

CHAPTER 1

INTRODUCTION: WHY WAVELETS?

Why wavelets? This is certainly a fair question for those interested in this book and also one that can be posed in multiple contexts. Students wonder why they need to learn about wavelets. Researchers ask why wavelets work so well in certain applications. Scientists want to know why they should consider using wavelets in applications instead of other popular tools. Finally, there is curiosity about the word itself — why *wavelets*? What are wavelets? Where did they come from?

In this short introduction to the book, we answer these questions and in the process, provide some information about what to expect from the chapters that follow.

Image Compression

In keeping with the true spirit of the book, let's start with an application in image compression.

Suppose that you wish to use the Internet to send a friend the digital image file plotted in Figure 1.1. Since the file is quite large, you first decide to *compress* it. Suppose that it is permissible to sacrifice image resolution in order to minimize

transmission time. The dimensions of the image, in pixels,¹ are 512 rows by 512 columns. The total number of elements that comprise the image is $512 \times 512 = 262,144$.

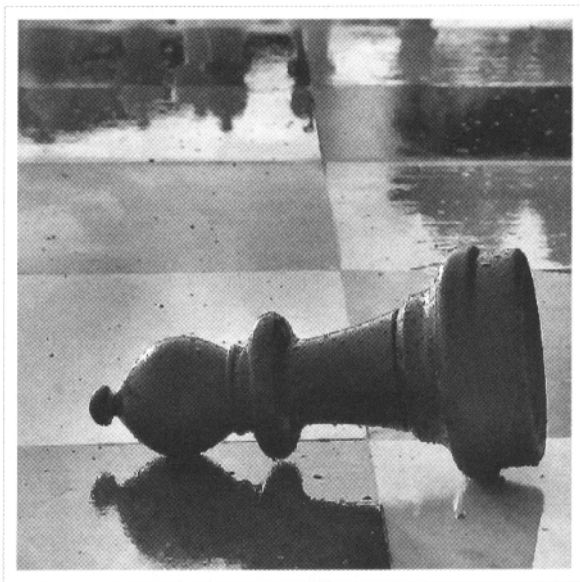


Figure 1.1 A digital image.

What does it even mean to compress the image? How do you suppose that we compress the image? How do we measure the effectiveness of the compression method?

As you will learn in Chapter 3, each pixel value is an integer from 0 (black) to 255 (white) and each of these integers is stored on a computer using 8 *bits* (the value of a bit is either 0 or 1). Thus we need $262,144 \times 8 = 2,097,152$ bits to represent the image. To compress an image, we simply wish to reduce the number of bits needed to store it. Most compression methods follow the basic algorithm given in Figure 1.2.

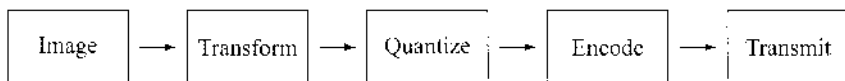


Figure 1.2 A basic algorithm for compressing a digital image.

¹You can think of a pixel as a small rectangle on your computer screen or paper that is rendered at some gray level between black and white.

Note that the first step is to *transform* the image. The goal here is to map the integers that comprise the image to a new set of numbers. In this new setting we alter (*quantize*) some or all of the values so that we can write (*encode*) the modified values using fewer bits. In Figure 1.3, we have plotted one such transformation of the image from Figure 1.1. The figure also contains a plot of the quantized transform. The transformation in Figure 1.3 is a *discrete wavelet transform*. In fact, it is the same transformation as that used in the JPEG2000 image compression standard.²

