges. Wavelet transforms are localized, so that only low-resolution components are required to represent most of such a signal or image. High-resolution components are required only at edges.

Wavelet transforms have three major applications:

- *Compression* of signals and images, so that they are represented (in the wavelet transform domain) by many fewer numbers than the original signal or image. The JPEG-2000 image compression standard uses the wavelet transform;

- *Compressed sensing* of signals and images. Since the signal or image in the wavelet transform domain requires many fewer numbers to represent it, it can be reconstructed from many fewer observations than would be required to reconstruct the original signal or image. An introduction to compressed sensing is presented later in this chapter;

- *Filtering* of signals and images. Since the signal or image in the wavelet transform domain requires many fewer numbers to represent it, thresholding to zero small values of the wavelet transform of a noisy signal or image will reduce the noise in the original signal or image more effectively than using the discrete Fourier transform.

We begin by presenting the *Haar* wavelet transform, which is the simplest wavelet transform, and yet illustrates many features of wavelet transforms. We then presents filter banks and quadrature-mirror filters (QMF's), derives the Smith-Barnwell condition for perfect reconstruction of the original signal from its wavelet transform, and derive the Daubechies wavelet function, which is the most commonly-used wavelet function. Finally, examples of signal and image compression and denoising are provided.

The cyclic or circular convolution of two signals $h[n]$ and $x[n]$, each having duration N, was defined in Eq. (7.175). The result is the signal $y[n]$, also of duration N

$$\begin{aligned} y[n] &= h[n] \copyright x[n] \\ &= \sum_{i=0}^{N-1} h[i]x[(n-i)\text{mod}(N)] \end{aligned} \tag{10.47}$$

$n\,\text{mod}(N)$ is the remainder when n is divided by N.

The duration L of $h[n]$ is usually much less than N, in which case $h[n]$ is zero-padded with $(N-L)$ zeros. In the sequel, it should be assumed that $h[n]$ has been zero-padded to the durations N of $x[n]$ and $y[n]$. This does not increase the computation.

The Haar transform is by far the simplest wavelet transform, and yet it illustrates many of the concepts of how the wavelet transform works. This section presents and demonstrates the Haar transform.

10.5.2 Single-Stage Decomposition

Consider the finite-duration signal $x[n]$

$$x[n] = \{\underline{a}, b, c, d, e, f, g, h\}. \tag{10.48}$$

Define the lowpass and highpass filters having impulse responses $g_{haar}[n]$ and $h_{haar}[n]$, respectively

$$\begin{aligned} g_{haar}[n] &= \{\underline{1}, 1\} \\ h_{haar}[n] &= \{\underline{1}, -1\}. \end{aligned} \tag{10.49}$$

Recall from Example 7-23 that the frequency responses of these filters are the DTFT's

$$\mathbf{G}_{haar}(e^{j\Omega}) = 1 + e^{-j\Omega} = 2\cos(\Omega/2)e^{-j\Omega/2} \tag{10.50}$$

$$\mathbf{H}_{haar}(e^{j\Omega}) = 1 - e^{-j\Omega} = 2\sin(\Omega/2)je^{-j\Omega/2}$$

which are lowpass and highpass, respectively.

Define the *average* (lowpass) signal $x_L[n]$

$$\begin{aligned} x_L[n] &= x[n]©g_{haar}[n] \qquad (10.51)\\ &= \{\underline{a+h}, b+a, c+b, d+c, e+d\ldots\} \end{aligned}$$

and the *detail* (highpass) signal $x_H[n]$

$$\begin{aligned} x_H[n] &= x[n]©h_{haar}[n] \qquad (10.52)\\ &= \{\underline{a-h}, b-a, c-b, d-c, e-d\ldots\}. \end{aligned}$$

Next, define the downsampled average signal $x_{LD}[n]$

$$\begin{aligned} x_{LD}[n] &= x_L[2n] \qquad (10.53)\\ &= \{\underline{a+h}, c+b, e+d, g+f\} \end{aligned}$$

and the downsampled detail signal $x_{HD}[n]$

$$\begin{aligned} x_{HD}[n] &= x_H[2n] \qquad (10.54)\\ &= \{\underline{a-h}, c-b, e-d, g-f\}. \end{aligned}$$

The signal $x[n]$ of duration eight has been replaced by the two signals $x_{LD}[n]$ and $x_{HD}[n]$ of durations four each, so no information about $x[n]$ has been lost.

10.5.3 Single-Stage Reconstruction

$x[n]$ can be reconstructed from $x_{LD}[n]$ and $x_{HD}[n]$ as follows. Again, all convolutions are cyclic.

Define the upsampled (zero-stuffed) signal $x_{LDU}[n]$

$$\begin{aligned} x_{LDU}[n] &= \begin{cases} x_{LD}[n/2] & \text{for } n \text{ even}\\ 0 & \text{for } n \text{ odd} \end{cases} \qquad (10.55)\\ &= \{\underline{a+h}, 0, c+b, 0, e+d, 0, g+f, 0\} \end{aligned}$$

and the upsampled (zero-stuffed) signal $x_{HDU}[n]$

$$\begin{aligned} x_{HDU}[n] &= \begin{cases} x_{HD}[n/2] & \text{for } n \text{ even}\\ 0 & \text{for } n \text{ odd} \end{cases} \qquad (10.56)\\ &= \{\underline{a-h}, 0, c-b, 0, e-d, 0, g-f, 0\}. \end{aligned}$$

Note downsampling by 2 followed by upsampling by 2 replaces values of $x[n]$ with zeros for odd times n.

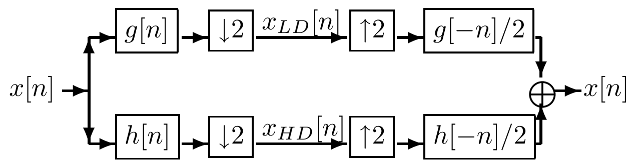
Next, filter $x_{LDU}[n]$ and $x_{HDU}[n]$ with the *synthesis* filters $g_{haar}[-n]$ and $h_{haar}[-n]$, respectively (note these are time-reverals of the *analysis* filters $g_{haar}[n]$ and $h_{haar}[n]$):

$$\begin{aligned} x_{LDU}[n]©g_{haar}[-n] &= \{\underline{a+h}, c+b, c+b, \ldots, a+h\}\\ x_{HDU}[n]©h_{haar}[-n] &= \{\underline{a-h}, b-c, c-b, \ldots, h-a\}. \end{aligned} \qquad (10.57)$$

Adding $x_{LD}[n]$ and $x_{HD}[n]$ and halving the sum gives $x[n]$. Halving can be incorporated by making the synthesis filters be $g_{haar}[-n]/2$ and $h_{haar}[-n]/2$:

$$\begin{aligned} x[n] &= x_{LDU}[n]©g_{haar}[-n]/2\\ &+ x_{HDU}[n]©h_{haar}[-n]/2 \qquad (10.58) \end{aligned}$$

The single-stage Haar decomposition and reconstruction is depicted in Fig. 10-11.



[Replace $g[n]$ and $h[n]$ with $g_{haar}[n]$ and $h_{haar}[n]$.]
Fig. 10-11: Single-Stage Haar Decomposition to $x_{LD}[n]$ and $x_{HD}[n]$, and Reconstruction of $x[n]$.

It is still not evident why this is worth doing. The following example shows why this is worth doing.

Example 10-10: Signal Compression using Haar Decomposition

Consider the finite-duration length-16 signal $x[n]$

$$x[n] = \begin{cases} 4 & \text{for } 0 \le n \le 4\\ 1 & \text{for } 5 \le n \le 9\\ 3 & \text{for } 10 \le n \le 14\\ 4 & \text{for } n = 15 \end{cases}. \qquad (10.59)$$

The Haar-transformed signals $x_{LD}[n]$ and $x_{HD}[n]$ are

$$\begin{aligned} x_{LD}[n] &= \{\underline{8}, 8, 8, 2, 2, 4, 6, 6\} \qquad (10.60)\\ x_{HD}[n] &= \{\underline{0}, 0, 0, 0, 0, 2, 0, 0\}. \end{aligned}$$

$x[n]$ can then be recovered from $x_{LD}[n]$ and $x_{HD}[n]$. The point is that $x_{HD}[n]$ is *sparse* (mostly zero-valued): only $x_{HD}[5]$=2 is nonzero. So the Haar transform allows $x[n]$, which has duration sixteen, to be represented using the eight values of $x_{LD}[n]$ and the single nonzero value (and its location $n = 5$) of $x_{HD}[n]$. This saves almost half of the storage required for $x[n]$. $x[n]$ has been *compressed*: only nine numbers are required to represent a signal of duration sixteen, a reduction of 43% in numbers to be stored.

10.5.4 Signal Compression

This is a simple example of signal *compression*. Although $x[n]$ is not sparse, it can be transformed, using the Haar transform, into a representation with the same number of samples that is sparse, meaning that most of the values of the Haar-transformed signal are zero-valued. This reduces the amount of memory required to store $x[n]$, since zero values need not be stored, only their times n. The few bits (0 or 1) required to store locations of nonzero values are considered to be negligible compared to the many bits required to store the actual nonzero values. Since the Haar transform is orthogonal, $x[n]$ can be recovered perfectly from its Haar-transformed values.

Compression occurred in Example 10-10 because $x[n]$ was piecewise constant. Replacing the Haar functions Eq. (10.49) with the $D2$ Daubechies functions Eq. (10.135) and Eq. (10.136) below transforms piecewise-linear signals $x[n]$ into sparse representations. Using the DL Daubechies functions transforms piecewise-$(L-1)^{th}$-degree polynomial signals into sparse representations. Many signals of real-world interest can be modelled as piecewise-$(L-1)^{th}$-degree polynomial signals, and they can be compressed into sparse representations, from which the original signal can be recovered.

More generally, signals can be transformed into representations consisting of a few large values and many very small values. The very small values require fewer bits of memory to store them, so the total amount of memory required is still reduced. Also, very small values whose absolute values are less than some threshold can be set to zero, for a sparse representation. The $x[n]$ recovered from this thresholded representation will not match the original $x[n]$ exactly, but the error is usually very small. We illustrate this in Example 10-15 below.

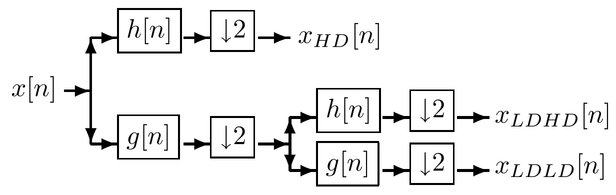
10.5.5 Multistage Decomposition and Reconstruction

Furthermore, $x_{LD}[n]$ can in turn be Haar-transformed into the two signals $x_{LDLD}[n]$ and $x_{LDHD}[n]$ by applying the above transformations to $x_{LD}[n]$ instead of to $x[n]$. Continuing Example 10-10, this gives

$$\begin{aligned} x_{LDLD}[n] &= \{\underline{14}, 16, 4, 10\} \\ x_{LDHD}[n] &= \{\underline{2}, 0, 0, 2\}. \end{aligned} \tag{10.61}$$

$x_{LDHD}[n]$ is again sparse: only two of its four values are nonzero. So $x[n]$ can now be represented by the four values of $x_{LDLD}[n]$, the two nonzero values of $x_{LDHD}[n]$, and the one nonzero value of $x_{HD}[n]$. This reduces by 57% the storage required for $x[n]$.

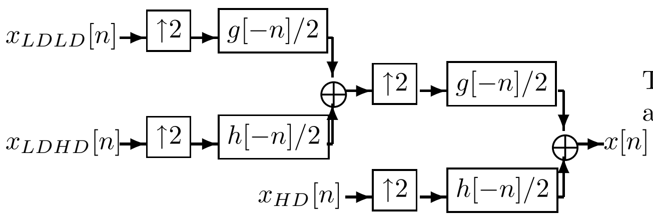
Multistage decomposition is shown in Fig. 10-12.



[Replace $g[n]$ and $h[n]$ with $g_{haar}[n]$ and $h_{haar}[n]$.]
Fig. 10-12. Multistage Haar Analysis Filter Bank.

The average signal $x_{LDLD}[n]$ can in turn be decomposed even further. The result is a *analysis filter bank* that computes the Haar wavelet transform of $x[n]$. This analysis filter bank consists of a series of sections like the left half of Fig. 10-11, connected as in Fig. 10-12, except that each $x_{L\text{any}}$ is further decomposed. The signals computed at the right end of this analysis filter bank constitute the Haar wavelet transform of $x[n]$.

The original signal $x[n]$ can be computed from its Haar wavelet transform using the *synthesis filter bank* shown in Fig. 10-13. This synthesis filter bank consists of a series of sections like the right half of Fig. 10-11, connected in a mirror image of the analysis filter bank Fig. 10-12. The output is the original signal $x[n]$.



[Replace $g[n]$ and $h[n]$ with $g_{haar}[n]$ and $h_{haar}[n]$.]
Fig. 10-13. Multistage Haar Synthesis Filter Bank.

10.5.6 Haar Wavelet Transform by Filter Bank

The Haar wavelet transform can be computed as follows. Let $x[n]$ have duration $N = 2^K$. All convolutions are cyclic.

- Start with $x[n]$. Then:
- $x[n] \rightarrow \boxed{h_{haar}[n]} \rightarrow \boxed{\downarrow 2} \rightarrow \hat{x}_1[n] = x_{HD}[n]$.
  $x[n] \rightarrow \boxed{g_{haar}[n]} \rightarrow \boxed{\downarrow 2} \rightarrow \hat{X}_1[n] = x_{LD}[n]$.
- $\hat{X}_1[n] \rightarrow \boxed{h_{haar}[n]} \rightarrow \boxed{\downarrow 2} \rightarrow \hat{x}_2[n] = x_{LDHD}[n]$.
  $\hat{X}_1[n] \rightarrow \boxed{g_{haar}[n]} \rightarrow \boxed{\downarrow 2} \rightarrow \hat{X}_2[n] = x_{LDLD}[n]$.
- $\hat{X}_2[n] \rightarrow \boxed{h_{haar}[n]} \rightarrow \boxed{\downarrow 2} \rightarrow \hat{x}_3[n]$.
  $\hat{X}_2[n] \rightarrow \boxed{g_{haar}[n]} \rightarrow \boxed{\downarrow 2} \rightarrow \hat{X}_3[n]$.
- Continue K decompositions.

The Haar transform of $x[n]$ consists of the $K + 1$ signals having durations

$$\{\underbrace{\hat{x}_1[n]}_{N/2}, \underbrace{\hat{x}_2[n]}_{N/4}, \underbrace{\hat{x}_3[n]}_{N/8} \ldots \underbrace{\hat{x}_K[n]}_{N/2^K}, \underbrace{\hat{X}_K[n]}_{N/2^K}\}. \qquad (10.62)$$

The total duration of all of the Haar transform signals together is

$$\frac{N}{2} + \frac{N}{4} + \frac{N}{8} + \ldots + \frac{N}{2^K} + \frac{N}{2^K} = N \qquad (10.63)$$

which equals the duration N of $x[n]$.

- The $\hat{x}_k[n]$ are called *detail* signals.
- The $\hat{X}_k[n]$ are called *average* signals.

The inverse Haar wavelet transform can be computed as follows. All convolutions are cyclic.

- Start with $\{\hat{X}_K[n], \hat{x}_K[n], \hat{x}_{K-1}[n] \ldots\}$. Then:
- $\hat{X}_K[n] \rightarrow \boxed{\uparrow 2} \rightarrow \boxed{g_{haar}[-n]/2} \rightarrow A_{K-1}[n]$.
  $\hat{x}_K[n] \rightarrow \boxed{\uparrow 2} \rightarrow \boxed{h_{haar}[-n]/2} \rightarrow B_{K-1}[n]$.
  $\hat{X}_{K-1}[n] = A_{K-1}[n] + B_{K-1}[n]$. Then:
- $\hat{X}_{K-1}[n] \rightarrow \boxed{\uparrow 2} \rightarrow \boxed{g_{haar}[-n]/2} \rightarrow A_{K-2}[n]$.
  $\hat{x}_{K-1}[n] \rightarrow \boxed{\uparrow 2} \rightarrow \boxed{h_{haar}[-n]/2} \rightarrow B_{K-2}[n]$.
  $\hat{X}_{K-2}[n] = A_{K-2}[n] + B_{K-2}[n]$.
- Continue until the final stage:
- $\hat{X}_1[n] \rightarrow \boxed{\uparrow 2} \rightarrow \boxed{g_{haar}[-n]/2} \rightarrow A_0[n]$.
  $\hat{x}_1[n] \rightarrow \boxed{\uparrow 2} \rightarrow \boxed{h_{haar}[-n]/2} \rightarrow B_0[n]$.
  $x[n] = A_0[n] + B_0[n]$ is the original signal.

Of course, if $x[n]$ is piecewise constant, then $x[n]$ could be represented by its constant values and the times at which $x[n]$ jumps. But the Haar decomposition reduces the storage automatically, without requiring any prior knowledge of $x[n]$.

Wavelet decompositions using DL Daubechies functions, instead of Haar functions, reduce storage for signals that are piecewise-$(L-1)^{th}$-degree polynomials, instead of just piecewise constant (see below).

10.5.7 Haar Wavelet Transform in Frequency Domain

Recall from Eq. (10.50) that $\mathbf{G}_{haar}(e^{j\Omega})$ is roughly lowpass and $\mathbf{H}_{haar}(e^{j\Omega})$ is roughly highpass. Then the signals at the first stage of the Haar wavelet transform have spectra

- $\hat{X}_1[n]$ is the lowpass part $0 \leq |\Omega| \leq \frac{\pi}{2}$ of $x[n]$.
- $\hat{x}_1[n]$ is the highpass part $\frac{\pi}{2} \leq |\Omega| \leq \pi$ of $x[n]$.

The signals at the second stage of the Haar wavelet transform have spectra

- $\hat{X}_2[n]$ is the lowpass part $0 \leq |\Omega| \leq \frac{\pi}{2}$ of $\hat{X}_1[n]$, which is the lowpass part $0 \leq |\Omega| \leq \frac{\pi}{4}$ of $x[n]$.
- $\hat{x}_2[n]$ is the highpass part $\frac{\pi}{2} \leq |\Omega| \leq \pi$ of $\hat{X}_1[n]$, which is the bandpass part $\frac{\pi}{4} \leq |\Omega| \leq \frac{\pi}{2}$ of $x[n]$.

The signals at the third stage of the Haar wavelet transform have spectra

- $\hat{X}_3[n]$ is the lowpass part $0 \leq |\Omega| \leq \frac{\pi}{2}$ of $\hat{X}_2[n]$, which is the lowpass part $0 \leq |\Omega| \leq \frac{\pi}{8}$ of $x[n]$.
- $\hat{x}_3[n]$ is the highpass part $\frac{\pi}{2} \leq |\Omega| \leq \pi$ of $\hat{X}_2[n]$, which is the bandpass part $\frac{\pi}{8} \leq |\Omega| \leq \frac{\pi}{4}$ of $x[n]$.

At each stage, downsampling expands the spectrum of each signal to the full range $0 \leq |\Omega| \leq \pi$.

So the Haar wavelet transform decomposes the spectrum $\mathbf{X}(e^{j\Omega})$ of $x[n]$ into *octaves*. The octave $\frac{\pi}{2^k} \leq \Omega \leq \frac{\pi}{2^{k-1}}$ is represented by $\hat{x}_k[n]$. Since the width of this band is $\frac{\pi}{2^k}$, the sampling rate can be reduced by a factor of 2^k using downsampling. The lowest band $0 \leq \Omega \leq \frac{\pi}{2^K}$ is represented by $\hat{X}_K[n]$.

This is illustrated in Fig. 10-14 for $K = 2$. This shows how the Haar wavelet transform roughly filters $x[n]$ into frequency bands, which are sampled at different rates. If the signal or image consists of slowly-varying segments with occasional fast-varying features, as many real-world signals and images do, then $\hat{x}_k[n]$ for small k, which have the most samples, will be sparse, so fewer samples are required to represent the Haar transform of the signal or image

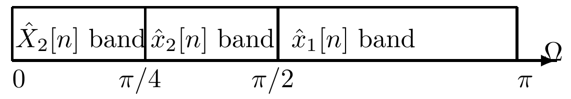


Fig. 10-14. Partitioning of Signal Spectrum by Haar Transform.

10.5.8 Normalized Haar Functions

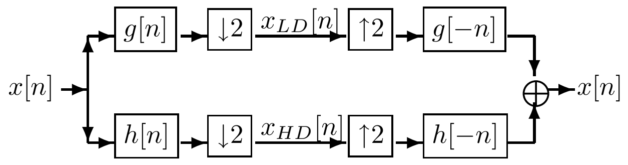
We can avoid having to divide by two in the synthesis stage by replacing Eq. (10.49) with the *normalized* functions

$$\begin{aligned} \tilde{g}_{haar}[n] &= \{\underline{1}, 1\}/\sqrt{2} \qquad (10.64) \\ \tilde{h}_{haar}[n] &= \{\underline{1}, -1\}/\sqrt{2}. \end{aligned}$$

These are the *normalized* Haar *scaling* (lowpass) and *wavelet* (highpass) functions, respectively. Some define the Haar functions as the time reversals of Eq. (10.64). They are also energy-normalized:

$$\sum_{n=0}^{1} \tilde{g}_{haar}[n]^2 = \sum_{n=0}^{1} \tilde{h}_{haar}[n]^2 = 1. \qquad (10.65)$$

Fig. 10-11 can be replaced with Fig. 10-15, in which the divisions by two in the synthesis filters are no longer needed.



[Replace $g[n]$ and $h[n]$ with $\tilde{g}_{haar}[n]$ and $\tilde{h}_{haar}[n]$, and $g[-n]$ and $h[-n]$ with $\tilde{g}_{haar}[-n]$ and $\tilde{h}_{haar}[-n]$]
Fig. 10-15: Single-Stage Haar Decomposition and Reconstruction Using Normalized Functions.

