

Plotting and Scheming with Wavelets

COLM MULCAHY

Dept. of Mathematics

Spelman College

Atlanta, Georgia 30314, USA

email: colm@spelman.edu

<http://www.spelman.edu/~colm>

1. INTRODUCTION

Wavelets are acquiring a visibility and popularity that may soon be on the scale first enjoyed by fractals a few years back. Like fractals, wavelets have attractive and novel features, both as mathematical entities and in numerous applications. They are often touted as worthwhile alternatives to classical Fourier analysis, which works best when applied to periodic data: wavelet methods make no such assumptions. However, the mathematics of wavelets can seem intractable to the novice. Indeed, most introductions to wavelets assume that the reader is already well versed in Fourier techniques.

Our main goal is simple: to convince the reader that at their most basic level, wavelets are fun, easy, and ideal for livening up dull conversations. We demonstrate how elementary linear algebra makes accessible this exciting and relatively new area at the border of pure and applied mathematics.

In **Plotting**, we explore several ways of visually representing data, with the help of *Matlab* software. In **Scheming**, we discuss a simple wavelet-based compression technique, whose generalizations are being used today in signal and image processing, as well as in computer graphics and animation. The basic technique uses only addition, subtraction, and division by two! Only later, in **Wavelets**, do we come clean and reveal what wavelets are, while unveiling the *multiresolution* setting implicit in any such analysis.

In **Averaging and Differencing with Matrices**, which may be read independently of **Wavelets**, we provide a matrix formulation of the compression scheme. In **Wavelets on the World Wide Web** we mention a natural form of progressive image transmission that lends itself to use by the emerging generation of web browsers (such wavelet-enhanced software is already on the market).

In **Wavelet Details**, we attempt to put everything in context, while hinting at the more sophisticated mathematics that must be mastered if one wishes to delve deeper into the subject. Finally, in **Closing Remarks**, we mention some other common applications of wavelets.

Along the way we find ourselves trying out an adaptive plotting technique for ordinary functions of one variable that differs from those currently employed by many of today's popular computer algebra packages. While this technique, as described here, is limited in its usefulness, it can be modified to produce acceptable results.

We were much inspired by Stollnitz, DeRose, and Salesin's fine wavelets primers ([15, 16]), which, along with [17], [18], [19], we recommend heartily to beginners who

desire more details. More general surveys can be found in [7] and [11]. Although the wavelets we discuss here had their origins in work of Haar early in this century, the subject proper really gathered momentum only in the last decade. The historical development of wavelets is quite complex, as the main concepts arose independently in several different fields. We do not cite the numerous groundbreaking papers in these fields, leaving that to the books and surveys listed in the bibliography. It's a fascinating story, combining ideas first studied by electrical engineers, physicists, and seismologists, as well as pure mathematicians. For an especially readable account of how it all happened, we recommend Barbara Burke Hubbard's *The World According to Wavelets* [10], a remarkable book which also goes into greater detail about wavelet applications than we do. A more mathematically concise version of this story can be found in Jawerth and Sweldens' survey paper [11].

2. PLOTTING

We begin by reviewing standard ways of plotting discrete data sets, in particular, sampled functions of the form $y = f(x)$, and two-dimensional digital images. The limitations inherent in attempts to plot functions by uniform sampling will lead us, in the next section, to suggest a wavelet-based scheme to work around this difficulty. The need for adaptive plotting techniques will become obvious. The real purpose of this section is to drum up support for some sort of data compression.

Suppose we have a finite set of planar data points (x, y) , which might be samples of a function $y = f(x)$. A common method of displaying these data is to plot the individual points and then join adjacent points with line segments; this is precisely what happens when many computer algebra packages graph functions. Graphing with *Matlab's* `plot` command, for instance, requires us to pick the x -values to be used. Plotting $y = \sin(15x)$ and $y = \sin(90x)$ this way, on the interval $[0, 1]$, using 32 equally spaced x -values, yields the pictures in Figure 1. The true nature of $y = \sin(15x)$ can be safely inferred from the first plot, as increasing the number of points sampled verifies. The second plot is another story, however.