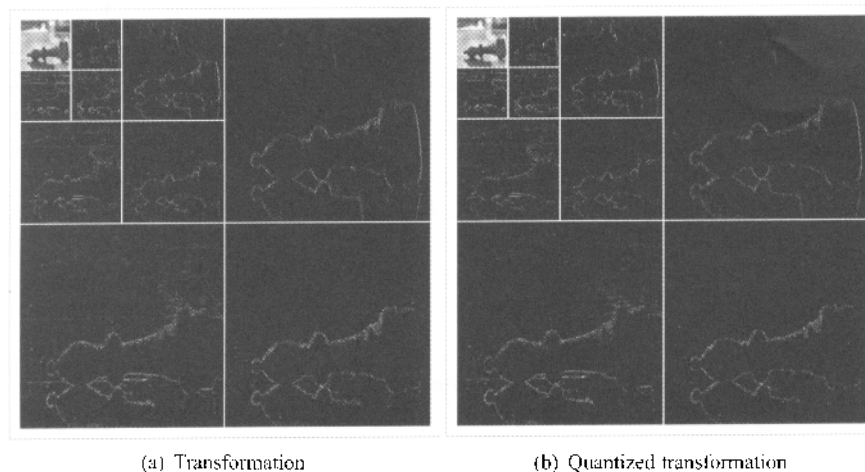


Figure 1.3 A transformation and the quantized transformation of the image in Figure 1.1. The white lines that appear in both images are superimposed to separate the different parts of the transform.

In mathematical terms, if the image is represented by the square matrix A , then the transformation is performed by constructing special matrices W and \tilde{W} ³ and then computing WAW^T . Of course, the key to the process is understanding the construction of W , \tilde{W} and how WAW^T concentrates most of the nonzero values (nonblack pixels) in the upper left-hand corner of the transformation. You might already begin to see why the wavelet transformation is useful for reducing the number of bits. The transformation has created large regions that are black or nearly black (i.e., regions where most of values are either zero or near zero). It is natural to believe that you would need less information to store these values than what is required for the original image.

The next step in the process is to quantize the transform. It turns out that zero is the only integer value in the transformed output. Thus each value must be converted

²You will learn about JPEG2000 in Chapter 12.

³In some cases, we choose $\tilde{W} = W$.

(rounded, for example) to an integer. The quantizer also makes decisions about certain values and may reduce them in size or even convert them to zero. Values that are reduced or converted to zero are those that the quantizer believes will not adversely affect the resolution of the compressed image. After quantization, it is impossible to exactly retrieve the original image⁴. It may be hard to see in Figure 1.3, but nearly all the values of the quantized transformation are different than the corresponding values in the transformation.

The final step before transmission is to *encode* the quantized transformation. That is, instead of using 8 bits to store each integer, we will try to group together like integers and possibly use a smaller number of bits to store those values that occur with greater frequency. Since the quantized wavelet transformation contains a large number of zeros (black pixels), we expect the encoded transform to require fewer bits. Indeed, using a very naive encoding scheme (see Section 3.4), we find that only 548,502 bits are needed to store the quantized wavelet transform. This is about 26% of the number of bits required to store the original!

To view the compressed image, the receiver decodes the transmitted file and then applies the inverse wavelet transform. The compressed image appears in Figure 1.4. It is very difficult to tell the images in Figures 1.1 and 1.4 apart!

Other Applications That Use Wavelet Transformations

Wavelets, or more precisely, wavelet transformations, abound in image-processing applications. The Federal Bureau of Investigation uses wavelets in their Wavelet/Scalar Quantization Specification to compress digital images of fingerprints [8]. In this case there is an objective measure of the effectiveness of the compression method — the uncompressed fingerprint file must be uniquely matched with the correct person. In many instances it is desirable to identify edges of regions that appear in digital images. Wavelets have proved to be an effective way to perform edge detection in images [56, 55]. Other image-processing applications where wavelets are used are image morphing [43] and digital watermarking [86, 32]. Image morphing is a visualization technique that transforms one image into another. You have probably seen image morphing as part of a special effect in a movie. Digital watermarking is the process by which information is added (either visible or invisible) to an image for the purpose of establishing the authenticity of the image.

Wavelets are used in applications outside image processing. For example, wavelet-based methods have proved effective in the area of signal denoising [28, 85, 81]. Cipra [19] recounts an interesting story of Yale University mathematician Ronald Coifman and his collaborators using wavelets to denoise an old cylinder recording (made of a radio broadcast) of Brahms playing his *First Hungarian Dance* [4]. It is impossible to identify the piano in the original but a restored version (using wavelets)

⁴For those readers with some background in linear algebra, W^{-1} , \tilde{W}^{-1} both exist so it is possible to completely recover A before the quantization step.