10.5.9 Haar Wavelet Transform by Formulae

Another way to write the wavelet transform is to specify $\hat{x}_k[n]$ and $\hat{X}_k[n]$ directly in terms of $x[n]$. Convolution with $\tilde{g}_{haar}[n]$ and $\tilde{h}_{haar}[n]$, followed by downsampling $y[n] = x[2n]$, repeated k times, gives

$$\begin{aligned} \hat{X}_k[n] &= \sum_{i=0}^{N-1} x[i]\tilde{g}_{haar}^{(k)}[(2^k n - i)\mathrm{mod}(N)] \\ \hat{x}_k[n] &= \sum_{i=0}^{N-1} x[i]\tilde{h}_{haar}^{(k)}[(2^k n - i)\mathrm{mod}(N)] \qquad (10.66) \end{aligned}$$

where filters $\tilde{g}_{haar}^{(k)}[n]$ and $\tilde{h}_{haar}^{(k)}[n]$ can be computed recursively ahead of time using the two formulae

$$\begin{aligned} \tilde{g}_{haar}^{(1)}[n] &= \tilde{g}_{haar}[n] \qquad (10.67) \\ \tilde{h}_{haar}^{(1)}[n] &= \tilde{h}_{haar}[n] \\ \tilde{g}_{haar}^{(k+1)}[n] &= \sum_{i=0}^{L} \tilde{g}_{haar}[i]\tilde{g}_{haar}^{(k)}[(n - 2^k i)\mathrm{mod}(N)] \end{aligned}$$

$$\tilde{h}^{(k+1)}_{haar}[n] = \sum_{i=0}^{L} \tilde{h}_{haar}[i]\tilde{g}^{(k)}_{haar}[(n-2^k i)\text{mod}(N)]$$

where $L = 1$ for $\tilde{g}_{haar}[n]$ and $\tilde{h}_{haar}[n]$. Note that $\tilde{g}^{(k)}_{haar}[n-2^k i]$ appears in both equations of Eq. (10.67), and the convolutions are cyclic.

We derive Eq. (10.66) and Eq. (10.67) for $k = 2$; the extension to larger values of k is similar. To save space, and because these formulae apply for non-Haar scaling and wavelet functions, we omit the subscript *haar* and the mod(N) throughout.

$\hat{X}_1[n]$ is the cyclic convolution of $x[n]$ and $g[n]$, followed by downsampling, which replaces n with $2n$:

$$\hat{X}_1[n] = \sum_{i=0}^{N-1} x[i]\tilde{g}[2n-i]. \tag{10.68}$$

$\hat{X}_2[n]$ is the cyclic convolution of $\hat{X}_1[n]$ and $\tilde{g}[n]$, followed by downsampling, which replaces n with $2n$:

$$\begin{aligned}\hat{X}_2[n] &= \sum_i \hat{X}_1[i]\tilde{g}[2n-i] \\ &= \sum_i\sum_j x[j]\tilde{g}[2i-j]\tilde{g}[2n-i] \\ &= \sum_j x[j]\sum_i \tilde{g}[2i-j]\tilde{g}[2n-i]\end{aligned} \tag{10.69}$$

Changing variables from i to $i' = 2n - i$ gives

$$\begin{aligned}\hat{X}_2[n] &= \sum_j x[j]\sum_{i'} \tilde{g}[2(2n-i')-j]\tilde{g}[i'] \\ &= \sum_j x[j]\sum_{i'} \tilde{g}[4n-2i'-j]\tilde{g}[i'] \\ &= \sum_j x[j]\tilde{g}^{(2)}[4n-j]\end{aligned} \tag{10.70}$$

where $\tilde{g}^{(2)}[n]$ is defined as

$$\tilde{g}^{(2)}[n] = \sum_{i'} \tilde{g}[n-2i']g[i']. \tag{10.71}$$

Replacing n with $4n-j$ gives Eq. (10.67) with $k = 1$.

$x[n]$ can be reconstructed directly from its wavelet transform $\{\hat{X}_K[n], \hat{x}_K[n], \hat{x}_{K-1}[n] \ldots x_1[n]\}$ using

$$\begin{aligned}x[n] &= \sum_{k=1}^{K}\sum_{i=0}^{N-1} \hat{x}_k[i]\tilde{h}^{(k)}_{haar}[(2^k i-n)\text{mod}(N)] \\ &+ \sum_{i=0}^{N-1} \hat{X}_K[i]\tilde{g}^{(K)}_{haar}[(2^K i-n)\text{mod}(N)]\end{aligned} \tag{10.72}$$

Note the time reversal between Eq. (10.66) and Eq. (10.72).

10.5.10 Haar Wavelet Transform by Matrix-Vector Product

The completely-decomposed normalized Haar wavelet transform of a length-eight signal $x[n]$ can be computed using the matrix-vector product

$$\begin{bmatrix}\hat{x}_1[0]\\ \hat{x}_1[1]\\ \hat{x}_1[2]\\ \hat{x}_1[3]\\ \hat{x}_2[0]\\ \hat{x}_2[1]\\ \hat{x}_3[0]\\ \hat{X}_3[0]\end{bmatrix} = \mathcal{H}\begin{bmatrix}x[0]\\ x[1]\\ x[2]\\ x[3]\\ x[4]\\ x[5]\\ x[6]\\ x[7]\end{bmatrix} \tag{10.73}$$

where the matrix $\mathcal{H}$ is defined as $\mathcal{H} = \frac{1}{2\sqrt{2}}\times$

$$\begin{bmatrix} -2 & 2 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & -2 & 2 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & -2 & 2 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & -2 & 2\\ -\sqrt{2} & -\sqrt{2} & \sqrt{2} & \sqrt{2} & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & -\sqrt{2} & -\sqrt{2} & \sqrt{2} & \sqrt{2}\\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\end{bmatrix} \tag{10.74}$$

The inverse length-eight Haar wavelet transform can then be computed using the matrix-vector product

$$\begin{bmatrix}x[0]\\ x[1]\\ x[2]\\ x[3]\\ x[4]\\ x[5]\\ x[6]\\ x[7]\end{bmatrix} = \mathcal{H}^T\begin{bmatrix}\hat{x}_1[0]\\ \hat{x}_1[1]\\ \hat{x}_1[2]\\ \hat{x}_1[3]\\ \hat{x}_2[0]\\ \hat{x}_2[1]\\ \hat{x}_3[0]\\ \hat{X}_3[0]\end{bmatrix}. \tag{10.75}$$

Orthogonality of Haar Wavelet Transform

Note $\mathcal{H}^{-1} = \mathcal{H}^T$, so $\mathcal{H}$ is an *orthogonal* matrix. This means that if $x[n]$ is perturbed to $x[n] + \delta x[n]$, then the wavelet transform $\{\hat{x}_k[n]\}$ of $x[n]$ (including $\hat{X}_0[n]$) is perturbed by $\delta\hat{x}_k[n]$, and the energies of the two perturbations are equal:

$$\sum_{n=0}^{7}(\delta x[n])^2 = \sum_{k=1}^{3}\sum_{n}(\delta\hat{x}_k[n])^2. \tag{10.76}$$

This implies that a small change in any of the $\hat{x}_k[n]$ will result in a small change in the reconstructed $x[n]$. So thresholding small values of $\hat{x}_k[n]$ to zero will have only a small effect on the reconstructed $x[n]$. This is useful for compression, as we shall see in Example 10-15 below.

The decomposition and reconstruction equations and figures presented here for the Haar wavelet transform generalize directly to the general wavelet transform.

The only difference is that $\tilde{g}_{haar}[n]$ and $\tilde{h}_{haar}[n]$ in Eq. (10.64) are replaced with more general scaling $g[n]$ and wavelet $h[n]$ functions, which must satisfy conditions we derive in the next section.

Exercise 10-3: Show that in the duration=8 example at the start of this chapter we can use linear instead of cyclic convolutions by defining downsampling and upsampling differently from $y[n] = x[2n]$ and $x[n/2]$.

Answer:
Use $y[n] = x[2n+1]$ for downsampling and $y[n] = x[(n+1)/2]$ for n odd for upsampling. This $y[n]$ is called a different *polyphase* component. Using linear instead of cyclic convolutions, we get

$$\begin{aligned}
x_L[n] &= \{\underline{a}, b+a, c+b, d+c, e+d\ldots\}\\
x_H[n] &= \{\underline{a}, b-a, c-b, d-c, e-d\ldots\}\\
x_{LD}[n] &= \{\underline{b+a}, d+c, f+e, h+g\}\\
x_{HD}[n] &= \{\underline{b-a}, d-c, f-e, h-g\}\\
x_{LDU}[n]*g[-n] &= \{\underline{b+a}, b+a, d+c, d+c\ldots\}\\
x_{HDU}[n]*h[-n] &= \{\underline{a-b}, b-a, c-d, d-c\ldots\}
\end{aligned}$$

This is a nicer presentation, but it only works for Haar functions, which have duration two.

Exercise 10-4: Show that $\tilde{g}_{haar}[n]$ and $\tilde{h}_{haar}[n]$ are energy-normalized functions.

Answer: From (10.64) we have
$(1/\sqrt{2})^2+(1/\sqrt{2})^2=(1/\sqrt{2})^2+(-1/\sqrt{2})^2=1.$

Exercise 10-5: Confirm $\mathcal{H}^{-1} = \mathcal{H}^T$ in Eq. (10.74).

Solution: Just check $\mathcal{H}\mathcal{H}^T = I$ in Eq. (10.74).

10.6 Non-Haar Wavelet and Scaling Functions

This section generalizes from the Haar functions in (10.64) to the general case of scaling $g[n]$ and wavelet $h[n]$ functions. It derives sufficient conditions on the scaling function $g[n]$ and wavelet function $h[n]$ so that $x[n]$ is reconstructed at the end of the structure in Fig. 10-15.

The wavelet transform is computed using an analysis filter bank that consists of copies of the left half of the structure of Fig. 10-15, recursively decomposed as in Fig. 10-12. Alternatively, the Haar wavelet transform formulae (10.66), now using non-Haar $g[n]$ and $h[n]$, may be used.

$x[n]$ is reconstructed from its wavelet transform using a synthesis filter bank that consists of copies of the right half of the structure of Fig. 10-15, recursively recombined as in Fig. 10-13. Alternatively, the inverse Haar wavelet transform formulae (10.72), now using non-Haar $g[n]$ and $h[n]$, may be used.

So if $g[n]$ and $h[n]$ are such that $x[n]$ is reconstructed at the right end of Fig. 10-15, then $x[n]$ will be reconstructed at the right end of Fig. 10-13 as well. This section derives sufficient conditions for non-Haar $g[n]$ and $h[n]$ to implement this *perfect reconstruction* situation.

We consider only FIR filters $g[n]$ and $h[n]$ having lengths $L+1$, where L is an odd integer. Although the time-reversed impulse responses $g[-n]$ and $h[-n]$ in the synthesis filter bank are noncausal, their delayed versions $g[-(n-L)] = g[L-n]$ and $h[-(n-L)] = h[L-n]$ are causal. Then the output of each stage shown in Fig. 10-15 is delayed by L. Since L is small (around seven) in practice, each filtering step can be implemented very quickly in practice, with transient responses of lengths L. IIR (infinite duration) filters $g[n]$ and $h[n]$ will not be considered in this book.

Historically, the use of tree-structured filter banks to compress signals arose from *subband coding* of speech signals.

10.6.1 Conditions for Single-Stage Perfect Reconstruction

The goal is to determine conditions on $h[n]$ and $g[n]$ so that the output of the filter structure in Fig. 10-15 is indeed $x[n]$. This is called *perfect reconstruction.*

We restate the downsampling and upsampling formulae for DTFT in Section 9-5 on multirate filtering:

$$y[n] = x[2n] \Leftrightarrow \mathbf{Y}(e^{j\Omega}) = \frac{1}{2}[\mathbf{X}(e^{j\Omega/2}) + \mathbf{X}(e^{j\frac{\Omega+2\pi}{2}})] \tag{10.77}$$

for downsampling and

$$y[n] = \begin{cases} x[n/2] & \text{for } n \text{ even} \\ 0 & \text{for } n \text{ odd} \end{cases} \Leftrightarrow \mathbf{Y}(e^{j\Omega}) = \mathbf{X}(e^{j2\Omega}) \tag{10.78}$$

for upsampling. To make the derivation easier to follow, and to maintain consistency with other references, we use "upsampling" to mean "zero-stuffing" and we use DTFT's throughout.

We also replace linear convolutions with cyclic convolutions throughout. This is tantamount to sampling all DTFT's at $\Omega = 2\pi\frac{k}{N}$ where N is the duration of $x[n]$. This replaces all DTFT's with N^{th}-order DFT's. This is sufficient for perfect reconstruction, since $x[n]$ has duration N. But computing the cyclic convolution of a long signal $x[n]$ with a very short filter $g[n]$ or $h[n]$ requires no more computation than a linear convolution.

Referring to Fig. 10-15 above, the DTFT's $\mathbf{X_{HD}}(e^{j\Omega})$ of $x_{HD}[n]$ and $\mathbf{X_{LD}}(e^{j\Omega})$ of $x_{LD}[n]$ are

$$\begin{aligned} \mathbf{X_{HD}}(e^{j\Omega}) &= [\mathbf{H}(e^{j\frac{\Omega}{2}})\mathbf{X}(e^{j\frac{\Omega}{2}}) \\ &+ \mathbf{H}(e^{j\frac{\Omega+2\pi}{2}})\mathbf{X}(e^{j\frac{\Omega+2\pi}{2}})]/2 \\ \mathbf{X_{LD}}(e^{j\Omega}) &= [\mathbf{G}(e^{j\frac{\Omega}{2}})\mathbf{X}(e^{j\frac{\Omega}{2}}) \\ &+ \mathbf{G}(e^{j\frac{\Omega+2\pi}{2}})\mathbf{X}(e^{j\frac{\Omega+2\pi}{2}})]/2 \end{aligned} \tag{10.79}$$

The perfect reconstruction condition is that the DTFT of the output equal the DTFT of the input:

$$\mathbf{X}(e^{j\Omega}) = \mathbf{H}(e^{-j\Omega})\mathbf{X_{HD}}(e^{j2\Omega}) + \mathbf{G}(e^{-j\Omega})\mathbf{X_{LD}}(e^{j2\Omega}). \tag{10.80}$$

Substituting Eq. (10.79) in Eq. (10.80) and noting that $\frac{2\Omega+2\pi}{2} = \Omega + \pi$ gives

$$\begin{aligned} \mathbf{X}(e^{j\Omega}) &= \mathbf{H}(e^{-j\Omega})[\mathbf{H}(e^{j\Omega})\mathbf{X}(e^{j\Omega}) \\ &+ \mathbf{H}(e^{j(\Omega+\pi)})\mathbf{X}(e^{j(\Omega+\pi)})]/2 \\ &+ \mathbf{G}(e^{-j\Omega})[\mathbf{G}(e^{j\Omega})\mathbf{X}(e^{j\Omega}) \\ &+ \mathbf{G}(e^{j(\Omega+\pi)})\mathbf{X}(e^{j(\Omega+\pi)})]/2. \end{aligned} \tag{10.81}$$

Eq. (10.81) must hold for any $\mathbf{X}(e^{j\Omega})$. This means the coefficients of $\mathbf{X}(e^{j\Omega})$ in Eq. (10.81) must sum to zero. This gives the condition

$$\mathbf{H}(e^{-j\Omega})\mathbf{H}(e^{j\Omega}) + \mathbf{G}(e^{-j\Omega})\mathbf{G}(e^{j\Omega}) = 2. \tag{10.82}$$

The coefficients of $\mathbf{X}(e^{j(\Omega+\pi)})$ in Eq. (10.81) must also sum to zero. This gives the condition

$$\mathbf{H}(e^{-j\Omega})\mathbf{H}(e^{j(\Omega+\pi)}) + \mathbf{G}(e^{-j\Omega})\mathbf{G}(e^{j(\Omega+\pi)}) = 0. \tag{10.83}$$

Both Eq. (10.82) and Eq. (10.83) must be satisfied by $\mathbf{G}(e^{j\Omega})$ and $\mathbf{H}(e^{j\Omega})$ for perfect reconstruction to occur.

Quadrature Mirror Filters (QMF's)

Recall $g[n]$ should be a lowpass filter and $h[n]$ a highpass filter. We will show Eq. (10.83) can be satisfied by

$$h[n] = (-1)^n g[L-n] = -(-1)^{L-n} g[L-n] \tag{10.84}$$

where L is any odd integer. In practice, $L+1$ is the duration of $g[n]$, so that $g[L-n]$ and $h[n]$ are causal.

We now show that Eq. (10.84) always satisfies Eq. (10.83).

Using the modulation and time delay properties of the DTFT, noting $e^{-j\pi L} = -1$ if L is odd, and recalling multiple signal transformations from Section 1-2, the frequency responses (DTFT's) of $h[n]$ and $g[n]$ are related by

$$\mathbf{H}(e^{j\Omega}) = -\mathbf{G}(e^{j(\pi-\Omega)})e^{-j\Omega L}. \tag{10.85}$$

If $\mathbf{G}(e^{j\Omega})$ is a lowpass filter, then $\mathbf{H}(e^{j\Omega})$ is a highpass filter since shifting a lowpass filter by $\Omega = \pi$ converts it to a highpass filter. Since their frequency responses are mirror images of each other about the $\Omega{=}\pi/2$ axis, $\mathbf{G}(e^{j\Omega})$ and $\mathbf{H}(e^{j\Omega})$ are called *quadrature mirror filters* (QMF's).

In the sequel, only $g[n]$ and $\mathbf{G}(e^{j\Omega})$ will appear. Then $h[n]$ and $\mathbf{H}(e^{j\Omega})$ are determined by Eq. (10.84) and Eq. (10.85). We use Eq. (10.84) instead of the simpler choice $h[n] = (-1)^n g[n]$ since Eq. (10.84) automatically satisfies Eq. (10.83), while the simpler choice $h[n] = (-1)^n g[n]$ does not. To see this, again use $e^{-j\pi L}{=}{-1}$ if L is odd. Then

$$\begin{aligned}\mathbf{H}(e^{j(\Omega+\pi)}) &= -\mathbf{G}(e^{-j\Omega})e^{-j(\Omega+\pi)L}\\ &= \mathbf{G}(e^{-j\Omega})e^{-j\Omega L}\\ \mathbf{H}(e^{-j\Omega}) &= -\mathbf{G}(e^{j(\pi+\Omega)})e^{j\Omega L} \qquad (10.86)\end{aligned}$$

from which Eq. (10.83) follows. Note the $e^{-j\Omega L}$ factor is necessary to change the sign of the first term, but it cancels otherwise. This is why the odd-valued time shift by the odd integer L in Eq. (10.84) is necessary.

Smith-Barnwell Condition for Perfect Reconstruction

Inserting Eq. (10.86) into Eq. (10.82) gives the *Smith-Barnwell condition for perfect reconstruction*

$$|\mathbf{G}(e^{j\Omega})|^2 + |\mathbf{G}(e^{j(\Omega+\pi)})|^2 = 2. \qquad (10.87)$$

Any $g[n]$ having a DTFT $\mathbf{G}(e^{j\Omega})$ that satisfies Eq. (10.87), with $h[n]$ then determined using Eq. (10.84), will result in perfect reconstruction of $x[n]$ at the output of Fig. 10-15.

10.6.2 Conditions for Single-Stage Perfect Reconstruction using z-Transforms

The above presentation, using the DTFT, is standard for deriving the Smith-Barnwell condition for perfect reconstruction. However, using $\mathbf{z}$-transforms instead of the DTFT throughout gives in important orthogonality relations that must be satisfied by $g[n]$ and $h[n]$ for perfect reconstruction. We now repeat the above derivation using $\mathbf{z}$-transforms.

Downsampling and Upsampling in the z-Transform Domain

The equations in this subsection are related to the corresponding equations in the previous subsection by the relations

$$\begin{aligned}\mathbf{z} &= e^{j\Omega}\\ 1/\mathbf{z} &= e^{-j\Omega}\\ -\mathbf{z} &= e^{j(\Omega+\pi)} = e^{j\Omega}e^{j\pi} \qquad (10.88)\end{aligned}$$

Two real-valued signals $x[n]$ and $y[n]$ are *orthogonal* if their correlation $\sum x[n]y[n] = 0$. Correlation was defined in Eq. (**??**).

Upsampling (zero-stuffing) by 2 is implemented by $\mathbf{Y}(\mathbf{z}) = \mathbf{X}(\mathbf{z}^2)$. To see this, note that

$$\begin{aligned}\mathbf{X}(\mathbf{z}) &= \ldots x[-2]\mathbf{z}^2{+}x[-1]\mathbf{z}{+}x[0]{+}x[1]\mathbf{z}^{-1}{+}x[2]\mathbf{z}^{-2}\ldots\\ \mathbf{X}(\mathbf{z}^2) &= \ldots x[-2]\mathbf{z}^4{+}x[-1]\mathbf{z}^2{+}x[0]{+}x[1]\mathbf{z}^{-2}{+}x[2]\mathbf{z}^{-4}\ldots\\ &= \mathcal{Z}\{\ldots x[-2], 0, x[-1], 0, \underline{x[0]}, 0, x[1], 0, x[2] \qquad (10.89)\end{aligned}$$

Downsampling by 2 is implemented by $\mathbf{Y}(\mathbf{z}) = \mathbf{X}(\mathbf{z}^{1/2})/2 + \mathbf{X}(-\mathbf{z}^{1/2})/2$. To see this, note that

$$\begin{aligned}\mathbf{X}(\mathbf{z}) &= \ldots x[-2]\mathbf{z}^2{+}x[-1]\mathbf{z}{+}x[0]{+}x[1]\mathbf{z}^{-1}{+}x[2]\mathbf{z}^{-2}\ldots\\ \mathbf{X}(-\mathbf{z}) &= \ldots x[-2]\mathbf{z}^2{-}x[-1]\mathbf{z}{+}x[0]{-}x[1]\mathbf{z}^{-1}{+}x[2]\mathbf{z}^{-2}\ldots \qquad (10.90)\end{aligned}$$

and the sum of these two equations is

$$\mathbf{X}(\mathbf{z}) + \mathbf{X}(-\mathbf{z}) = \ldots 2x[-2]\mathbf{z}^2 + 2x[0] + 2x[2]\mathbf{z}^{-2}\ldots \qquad (10.91)$$

All odd powers of $\mathbf{z}$, and $x[n]$ for odd times n, have been eliminated. Since only even powers of $\mathbf{z}$ remain, substituting $\mathbf{z}^{1/2}$ for $\mathbf{z}$ and dividing by two yields the $\mathbf{z}$-transform of $x[n]$ downsampled by 2.

Perfect Reconstruction Derivation using z-Transforms

Referring again to Fig. 10-15 above, the $\mathbf{z}$-transforms $\mathbf{X_{HD}}(\mathbf{z})$ of $x_{HD}[n]$ and $\mathbf{X_{LD}}(\mathbf{z})$ of $x_{LD}[n]$ are

$$\begin{aligned}\mathbf{X_{HD}}(\mathbf{z}) &= [\mathbf{H}(\mathbf{z}^{1/2})\mathbf{X}(\mathbf{z}^{1/2})\\ &+ \mathbf{H}(-\mathbf{z}^{1/2})\mathbf{X}(\mathbf{z}^{1/2})]/2\\ \mathbf{X_{LD}}(\mathbf{z}) &= [\mathbf{G}(\mathbf{z}^{1/2})\mathbf{X}(\mathbf{z}^{1/2})\\ &+ \mathbf{G}(-\mathbf{z}^{1/2})\mathbf{X}(-\mathbf{z}^{1/2})]/2. \qquad (10.92)\end{aligned}$$

These equations are the $\mathbf{z}$-transform versions of Eq. (10.79).

The perfect reconstruction condition is that the $\mathbf{z}$-transform of the output equal the $\mathbf{z}$-transform of the input:

$$\mathbf{X}(\mathbf{z}) = \mathbf{H}(1/\mathbf{z})\mathbf{X_{HD}}(\mathbf{z}^2)+\mathbf{G}(1/\mathbf{z})\mathbf{X_{LD}}(\mathbf{z}^2). \quad (10.93)$$

Substituting Eq. (10.92) in Eq. (10.93) gives

$$\begin{aligned}\mathbf{X}(\mathbf{z}) &= \mathbf{H}(1/\mathbf{z})[\mathbf{H}(\mathbf{z})\mathbf{X}(\mathbf{z}) \\ &+ \mathbf{H}(-\mathbf{z})\mathbf{X}(-\mathbf{z})]/2 \\ &+ \mathbf{G}(1/\mathbf{z})[\mathbf{G}(\mathbf{z})\mathbf{X}(\mathbf{z}) \\ &+ \mathbf{G}(-\mathbf{z})\mathbf{X}(-\mathbf{z})]/2. \quad (10.94)\end{aligned}$$

Eq. (10.94) must hold for any $\mathbf{X}(\mathbf{z})$. This means the coefficients of $\mathbf{X}(\mathbf{z})$ in Eq. (10.94) must sum to zero. This gives the condition

$$\mathbf{H}(1/\mathbf{z})\mathbf{H}(\mathbf{z}) + \mathbf{G}(1/\mathbf{z})\mathbf{G}(\mathbf{z}) = 2. \quad (10.95)$$

The coefficients of $\mathbf{X}(-\mathbf{z})$ in Eq. (10.94) must also sum to zero. This gives the condition

$$\mathbf{H}(1/\mathbf{z})\mathbf{H}(-\mathbf{z}) + \mathbf{G}(1/\mathbf{z})\mathbf{G}(-\mathbf{z}) = 0. \quad (10.96)$$

Both Eq. (10.95) and Eq. (10.96) must be satisfied by $\mathbf{G}(\mathbf{z})$ and $\mathbf{H}(\mathbf{z})$ for perfect reconstruction to occur.

Eq. (10.96) is satisfied by setting

$$\mathbf{H}(\mathbf{z}) = -\mathbf{G}(-1/\mathbf{z})\mathbf{z}^{-L}. \quad (10.97)$$

Note that the factor of $\mathbf{z}^{-L}$, with L an odd integer, is needed so that the substitution $\mathbf{z} \to -\mathbf{z}$ changes the sign of the right side of Eq. (10.97). This is more evident than in Eq. (10.85). The inverse $\mathbf{z}$-transform of Eq. (10.97) is Eq. (10.84). This shows where Eq. (10.84) comes from.

Substituting Eq. (10.97) in Eq. (10.95) gives the Smith-Barnwell condition for perfect reconstruction in $\mathbf{z}$-transform form:

$$\mathbf{G}(\mathbf{z})\mathbf{G}(1/\mathbf{z}) + \mathbf{G}(-\mathbf{z})\mathbf{G}(-1/\mathbf{z}) = 2. \quad (10.98)$$

10.6.3 Orthogonality Relations

The point of using $\mathbf{z}$-transforms will now become evident. We show that:

(1) $g[n]$ is orthogonal to nonzero even-valued translations of itself;

(2) $h[n]$ is orthogonal to nonzero even-valued translations of itself;

(3) $g[n]$ and $h[n]$ are orthogonal to even-valued translations of each one relative to the other.

Hence the inverse wavelet formula Eq. (10.72) is an orthogonal expansion of $x[n]$ in orthogonal basis functions (remember the downsampling).

From Section 9-6, the autocorrelation of $g[n]$ is

$$r_g[n] = g[n] * g[-n]. \quad (10.99)$$

Using the time-reversal property of $\mathbf{z}$-transforms (see Table 7-6), the $\mathbf{z}$-transform of this is

$$\mathbf{R_g}(\mathbf{z}) = \mathbf{G}(\mathbf{z})\mathbf{G}(1/\mathbf{z}). \quad (10.100)$$

The $\mathbf{z}$-transform of the Smith-Barnwell condition is

$$\mathbf{R_g}(\mathbf{z}) + \mathbf{R_g}(-\mathbf{z}) = 2. \quad (10.101)$$

Using Eq. (10.91), the inverse $\mathbf{z}$-transform of the Smith-Barnwell condition Eq. (10.98) is just $r_g[n] = \delta[n]$ for n even. This means that $g[n]$ *must be orthogonal to (nonzero) even-valued translations of itself* for perfect reconstruction:

$$\sum_i g[i]g[i-2n] = \delta[2n] = \delta[n]. \quad (10.102)$$

Furthermore, if $h[n]$ is determined using Eq. (10.84), then $h[n]$ *must be orthogonal to (nonzero) even-valued translations of itself* for perfect reconstruction. To see this, note that $(-1)^2$=1 and change variables from i to $i' = L - i$ in the middle of the following:

$$\begin{aligned}\sum_i h[i]h[i-2n] &= \sum_i g[L-i]g[L-i+2n] \\ &= \sum_{i'} g[i']g[i'+2n] \\ &= \delta[n]. \quad (10.103)\end{aligned}$$

Finally, if $h[n]$ is determined using Eq. (10.84), then $h[n]$ and $g[n]$ *are orthogonal for all even-valued translations of one relative to the other:*

$$\sum_i g[i]h[i-2n] = 0. \quad (10.104)$$

The derivation of this relation is much longer, and is omitted here.

All of these orthogonality relations require that the Smith-Barnwell condition Eq. (10.98) be satisfied.

The next section shows how to construct wavelet scaling functions $g[n]$ having DTFT's $\mathbf{G}(e^{j\Omega})$ that satisfy the Smith-Barnwell condition Eq. (10.87).

Exercise 10-6: Show that the normalized Haar scaling function $\tilde{g}_{haar}[n]$ in Eq. (10.64) satisfies the Smith-Barnwell condition Eq. (10.87).

Solution: From Eq. (10.50) we have

$$\begin{aligned}
|\tilde{\mathbf{G}}_{haar}(e^{j\Omega})| &= |(1+e^{-j\Omega})/\sqrt{2}| \\
&= \sqrt{2}|\cos(\Omega/2)| \cdot |e^{-j\Omega/2}| \\
&= \sqrt{2}|\cos(\Omega/2)|
\end{aligned}$$

and $|\tilde{\mathbf{G}}_{haar}(e^{j(\Omega+\pi)})| = \sqrt{2}|-\sin(\Omega/2)|$. Then

$$\begin{aligned}
|\mathbf{G}(e^{j\Omega})|^2 + |\mathbf{G}(e^{j(\Omega+\pi)})|^2 &= \\
2\cos^2(\Omega/2) + 2\sin^2(\Omega/2) &= 2.
\end{aligned}$$

Exercise 10-7: Show that Eq. (10.84) with $L=1$ holds for the normalized Haar scaling and wavelet basis functions in Eq. (10.64).

Solution: $\tilde{g}_{haar}[n] = [\underline{1}, 1]/\sqrt{2}$ implies that $(-1)^n g[1-n] = [\underline{1}, -1]/\sqrt{2} = \tilde{h}_{haar}[n]$.

10.7 Derivation of Daubechies Wavelet Scaling Function

The most commonly used wavelet transform uses the Daubechies scaling and wavelet functions. The DL Daubechies scaling function $g[n]$ is characterized by its $\mathbf{z}$-transform $\mathbf{G}(\mathbf{z})$ having L zeros at $\mathbf{z}=-1$, in addition to satisfying the Smith-Barnwell condition Eq. (10.98). From Eq. (10.97), the DL Daubechies wavelet function $h[n]$ is characterized by its $\mathbf{z}$-transform $\mathbf{H}(\mathbf{z})$ having L zeros at $\mathbf{z}=1$. We show below that signals consisting of segments of various polynomials of degree less than L are sparsified (made mostly zero-valued) by the wavelet transform using DL Daubechies wavelet functions. We note that some use $D2L$, instead of DL, to designate Daubechies wavelet functions with L zeros at $\mathbf{z}=1$.

The next subsection derives this sparsification, and the following subsection shows how to compute the DL Daubechies scaling function. An example computes the $D2$ Daubechies scaling function.

The next subsection is an insert.

10.7.1 Multiple Zeros at One Sparsify Piecewise-Polynomial Signals

Definition of Piecewise-Polynomial Signals

A signal $x[n]$ is defined to be *piecewise-L^{th}-degree polynomial* if it has the form

$$\begin{aligned}
x[n] &= \sum_{k=0}^{L} a_{0,k} n^k, -\infty < n \leq N_0 \\
&= \sum_{k=0}^{L} a_{1,k} n^k, N_0 < n \leq N_1 \\
&= \sum_{k=0}^{L} a_{2,k} n^k, N_1 < n \leq N_2 \\
&\vdots \quad \vdots
\end{aligned} \tag{10.105}$$

$x[n]$ can be segmented into intervals, and in each interval $x[n]$ is a polynomial in time n of degree L. The times N_i at which the coefficients $\{a_{i,k}\}$ change values are sparse, meaning that they are scattered over time n. In continuous time, such a signal would be a *spline*, except that for a spline the derivatives of the signal must match at the *knots* (the times where the coefficients $\{a_{i,k}\}$ change values). The idea here is that the coefficients $\{a_{i,k}\}$ can change completely at the times N_i; there is no "smoothness" requirement. Indeed, these times N_i are the edges of $x[n]$.

We examine first piecewise-constant, then piecewise-linear, then piecewise polynomial signals, and the effect of zeros at $\mathbf{z}=1$ in each case.

Piecewise-Constant Signals

Let a transfer function $\mathbf{H}(\mathbf{z})$ have a zero at $\mathbf{z} = 1$. Then $\mathbf{H}(\mathbf{z})$ has the form

$$\mathbf{H}(\mathbf{z}) = (\mathbf{z} - 1)\mathbf{Q}(\mathbf{z}) \qquad (10.106)$$

for some rational function $\mathbf{Q}(\mathbf{z})$. The inverse **z**-transform of $(\mathbf{z} - 1)$ is $\{1, \underline{-1}\}$. The overall impulse response of LTI systems connected in series is the convolution of the impulse responses (see Section 2-5.2). So the system with transfer function $\mathbf{H}(\mathbf{z})$ can be implemented by two systems connected in series:

- $x[n] \rightarrow \boxed{w_1[n] = x[n+1] - x[n]} \rightarrow w_1[n]$
- $w_1[n] \rightarrow \boxed{q[n]} \rightarrow y[n]$

where $q[n]$ is the inverse **z**-transform of $\mathbf{Q}(\mathbf{z})$.

Now let $x[n]$ be piecewise constant, meaning that $x[n]$ has the form

$$\begin{aligned} x[n] &= a_0, -\infty < n \le N_0 \\ &= a_1, N_0 < n \le N_1 \\ &= a_2, N_1 < n \le N_2 \\ &\vdots \quad \vdots \end{aligned} \qquad (10.107)$$

The value of $x[n]$ changes only at a few scattered times. The amount by which $x[n]$ changes between $n = N_i$ and $n = N_i + 1$ is the jump $a_{i+1} - a_i$:

$$x[n+1] - x[n] = \begin{cases} 0 & \text{for } n \neq N_i \\ a_{i+1} - a_i & \text{for } n = N_i \end{cases} \qquad (10.108)$$

This can be restated as

$$x[n+1] - x[n] = \sum_i (a_{i+1} - a_i)\delta[n - N_i] \qquad (10.109)$$

This is illustrated in Fig. 10-18.

Then $x[n]$ is sparsified by the system with transfer function $\mathbf{H}(\mathbf{z})$, since

$$\begin{aligned} x[n] * h[n] &= x[n] * \{1, \underline{-1}\} * q[n] && (10.110) \\ &= (x[n+1] - x[n]) * q[n]. \\ &= \sum_i (a_{i+1} - a_i)\delta[n - N_i] * q[n] \\ &= \sum_i (a_{i+1} - a_i) q[n - i] && (10.111) \end{aligned}$$

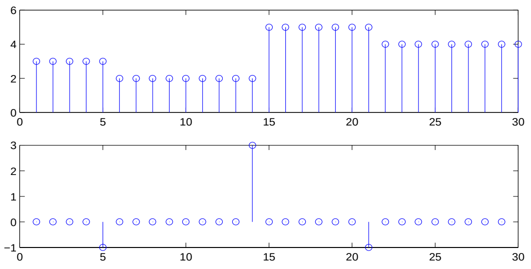


Figure 10.16: A System with a Zero at $\mathbf{z} = 1$ Compresses Piecewise-Constant Signals. Top: A Piecewise-Constant Signal. Bottom: Differences of the Signal.

using the time-shift property of convolution (entry #5 in Table #7-4). In practice, $q[n]$ will turn out to be a very short (length 3 or 4) FIR filter, so $x[n]*h[n]$ is still mostly zero-valued.

The Haar wavelet function $\tilde{h}_{haar}[n] = \frac{1}{\sqrt{2}}\{\underline{1}, -1\}$ has transfer function $\tilde{\mathbf{H}}_{\mathbf{haar}}(\mathbf{z}) = (\mathbf{z} - 1)/(\sqrt{2}\mathbf{z})$, which has a zero at $\mathbf{z} = 1$. So the Haar wavelet transform compresses piecewise constant signals. For the Haar wavelet function, $\mathbf{Q}(\mathbf{z}) = 1//(\sqrt{2}\mathbf{z})$.

Piecewise-Linear Signals

Let a transfer function $\mathbf{H}(\mathbf{z})$ have two zeros at $\mathbf{z} = 1$. Then $\mathbf{H}(\mathbf{z})$ has the form

$$\mathbf{H}(\mathbf{z}) = (\mathbf{z} - 1)^2\mathbf{Q}(\mathbf{z}) \qquad (10.112)$$

for some rational function $\mathbf{Q}(\mathbf{z})$. The inverse **z**-transform of $(\mathbf{z} - 1)$ is $\{1, \underline{-1}\}$, so the inverse **z**-transform of $(\mathbf{z} - 1)^2$ is

$$\mathcal{Z}^{-1}\{(\mathbf{z} - 1)^2\} = \{1, \underline{-1}\} * \{1, \underline{-1}\}. \qquad (10.113)$$

The overall impulse response of LTI systems connected in series is the convolution of the impulse responses (see Section 2-5.2). So the system with transfer function $\mathbf{H}(\mathbf{z})$ can be implemented by three systems connected in series:

- $x[n] \rightarrow \boxed{w_1[n] = x[n+1] - x[n]} \rightarrow w_1[n]$
- $w_1[n] \rightarrow \boxed{w_2[n] = w_1[n+1] - w_1[n]} \rightarrow w_2[n]$

- $w_2[n] \rightarrow \boxed{q[n]} \rightarrow y[n]$

where $q[n]$ is the inverse $\mathbf{z}$-transform of $\mathbf{Q}(\mathbf{z})$.

Now let $x[n]$ be piecewise linear, meaning that $x[n]$ has the form

$$\begin{aligned} x[n] &= a_{0,1}n + a_{0,0}, -\infty < n \leq N_0 \\ &= a_{1,1}n + a_{1,0}, N_0 < n \leq N_1 \\ &= a_{2,1}n + a_{2,0}, N_1 < n \leq N_2 \\ \vdots &\quad \vdots \end{aligned} \tag{10.114}$$

Proceeding as in the case of piecewise-constant $x[n]$, taking differences, and then taking differences of the differences, will sparsify a piecewise-linear signal. This is illustrated in Fig. 10-19.

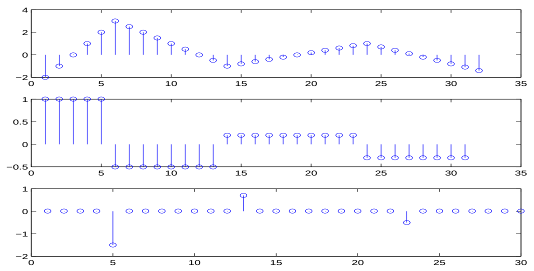


Figure 10.17: A System with Two Zeros at $\mathbf{z} = 1$ Compresses Piecewise-Linear Signals. Top: A Piecewise-Linear Signal. Middle: Differences of the Top Signal. Bottom: Differences of the Middle Signal.

The bottom signal in Fig. 10-19 is in turn convolved with $q[n]$, resulting in a series of scaled and delayed $q[n]$'s In practice, $q[n]$ will turn out to be a very short (length 3 or 4) FIR filter, so $x[n] * h[n]$ is still mostly zero-valued.

10.7.2 Piecewise-Polynomial Signals and Multiple Zeros at $\mathbf{z} = 1$

Let a system has L zeros at $\mathbf{z} = 1$. Then its transfer function $\mathbf{H}(\mathbf{z})$ must have the form

$$\mathbf{H}(\mathbf{z}) = (\mathbf{z} - 1)^L \mathbf{Q}(\mathbf{z}) \tag{10.115}$$

for some rational function $\mathbf{Q}(\mathbf{z})$. The system can be regarded as a series connection of the $L + 1$ systems

- $x[n] \rightarrow \boxed{w_1[n] = x[n+1] - x[n]} \rightarrow w_1[n]$
- $w_1[n] \rightarrow \boxed{w_2[n] = w_1[n+1] - w_1[n]} \rightarrow w_2[n]$
- $w_2[n] \rightarrow \boxed{w_3[n] = w_2[n+1] - w_2[n]} \rightarrow w_3[n] \ldots$
- $w_L[n] \rightarrow \boxed{q[n]} \rightarrow y[n]$

where $q[n]$ is the inverse $\mathbf{z}$-transform of $\mathbf{Q}(\mathbf{z})$.

If $x[n]$ is a monic polynomial n^{L-1} of degree $L - 1$ then $w_1[n]$ will be a polynomial of degree $L - 2$, since

$$\begin{aligned} w_1[n] &= x[n+1] - x[n] \\ &= (n+1)^{L-1} - n^{L-1} \\ &= (L-1)n^{L-2} + (L-1)(L-2)/2n^{L-3} + \ldots \end{aligned} \tag{10.116}$$

Each successive difference system reduces the degree of its input by one. $w_{L-1}[n]$ will have degree zero–it is constant. Then $w_L[n]$, and hence $y[n]$, will be zero. This is sometimes referred to as the *moments* $\sum_{n=-\infty}^{\infty} h[n]n^k = 0$ for $k = 0, 1 \ldots L - 1$.

Now let $x[n]$ be any polynomial of degree $L-1$. The term of highest degree will produce a zero output, and terms of lower degrees will also produce zero outputs. Since the system is linear, the system will eliminate any input that is a polynomial of degree $L - 1$.

The continuous-time analogue of this is that

$$y(t) = \frac{d^L x}{dt^L} = 0 \tag{10.117}$$

for any $(L-1)^{th}$-degree polynomial $x(t)$. Note the transfer function $\mathbf{H}(\mathbf{s}) = \mathbf{s}^L$ has L zeros at the origin.

Signals that are piecewise-polynomial functions of time n of degree $L - 1$ will be compressed to zero, except in the vicinity of the times at which the polynomial coefficients change. At these times, scaled and delayed versions of the impulse response $q[n] = \mathcal{Z}^{-1}\{\mathbf{Q}(\mathbf{z})\}$ will appear. In practice $q[n]$ is a short FIR filter of length 3 or 4, so $y[n]$ is still mostly zero.

End of insert.

10.7.3 Significance of Multiple Zeros at $\mathbf{z} = 1$

The Haar wavelet transform compresses signals that are piecewise constant, since

$$x[n] * \tilde{h}_{haar}[n] = (x[n] - x[n-1])/\sqrt{2} \qquad (10.118)$$

is zero at times n when $x[n]$ is constant and nonzero only at times n when $x[n]$ changes. The transfer function $\tilde{\mathbf{H}}_{\mathbf{haar}}(\mathbf{z}) = (\mathbf{z} - 1)/(\sqrt{2}\mathbf{z})$ has a zero at $\mathbf{z} = 1$.

Let a system has L zeros at $\mathbf{z} = 1$. Then its transfer function $\mathbf{H}(\mathbf{z})$ must have the form

$$\mathbf{H}(\mathbf{z}) = (\mathbf{z} - 1)^L \mathbf{Q}(\mathbf{z}) \qquad (10.119)$$

for some rational function $\mathbf{Q}(\mathbf{z})$. The system can be regarded as a series connection of the systems

- $x[n] \rightarrow \boxed{w_1[n] = x[n+1] - x[n]} \rightarrow w_1[n]$
- $w_1[n] \rightarrow \boxed{w_2[n] = w_1[n+1] - w_1[n]} \rightarrow w_2[n]$
- $w_2[n] \rightarrow \boxed{w_3[n] = w_2[n+1] - w_2[n]} \rightarrow w_3[n] \ldots$
- $w_L[n] \rightarrow \boxed{\mathbf{Q}(\mathbf{z})} \rightarrow y[n]$

If $x[n]$ is a monic polynomial n^{L-1} of degree $L-1$ then $w_1[n]$ will be a polynomial of degree $L-2$, since

$$(n+1)^{L-1} - n^{L-1} = (L-1)n^{L-2} + \ldots \qquad (10.120)$$

Each successive difference system reduces the degree of its input by one. $w_{L-1}[n]$ will have degree zero–it is constant. Then $w_L[n]$, and hence $y[n]$, will be zero, regardless of what $\mathbf{Q}(\mathbf{z})$ is.

Now let $x[n]$ be any polynomial of degree $L-1$. The term of highest degree will produce a zero output, and terms of lower degrees will also produce zero outputs. Since the system is linear, the system will eliminate any input that is a polynomial of degree $L-1$ or lower.

Hence systems with L zeros at $\mathbf{z} = 1$ will compress to zero inputs that are polynomial functions of n of degree $L-1$. Signals that are piecewise-polynomial functions of time n of degree $L-1$ will be compressed to zero, except in the vicinity of the times at which the polynomial coefficients change. The transient response of the overall system $\mathbf{H}(\mathbf{z})$ will appear at these times. But the transient response has short duration vs. $x[n]$, so $y[n]$ is still mostly zero.

The continuous-time analogue of this is that

$$y(t) = \frac{d^L x}{dt^L} = 0 \qquad (10.121)$$

for any $(L-1)^{th}$-degree polynomial $x(t)$. Note the transfer function $\mathbf{H}(\mathbf{s}) = \mathbf{s}^L$ has L zeros at the origin.

From Eq. (10.97), the $\mathbf{z}$-transform $\mathbf{H}(\mathbf{z})$ of the wavelet function $h[n]$ will have L zeros at $\mathbf{z} = 1$ if the $\mathbf{z}$-transform $\mathbf{G}(\mathbf{z})$ of the scaling function $g[n]$ has L zeros at $\mathbf{z} = -1$.