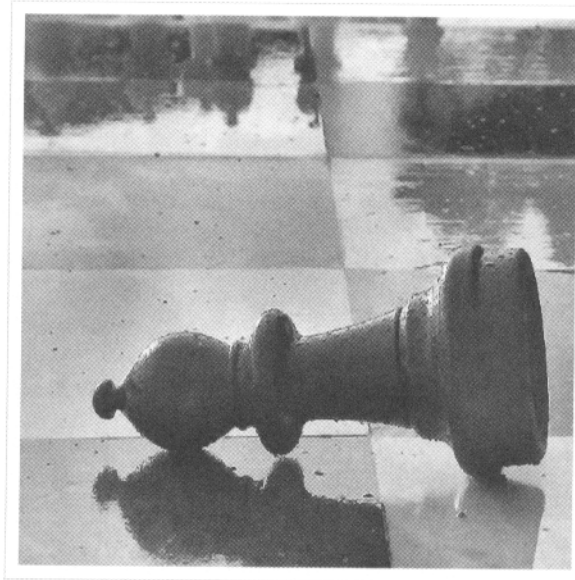


Figure 1.4 The decompressed digital image. Whereas each pixel in the original image is stored using 8 bits, the pixels in this image require 2.09 bits storage on average.

clearly portrays the piano melody. Stanford University music professor and Coifman collaborator Jonathan Berger maintains a very interesting Web site on this project [5]. Wavelet-based denoising (wavelet shrinkage) is also used in applications in finance and economics [38]. We study wavelet shrinkage in Chapter 9.

Wavelets have been used to examine electroencephalogram data in order to detect epileptic seizures [3], model distant galaxies [6], and analyze seismic data [45, 46]. Finally, in more mathematical applications, wavelets are used to estimate densities and to model time series in statistics [81] and in numerical methods for solving partial differential equations [22].

Wavelet Transforms Are Local Transforms

We have by no means exhausted the list of applications that utilize wavelets. Understand as well that wavelets are not always the best tool to use for a particular application. There are many applications in audio and signal processing for which Fourier methods are superior, and applications in image processing where filtering methods other than wavelet-based methods are preferred.

There are several reasons that wavelets work well in many applications. Probably the most important is the fact that a discrete wavelet transform is a *local transform*. To

understand what is meant by this term, consider the 100-term signal in Figure 1.5(a). Four of the signal values are altered and the result is plotted in Figure 1.5(b).

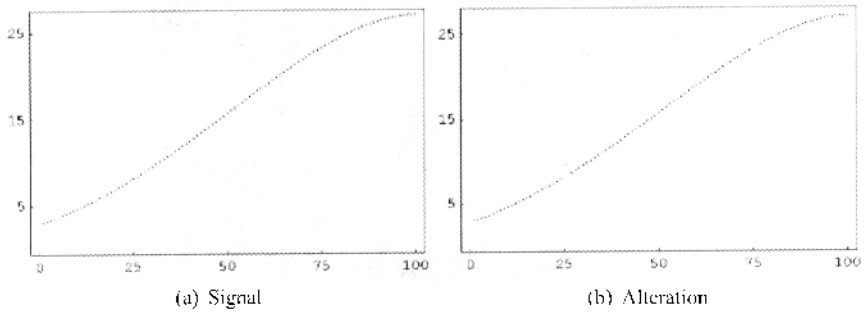


Figure 1.5 A signal and alteration of the signal.

In Figure 1.6 we have plotted a discrete wavelet transformation of each signal from Figure 1.5. The wavelet transform in Figure 1.6(a) is again simply a matrix we use to multiply with the input vector. The output consists of two parts. The first 50 components of the transform serve as an approximation of the original signal. The last 50 elements basically describe the differences between the original signal and the approximation. Since the original signal is relatively smooth, it is not surprising that the differences between the original signal and the approximation are quite small.

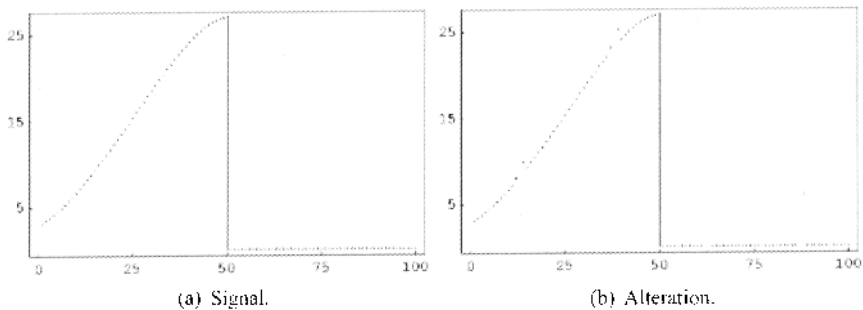


Figure 1.6 Discrete wavelet transformations of each of the signals in Figure 1.5.

Figure 1.7 gives a better view of the difference portion of the wavelet transform in Figure 1.6(a). Now look at the transformation in Figure 1.6(b). In both portions of the transform, we see two small regions that are affected by the changes we made to produce the altered signal in Figure 1.5(b). The important point to note is that the regions where the transform was affected by the altered data are relatively small. Thus,

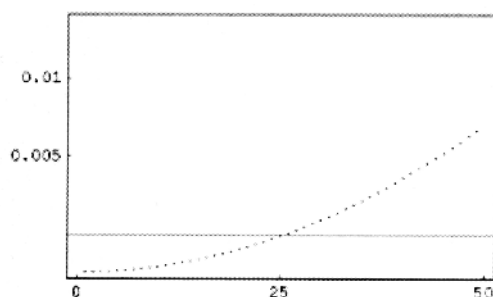


Figure 1.7 The difference portion of the discrete wavelet transformation from Figure 1.6(a).

small changes in the input data result in small changes in the wavelet-transformed data.

One of the most popular and useful tools for signal-processing applications is the *discrete Fourier transformation* (DFT). It is built using sampled values of cosine and sine. Although we will not delve into the specifics of the construction of the transform and how it is used in applications, it is worthwhile to view the DFT of the signals in Figure 1.5. We have plotted the modulus (see Section 4.1) of these DFTs in Figure 1.8.

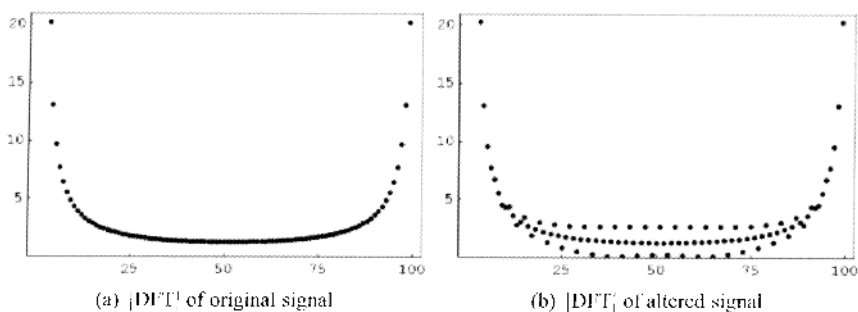


Figure 1.8 The moduli of the discrete Fourier transforms of each of the signals in Figure 1.5.

Look at what happened to the DFT of the altered data. Although we changed only four values, the effect of these changes on the transform is *global*⁵. The DFT is constructed from samples of sine and cosine functions.⁶ These functions oscillate

⁵After examining the images in Figure 1.8, an expert in Fourier analysis would undoubtedly be able to ascertain that a few of the values in the original signal had been altered but would have no way of knowing the locations of the altered values.

⁶A nice derivation of the DFT appears in Kammler [51].

between -1 and 1 for all time and never decay to 0 . Thus the effects of any minor change in the input data will be felt throughout the entire transformed output.