We now derive the *DL* Daubechies scaling function, whose $\mathbf{z}$-transform has L zeros at $\mathbf{z}$=–1 and also satisfies the Smith-Barnwell condition Eq. (10.98) for perfect reconstruction. The *DL* Daubechies scaling function compresses piecewise-polynomial signals of degree $L-1$ to zero, except near the times n at which the polynomial coefficients change.

10.7.4 Computation of the *DL* Daubechies Scaling Function

Let $g[n]$ be the *DL* Daubechies scaling function. For its $\mathbf{z}$-transform $\mathbf{G}(\mathbf{z})$ to have L zeros at $\mathbf{z} = -1$, and for $g[n]$ to be causal, $\mathbf{G}(\mathbf{z})$ must have the form

$$\mathbf{G}(\mathbf{z}) = (1 + \mathbf{z}^{-1})^L \mathbf{Q}(\mathbf{z}) \qquad (10.122)$$

for some polynomial $\mathbf{Q}(\mathbf{z})$ in $\mathbf{z}^{-1}$ (so $q[n]$ and $g[n]$ will be causal) of degree $L-1$. After specifying the scaling function design procedure, it will be evident why $\mathbf{Q}(\mathbf{z})$ should have degree $L-1$ (see below).

$\mathbf{G}(\mathbf{z})$ must also satisfy the Smith-Barnwell condition Eq. (10.98), rewritten as

$$\mathbf{G}(\mathbf{z})\mathbf{G}(\mathbf{z}^{-1}) + \mathbf{G}(-\mathbf{z})\mathbf{G}(-\mathbf{z}^{-1}) = 2. \qquad (10.123)$$

Now define $r[n]$ as the autocorrelation of $q[n]$

$$r[n] = r[-n] = q[n] * q[-n]. \qquad (10.124)$$

The $\mathbf{z}$-transform $\mathbf{R}(\mathbf{z})$ of $r[n]$ is

$$\mathbf{R}(\mathbf{z}) = \mathbf{R}(1/\mathbf{z}) = \mathbf{Q}(\mathbf{z})\mathbf{Q}(1/\mathbf{z}). \qquad (10.125)$$

Substituting Eq. (10.122) and Eq. (10.125) in Eq. (10.123) gives

$$(1+\mathbf{z})^L(1+\mathbf{z}^{-1})^L\mathbf{R}(\mathbf{z})+(1-\mathbf{z})^L(1-\mathbf{z}^{-1})^L\mathbf{R}(-\mathbf{z}) = 2. \tag{10.126}$$

This is a linear system of equations of size L in the unknown $r[n]$. Solving this system determines $\mathbf{R}(\mathbf{z})$.

Now that the complete procedure has been specified, we see why $\mathbf{Q}(\mathbf{z})$ should have degree $L-1$:

- From Eq. (10.122), $\mathbf{G}(\mathbf{z})$ has degree $2L-1$.
- From Eq. (10.125), $\mathbf{R}(\mathbf{z})$ has degree $2L-2$.
- In Eq. (10.126), each term has degree $4L-2$.
- In Eq. (10.126), equations for negative powers of $\mathbf{z}$ are identical to those for positive powers of $\mathbf{z}$. So the coefficients of odd powers of $\mathbf{z}$ cancel.
- Eq. (10.126) is L equations in L unknowns $r[n]$.

Next, we show how to compute $\mathbf{Q}(\mathbf{z})$ from $\mathbf{R}(\mathbf{z})$.

From Eq. (10.125), $\mathbf{R}(\mathbf{z}) = \mathbf{R}(1/\mathbf{z})$, so if $\mathbf{z}_o$ is a zero of $\mathbf{R}(\mathbf{z})$ then $1/\mathbf{z}_o$ is also a zero of $\mathbf{R}(\mathbf{z})$. Since $r[n]$ is real-valued, the zeros also occur in complex conjugate pairs: $\mathbf{z}_o^*$ is also a zero. Combining these, the zeros of $\mathbf{R}(\mathbf{z})$ occur in *conjugate reciprocal quadruples*, each of which has the form, for some zero $\mathbf{z}_o$,

$$\{\mathbf{z}_o, \mathbf{z}_o^*, 1/\mathbf{z}_o, 1/\mathbf{z}_o^*\}$$

$\mathbf{Q}(\mathbf{z})$ is then computed from $\mathbf{R}(\mathbf{z})$ by performing a *spectral factorization* of $\mathbf{R}(\mathbf{z})$, as follows:

- Zeros of $\mathbf{R}(\mathbf{z})$ inside the unit circle ($|\mathbf{z}_o| < 1$) are assigned to be the zeros of $\mathbf{Q}(\mathbf{z})$.
- Zeros of $\mathbf{R}(\mathbf{z})$ outside the unit circle ($|\mathbf{z}_o| > 1$) are assigned to be the zeros of $\mathbf{Q}(1/\mathbf{z})$.

This determines the minimum-phase $\mathbf{Q}(\mathbf{z})$, hence $\mathbf{G}(\mathbf{z})$, to a scale factor, which is determined by Eq. (10.126).

Example 10-11: Computation of the $D2$ Daubechies Scaling Function

For $L = 2$, $\mathbf{R}(\mathbf{z})$ can be written as

$$\mathbf{R}(\mathbf{z}) = r[1]\mathbf{z} + r[0] + r[1]\mathbf{z}^{-1} \tag{10.127}$$

since $r[n] = r[-n]$ from Eq. (10.124). Then Eq. (10.126) becomes

$$(1+\mathbf{z})^2(1+\mathbf{z}^{-1})^2(r[1]\mathbf{z}+r[0]+r[1]\mathbf{z}^{-1}) + (1-\mathbf{z})^2(1-\mathbf{z}^{-1})^2(-r[1]\mathbf{z}+r[0]-r[1]\mathbf{z}^{-1}) = 2. \tag{10.128}$$

Writing this equation out in detail gives

$$\begin{aligned} (\mathbf{z}^2+4\mathbf{z}+6+4\mathbf{z}^{-1}+\mathbf{z}^{-2}) &\quad \times \\ (r[1]\mathbf{z}+r[0]+r[1]\mathbf{z}^{-1}) &\quad + \\ (\mathbf{z}^2-4\mathbf{z}+6-4\mathbf{z}^{-1}+\mathbf{z}^{-2}) &\quad \times \\ (-r[1]\mathbf{z}+r[0]-r[1]\mathbf{z}^{-1}) &= 2. \end{aligned} \tag{10.129}$$

In Eq. (10.123), the coefficients of odd powers of $\mathbf{z}$ automatically cancel out, and the coefficients of negative powers of $\mathbf{z}$ are identical to the coefficients of the corresponding positive powers of $\mathbf{z}$. So there are only two non-trivial equations. Equating coefficients of $\mathbf{z}^2$ and $\mathbf{z}^0$, respectively, gives the two linear equations

$$\begin{aligned} 2r[0]+8r[1] &= 0 \\ 12r[0]+16r[1] &= 2 \end{aligned} \tag{10.130}$$

which has the solution

$$r[0] = 1/4; \quad r[1] = -1/16. \tag{10.131}$$

Then $\mathbf{R}(\mathbf{z})$ is

$$\mathbf{R}(\mathbf{z}) = -(1/16)\mathbf{z}^{-1} + 1/4 - (1/16)\mathbf{z}. \tag{10.132}$$

The two zeros of $\mathbf{R}(\mathbf{z})$ (the roots of $\mathbf{R}(\mathbf{z})$=0) are

$$\begin{aligned} \mathbf{z}_o = \mathbf{z}_o^* &= 2-\sqrt{3} = 0.2679 \\ \frac{1}{\mathbf{z}_o} = \frac{1}{\mathbf{z}_o^*} &= 2+\sqrt{3} = 3.7321. \end{aligned} \tag{10.133}$$

The complex conjugate quadruple is just a pair, since both zeros are real-valued.

The spectral factorization of $\mathbf{R}(\mathbf{z})$ is performed by choosing $\mathbf{z_o}$ to be the zero of $\mathbf{Q}(\mathbf{z})$, since $|\mathbf{z_o}| < 1$. Then Eq. (10.122) becomes, for some constant C,

$$\begin{aligned} \mathbf{G}(\mathbf{z}) &= C(1+\mathbf{z}^{-1})^2(1-0.2679\mathbf{z}^{-1}) \qquad (10.134) \\ &= C(1+1.7321\mathbf{z}^{-1}+0.4642\mathbf{z}^{-2}-0.2679\mathbf{z}^{-3}). \end{aligned}$$

The constant C is computed by inserting this expression for $\mathbf{G}(\mathbf{z})$ into Eq. (10.123). This gives

$C = 0.4830$. Then the $D2$ Daubechies scaling function $g[n]$ is

$$g[n] = \{\underline{.4830}, .8365, .2242, -.1294\}. \qquad (10.135)$$

The causal $D2$ Daubechies wavelet function is then

$$h[n] = \{\underline{-.1294}, -.2242, .8365, -.4830\}. \qquad (10.136)$$

Note that $g[n]$ and $h[n]$ are energy-normalized:

$$\sum_{n=0}^{3} g[n]^2 = \sum_{n=0}^{3} h[n]^2 = 1. \qquad (10.137)$$

The coefficients of the $D1$, $D2$, $D3$ and $D4$ Daubechies scaling functions are listed in Table 10-1. The corresponding wavelet functions can be obtained from these scaling functions using Eq. (10.84).

$g[n]$	D1	D2	D3	D4
$g[0]$	.7071	.4830	.3327	.2304
$g[1]$	.7071	.8365	.8069	.7148
$g[2]$	0	.2241	.4599	.6309
$g[3]$	0	−.1294	−.1350	−.0280
$g[4]$	0	0	−.0854	−.1870
$g[5]$	0	0	.0352	.0308
$g[6]$	0	0	0	.0329
$g[7]$	0	0	0	−.0106

Table 10-1. Coefficients of Daubechies Scaling Functions.

10.7.5 Amount of Computation

The total amount of computation required to compute the DL Daubechies wavelet transform of a signal $x[n]$ of duration N can be computed as follows. The duration of the DL Daubechies wavelet and scaling functions is $2L$. Convolving both of these with $x[n]$ requires $2(2L)N = 4LN$ multiplications-and-additions (MAD's). But since the results will be downsampled by two, only half of the convolution outputs must be computed, halving this to $2LN$.

At each successive decomposition, these functions are convolved with the average signal from the previous stage. So these functions are convolved with the signals, with respective durations,

$$\{\underbrace{\hat{X}_1[n]}_{N/2}, \underbrace{\hat{X}_2[n]}_{N/4}, \ldots, \underbrace{\hat{X}_K[n]}_{N/2^K}\}.$$

The total number of MAD's required is thus

$$2L\left(N + \frac{N}{2} + \frac{N}{4} + \ldots + \frac{N}{2^K}\right) < 4LN. \qquad (10.138)$$

The additional computation for computing more decompositions (i.e., increasing K) is minimal.

Since L is small, this is comparable to the amount of computation $\frac{N}{2}\log_2(N)$ required to compute the DFT, using the FFT, of a signal $x[n]$ of duration N. But the DFT requires complex additions and multiplications, while the wavelet transform uses only real additions and multiplications. So the savings are even greater than they first appear.

In summary, the DL Daubechies scaling $g[n]$ and wavelet $h[n]$ functions have the following properties:

- Both $g[n]$ and $h[n]$ have durations $2L$.
- Lowpass $\mathbf{G}(\mathbf{z})$ has $L-1$ zeros at $\mathbf{z} = -1$.
- Highpass $\mathbf{H}(\mathbf{z})$ has $L-1$ zeros at $\mathbf{z} = 1$.
- Convolution with $h[n]$ sparsifies signals that are piecewise-$(L-1)^{th}$-degree polynomial signals.
- Computation of the complete wavelet transform of $x[n]$ of duration N requires $4LN$ real MAD's.

The following example shows how the $D2$ Daubechies wavelet transform can be used to compress piecewise-linear signals.

Example 10-12: Linear Signal Compression using the $D2$ Daubechies Wavelet Transform

The goal is to compute the $D2$ Daubechies wavelet transform of the piecewise-linear signal $x[n]$ shown in Fig. 10-16.

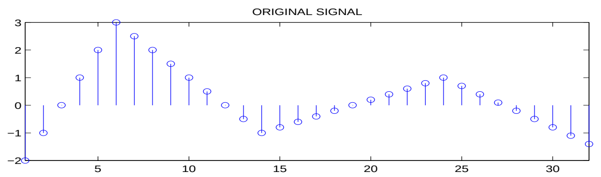


Figure 10.18: Piecewise-Constant Signal $x[n]$.

The average $\hat{X}_k[n]$ and detail $\hat{x}_k[n]$ signals are shown in Fig. 10-17. At the k^{th} stage, note the:

- Average signal $\hat{X}_k[n]$ is a low-resolution $x[n]$;
- Detail signal $\hat{x}_k[n]$ is sparse (mostly zero-valued);
- Nonzero values of $\hat{x}_k[n]$ are small (note the scale);
- Detail signal $\hat{x}_k[n]$ is the additional information needed to reconstruct $\hat{X}_{k-1}[n]$ from $\hat{X}_k[n]$.

This explains the terms "average" and "detail."

The Matlab code for this example is on the CD.

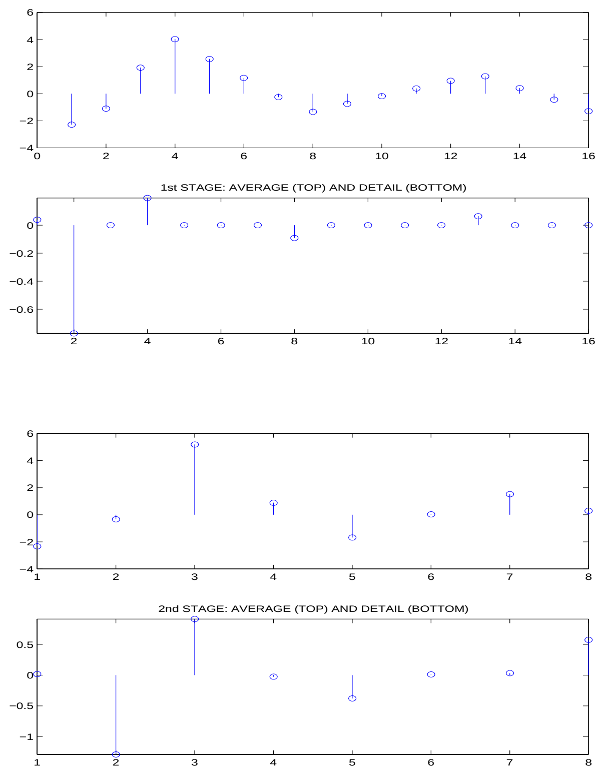


Figure 10.19: (a) Average $\hat{X}_1[n]$. (b) Detail $\hat{x}_1[n]$. (c) Average $\hat{X}_2[n]$. (d) Detail $\hat{x}_2[n]$.

Exercise 10-8: Show that $D1$ Daubechies scaling function $g[n]$ is the normalized Haar scaling function $\tilde{g}_{haar}[n]$.

Answer: Eq. (10.122) is $\mathbf{G}_{haar}(\mathbf{z}) = (1+\mathbf{z}^{-1})Q$ for some constant Q. Inserting into Eq. (10.123) gives $Q = 1/\sqrt{2}$.

Exercise 10-9: Show $D1$ Daubechies scaling function $g[n]$ is orthogonal to even-valued translations of $g[n]$. **Answer:** From Table 10-1,

$\sum g[n]g[n+2] = g[0]g[2] + g[1]g[3]$
$=(.4830)(.2241)+(.8365)(-.1294)=0.$
$\sum g[n]g[n+4] = 0$ since $g[n]$ has duration 4. $h[n]$ is also orthogonal to even-valued translations of itself.

Exercise 10-10: Show a system with two zeros at $\mathbf{z} = 1$ compresses signals linear in time n to zero.

Answer:
If $\mathbf{H}(\mathbf{z})$ has two zeros at $\mathbf{z} = 1$, it must have the form $\mathbf{H}(\mathbf{z}) = (\mathbf{z}-1)^2\mathbf{P}(\mathbf{z}) = (\mathbf{z}^2 - 2\mathbf{z}+1)\mathbf{P}(\mathbf{z})$.

Let $x[n] = an+b$ for constants a and b. $x[n]*h[n] = x[n]*\{1,-2,\underline{1}\}*p[n] = (x[n+2]-2x[n+1]+x[n])*p[n] = 0*p[n] = 0$ since we have $x[n+2]-2x[n+1]+x[n] = (a(n+2)+b)-2(a(n+1)+b)+(an+b) = 0$.

10.8 2-D Wavelet Transform

The real power of the wavelet transform becomes apparent when it is applied to images. A 512 × 512 image has more than a quarter-million pixel values. Storing a sparse representation of an image, rather than the image, saves a huge amount of memory. Compressed sensing (covered in Section 10-? below) becomes very powerful when applied to images.

10.8.1 Downsampling and Upsampling in 2-D

Downsampling (decimation) in 2-D is defined as 1-D downsampling in both directions:

$$x[m,n] \rightarrow \boxed{\downarrow(2,2)} \rightarrow x[2m,2n]. \qquad (10.139)$$

A tiny example of 2-D downsampling is

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \rightarrow \boxed{\downarrow (2,2)} \rightarrow \begin{bmatrix} 1 & 3 \\ 9 & 11 \end{bmatrix}. \tag{10.140}$$

Upsampling (zero-stuffing) in 2-D is defined as 1-D upsampling in both directions:

$$x[m,n] \rightarrow \boxed{\uparrow (2,2)} \rightarrow \begin{cases} x\left[\frac{m}{2}, \frac{n}{2}\right] & m,n \text{ even} \\ 0 & \text{otherwise} \end{cases}. \tag{10.141}$$

A tiny example of 2-D upsampling is

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \rightarrow \boxed{\uparrow (2,2)} \rightarrow \begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 \\ 3 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \tag{10.142}$$

2-D cyclic convolutions are defined as 1-D cyclic convolutions in both directions. Recall that images are just 2-D signals.

10.8.2 Image Analysis Filter Bank

The generalization of the wavelet transform from signals to images is straightforward if *separable* scaling and wavelet functions are used. In this book we restrict attention to separable functions. Let $g[n]$ and $h[n]$ be scaling and wavelet functions, such as the Haar or Daubechies functions. The image analysis filter bank then performs the following operations:

- Start with the original image $x[m,n]$. Then:

$$x[m,n] \rightarrow \boxed{g[m]g[n]} \rightarrow \boxed{\downarrow (2,2)} \rightarrow \hat{x}_{LL}^{(1)}[m,n]$$
$$x[m,n] \rightarrow \boxed{g[m]h[n]} \rightarrow \boxed{\downarrow (2,2)} \rightarrow \hat{x}_{LH}^{(1)}[m,n]$$
$$x[m,n] \rightarrow \boxed{h[m]g[n]} \rightarrow \boxed{\downarrow (2,2)} \rightarrow \hat{x}_{HL}^{(1)}[m,n]$$
$$x[m,n] \rightarrow \boxed{h[m]h[n]} \rightarrow \boxed{\downarrow (2,2)} \rightarrow \hat{x}_{HH}^{(1)}[m,n]$$

- Perform the same decomposition on $\hat{x}_{LL}^{(1)}[m,n]$:

$$\hat{x}_{LL}^{(1)}[m,n] \rightarrow \boxed{g[m]g[n]} \rightarrow \boxed{\downarrow (2,2)} \rightarrow \hat{x}_{LL}^{(2)}[m,n]$$
$$\hat{x}_{LL}^{(1)}[m,n] \rightarrow \boxed{g[m]h[n]} \rightarrow \boxed{\downarrow (2,2)} \rightarrow \hat{x}_{LH}^{(2)}[m,n]$$
$$\hat{x}_{LL}^{(1)}[m,n] \rightarrow \boxed{h[m]g[n]} \rightarrow \boxed{\downarrow (2,2)} \rightarrow \hat{x}_{HL}^{(2)}[m,n]$$
$$\hat{x}_{LL}^{(1)}[m,n] \rightarrow \boxed{h[m]h[n]} \rightarrow \boxed{\downarrow (2,2)} \rightarrow \hat{x}_{HH}^{(2)}[m,n]$$

- Continue decomposing $\hat{x}_{LL}^{(k)}[m,n]$ as above.

The final wavelet transform of $x[m,n]$ consists of:

- The coarsest average image $\hat{x}_{LL}^{(K)}[m,n]$;
- The three detail images at each stage: $\{\hat{x}_{LH}^{(K-1)}[m,n], \hat{x}_{HL}^{(K-1)}[m,n], \hat{x}_{HH}^{(K-1)}[m,n]\}$; $\{\hat{x}_{LH}^{(K-2)}[m,n], \hat{x}_{HL}^{(K-2)}[m,n], \hat{x}_{HH}^{(K-2)}[m,n]\}$; up to the largest (in size) three detail images: $\{\hat{x}_{LH}^{(1)}[m,n], \hat{x}_{HL}^{(1)}[m,n], \hat{x}_{HH}^{(1)}[m,n]\}$.

The average images $\hat{x}_{LL}^{(k)}[m,n]$ are analogous to the average signals $\hat{X}_k[n]$, except that they are low-resolution versions of an image $x[m,n]$ instead of a signal $x[n]$. But in 2-D there are now three detail images, while in 1-D there is only one detail signal.

In 1-D, the detail signals are zero except near *edges*, which are abrupt changes in the signal or in its slope. Detail signals pick up edges, as shown in Example 10-12 above. In 2-D, the detail images

- $\hat{x}_{LH}^{(k)}[m,n]$ picks up vertical edges;
- $\hat{x}_{HL}^{(k)}[m,n]$ picks up horizontal edges;
- $\hat{x}_{HH}^{(k)}[m,n]$ picks up diagonal edges.

10.8.3 Image Synthesis Filter Bank

The image synthesis filter bank combines all of the detail images, and the coarsest average image $\hat{x}_{LL}^{(K)}[m,n]$, into the original image $x[m,n]$, as follows:

$$\hat{x}_{LL}^{(K)}[m,n] \rightarrow \boxed{\uparrow (2,2)} \rightarrow \boxed{g[-m]g[-n]} \rightarrow A_{LL}^{(K)}[m,n];$$
$$\hat{x}_{LH}^{(K)}[m,n] \rightarrow \boxed{\uparrow (2,2)} \rightarrow \boxed{g[-m]h[-n]} \rightarrow A_{LH}^{(K)}[m,n];$$
$$\hat{x}_{HL}^{(K)}[m,n] \rightarrow \boxed{\uparrow (2,2)} \rightarrow \boxed{h[-m]g[-n]} \rightarrow A_{HL}^{(K)}[m,n];$$
$$\hat{x}_{HH}^{(K)}[m,n] \rightarrow \boxed{\uparrow (2,2)} \rightarrow \boxed{h[-m]h[-n]} \rightarrow A_{HH}^{(K)}[m,n];$$

$\hat{x}_{LL}^{(K-1)}[m,n]$ is the sum of the above four outputs: $A_{LL}^{(K)}[m,n]+A_{LH}^{(K)}[m,n]+A_{HL}^{(K)}[m,n]+A_{HH}^{(K)}[m,n]$.