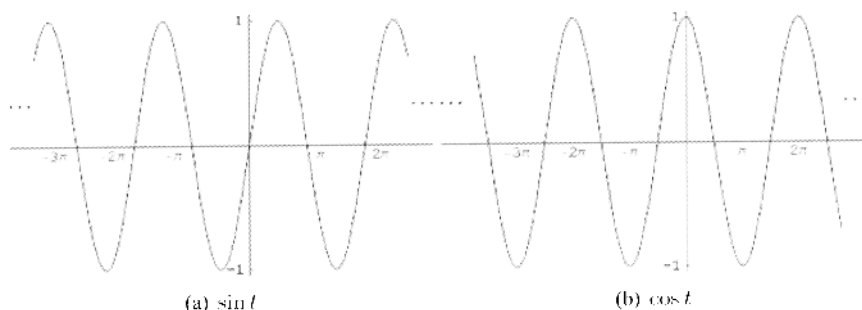


Figure 1.9 Sine and cosine. Both functions oscillate between -1 and 1 and never decay to zero.

Classical Wavelet Theory in a Nutshell

The classical approach to wavelet theory utilizes the setting of square integrable functions on \mathbb{R} and seeks to elicit the elements of the discrete transform from oscillatory functions (e. g., sine and cosine; Figure 1.9) that are required to decay to zero. That is, the underlying functions are tiny waves or *wavelets*. A couple of popular wavelet functions are plotted in Figure 1.10.

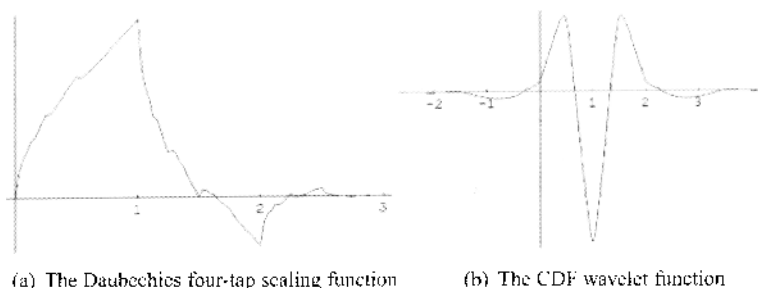


Figure 1.10 Some functions from classical wavelet theory.

Note that the functions in Figure 1.10 are oscillatory and have value zero outside a finite interval. The function in Figure 1.10(a) is probably one of the most famous in all of wavelet theory. It is Ingrid Daubechies' *four-tap scaling function*⁷ [26]. It gives rise

⁷A scaling function is called a *father wavelet function* by some researchers.

to the four numbers we use to build a wavelet transformation matrix in Section 7.1. The function in 1.10(b) is a wavelet function constructed by Albert Cohen, Ingrid Daubechies, and Jean-Christophe Feauveau [20]. It provides the elements of the wavelet transform matrix used in the JPEG2000 image compression standard.

If these functions really do lead to the discrete wavelet transformations used in so many applications, several questions naturally arise: Where do these functions come from? How are they constructed? Once we have built them, how do we extract the elements that are used in the discrete wavelet transforms?

Although the mathematics of wavelets can be traced back to the early twentieth century,⁸ most researchers mark the beginning of modern research in wavelet theory by the 1984 work [41] of French physicists Jean Morlet and Alexander Grossmann. Yves Meyer [59] (in 1992) and Stéphane Mallat [57] (in 1989) produced foundational work on the so-called *multiresolution analysis* and using this construct, Ingrid Daubechies constructed wavelet functions [23, 24] (in 1988, 1993, respectively) that give rise to essentially all the discrete wavelet transforms found in this book.