Here $\{A_{LL}^{(K)}[m,n], A_{LH}^{(K)}[m,n], A_{HL}^{(K)}[m,n], A_{HH}^{(K)}[m,n]\}$ are just four temporary quantities to be added. The analogy to signal analysis and synthesis filter banks is evident, except that at each stage there are three detail images instead of one detail signal.

The condition for perfect reconstruction is the 2-D version of the Smith-Barnwell condition Eq. (10.87):

$$|\mathbf{G_2}(e^{j\Omega_1}, e^{j\Omega_2})|^2 + |\mathbf{G_2}(e^{j(\Omega_1+\pi)}, e^{j(\Omega_2+\pi)})|^2 + \tag{10.143}$$
$$|\mathbf{G_2}(e^{j(\Omega_1+\pi)}, e^{j\Omega_2})|^2 + |\mathbf{G_2}(e^{j\Omega_1}, e^{j(\Omega_2+\pi)})|^2 = 4$$

where $\mathbf{G_2}(e^{j\Omega_1}, e^{j\Omega_2})$ is the DSFT of the 2-D scaling function $g_2[m,n] = g[m]g[n]$. Since the 2-D scaling function $g[m]g[n]$ is separable, its DSFT is also separable:

$$\mathbf{G_2}(e^{j\Omega_1}, e^{j\Omega_2}) = \mathbf{G}(e^{j\Omega_1})\mathbf{G}(e^{j\Omega_2}) \tag{10.144}$$

where $\mathbf{G}(e^{j\Omega_1})$ is the DTFT of $g[n]$. The 2-D Smith-Barnwell condition is satisfied if the 1-D Smith-Barnwell condition Eq. (10.87) is satisfied (see Exercise 10-11 below).

Example 10-13: 2-D Haar Wavelet Transform of Shepp-Logan Phantom

The *Shepp-Logan phantom* is a piecewise-constant image that has been a test image for tomography algorithms since the 1970's. Compute the three-stage 2-D Haar wavelet transform of the 256 × 256 Shepp-Logan phantom depicted in Fig. 10-18.

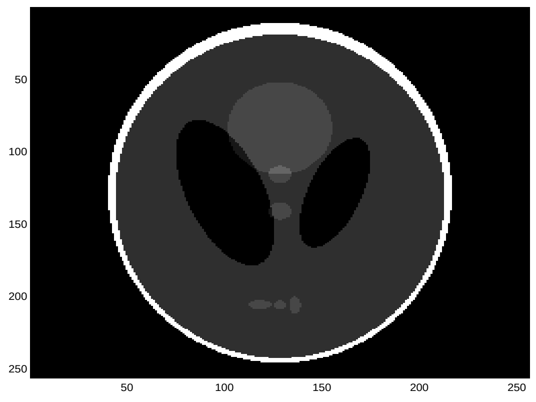


Figure 10.20: 256 × 256 Shepp-Logan Phantom.

Solution:

The 2-D Haar wavelet transform is shown in Fig. 10-19, which demonstrates how to display the wavelet transform of an image:

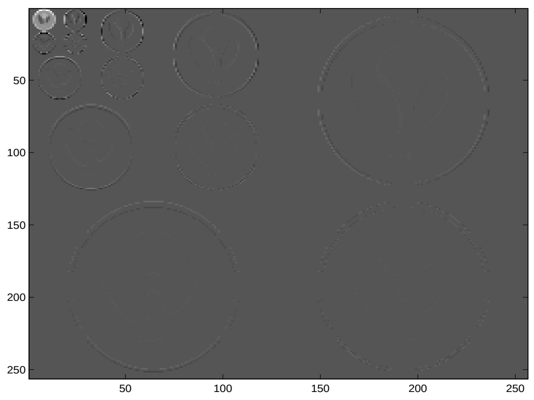


Figure 10.21: 2-D Haar Wavelet Transform of Shepp-Logan Phantom.

- The coarsest average image $\hat{x}_{LL}^{(K)}[m,n]$ is the thumbnail image at the upper left;
- The three detail images are clockwise around it:

$X^{(K)}=$

$\hat{x}_{LL}^{(K)}[m,n]$	$\hat{x}_{LH}^{(K)}[m,n]$
$\hat{x}_{HL}^{(K)}[m,n]$	$\hat{x}_{HH}^{(K)}[m,n]$

- The next stage of detail images, which are twice as large, are clockwise around this:

$\hat{x}_{LL}^{(K)}[m,n]$	$\hat{x}_{LH}^{(K)}[m,n]$	$\hat{x}_{LH}^{(K-1)}[m,n]$
$\hat{x}_{HL}^{(K)}[m,n]$	$\hat{x}_{HH}^{(K)}[m,n]$	
$\hat{x}_{HL}^{(K-1)}[m,n]$		$\hat{x}_{HH}^{(K-1)}[m,n]$

- The pattern continues to repeat itself as

$\hat{X}^{(k)}$	$\hat{x}_{LH}^{(k-1)}[m,n]$
$\hat{x}_{HL}^{(k-1)}[m,n]$	$\hat{x}_{HH}^{(k-1)}[m,n]$

Note that the coarsest average image, the thumbnail at the upper left, is only 32 × 32. Yet it contains almost all the large numbers in the 2-D wavelet transform of the image. The detail images are almost entirely zero. So the original image, with 256^2=65536 pixels, has been compressed to only 3619 nonzero pixels. So $\frac{3619}{65536}$=94.5% were set to zero, leaving only $1 - \frac{3619}{65536}$=5.5% nonzero values in the wavelet transform.

Example 10-14: 2-D $D3$ Daubechies Wavelet Transform of Clown Image.

Use the $D3$ Daubechies wavelet, whose coefficients are listed in Table 10-1, to compute the three-stage 2-D wavelet transform of the clown image shown in Fig. 10-20.

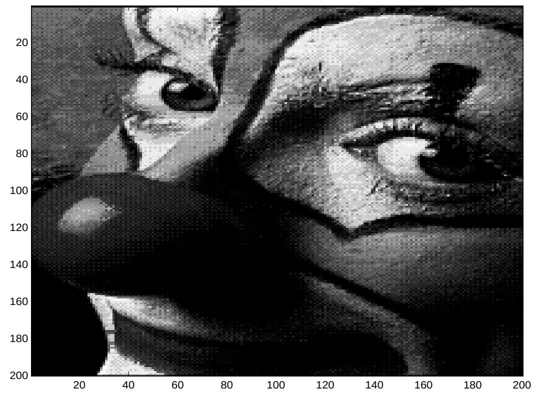


Figure 10.22: 200 × 200 Clown Image.

Solution:
The result is shown in Fig. 10-21. The coarsest-scale average image $\hat{x}_{LL}^{(K)}[m,n]$ is the thumbnail in the upper left corner. The detail images all have very small (but not nonzero) pixel values, so the 2-D wavelet transform of the image can be stored using many fewer bits than the original image.

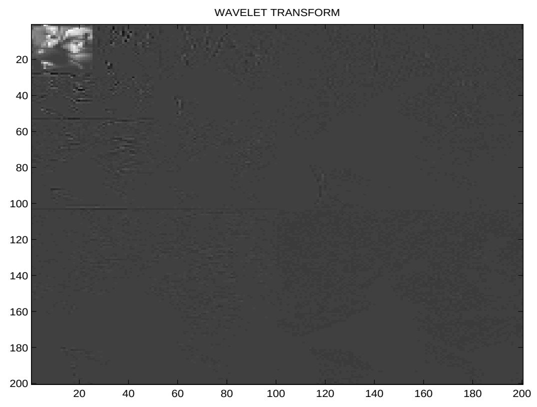


Figure 10.23: $D3$ 2-D Wavelet Transform of the Clown Image.

10.8.4 Image Compression by Thresholding the Wavelet Transform

Examples 10-11 and 10-12 suggest that a good way to compress a signal or image is to *threshold* sufficiently small (in absolute value) values of its wavelet transform to zero. Thresholding a signal $x[n]$ with a threshold value of λ means replacing $x[n]$ with $y[n]$, where

$$y[n] = \begin{cases} x[n] & \text{if } |x[n]| > \lambda \\ 0 & \text{if } |x[n]| < \lambda \end{cases}. \tag{10.145}$$

Thresholding was used in Example 8-12 to denoise a noisy periodic signal. The spectrum of a periodic signal consists of only a few harmonics, but the spectrum of a periodic signal with noise added to it is nonzero everywhere, with harmonics sticking up like dandelions from an unmown lawn. Thresholding the spectrum of a noisy periodic signal virtually eliminated the noise.

Since most wavelet transform values are very small (see Examples 10-11 and 10-12 above), little information about the image is lost by setting small values of its wavelet transform to zero, so that these values need no longer be stored. Since the wavelet transform is orthogonal, a small change in the wavelet transform of an image will produce only a small change in the image reconstructed from these values. This was shown for the 1-D Haar transform of duration eight in Eq. (10.76); this generalizes to all orthogonal wavelet transforms.

In fact, compression by thresholding works very well, as the following example demonstrates.

Example 10-15: Compression of Clown Image by Thresholding its Wavelet Transform.

The $D3$ Daubechies wavelet transform of the clown image shown in Fig. 10-20 was computed as in Example 10-14. All wavelet transform values less than 0.11 in absolute value were thresholded to zero by applying Eq. (10.145) to the wavelet transform of the clown image with $\lambda = 0.11$. This set 94% of the wavelet transform to zero, so that only 6% of it was nonzero, almost the same as the result of applying the Haar wavelet transform to the piecewise-constant Shepp-Logan phantom.

Solution:
The image was then reconstructed from its thresholded wavelet transform. The result is called the compressed image and is shown in Fig. 10-22. There is no apparent difference between the original (Fig. 10-20) and compressed (Fig. 10-22) images, even though the compressed image requires only 6% as much memory as the original image.

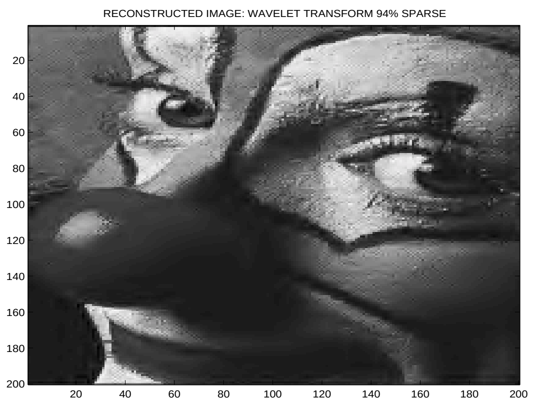


Figure 10.24: 200 $\times$ 200 Compressed Clown Image. This Requires only 6% as much Storage as the Original.

There is far, far more to the topic of image compression. But this does show the utility of the wavelet transform for image compression.

Exercise 10-11: Show that for separable 2-D scaling and wavelet functions, the 2-D Smith-Barnwell condition Eq. (10.143) is satisfied if the 1-D Smith-Barnwell condition Eq. (10.87) is satisfied.

Solution: Inserting Eq. (10.144) into Eq. (10.143) gives

$$\begin{aligned}|\mathbf{G}(e^{j\Omega_1})\mathbf{G}(e^{j\Omega_2})|^2 + |\mathbf{G}(e^{j(\Omega_1+\pi)})\mathbf{G}(e^{j(\Omega_2+\pi)})|^2 &+\\ |\mathbf{G}(e^{j(\Omega_1+\pi)})\mathbf{G}(e^{j\Omega_2})|^2 + |\mathbf{G}(e^{j\Omega_1})\mathbf{G}(e^{j(\Omega_2+\pi)})|^2 &=\\ \left(|\mathbf{G}(e^{j\Omega_1})|^2 + |\mathbf{G}(e^{j(\Omega_1+\pi)})|^2\right)|\mathbf{G}(e^{j\Omega_2})|^2 &+\\ \left(|\mathbf{G}(e^{j\Omega_1})|^2 + |\mathbf{G}(e^{j(\Omega_1+\pi)})|^2\right)|\mathbf{G}(e^{j(\Omega_2+\pi)})|^2 &=\\ 2|\mathbf{G}(e^{j\Omega_2})|^2 + 2|\mathbf{G}(e^{j(\Omega_2+\pi)})|^2 &= 4\end{aligned}$$

10.9 Denoising by Thresholding and Shrinking

The denoising problem is to reduce the noise in an image to which noise has been added. In a denoising problem we observe

$$y[m,n] = x[m,n] + v[m,n] \tag{10.146}$$

where $x[m,n]$ is the desired image and $v[m,n]$ is noise to be reduced. The goal is to recover the original image $x[m,n]$ from the noisy image $y[m,n]$. We show that the obvious approach of simply thresholding (defined in Eq. (10.145) above) the wavelet transform of the image doesn't work. Then we show that thresholding and shrinking (defined in Eq. (10.184) below) the wavelet transform of the image does work.

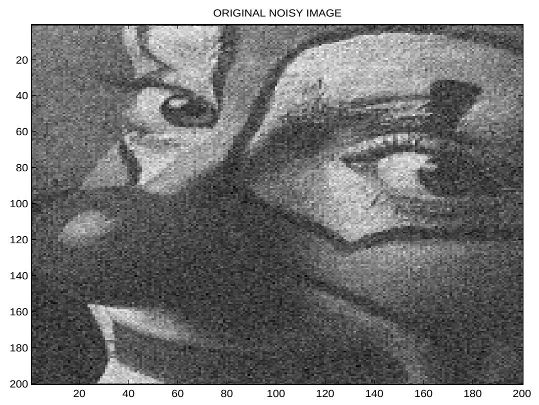
10.9.1 Wavelet-Based Denoising: Thresholding Alone

One approach to denoising is to threshold the wavelet transform of $y[m,n]$. The idea is that for small wavelet transform values, the signal-to-noise ratio is low, so little of value is lost by thresholding these small values to zero. For large wavelet transform values, the signal-to-noise ratio is large, so these large values should be kept. This approach worked quite well in Example 8-12. But this approach works poorly on wavelet transforms of noisy images, as the following example shows.

Example 10-16: Denoising the Clown Image by Thresholding its $D3$ Wavelet Transform.
Zero-mean 2-D white Gaussian noise with standard deviation 0.1 was added to the clown image. Its $D3$ wavelet transform was computed, thresholded with threshold $\lambda = 0.11$, and the image reconstructed from its thresholded wavelet transform.

Solution: The noisy and denoised images are shown in Fig. 10-23. Thresholding failed to reduce the noise much.

This shows the danger in assuming that an approach that worked on one type of problem will work on another type of problem. The problem is that the



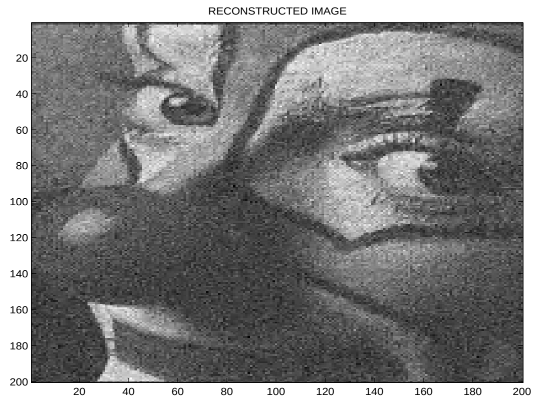


Figure 10.25: Denoising the Clown image. (a) Noisy Clown Image. (b) Denoised by Thresholding its Wavelet Transform. This Didn't Work.

$D3$ wavelet transform of the clown image has many more nonzero values than the DTFS of the periodic signal in Example 8-12. Clearly a different approach is clearly needed for image denoising.

10.9.2 Wavelet-Based Denoising: Thresholding and Shrinkage

We now show that a combination of *thresholding* small wavelet transform values to zero and *shrinking* other wavelet transform values by a small number λ performs much better in denoising images. First we show that shrinkage comes from minimizing a cost functional, just as Wiener filtering Eq. (10.43) came from minimizing the Tikhonov cost functional Eq. (10.41).

Sparsity-Inducing LASSO Cost Functional Λ

Suppose we are given noisy observations $y[n]$ of a signal $x[n]$ that is known to be sparse (mostly zero-valued)

$$y[n] = x[n] + v[n]. \tag{10.147}$$

All three signals have durations N and the noise $v[n]$ is known to have zero-mean. The goal is to estimate $x[n]$ from the noisy observations $y[n]$, i.e., denoise $y[n]$ with the additional information that $x[n]$ is mostly zero-valued.

The prior knowledge that $x[n]$ is sparse can be incorporated by estimating $x[n]$ from $y[n]$, not by simply using $y[n]$ as an estimator for $x[n]$, but by minimizing over $x[n]$ the *LASSO cost functional*

$$\Lambda = \underbrace{\frac{1}{2}\sum_{n=0}^{N-1}(y[n]-x[n])^2}_{\text{fidelity to data y[n]}} + \lambda \underbrace{\sum_{n=0}^{N-1}|x[n]|}_{\text{sparsity}}. \tag{10.148}$$

Readers familiar with basic estimation theory will note that Λ is the negative log-likelihood function for zero-mean white Gaussian noise $v[n]$ with independent Laplacian *a priori* distributions for each $x[n]$. λ is a tradeoff between fidelity to the data $y[n]$ and imposition of sparsity. If $\lambda = 0$, then the estimator $\hat{x}[n]$ of $x[n]$ is just $\hat{x}[n] = y[n]$. Nonzero λ emphasizes sparsity, while allowing some difference between $\hat{x}[n]$ and $y[n]$, which takes into account the noise $v[n]$.

"LASSO" is an acronym for Least Absolute Shrinkage and Selection Operator.

Minimization of LASSO Cost Functional Λ

The minimization of Λ decouples in time n, so each $\hat{x}[n]$ can be computed separately for different n. Setting the derivative of each term of Λ to zero separately gives the following equation for $\hat{x}[n]$:

$$\begin{aligned}&\frac{d}{dx[n]}[(y[n]-x[n])^2/2 + \lambda|x[n]|]\\ &= (\hat{x}[n]-y[n]) + \lambda\,\mathrm{sign}(\hat{x}[n]) = 0\end{aligned} \tag{10.149}$$

since, recalling that derivatives are slopes,

$$\frac{d|x|}{dx} = \begin{cases} +1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \end{cases} = \text{sign}[x] \qquad (10.150)$$

$\frac{d|x|}{dx}$ is undefined at $x = 0$, but this case is handled separately. We now consider three cases:

$\underline{y[n] > \lambda}$: $\hat{x}[n] = y[n] - \lambda$ solves Eq. (10.149).

$\underline{y[n] < -\lambda}$: $\hat{x}[n] = y[n] + \lambda$ solves Eq. (10.149).

$\underline{|y[n]| < \lambda}$: Eq. (10.149) has no solution! Setting the derivative to zero does not yield the solution. Instead, minimization occurs for $\hat{x}[n] = 0$.

Thresholding and Shrinkage

The solution to Eq. (10.149) can be summarized as

$$\hat{x}[n] = \begin{cases} y[n] - \lambda & \text{for } y[n] > +\lambda \\ y[n] + \lambda & \text{for } y[n] < -\lambda\,. \\ 0 & \text{for } |y[n]| < \lambda \end{cases} \qquad (10.151)$$

Values of $y[n]$ smaller in absolute value than the threshold λ are *thresholded* (set) to zero. Values of $y[n]$ larger in absolute value than the threshold λ are *shrunk* by λ, making their absolute values smaller. So $\hat{x}[n]$ is computed by *thresholding and shrinking* $y[n]$. This is usually called (ungrammatically) "thresholding and shrinkage."

The next example shows that denoising images works much better with thresholding and shrinkage than with thresholding alone.

Example 10-17: Denoising the Clown Image by Thresholding and Shrinking its $D3$ Wavelet Transform.

Repeat Example 10-16 using thresholding and shrinkage, instead of just thresholding. Use $\lambda = 0.11$.

Solution:

The noisy and denoised images are shown in Fig. 10-24. Thresholding and shrinking greatly reduced the noise in the image.

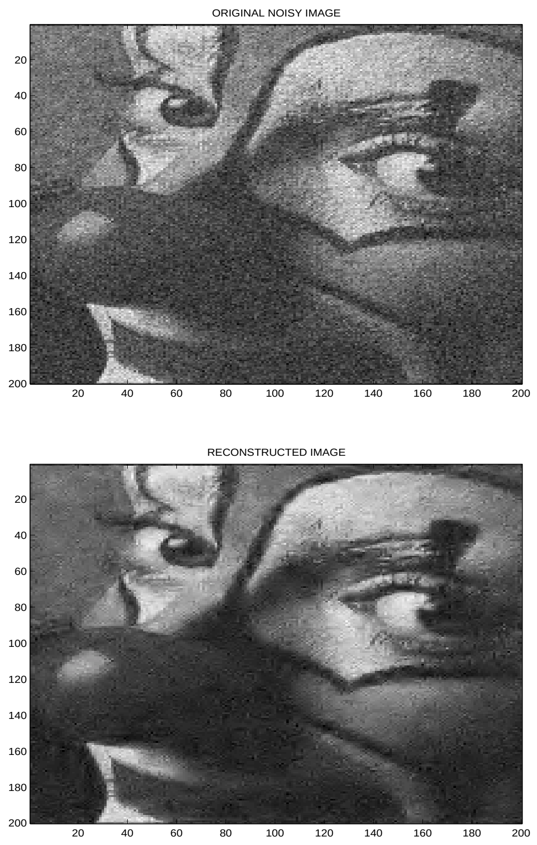


Figure 10.26: Denoising the Clown image. (a) Noisy Clown Image. (b) Denoised by Thresholding and Shrinking its Wavelet Transform. This Worked Well.

10.10 Compressed Sensing

Overview

An *inverse* problem in signal and image processing is to reconstruct an unknown signal or image from measurements (known linear combinations) of the values of the signal or image. Such inverse problems arise in medical imaging, radar imaging, optics, and many other fields. For example, in tomography and magnetic resonance imaging (MRI), the inverse problem is to reconstruct an image from measurements of its 2-D Fourier transform. If the number of measurements equals or exceeds the size of the unknown signal or image, solution of the inverse problem becomes solution of a linear system of equations.

We have seen that many real-world signals and images can be compressed using the wavelet transform into a sparse representation in which most of the values are zero. This suggests that the number of measurements needed to reconstruct the signal or image should be less than the size of the signal or image, since in the wavelet transform domain most of the values to be reconstructed are known to be zero. However, the locations of the nonzero values are unknown, so the problem is not just the solution of a smaller (than the original) linear system of equations.