To (over)simplify the central idea of Daubechies, we seek a function $\phi(t)$ (use the function in Figure 1.10(a) as a reference) that satisfies a number of properties. In particular, $\phi(t)$ should be zero outside a finite interval and $\phi(t)$ and its integer translates $\phi(t - k)$, $k = 0, \pm 1, \pm 2, \pm 3, \dots$, (see Figure 1.11) should form a *basis* for a particular space. The function $\phi(t)$ should be suitably smooth⁹ and the functions should also satisfy

$$\int_{\mathbb{R}} \phi(t)\phi(t - k) dt = 0, \quad k = \pm 1, \pm 2, \dots \quad (1.1)$$

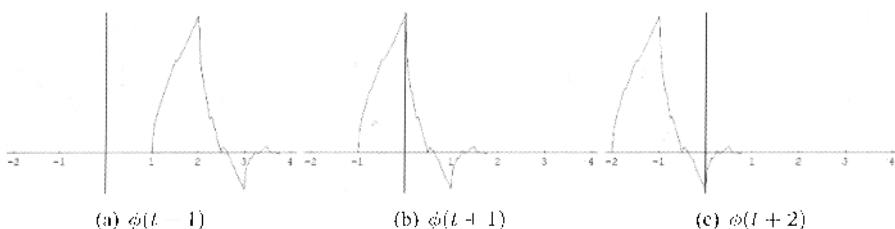


Figure 1.11 Some integer translates of the function $\phi(t)$ from Figure 1.10(a).

Finally, we should be able to write $\phi(t)$ as a combination of dilations (contractions, actually) and translations of itself. For example, the function $\phi(t)$ in Figure 1.10(a) satisfies the *dilation equation*

⁸Yves Meyer [59] gives a nice discussion of the origins of wavelet theory in the twentieth century, and a wonderful exposition on the history of wavelet theory, its origins, and uses in applications can be found in a 1998 book by Barbara Burke Hubbard [47].

⁹Believe it or not, in a certain sense, the function in Figure 1.10(a) and its integer translates can be used to represent linear polynomials!

$$\phi(t) = h_0\phi(2t) + h_1\phi(2t-1) + h_2\phi(2t-2) + h_3\phi(2t-3) \quad (1.2)$$

The function $\phi(t)$ in Figure 1.10(a) is nonzero only on the interval $[0, 3]$. The function $\phi(2t)$ is a contraction — it is nonzero only on $[0, \frac{3}{2}]$. For $k = 1$, $\phi(2t-1) = \phi(2(t - \frac{1}{2}))$, so we see that $\phi(2t-1)$ is simply $\phi(2t)$ translated $\frac{1}{2}$ unit right. In a similar way, we see that $\phi(2t-2)$ and $\phi(2t-3)$ are obtained by translating $\phi(2t)$ 1 and $\frac{3}{2}$ units right, respectively. The functions on the right side of (1.2) are plotted in Figure 1.12.

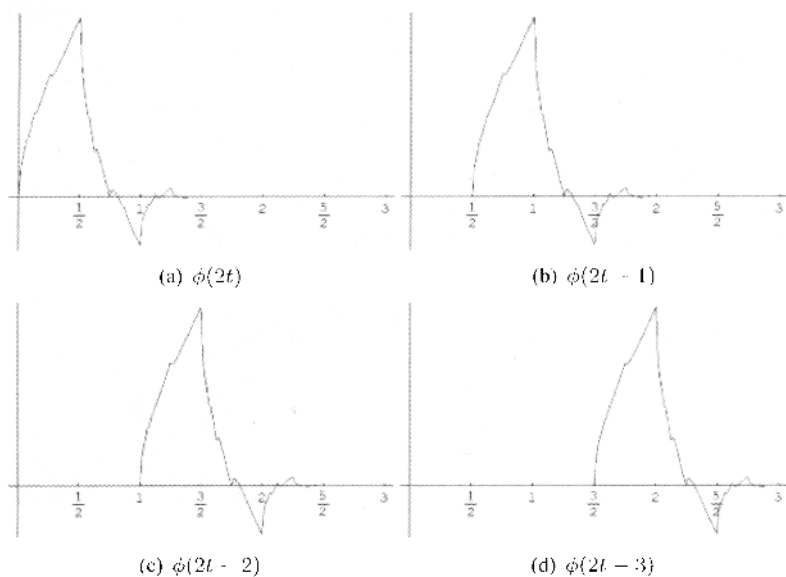


Figure 1.12 The functions that comprise the right-hand side of (1.2).

It turns out that the four numbers h_0, \dots, h_3 we need to use in combination with the functions in Figure 1.12 to build $\phi(t)$ are

$$h_0 = \frac{1 + \sqrt{3}}{4\sqrt{2}} \approx 0.482963 \quad h_1 = \frac{3 + \sqrt{3}}{4\sqrt{2}} \approx 0.836516$$

$$h_2 = \frac{3 - \sqrt{3}}{4\sqrt{2}} \approx 0.224144 \quad h_3 = \frac{1 - \sqrt{3}}{4\sqrt{2}} \approx -0.129410$$

We derive these numbers in Section 7.1. Each of the properties satisfied by $\phi(t)$ affect the discrete wavelet transform. The integral condition (1.1) ensures that the discrete wavelet transform matrix W is not only invertible, but *orthogonal*. As we

will see in Section 2.2, orthogonal matrices satisfy the simple inverse formula $W^{-1} = W^T$. Insisting that $\phi(t)$ is suitably smooth allows us to form accurate approximations of smooth data. Requiring $\phi(t)$ to be zero outside a finite interval results in a finite list of numbers h_0, \dots, h_N that are used in the discrete wavelet transform. Finally, the dilation equation allows us to build a transform that can “zoom” in on complicated segments of a signal or image. This zoom-in property is not entirely obvious — we talk more about it in Chapters 6 and 7.

The Approach in This Book

Using the classical approach of Daubechies [23] to derive the numbers h_0, h_1, h_2 , and h_3 for the function in Figure 1.10(a) is a journey through some beautiful mathematics, but it is difficult and requires more mathematical background than most undergraduate students possess.¹⁰

On the other hand, the matrix used to produce the output in Figure 1.6 is quite simple in structure. It is 100 rows by 100 columns and looks as follows:

$$W = \begin{bmatrix} \frac{3}{4} & \frac{1}{2} & -\frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ -\frac{1}{8} & \frac{1}{4} & \frac{3}{4} & \frac{1}{4} & -\frac{1}{8} & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & -\frac{1}{8} & \frac{1}{4} & \frac{3}{4} & \frac{1}{4} & -\frac{1}{8} & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & -\frac{1}{8} & \frac{1}{4} & \frac{3}{4} & \frac{1}{4} & -\frac{1}{8} & 0 & \cdots & 0 \\ \vdots & & & & & & & & & & & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & -\frac{1}{8} & \frac{1}{4} & \frac{3}{4} & \frac{1}{4} & -\frac{1}{8} & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & -\frac{1}{8} & \frac{1}{4} & \frac{3}{8} & \frac{1}{4} \\ \hline -\frac{1}{2} & 1 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \frac{1}{2} & 1 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & -\frac{1}{2} & 1 & -\frac{1}{2} & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{1}{2} & 1 & -\frac{1}{2} & 0 & \cdots & 0 & 0 \\ \vdots & & & & & & & & & & & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & -\frac{1}{2} & 1 & -\frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & -1 & 1 \end{bmatrix} \quad (1.3)$$

Note that we have placed a line between the first 50 rows and the last 50 rows. There are a couple of reasons for partitioning the matrix in this way. First, you might notice that rows 2 through 49 are built using the same sequence of five nonzero values

¹⁰For those students interested in a more classical approach to wavelet theory, I strongly recommend the books by David Walnut [83], Albert Boggess and Francis Narcowich [7], or Michael Frazier [35].

$(-\frac{1}{8}, \frac{1}{4}, \frac{3}{4}, \frac{1}{4}, -\frac{1}{8})$, and rows 51 through 99 are built using the same sequence of three nonzero values $(-\frac{1}{2}, 1, -\frac{3}{2})$.¹¹ Another reason for the separator is the fact that the product of W with an input signal consisting of 100 elements has to produce an approximation to the input (recall Figure 1.6) that consists of 50 elements. So the first 50 rows of W perform that task. In a similar manner, the last 50 rows of W produce the differences we need to combine with the approximation to recover the original signal.