Compressed sensing is the field of reconstructing wavelet-compressible signals and images from a number of measurements much smaller than the size of the signal or image. In tomography, this reduces patient exposure to radiation. In MRI, this reduces acquisition time inside the MRI machine. In digital cameras, such as smartphone cameras, this reduces exposure time and energy required to acquire an image. Since the turn of the millennium, compressed sensing has been an active area of research and application in signal and image processing.

This section presents the basic concepts behind compressed sensing and applies these concepts to signal and image inverse problems. We do not present proofs or details on precisely when compressed sensing works, since this is an active area of research.

10.10.1 Problem Formulation

Define the following quantities:

- $\{x[n], n = 0 \dots N-1\}$ is an unknown signal;
- The unknown wavelet transform of $x[n]$ is $\{\hat{x}_1[n], \hat{x}_2[n], \dots \hat{x}_L[n], \hat{X}_L[n]\}$;
- The wavelet transform of $x[n]$ is **sparse:** Only K values of all of the $\{\hat{x}_k[n]\}$ are nonzero;
- $\{y[n], n = 0 \dots M-1\}$ are known *measurements* (linear combinations) $\sum_{i=0}^{N-1} a_{n,i}x[i]$ of $x[n]$;
- $\{a_{n,i}, n = 0 \dots M-1, i = 0 \dots N-1\}$ are known;
- K is unknown, but we do know $K << M << N$.

The goal is to compute $\{x[n], n = 0 \dots N-1\}$ from the M known measurements $\{y[n], n = 0 \dots M-1\}$.

This problem can be formulated as a linear algebra problem by defining the vectors and matrices

- $\underline{x} = [x[0], x[1] \dots x[N-1]]^T$;
- $\underline{y} = [y[0], y[1] \dots y[M-1]]^T$;
- $\underline{z}$ is $\{\hat{x}_1[n], \hat{x}_2[n], \dots \hat{x}_L[n], \hat{X}_L[n]\}$ collected into a column vector of length N (see (10.62));
- A is the $M \times N$ matrix with $(i,j)^{th}$ element $a_{i,j}$;
- W is the $N \times N$ matrix such that $\underline{z} = W\underline{x}$ implements the wavelet transform of $x[n]$. For example, for a Haar transform with $N = 8$, W is $\mathcal{H}$, the (8×8) matrix in (10.74).

Then the problem can be formulated as finding the solution $\underline{x}$ to the underdetermined linear system of equations $\underline{y} = A\underline{x}$ such that $\underline{z} = W\underline{x}$ has only K nonzero elements, where K is unknown.

Since the wavelet transform is invertible, this in turn can be formulated as finding the solution $\underline{z}$ to the underdetermined linear system of equations $\underline{y} = AW^{-1}\underline{z}$ such that $\underline{z}$ has only K nonzero elements. Then $\underline{x} = W^{-1}\underline{z}$ recovers $\underline{x}$ from $\underline{z}$.

For the orthogonal wavelet transforms (Haar and Daubechies) covered in Sections 10-4 and 10-6, $W^{-1} = W^T$, so the inverse wavelet transform can be computed as easily as the wavelet transform. In practice, both are computed using analysis and synthesis filter banks.

The $M \times N$ linear system of equations $\underline{y} = AW^{-1}\underline{z}$ is underdetermined since $M << N$. A solution $\underline{z}$ with $N - M$ zero values is easily found by deleting any $N-M$ columns of AW^{-1}, leaving an $M \times M$ matrix, solving the resulting linear system of equations for nonzero values of $\underline{z}$, and setting the values of $\underline{z}$ corresponding to deleted columns to zero. A solution $\underline{z}$ with more than $N - M$ zero values, i.e., fewer than M nonzero values, usually does not exist.

In the sequel, we will assume that a *K-sparse* solution $\underline{z}$ with only $K << M$ nonzero values exists, where K is unknown. The problem is how to find it.

10.10.2 Inducing Sparsity

History

In seismic signal processing, explosions are set off on the earth's surface, and echoes of the seismic waves created by this explosion are measured by seismometers. In the 1960's, sedimentary media (such as the bottom of the Gulf of Mexico) were modelled as a stack of layers, so the seismometers would record occasional sharp pulses reflected off of the interfaces between the layers. The amplitudes and times of the pulses would allow the layered medium to be reconstructed. However, the occasional pulses had to be deconvolved from the source pulse created by the explosions. The deconvolution problem was modelled as an underdetermined linear system of equations.

It was known heuristically that sparse solutions to a system of equations could be found by choosing the solution that minimized the sum of absolute values of the solution. This is the *minimum ℓ_1 norm* solution. The ℓ_1 and squared ℓ_2 norms of an N-vector $\underline{z}$ are

$$\begin{aligned} ||\underline{z}||_1 &= \sum_{n=1}^{N} |z_n| \\ ||\underline{z}||_2^2 &= \sum_{n=1}^{N} z_n^2. \end{aligned} \tag{10.152}$$

The minimum ℓ_1 norm solution to a linear system of equations has been a centerpiece of compressed sensing, although it is not the only approach.

An extensive amount of research since the turn of the millennium has showed that under mild assumptions on the matrix A, the minimum ℓ_1 norm solution is the sparsest solution, if K is small enough.

We present a tiny example that illustrates how the minimum ℓ_1 norm solution can be sparse when the minimum ℓ_2 norm solution is not.

Example 10-18: Comparing Minimum ℓ_1 and ℓ_2 Norm Solutions

Compute the minimum ℓ_1 and ℓ_2 norm solutions to the single equation in two unknowns ($M = 1$ and $N = 2$) $z_1 + 2z_2 = 10$, which is a straight line.

Solution:

The problem is illustrated in Fig. 10-25, which shows contours of constant

- ℓ_1 norm (left): $|z_1| + |z_2|$ is constant. The diamond has "corners" on the axes, representing sparse solutions. A corner hits the line.
- ℓ_2 norm (right): $z_1^2 + z_2^2$ is constant.

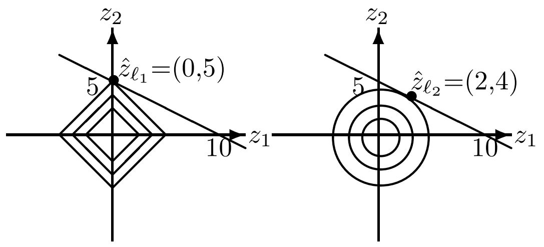


Fig. 10-25. Solutions Minimizing $||\underline{z}||_1$ and $||\underline{z}||_2$.

The minimum ℓ_1 norm solution is found by inspection to be $(z_1, z_2)_{\ell_1} = (0, 5)$. It is sparse since one of two components is zero.

The minimum ℓ_2 norm solution is found by substituting $z_1 = 10 - z_2$ in the squared ℓ_2 norm $(z_1^2 + z_2^2)$ and setting its derivative to zero:

$$\begin{aligned} 0 &= \frac{d}{dz_2}[(10 - 2z_2)^2 + z_2^2] \\ &= -2(2)(10 - 2z_2) + 2z_2 \\ &= 10z_2 - 40 \rightarrow z_2 = 4. \end{aligned} \tag{10.153}$$

The minimum ℓ_2 norm solution is $(z_1, z_2)_{\ell_2} = (2, 4)$. It is not sparse since neither component is zero.

For matrices A composed of zero-mean independent Gaussian random variables, the minimum ℓ_1 norm reliably recovers a K-sparse solution if K is less than the value listed in Table 10-2 for various M and N:

K	15	16	20	26	45
M	50	64	128	100	250
N	250	512	512	200	1000

Table 10-2. Largest Value of Sparsity K for which a Sparse Solution can be Reliably Reconstructed for a Random $M \times N$ Matrix A.

Basis Pursuit

So one way of computing the sparse solution to the underdetermined system of equations is to solve

$$\begin{array}{rl} \text{Minimize} & \sum_{n=1}^{N}|z_n| \\ \text{Subject to} & \underline{y}=AW^{-1}\underline{z}. \end{array} \tag{10.154}$$

This is called *basis pursuit*. It can be formulated as a *linear programming* problem as follows. Define the positive $\underline{z}^+$ and negative $\underline{z}^-$ parts of $\underline{z}$ as

$$z_i^+=\begin{cases}+z_i & \text{if } z_i\geq 0\\ 0 & \text{if } z_i\leq 0\end{cases} \quad z_i^-=\begin{cases}-z_i & \text{if } z_i\leq 0\\ 0 & \text{if } z_i\geq 0\end{cases}\geq 0. \tag{10.155}$$

Then we have

$$\begin{aligned} \underline{z} &= \underline{z}^+-\underline{z}^- \\ ||\underline{z}||_1 &= \sum_{i=1}^{N}(z_i^+ + z_i^-). \end{aligned} \tag{10.156}$$

So the basis pursuit problem Eq. (10.154) becomes

$$\begin{array}{rl} \text{Minimize} & \sum_{i=1}^{N}(z_i^+ + z_i^-) \\ \text{Subject to} & \underline{y}=AW^{-1}\underline{z}^+ - AW^{-1}\underline{z}^- \\ \text{Subject to} & z_i^+, z_i^-\geq 0 \end{array} \tag{10.157}$$

which is a straightforward linear programming problem that can be solved in Matlab using `linprog` in Matlab's Optimization Toolbox.

LASSO Functional

A problem with basis pursuit is that it does not account for noise in the observations $\underline{y}$. Noise can be accounted for by minimizing the LASSO functional defined in Eq. (10.148), which for the current problem becomes

$$\Lambda=\frac{1}{2}||\underline{y}-AW^{-1}\underline{z}||_2^2+\lambda||\underline{z}||_1 \tag{10.158}$$

where λ is a tradeoff parameter between sparsity of $\underline{z}$ and fidelity to the measurements $\underline{y}$. As $\lambda\to 0$, the LASSO minimization solution approaches the basis pursuit solution. This is basis pursuit denoising.

But minimization of Eq. (10.158), unlike minimization of Eq. (10.148), does not decouple in z_n, so it cannot be solved directly. Instead, an iterative algorithm must be used. We now derive two commonly used iterative algorithms for minmizing Eq. (10.158): Iterative Reweighted Least Squares (IRLS); and Iterative Shrinkage and Thresholding (IST).

Iterative Reweighted Least Squares (IRLS) Algorithm

IRLS is an application of Tikhonov regularization, defined in Eq. (10.41), with an additional weighting matrix D on the unknown. For convenience we define $\tilde{A}=AW^{-1}$. The Tikhonov cost functional for the present problem is

$$\mathcal{T}=\frac{1}{2}\underbrace{||\underline{y}-\tilde{A}\underline{z}||_2^2}_{\text{fidelity}}+\lambda\underbrace{||D\underline{z}||_2^2}_{\text{size}}. \tag{10.159}$$

λ is a tradeoff parameter between fidelity to the data and size of the unknown $\underline{z}$. The idea is to trade off small differences between $\underline{y}$ and $\tilde{A}\underline{z}$ to keep $\underline{z}$ small.

The $\underline{z}$ minimizing $\mathcal{T}$ can be computed in closed form by rewriting Eq. (10.159) as

$$\begin{aligned} \mathcal{T} &= \frac{1}{2}||\underline{y}-\tilde{A}\underline{z}||_2^2+\lambda||D\underline{z}||_2^2 \\ &= \frac{1}{2}||\underbrace{\begin{bmatrix}\underline{y}\\0\end{bmatrix}}_{\mathbf{y}}-\underbrace{\begin{bmatrix}\tilde{A}\\ \sqrt{2\lambda}D\end{bmatrix}}_{\mathbf{A}}\underline{z}\,||_2^2. \end{aligned} \tag{10.160}$$

The solution to Eq. (10.160) is the well-known pseudo-inverse

$$\begin{aligned} \hat{\underline{z}} &= (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{y} \\ &= \left([\tilde{A}^T\ \sqrt{2\lambda}D^T]\begin{bmatrix}\tilde{A}\\ \sqrt{2\lambda}D\end{bmatrix}\right)^{-1}[\tilde{A}^T\ \sqrt{2\lambda}D^T]\begin{bmatrix}\underline{y}\\0\end{bmatrix} \\ &= (\tilde{A}^T\tilde{A}+2\lambda D^TD)^{-1}\tilde{A}^Ty. \end{aligned} \tag{10.161}$$

As always, this should be computed by solving

$$(\tilde{A}^T\tilde{A}+2\lambda D^TD)\hat{\underline{z}}=\tilde{A}^Ty. \tag{10.162}$$

The LASSO functional Eq. (10.158) is the Tikhonov functional Eq. (10.159) with the matrix D set to

$$D=\text{diag}\,[1/\sqrt{|z_n|}] \tag{10.163}$$

since

$$||D\underline{z}||_2^2 = \sum_{n=1}^{N} \frac{z_n^2}{|z_n|} = \sum_{n=1}^{N} |z_n| = ||\underline{z}||_1. \qquad (10.164)$$

Eq. (10.162) cannot be solved in closed form, since D depends on the unknowns z_n. But it can be solved using the following iterative algorithm, called IRLS. Let $\hat{\underline{z}}^{(k)}$ be the estimate of $\hat{\underline{z}}$ at the k^{th} iteration. Then:

- **Initialize:** $\hat{\underline{z}}^{(1)}$ solves Eq. (10.162) with D=I.
- **Reweight:** Compute $D^{(k)}$ from $\hat{\underline{z}}^{(k)}$ using $D^{(k)} = \text{diag}\,[1/(|\hat{\underline{z}}_n^{(k)}| + \epsilon)^{1/2}]$ for some tiny ϵ to keep $D^{(k)}$ finite when elements of $\hat{\underline{z}}_n^{(k)} \to 0$;
- **Recursion:** $\hat{\underline{z}}^{(k+1)}$ is the solution to Eq. (10.162) with $D^{(k)}$.

It can be shown that this algorithm is guaranteed to converge under mild assumptions. It is also called FOCUSS. But the requirement of solution of a large system of equations at each iteration makes it unsuitable for most signal and image processing problems.

10.10.3 Iterative Shrinkage and Thresholding Algorithm

Iterative Shrinkage and Thresholding (IST) is an application of the Landweber iteration, which we now derive. IST is also the application of a gradient descent iterative algorithm to the LASSO functional.

Because this section derives several results not yet encountered, for notational convenience we replace the system $\underline{y}$=$\tilde{A}\underline{z}$ with $y = Ax$ in the following section alone.

Landweber Iteration

Let B be a square matrix. Then for any integer N

$$I - B^N = (I - B)(I + B + B^2 + B^3 + \ldots B^{N-1}). \qquad (10.165)$$

If the eigenvalues of B are all less than one in magnitude, then $I - B$ is nonsingular and $\underset{N\to\infty}{\text{LIM}} B^N = 0$. Then the following infinite series converges:

$$(I - B)^{-1} = I + B + B^2 + B^3 + \ldots \qquad (10.166)$$

Compare this to the infinite geometric series

$$\frac{1}{1-r} = 1 + r + r^2 + r^3 + \ldots \text{ if } |r| < 1. \qquad (10.167)$$

Now consider the underdetermined linear system

$$y = Ax. \qquad (10.168)$$

The solution $\hat{x}$ with minimum ℓ_2 norm $||x||_2$ is

$$\hat{x} = A^T(AA^T)^{-1}y. \qquad (10.169)$$

Note that this is different from Eq. (10.161).

Now suppose that the eigenvalues of AA^T (which are real-valued) all lie in the interval $(0, 2)$. Then the eigenvalues of $I - AA^T$ all lie in the interval (–1,1). Setting $B = I - AA^T$ in Eq. (10.166) gives

$$(AA^T)^{-1} = \sum_{k=0}^{\infty}(I - AA^T)^k \qquad (10.170)$$

Premultiplying by A^T and postmultiplying by y gives

$$\hat{x} = \sum_{k=0}^{\infty} A^T(I - AA^T)^k y. \qquad (10.171)$$

Recognizing that

$$A^T(I - AA^T) = (I - A^TA)A^T \qquad (10.172)$$

and applying it k times gives

$$\hat{x} = \sum_{k=0}^{\infty}(I - A^TA)^k A^T y. \qquad (10.173)$$

The eigenvalues of A^TA are the eigenvalues of AA^T, so this series still converges. It can be implemented recursively as ($x^{(k)}$ is the result of the k^{th} iteration)

$$\begin{aligned} x^{(0)} &= 0 \\ x^{(k+1)} &= A^Ty + (I - A^TA)x^{(k)} \end{aligned} \qquad (10.174)$$

This can be rewritten as

$$\begin{aligned} x^{(0)} &= 0 \\ x^{(k+1)} &= x^{(k)} + A^T(y - Ax^{(k)}). \end{aligned} \qquad (10.175)$$

This is called the *Landweber iteration.* In optics it is also known as the Van Cittert iteration.

The Landweber iteration Eq. (10.175) can also be used to compute the pseudoinverse Eq. (10.161) for overdetermined systems $y = Ax$. Setting $B = I - A^T A$ in Eq. (10.166) and postmultiplying by $A^T y$ gives

$$\begin{aligned} \hat{x} &= (A^T A)^{-1} A^T y \qquad (10.176) \\ &= \sum_{k=0}^{\infty} (I - A^T A)^k A^T y. \end{aligned}$$

which is the same as Eq. (10.173). So the Landweber iteration computes the pseudoinverse Eq. (10.161) for overdetermined systems $y = Ax$.

Landweber for Elastic Net

The Landweber iteration can be used to minimize the *elastic net* functional $\mathcal{E}$, which is a combination of the LASSO and Tikhonov functionals:

$$\mathcal{E} = \frac{1}{2}||y - Ax||_2^2 + \lambda ||x||_1 + \mu ||x||_2^2 \qquad (10.177)$$

The third term is also called a *ridge* penalty term.

The ℓ_1 norm is handled as it was in basis pursuit. Define the positive x^+ and negative x^- parts of x as

$$x_i^+ = \begin{cases} +x_i & \text{if } x_i \geq 0 \\ 0 & \text{if } x_i \leq 0 \end{cases} \quad x_i^- = \begin{cases} -x_i & \text{if } x_i \leq 0 \\ 0 & \text{if } x_i \geq 0 \end{cases} \geq 0. \qquad (10.178)$$

Then we have

$$\begin{aligned} x &= x^+ - x^- \qquad (10.179) \\ ||x||_1 &= \sum_{i=1}^{N} (x_i^+ + x_i^-) \\ ||x||_2^2 &= ||x^+||_2^2 + ||x^-||_2^2 \end{aligned}$$

Now consider the still-underdetermined problem

$$\underbrace{\begin{bmatrix} y \\ -\frac{\lambda}{\sqrt{2\mu}}\mathbf{1} \end{bmatrix}}_{\tilde{y}} = \underbrace{\begin{bmatrix} A & -A \\ \sqrt{2\mu}I & \sqrt{2\mu}I \end{bmatrix}}_{\tilde{A}} \underbrace{\begin{bmatrix} x^+ \\ x^- \end{bmatrix}}_{\tilde{x}} \qquad (10.180)$$

where $\mathbf{1}$=[1,1...1]^T, and both $x_i^+ \geq 0$ and $x_i^- \geq 0$.

The squared ℓ_2 error is then (note that $x_i^+ x_i^-$=0)

$$\begin{aligned} ||\tilde{y} - \tilde{A}\tilde{x}||_2^2 &= ||y - A(x^+ - x^-)||_2^2 \qquad (10.181) \\ &+ ||\sqrt{2\mu}(x^+ + x^-) + \frac{\lambda}{\sqrt{2\mu}}||_2^2 \\ &= ||y - Ax||_2^2 \\ &+ 2\mu ||x||_2^2 + 2\lambda ||x||_1 + \frac{N\lambda^2}{2\mu} \\ &= 2\mathcal{E} + \frac{N\lambda^2}{2\mu} \end{aligned}$$

The final term in Eq. (10.181) does not affect the argmax, so computing the *non-negative least-squares* solution to Eq. (10.180) minimizes the elastic net cost function $\mathcal{E}$.

Substitution of Eq. (10.180) in Eq. (10.175) gives

$$\begin{aligned} z^{(k+1)} &= y - A[(x^+)^{(k)} - (x^-)^{(k)}] \\ v^{(k+1)} &= \lambda \mathbf{1} + 2\mu[(x^+)^{(k)} + (x^-)^{(k)}] \\ (x^+)^{(k+1)} &= (x^+)^{(k)} + A^T z^{(k+1)} - v^{(k+1)} \\ (x^-)^{(k+1)} &= (x^-)^{(k)} - A^T z^{(k+1)} - v^{(k+1)} \qquad (10.182) \end{aligned}$$

followed by the non-negativity constraints

$$\begin{aligned} (x^+)_i^{(k+1)} &= \max[(x^+)_i^{(k+1)}, 0] \\ (x^-)_i^{(k+1)} &= \max[(x^-)_i^{(k+1)}, 0]. \qquad (10.183) \end{aligned}$$

Let $\mu \to 0$ in the elastic net criterion Eq. (10.177). This gives the LASSO criterion. We now examine what letting $\mu \to 0$ does to the Landweber iteration Eq. (10.175).

Iterative Shrinkage and Thresholding (IST) Algorithm

Let $\mu \to 0$ in (10.175). This gives the iteration

$$\begin{aligned} z^{(k+1)} &= y - A[(x^+)^{(k)} - (x^-)^{(k)}] \\ (x^+)^{(k+1)} &= (x^+)^{(k)} + A^T z^{(k+1)} - \lambda \mathbf{1} \\ (x^-)^{(k+1)} &= (x^-)^{(k)} - A^T z^{(k+1)} - \lambda \mathbf{1} \qquad (10.184) \end{aligned}$$

followed by the non-negativity constraints

$$\begin{aligned} (x^+)_i^{(k+1)} &= \max[(x^+)_i^{(k+1)}, 0] \\ (x^-)_i^{(k+1)} &= \max[(x^-)_i^{(k+1)}, 0]. \qquad (10.185) \end{aligned}$$

Since x^k=$(x^+)^{(k)}$–$(x^-)^{(k)}$, and $-(x^-)^{(k)}$ is the negative values of $x^{(k)}$, Eq. (10.184) is the usual Landweber iteration, followed by *thresholding* values of $|x^{(k)}| < \lambda$ to 0 and *shrinkage* of $|x^{(k)}|$ by λ. This is the *Iterated Shrinkage and Thresholding (IST) algorithm*:

$$x^{(k+1)} = x^{(k)} + A^T(y - A(k)), \quad x^{(0)} = 0 \quad (10.186)$$

followed by thresholding and shrinkage

$$x_i^{(k+1)} = \begin{cases} x_i^{(k+1)} - \lambda & \text{if } x_i^{(k+1)} > +\lambda \\ x_i^{(k+1)} + \lambda & \text{if } x_i^{(k+1)} < -\lambda \\ 0 & \text{if } |x_i^{(k+1)}| < \lambda. \end{cases} \quad (10.187)$$

Each iteration for minimizing the LASSO functional consists of a Landweber iteration and a *threshold and shrinkage* step which thresholds values of $|x_n^{(k)}| < \lambda$ to zero and shrinks the remaining $|x_n^{(k)}|$ by λ, making theshrunken $x_n^{(k)}$ values closer to zero.