There is additional structure evident in W . If we consider row 2 as a list of 100 numbers, then row 3 is obtained by cyclically shifting the elements in row 2 to the right 2 units. Row 4 is obtained from row 3 in a similar manner, and the process is continued to row 49. Row 52 can be constructed from row 51 by cyclically shifting the list $(-\frac{1}{2}, 1, -\frac{1}{2}, 0, 0, \dots, 0, 0)$ two unit right. It is easy to see how to continue this process to construct rows 53 through 99. Rows 1, 50, and 100 are different from the other rows and you will have to wait until Section 11.3 to find out why!

Let's look closer at the list $(-\frac{1}{8}, \frac{1}{4}, \frac{3}{4}, \frac{1}{4}, -\frac{1}{8})$. When we compute the dot product of row 2 say with an input signal (x_1, \dots, x_{100}) , we obtain

$$-\frac{1}{8}x_1 - \frac{1}{4}x_2 + \frac{3}{4}x_3 + \frac{1}{4}x_4 - \frac{1}{8}x_5$$

If we add the elements of our list, we obtain

$$\frac{1}{8} + \frac{1}{4} - \frac{3}{4} + \frac{1}{4} - \frac{1}{8} = 1$$

and see that the list creates a weighted average of the values (x_1, \dots, x_5) . The idea here is no different from what your instructor uses to compute your final course grade. For example, if three hourly exams are worth 100 points each and the final exam is worth 200 points, then your grade is computed as

$$\text{grade} = \frac{1}{5} \cdot \text{exam 1} + \frac{1}{5} \cdot \text{exam 2} + \frac{1}{5} \cdot \text{exam 3} + \frac{2}{5} \cdot \text{final}$$

Of course, most instructors do not use negative weights like we did in W ! The point here is that by computing weighted averages, we are creating an approximation of the original input signal.

Taking the dot product of row 51 say with the input signal produces

$$-\frac{1}{2}x_1 + 1 \cdot x_2 - \frac{1}{2}x_3$$

This value will be near zero if x_1, x_2 , and x_3 are close in size. If there is a large change between x_1 and x_2 or x_2 and x_3 then the output will be a relatively large

¹¹We mentioned earlier in this chapter that we seek orthogonal matrices for use as wavelet transforms. The reader with a background in linear algebra will undoubtedly realize that W in (1.3) is not orthogonal. As we will see late in the book, it is desirable to relinquish orthogonality so that our wavelet matrices can be built with lists of numbers that are symmetric.

(absolute) value. For applications, we may want to convert small changes to zero (compression) or identify the large values (edge detection).

The analysis of W and its entries leads to several questions: How do we determine the structure of W ? How do we build the lists of numbers that populate it? How are the rows at the top and bottom of the different portions of W formed? How do we design W for use in different applications? How do we make decisions about the output of the transformation? The matrix W consists of a large number of zeros; can we write a program that exploits this fact and computes the product of W and the input signal faster than by using conventional matrix and vector multiplication?

We answer all of these questions and many more in the chapters that follow. But we will consider the problem in the discrete setting. We will see that our lists of numbers are called *filters* by electrical engineers and other scientists. We first learn about the structure of W by understanding how signals are processed by filters. We then learn how to create filters that perform tasks such as approximating signals or creating the differences portion of the transform. These are the ideas that are covered in Chapters 4 and 5.

We further refine W in Chapter 6 to build the *discrete Haar wavelet transformation* and then use this transform to detect edges in images and to perform naive data compression. More sophisticated transforms are constructed in Chapter 7. Unfortunately, as we ask more of the transformations, the method we use to construct them increases in complexity. But at this point, you should have a good understanding of the discrete wavelet transform, how to use it in applications, how to write software for its implementation, and what limitations we still face. With this understanding comes an appreciation of the need to step back and model our construct in a more general mathematical setting. In the second half of the book we take a more theoretical tack toward transform design. The payoff is the ability to design wavelet transforms like that given in (1.3) and use them in current applications such as JPEG2000 or signal and image denoising.

Thus, in the pages that follow, you will discover the answers to the questions asked at the beginning of the chapter. It is my sincere hope after working through the material, problem sets, software, and computer labs that your response to these questions is: *Why wavelets, indeed!*