Thresholding and shrinkage was encountered previously in Section 10-8 as part of denoising using wavelets. Thresholding and shrinkage is often called *soft thresholding*, while thresholding small values to zero without shrinking is called *hard thresholding.*

The bottom line is that the LASSO functional can be minimized using a Landweber iteration with thresholding and shrinkage at each iteration.

IST Algorithm Convergence

Since the cost functional $\mathcal{E}$ and the non-negativity constraints $x_i^+ \geq 0$ and $x_i^- \geq 0$ are all convex, the non-negative Landweber iteration is guaranteed to converge if the maximum eigenvalue of $\tilde{A}^T\tilde{A} < 2$. The nonzero eigenvalues of $\tilde{A}^T\tilde{A}$ are the eigenvalues of $\tilde{A}\tilde{A}^T$, which from

$$\begin{aligned} \tilde{A}\tilde{A}^T &= \begin{bmatrix} A & -A \\ \sqrt{2\mu}I & \sqrt{2\mu}I \end{bmatrix} \begin{bmatrix} A^T & \sqrt{2\mu}I \\ -A^T & \sqrt{2\mu}I \end{bmatrix} \\ &= \begin{bmatrix} 2AA^T & 0 \\ 0 & 4\mu I \end{bmatrix} \end{aligned} \quad (10.188)$$

are double the eigenvalues of AA^T, and 4μ. The non-negative Landweber iteration will converge if $\mu < \frac{1}{2}$ and all the singular values of A are less than unity. Since $\mu \to 0$ here, the former condition is satisfied.

In many problems of interest (see the two examples below), $AA^T = I$ since the wavelet transform used to sparsify x and the matrix A represents a unitary matrix, like the DFT. The IST algorithm still converges.

IST as a Gradient Algorithm

The IST algorithm can also be derived as a gradient algorithm for minimizing the LASSO functional. As an alternative, we present this derivation as well.

A *gradient* or *descent* algorithm for minimizing a function $f(x)$ is an iterative algorithm of the form

$$x^{(k+1)} = x^{(k)} - a\frac{df}{dx}(x^{(k)}) \quad (10.189)$$

where $x^{(k)}$ is the estimate at the k^{th} iteration of the value of x that minimizes $f(x)$ and a is the step size. For a function $f(\underline{x})$ of several variables $\{x_1 \dots x_N\}$, we apply Eq. (10.189) to each x_i and stack the results in an N-vector. This gives

$$\underline{x}^{(k+1)} = \underline{x}^{(k)} - a\nabla f(\underline{x}^{(k)}) \quad (10.190)$$

where the gradient vector $\nabla f(\underline{x})$ is the N-vector

$$\nabla f(\underline{x}) = \left[\frac{\partial f}{\partial x_1} \cdots \frac{\partial f}{\partial x_N}\right]^T. \quad (10.191)$$

The LASSO functional Eq. (10.158) in terms of each x_i is

$$\Lambda = \frac{1}{2}\sum_{n=1}^{M}(y_n - \sum_{i=1}^{N} a_{n,i}x_i)^2 + \lambda\sum_{i=1}^{N}|x_i|. \quad (10.192)$$

The partial derivative of Λ with respect to x_1 is

$$\frac{\partial\Lambda}{\partial x_1} = -\sum_{n=1}^{M} a_{n,1}(y_n - \sum_{i=1}^{N} a_{n,i}x_i) + \lambda\,\text{sign}[x_1]. \quad (10.193)$$

Repeating this for x_i for $i = 2 \dots N$ and then stacking the results into an N-vector gives

$$\nabla\Lambda(\underline{x}) = -A^T(\underline{y} - A\underline{x}) + \lambda\,\text{sign}[\underline{x}] \quad (10.194)$$

Inserting Eq. (10.194) into Eq. (10.190) and setting $a = 1$ gives

$$\underline{x}^{(k+1)} = \underbrace{\underline{x}^{(k)} + A^T(\underline{y} - A\underline{x}^{(k)})}_{\text{Landweber iteration}} - \underbrace{\lambda\,\text{sign}[\underline{x}^{(k)}]}_{\text{shrinkage}} \tag{10.195}$$

So each iteration of the gradient or descent algorithm for minimizing the LASSO functional consists of a *Landweber* iteration, which minimizes $\frac{1}{2}||\underline{y} - A\underline{x}||_2^2$, and a *shrinkage* step, which reduces $|\underline{x}^{(k)}|$ by λ, i.e., makes $\underline{x}^{(k)}$ closer to zero.

There are several variations of the IST algorithm. Some simply threshold sufficiently small values of $|x^{(k)}|$ to zero, and some use both shrinkage and thresholding. The most commonly-used form is

$$\begin{aligned} \underline{w}^{(k+1)} &= \underline{x}^{(k)} + \tilde{A}^T(\underline{y} - A\underline{x}^{(k)}) \qquad (10.196) \\ x_i^{(k+1)} &= \begin{cases} w_i^{(k+1)} - \lambda & \text{if } w_i^{(k+1)} \geq +\lambda \\ w_i^{(k+1)} + \lambda & \text{if } w_i^{(k+1)} \leq -\lambda \\ 0 & \text{if } |w_i^{(k+1)}| \leq \lambda \end{cases} \end{aligned}$$

Reversing the order of Landweber iteration and shrinking-or-thresholding is irrelevant, since IST is an iterative algorithm with many iterations.

IST Algorithm Examples

We now present two examples of the IST algorithm.

Example 10-19: Shepp-Logan Phantom Reconstruction by IST and Least-Squares

The 2-D DFT of the 256×256 Shepp-Logan phantom is computed at randomly-selected frequencies:

- 35755 known of $(256)^2$=65536 values of $\mathbf{X}^*_{k_1,k_2}$;
- If $\mathbf{X}_{k_1,k_2}$ was selected, then conjugate location $X_{256-k_1,256-k_2} = \mathbf{X}^*_{k_1,k_2}$ was also selected;
- In terms of the original formulation, we have: $M = 35755$; $N = 65536$; $K = 3764$ (is unknown).

The locations of known DFT values are depicted as white dots in Fig. 10-26.

The goal is to reconstruct the image from this subset of its 2-D DFT values, using:

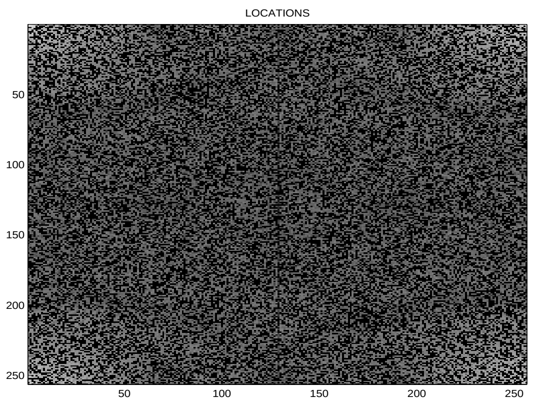
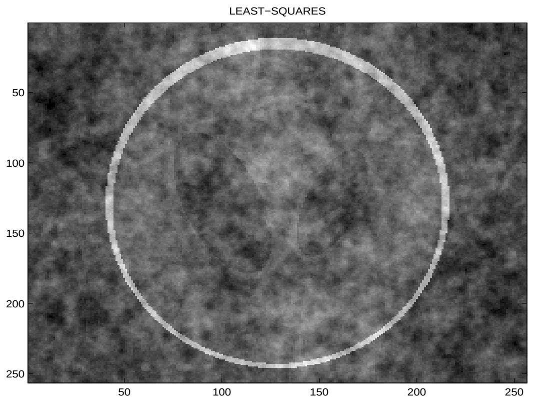


Figure 10.26: Locations of Known Values of $\mathbf{X}_{k_1,k_2}$.

- Least-squares: Unknown values of $\mathbf{X}_{k_1,k_2}$ are set to zero, and the inverse 2-D DFT computed;
- IST algorithm: with $\lambda = 0.01$ and 1000 iterations.

Solution:
The results are depicted in Fig. 10-27.



The least-squares reconstruction recovers only the outer band; no other image features are apparent.

The IST reconstruction is an excellent reconstruction of the original image, with no visual difference.

Despite the size of the problem, the IST required only a few seconds. This is due to the minimal amount of computation required at each iteration for

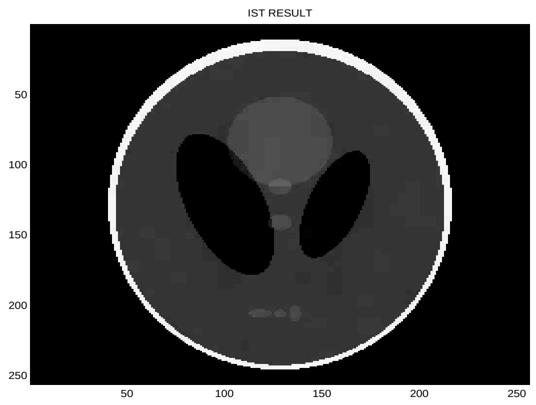


Figure 10.27: Reconstruction of Shepp-Logan Phantom Image from a Subset of its 2-D DFT Values using: (a) Least-Squares; (b) IST.

the 2-D FFT and Haar algorithms. The 35755×65536 matrix is never constructed or accessed, and no matrix-vector products used.

Example 10-20: Image Inpainting

About half of the clown image pixel values, at random locations, were considered to be unknowns. The goal is to reconstruct the clown image from the remaining known half of the clown image pixel values. This is called the *image inpainting* problem. The image with the unknown pixel values set to zero is depicted in Fig. 10-28.

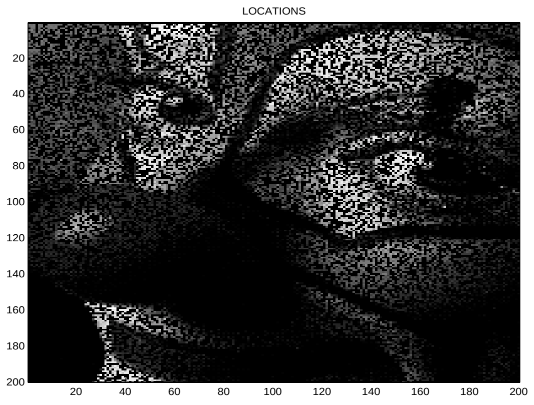


Figure 10.28: Locations of Known Values of Image.

At first glance this seems unsolvable–the unknown pixel values could be anything. But under the assumption that the clown image is sparsifiable using the $D3$ Daubechies wavelet function, this is just another problem of reconstructing an image from some linear combinations of its values, even though each linear combination is actually just a single pixel value. In terms of the original problem formulation, $M = 20227$ and $N = 40000$.

Solution:

The result is depicted in Fig. 10-29. The image has been reconstructed almost perfectly.

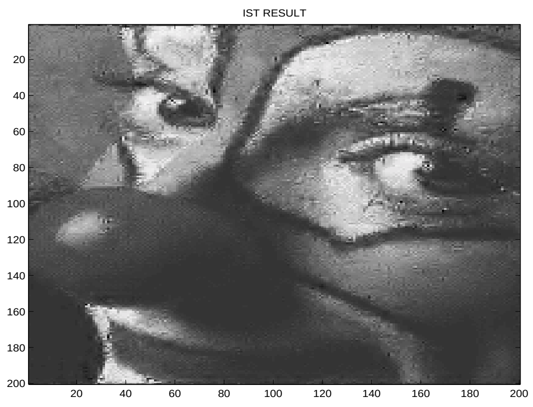


Figure 10.29: Reconstructed (Inpainted) Image.

10.10.4 SPARSA Algorithm

A major problem with IST is that it converges very slowly. Often, thousands of iterations are required for convergence. Many modifications of IST have been proposed. We present one of the simplest, called SPARSA (SPArse Reconstruction by Separable Approximation), applied to the LASSO functional.

The basic idea behind SPARSA is to modify the gradient algorithm Eq. (10.190) to the iteration

$$\underline{x}^{(k+1)} = \underline{x}^{(k)} - (\nabla^2 f(\underline{x}^{(k)}))^{-1} \nabla f(\underline{x}^{(k)}) \qquad (10.197)$$

which is *Newton's method* applied to finding the solution to $\nabla f(\underline{x}) = \underline{0}$, which is the minimum of $f(\underline{x})$.

Eq. (10.197) cannot be used directly on LASSO, since $(\nabla^2 f(\underline{x}))^{-1} = (A^T A)^{-1}$, which does not exist since A^TA has rank $M < N$. Instead, $\nabla^2 f(\underline{x})$ is approximated by $c^{(k)}I$, where

$$c^{(k)} = \frac{||A(x^{(k)} - x^{(k-1)})||_2^2}{||x^{(k)} - x^{(k-1)}||_2^2} \tag{10.198}$$

so that $a = \frac{1}{c^{(k)}}$ is the step size used in Eq. (10.190).

This form of SPARSA can be summarized as:

Initialize: $x^{(0)}=\underline{0}$ and $c^{(0)}=1$.

Landweber: $x^{(k+1)}) = x^{(k)} + A^T(\underline{y} - Ax^{(k)})/c^{(k)}$.

Soft Threshold: Modify λ to $\frac{\lambda}{c^{(k)}}$, giving:

$$x_i^{(k+1)} = \begin{cases} x_i^{(k+1)} - \frac{\lambda}{c^{(k)}} & \text{if } x_i^{(k+1)} > +\frac{\lambda}{c^{(k)}} \\ x_i^{(k+1)} + \frac{\lambda}{c^{(k)}} & \text{if } x_i^{(k+1)} < -\frac{\lambda}{c^{(k)}} \\ 0 & \text{if } |x_i^{(k+1)}| < \frac{\lambda}{c^{(k)}}. \end{cases}$$

Step Size: $c^{(k+1)}=\frac{||A(x^{(k+1)}-x^{(k)})||_2^2}{||x^{(k+1)}-x^{(k)}||_2^2}$.

Stop: When $||x^{(k+1)} - x^{(k)}||_2^2 < \epsilon$ for some $\epsilon << 1$.

SPARSA converges much faster than ITA. But the LASSO functional can increase for a few iterations, and then continue decreasing. Both of these effects are illustrated in the following example.

Example 10-21: SPARSA vs. ITA

Compare the performances of SPARSA and ITA on a small LASSO minimization problem with:

- $K = 12; M = 50; N = 100; \lambda = 0.01$.
- $a_{i,j}$ is zero-mean white Gaussian.
- x_n are at 12 randomly-chosen locations.

Solution:

Results are shown in Figs. 10-30 through 10-32.

Fig. 10-30 plots the ℓ_2 norm errors for ITA and SPARSA vs. number of iterations. SPARSA (in blue) has converged after 50 iterations, while IST has barely decreased.

Fig. 10-31 plots the reconstructed x_n vs. n using IST. The reconstruction is poor after 65 iterations.

Fig. 10-32 plots the reconstructed x_n vs. n using SPARSA. The reconstruction is quite good.

The Matlab code for this example is on the CD.

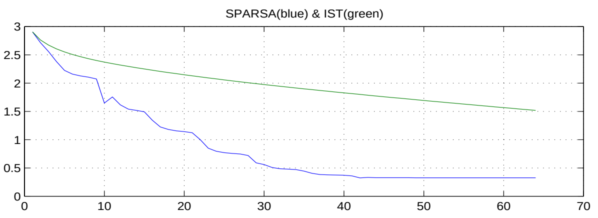


Figure 10.30: ℓ_2 norm errors for SPARSA (blue) and ITA (green) vs. number of iterations. SPARSA converged after 50 iterations, while ITA has barely decreased.

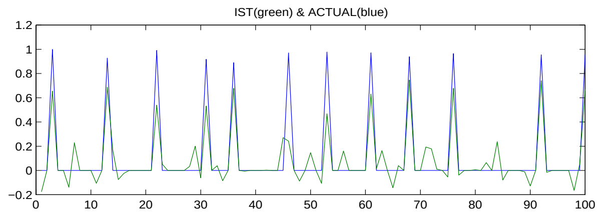


Figure 10.31: Reconstructed x_n vs. n using IST. The reconstruction is not good after 65 iterations.

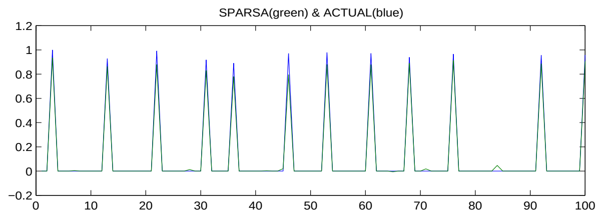


Figure 10.32: Reconstructed x_n vs. n using SPARSA. The reconstruction is quite good after 65 iterations.

Conclusion

There are many other approaches and algorithms for reconstruction of signals and images that are sparse or sparsifiable by wavelet (or other) transforms. However, many of these (such as the family of matching pursuit algorithms) require that the matrix A be formed or accessed. This greatly increases memory and computational requirements.

The major advantage of the IST algorithm over other commonly-used algorithms for sparse reconstruction is that the only significant operations are two matrix-vector products using $\tilde{A}$ and $\tilde{A}^T$. But

$$\begin{aligned}\tilde{A}x &= AW^{-1}x \\ \tilde{A}^T x &= W^{-T}A^T x = WA^T x\end{aligned} \tag{10.199}$$

Multiplication by W implements a wavelet transform and multiplication by W^{-1} implements an inverse wavelet transform. Often multiplication by A can be implemented using an algorithm like the FFT. So the matrices A, W, A^T, W^{-1} need never even be formed, let alone multiplied by vectors. This is a huge savings in storage and computational requirements.

We have not discussed sufficient conditions on A and x for ℓ_1-norm minimization to determine the sparse solution. One such condition for A is *restricted isometry*, but this is impractical to check. A general guideline is that the more random are A and nonzero locations of x, the more likely is ℓ_1-norm minimization (or any other approach) to work. This is still very much an active area of research.

Chapter 10 Summary

Concepts

- LTI, convolution, impulse response (now PSF), DTFT (now DSFT), frequency (now wavenumber) response, DFT, and FFT all generalize directly from 1-D (signal) to 2-D (image) processing.
- Deconvolution from noisy data requires Tikhonov regularization, which is performed by a Wiener filter.
- The discrete-time wavelet transform splits signals into average signals and detail signals, whose total length matches that of the original signal. But the detail signals are sparse (mostly zero). This also applies to images. It is very fast.
- The Daubechies DL wavelet transform models signals as piecewise-$(L-1)^{th}$ polynomials. The Haar transform is a $D1$ wavelet transform.
- Wavelet transforms are useful for compressing, denoising, deconvolving, reconstructing, and inpainting images.
- Compressed sensing allows reconstruction of sparsifiable signals or images using many fewer linear combinations of the signal or image values than the number of values.
- IRLS, IST and SPARSA are iterative algorithms for compressed sensing problems.

Mathematical Models

- Wiener Filter: $\mathbf{G}_{k_1,k_2} = \mathbf{H}^*_{k_1,k_2}/[|\mathbf{H}_{k_1,k_2}|^2 + \lambda^2]$
- ℓ_1 Norm: $||x[n]||_1 = \sum |x[n]|$
- LASSO cost functional: $\Lambda = \frac{1}{2}||\underline{y} - AW^{-1}\underline{z}||_2^2 + \lambda||\underline{z}||_1$
- Landweber Iteration: $x^{(k+1)} = x^{(k)} + A^T(y - Ax^{(k)})$
- Shrinkage: $y[n] = x[n] - \lambda \cdot \text{sign}(x[n])$
- Threshold: $y[n] = x[n]$ if $|x[n]| > \lambda$; $y[n] = 0$ if $|x[n]| < \lambda$
- IST Algorithm: Landweber with thresholding and shrinkage

Glossary of Important Terms

Provide definitions or explain the meaning of the following terms:

Analysis Filter Bank	Average Signal
Compressed Sensing	Daubechies Wavelets
Image Deconvolution	Image Denoising
Detail Signal	DSFT (2-D DTFT)
Haar Wavelets	IST Algorithm
ℓ_1 Norm	Perfect Reconstruction
Piecewise Polynomial	Point-Spread Function
Scaling Function	Shepp-Logan Phantom
Smith-Barnwell Condition	SPARSA Algorithm
Sparse Signal or Image	Synthesis Filter Bank
Threshold and Shrink	Tikhonov Regularization
Wavelet Function	Wavelet Transform
Wavenumber Response	Wiener Filter

10-18.3 Valid 2-D Deconvolution

(a) Definition of valid convolution

Given an $M \times M$ image $x[m,n]$ and an $L \times L$ point spread function (PSF) $h[m,n]$, their 2-D convolution generates an $(M+L-1)\times(M+L-1)$ blurred image $y[m,n]$. The process is reversible: the blurred image can be deconvolved to reconstruct $x[m,n]$ by subjecting $y[m,n]$ to a Wiener filter, as described earlier in Section 10-7. To do so, however, requires that all of $y[m,n]$ be known.

Often, we encounter deconvolution applications where only a fraction of $y[m,n]$ is known, specifically, the part of $y[m,n]$ called the ***valid convolution***. This is the part whose convolution computation does not require the image $x[m,n]$ to be zero-valued outside the square $0 \le m,n \le M-1$.

For $L < M$ (image larger than PSF), the valid 2-D convolution of $h[m,n]$ and $x[m,n]$ is defined as

$$\begin{aligned} y_V[m,n] &= \sum_{i=0}^{M-1}\sum_{j=0}^{M-1} x[i,j]\,h[m-i,\ n-j] \\ &= h[m,n] * x[m,n],\ \text{restricted to} \\ &\qquad \{\,L-1 \le m,n \le M-1\,\}. \end{aligned} \tag{10.210}$$

A valid convolution omits all end effects in 1-D convolution and all edge effects in 2-D convolution. Consequently, the size of $y_V[m,n]$ is $(M-L+1)\times(M-L+1)$, instead of $(M+L-1)\times(M+L-1)$ for the complete convolution $y[m,n]$.

To further illustrate the difference between $y[m,n]$ and $y_V[m,n]$, let us consider the following example:

$$x[m,n] = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

and

$$h[m,n] = \begin{bmatrix} 11 & 12 \\ 13 & 14 \end{bmatrix}.$$

Since $M = 3$ and $L = 2$, the 2-D convolution is $(M+L-1)\times(M+L-1) = 4\times 4$, and $y[m,n]$ is given by

$$y[m,n] = \begin{bmatrix} 11 & 34 & 57 & 36 \\ 57 & 143 & 193 & 114 \\ 129 & 293 & 343 & 192 \\ 91 & 202 & 229 & 126 \end{bmatrix}.$$

In contrast, the size of the valid 2-D convolution is $(M-L+1)\times(M-L+1) = 2\times 2$, and $y_V[m,n]$ is given by

$$y_V[m,n] = \begin{bmatrix} 143 & 193 \\ 293 & 343 \end{bmatrix}.$$

The valid convolution $y_v[m,n]$ is the central part of $y[m,n]$, obtained by deleting the edge rows and columns from $y[m,n]$. In MATLAB, the valid 2-D convolution of X and H can be computed using the command

```
Y=conv2(X,H,'valid').
```

(b) Reconstruction from $y_V[m,n]$

The valid 2-D deconvolution problem is to reconstruct an unknown image from its valid 2-D convolution with a known PSF. The 2-D DFT and Wiener filter cannot be used here, since not all of the blurred image $y[m,n]$ is known. It may seem that we may simply ignore, or set to zero, the unknown parts of $y[m,n]$ and still obtain a decent reconstructed image using a Wiener filter, but as we will demonstrate with an example, such an approach does not yield fruitful results.

The valid 2-D deconvolution problem is clearly underdetermined, since the $(M-L+1)\times(M-L+1)$ portion of the blurred image is smaller than the $M\times M$ unknown image. But if $x[m,n]$ is sparsifiable, then valid 2-D deconvolution can be formulated as a compressed sensing problem and solved using the ISTA. The matrix A turns out to be a ***block Toeplitz with Toeplitz blocks*** matrix, but multiplication by A is implemented as a valid 2-D convolution. Multiplication by A^T is implemented as a valid 2-D convolution.

The valid 2-D convolution can be implemented as $\underline{y}_V = A\underline{x}$ where

$$\underline{x} = [1\ \ 2\ \ 3\ \ 4\ \ 5\ \ 6\ \ 7\ \ 8\ \ 9]^T,$$

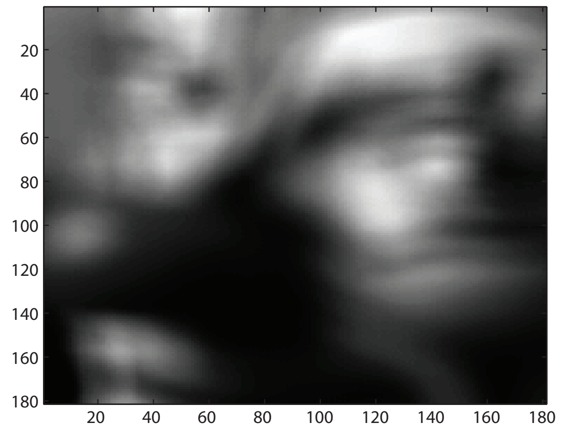
$$\underline{y}_V = [143\ \ 193\ \ 293\ \ 343]^T,$$

and the matrix A is composed of the elements of $h[m,n]$ as follows:

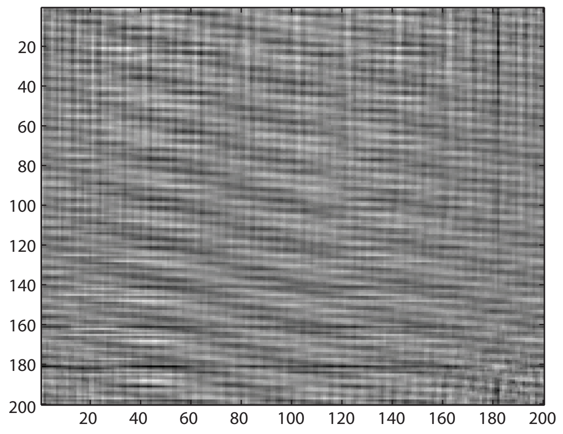
$$A = \begin{bmatrix} 14 & 13 & 0 & 12 & 11 & 0 & 0 & 0 & 0 \\ 0 & 14 & 13 & 0 & 12 & 11 & 0 & 0 & 0 \\ 0 & 0 & 0 & 14 & 13 & 0 & 12 & 11 & 0 \\ 0 & 0 & 0 & 0 & 14 & 13 & 0 & 12 & 11 \end{bmatrix}.$$

Note that A is a 2×3 block matrix of 2×3 blocks. Each block is constant along its diagonals, and the blocks are constant along block diagonals. This is the block Toeplitz with Toeplitz blocks structure. Also note that images $x[m,n]$ and $y_V[m,n]$ have been unwrapped row by row, starting with the top row, and the transposes of the rows are stacked into a column vector. Finally, note that multiplication by A^T can be implemented as a valid 2-D convolution with the doubly reversed version of $h[m,n]$. For example, if

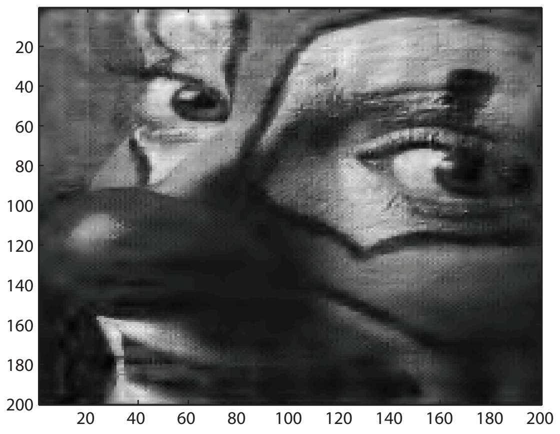
$$z[m,n] = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$



(a) Valid 2-D convolution $y_V[m,n]$ of clown image



(b) 2-D deconvolution using $y_V[m,n]$ and Wiener filter



(c) 2-D deconvolution using $y_V[m,n]$ and ISTA

Figure 10-36: (a) Valid 2-D convolution $y_V[m,n]$ of clown image, (b) deconvolution using Wiener filter, and (c) deconvolution using ISTA.

and

$$g[m,n] = \begin{bmatrix} 14 & 13 \\ 12 & 11 \end{bmatrix} = h[1-m,\ 1-n], \quad \text{with } m,n = 0,1,$$

then the valid 2-D convolution of $z[m,n]$ and $g[m,n]$ is

$$w_V[m,n] = \begin{bmatrix} 184 & 234 & 284 \\ 384 & 434 & 484 \\ 584 & 634 & 684 \end{bmatrix}.$$

This valid 2-D convolution can also be implemented as $\underline{w} = A^T \underline{z}$ where

$$\underline{z} = [1\ \ 2\ \ 3\ \ 4\ \ 5\ \ 6\ \ 7\ \ 8\ \ 9\ \ 10\ \ 11\ \ 12\ \ 13\ \ 14\ \ 15\ \ 16]^T$$

and

$$\underline{w} = [184\ \ 234\ \ 284\ \ 384\ \ 434\ \ 484\ \ 584\ \ 634\ \ 684]^T.$$

To illustrate the process with an image, we computed the valid 2-D convolution of the 200×200 clown image with a 20×20 PSF. The goal is to reconstruct the clown image from the 181×181 blurred image shown in **Fig. 10-36(a)**. The $D3$ Daubechies wavelet function was used to sparsify the image. Here, $M = 200$ and $L = 20$, so the valid 2-D convolution has size $(M - L + 1) \times (M - L + 1) = 181 \times 181$. In terms of $\underline{y}_V = A\underline{x}$, A is $181^2 \times 200^2 = 32761 \times 40000$.

Parts (b) and (c) of **Fig. 10-36** show reconstructed versions of the clown image, using a Wiener filter and ISTA, respectively. Both images involve deconvolution using the restricted valid convolution data $y_V[m,n]$. In the Wiener-image approach, the unknown parts of the blurred image (beyond the edges of $y_V[m,n]$) were ignored, and the resultant image bears no real resemblance to the original clown image. In contrast, the ISTA approach provides excellent reconstruction of the original image. This is because ISTA is perfectly suited for solving underdetermined systems of linear equations with sparse solutions.

10-18.4 Computed Axial Tomography (CAT)

Computed axial tomography, also known as ***CAT scan***, is a technique capable of generating 3-D images of the X-ray attenuation (absorption) properties of an object, such as the human body. The X-ray absorption coefficient of a material is strongly dependent on the density of that material. CAT has the sensitivity necessary to image body parts across a wide range of densities, from soft tissue to blood vessels and bones.

As depicted in **Fig. 10-37(a)**, a CAT scanner uses an X-ray source, with a narrow slit to generate a fan-beam, wide enough to encompass the extent of the body, but only a few millimeters in thickness. The attenuated X-ray beam is captured by an array of ~ 700 detectors. The X-ray source and the detector array are mounted on a circular frame that rotates in steps of a fraction of a degree over a full 360° circle around the object or patient, each time recording an X-ray attenuation profile from a different angular direction. Typically, on the order of 1000 such profiles are recorded, each composed of measurements by 700 detectors. For each horizontal slice of the body, the process is completed in less than 1 second. CAT performs a deconvolution to generate a 2-D image of the absorption coefficient of that horizontal slice. To image an entire part of the body, such as the chest or head, the process is repeated over multiple slices (layers). Our current interest is in the deconvolution process, so we limit our treatment to the 2-D case.

For each anatomical slice, the CAT scanner generates on the order of 7×10^5 measurements (1000 angular orientations $\times$700 detectors). In terms of the coordinate system shown in **Fig. 10-37(b)**, we define $\alpha(\xi, \eta)$ as the absorption coefficient of the object under test at location (ξ, η). The X-ray beam is directed along the ξ direction at $\eta = \eta_0$. The X-ray intensity received by the detector located at $\xi = \xi_0$ and $\eta = \eta_0$ is given by

$$I(\xi_0, \eta_0) = I_0 \exp\left[-\int_0^{\xi_0} \alpha(\xi, \eta_0)\, d\xi\right], \qquad (10.211)$$

where I_0 is the X-ray intensity radiated by the source. Outside the body, $\alpha(\xi, \eta) = 0$. The corresponding logarithmic ***path attenuation*** $p(\xi_0, \eta_0)$ is defined as

$$p(\xi_0, \eta_0) = -\log \frac{I(\xi_0, \eta_0)}{I_0} = \int_0^{\xi_0} \alpha(\xi, \eta_0)\, d\xi. \qquad (10.212)$$

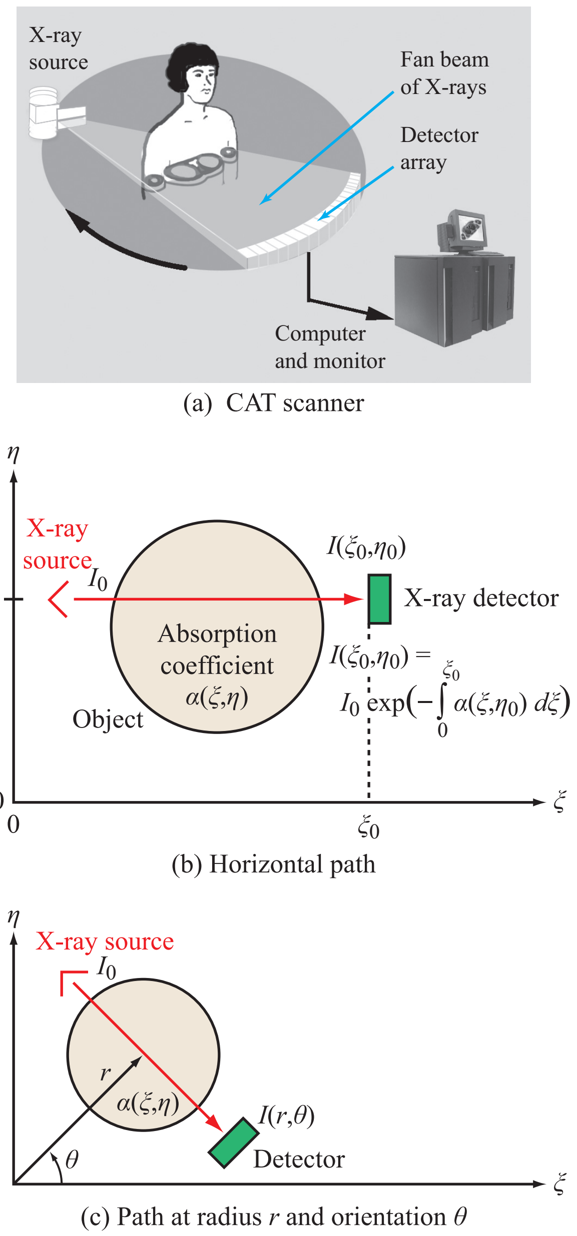


Figure 10-37: (a) CAT scanner, (b) X-ray path along ξ, and (c) X-ray path along arbitrary direction.

The path attenuation $p(\xi_0, \eta_0)$ is the integrated absorption coefficient across the X-ray path.

In the general case, the path traversed by the X-ray source is at a range r and angle θ in a polar coordinate system, as depicted in **Fig. 10-37(c)**. The direction of the path is orthogonal to the direction of r. For a path corresponding to a specific set (r, θ), Eq. (10.212) becomes

$$p(r,\theta) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \alpha(\xi,\eta)\ \delta(r - \xi\cos\theta - \eta\sin\theta)\ d\xi\ d\eta, \tag{10.213}$$

where the impulse function $\delta(r - \xi\cos\theta - \eta\sin\theta)$ dictates that only those points in the (ξ, η) plane that fall along the path specified by fixed values of (r, θ) are included in the integration.

The relation between $p(r, \theta)$ and $\alpha(\xi, \eta)$ is known as the 2-D ***Radon transform*** of $\alpha(\xi, \eta)$. The goal of CAT is to reconstruct $\alpha(\xi, \eta)$ from the measured path attenuations $p(r, \theta)$, by inverting the Radon transform given by Eq. (10.213). We do so with the help of the Fourier transform.

Recall from entry #1 in **Table 5-6** that for variable r, $\mathcal{F}\{\delta(r)\} = 1$, and from entry #4 in **Table 5-7** that the shift property is

$$\mathcal{F}\{x(r - r_0)\} = \mathbf{X}(\omega)\ e^{-j\omega r_0}.$$

The combination of the two properties leads to

$$\begin{aligned}\mathcal{F}\{\delta(r - \xi\cos\theta - \eta\sin\theta)\}\\ &= \int_0^{\infty} \delta(r - \xi\cos\theta - \eta\sin\theta)\ e^{-j\omega r}\ dr\\ &= e^{-j\omega(\xi\cos\theta+\eta\sin\theta)} = e^{-j(\omega_1\xi+\omega_2\eta)},\end{aligned} \tag{10.214}$$

where we define angular frequencies ω_1 and ω_2 as

$$\omega_1 = \omega\cos\theta, \tag{10.215a}$$

$$\omega_2 = \omega\sin\theta. \tag{10.215b}$$

Next, let us define $\mathbf{A}$ as the 2-D Fourier transform of the absorption coefficient $\alpha(\xi, \eta)$ using the relationship given by Eq. (5.143a):

$$\mathbf{A}(\omega_1,\omega_2) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \alpha(\xi,\eta)\ e^{-j\omega_1\xi} e^{-j\omega_2\eta}\ d\xi\ d\eta. \tag{10.216}$$

If we know $\mathbf{A}(\omega_1, \omega_2)$, we can perform an inverse 2-D Fourier transform to retrieve $\alpha(\xi, \eta)$. To do so, we need to relate $\mathbf{A}(\omega_1, \omega_2)$ to the measured path attenuation profiles $p(r, \theta)$. To that end, we use Eq. (10.213) to compute $\mathbf{P}(\omega, \theta)$, the 1-D Fourier transform of $p(r, \theta)$:

$$\begin{aligned}\mathbf{P}(\omega,\theta) &= \int_0^{\infty} p(r,\theta)\ e^{-j\omega r}\ dr\\ &= \int_0^{\infty}\Bigg[\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \alpha(\xi,\eta)\\ &\qquad \cdot\ \delta(r - \xi\cos\theta - \eta\sin\theta)\ d\xi\ d\eta\Bigg] e^{-j\omega r}\ dr.\end{aligned} \tag{10.217}$$

By reversing the order of integration, we have

$$\begin{aligned}\mathbf{P}(\omega,\theta) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \alpha(\xi,\eta)\\ &\quad \cdot\left[\int_0^{\infty} \delta(r - \xi\cos\theta - \eta\sin\theta)\ e^{-j\omega r}\ dr\right] d\xi\ d\eta.\end{aligned} \tag{10.218}$$

We recognize the integral inside the square bracket as the Fourier transform of the shifted impulse function, as given by Eq. (10.214). Hence, Eq. (10.218) simplifies to

$$\mathbf{P}(\omega,\theta) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \alpha(\xi,\eta)\ e^{-j(\omega_1\xi+\omega_2\eta)}\ d\xi\ d\eta, \tag{10.219}$$

which is identical to Eq. (10.216). Hence,

$$\mathbf{A}(\omega_1,\omega_2) = \mathbf{P}(\omega,\theta), \tag{10.220}$$

where $\mathbf{A}(\omega_1, \omega_2)$ is the 2-D Fourier transform of $\alpha(\xi, \eta)$, and $\mathbf{P}$ is the 1-D Fourier transform (with respect to r) of $p(r, \theta)$. The variables (ω_1, ω_2) and (ω, θ) and related by Eq. (10.215).

If $p(r, \theta)$ is measured for all r across the body of interest and for all directions θ, then its 1-D Fourier transform $\mathbf{P}(\omega, \theta)$ can be computed, and then converted to $\mathbf{A}(\omega_1, \omega_2)$ using Eq. (10.215). The conversion is called the ***projection-slice theorem***. In practice, however, $p(r, \theta)$ is measured for only a finite number of angles θ, so $\mathbf{A}(\omega_1, \omega_2)$ is known only along radial slices in the 2-D wavenumber domain (ω_1, ω_2). Reconstruction to find $\alpha(\xi, \eta)$ from a subset of its 2-D Fourier transform values is a perfect example of compressed sensing.

Image reconstruction from partial radial slices

To demonstrate the reconstruction process, we computed the 2-D DFT $\mathbf{X}[k_1, k_2]$ of a 256×256 Shepp-Logan phantom image, and then retained the data values corresponding to only 12 radial slices, as shown in **Fig. 10-38(a)**. These radial slices simulate $\mathbf{P}(\omega, \theta)$, corresponding to 12 radial measurements $p(r, \theta)$. In terms of $\underline{y} = A\underline{x}$, the number of pixels in the frequency domain image is $N = 65536$, and the number of values contained in the 12 radial slices is $M = 11177$.

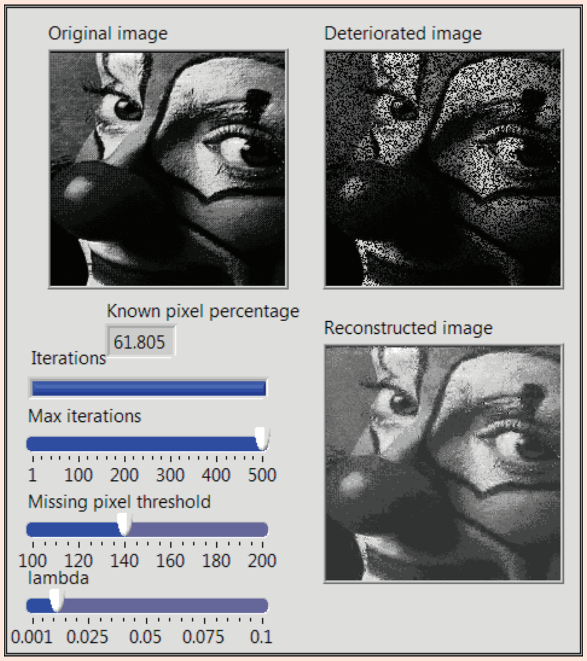
(a) Least-squares reconstruction: Unknown values of $\mathbf{X}[k_1, k_2]$ were set to zero, and then the inverse 2-D DFT was computed. The resulting image is displayed in **Fig. 10-38(b)**.

(b) ISTA reconstruction: Application of ISTA with $\lambda = 0.01$ for 1000 iterations led to the image in **Fig. 10-38(c)**, which bears very good resemblance to the original image.

Concept Question 10-20: Why is it possible to reconstruct a real-world image almost perfectly from only a subset of its 2-D DFT values, or a subset of its pixel values?

Exercise 10-15: Use LabVIEW Module 10.7 to inpaint the clown image. Use lambda $= 0.01$, missing pixel threshold $= 140$, and max iterations $= 500$.

Answer:



Original image

Deteriorated image

Known pixel percentage 61.805

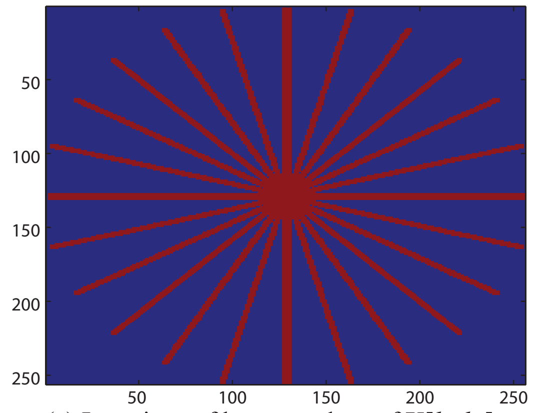
Iterations

Max iterations

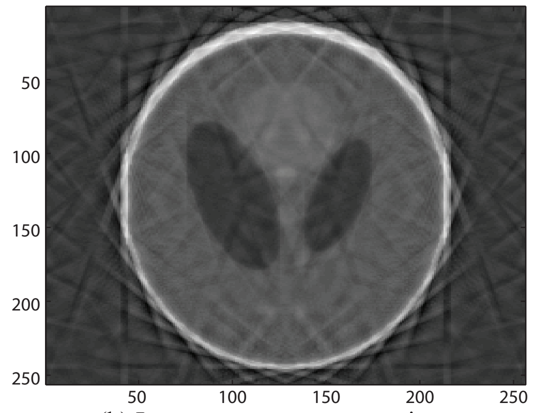
Missing pixel threshold

lambda

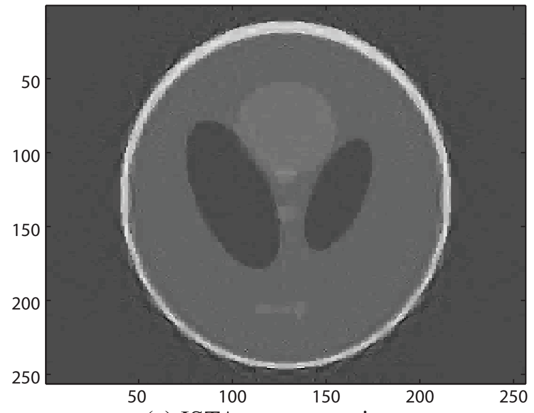
Reconstructed image



(a) Locations of known values of $\mathbf{X}[k_1,k_2]$



(b) Least-squares reconstruction



(c) ISTA reconstruction

Figure 10-38: Shepp-Logan phantom image reconstruction from partial radial slices of its 2-D DFT: (a) radial slices of $\mathbf{X}[k_1, k_2]$, (b) least-squares reconstruction, and (c) ISTA reconstruction.