Figure 1(b) suggests a function whose oscillations exhibit a pulsing pattern, although, symbolically, we expect a horizontally telescoped version of the preceding graph. The apparent pulsing behavior is an artifact of sampling uniformly at 32 points: $\sin(90x)$ has frequency $\frac{90}{2\pi} \approx 14.3239$, which is just under half the sampling frequency.

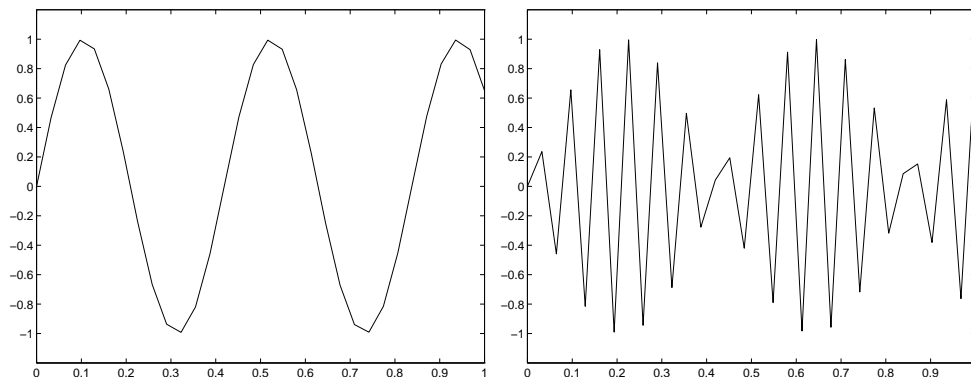


FIGURE 1

Plots of $y = \sin(15x)$ and $y = \sin(90x)$, using 32 uniformly sampled points

While we have just enough information to determine or reconstruct the function [8, p. 340], the graphical anomaly (known as *aliasing*) is not too surprising. Using a few more points yields the anticipated 6-fold repetition of the pattern seen in Figure 1(a).

Figure 1 makes one thing abundantly clear: using uniformly-spaced sample points isn't the smartest approach, unless we are prepared to use a lot of points. Much better pictures are obtained from *Matlab's* `fp1ot` command, and from the corresponding *Maple* or *Mathematica* commands. These commands produce adaptive plots, with points clustered where the function seems to exhibit great variation. These adaptive plotting routines examine angles between connecting line segments in provisional internally-generated plots based on uniform sampling. Having identified regions of great variation, they subdivide certain intervals further before producing a visible plot. (For details, see [12, p. 216], [9, p. 303-304], [22, p. 579-584].) In the next section we will illustrate how wavelets give rise to an adaptive plotting scheme that does not require us, or the computer, to consider angles first while peeking at default plots.

For the sorts of (differentiable) functions considered so far, intuition correctly suggests that, on the one hand, if we continually replot a function, sampling more and more frequently (uniformly or otherwise), we get a sequence of pictures that converges to the true graph. On the other hand, no matter what scale we view (or print) at, there comes a stage past which it is impossible to detect the use of additional sampled points. Just how many points need we plot to give the illusion of a correct graph? The answer depends very much on the amount of variation the function possesses over the interval in question, as well as on the size of the picture we are going to look at, as the next examples make clear.

Some functions are beyond redemption from the point of view of plotting and displaying at any reasonable scale. A function like $y = \sin(\frac{1}{x})$, which has infinitely many extrema on $(0, 1]$, is going to give this or any other plotting routine a run for its money. The (algebraically) innocent-looking function $y = \sin(e^{2x+9})$ achieves so many extrema (a staggering $\lfloor \frac{e^{11}}{\pi} - \frac{1}{2} \rfloor - \lfloor \frac{e^9}{\pi} - \frac{1}{2} \rfloor + 1 = 19058 - 2579 + 1 = 16480$, to be precise) in the interval $[0, 1]$, that the plot in Figure 2(a), which is based on uniform sampling at 32 points, is totally misleading. Worse still, the more points we plot (even if we plot adaptively) the denser the pictures appear, on account of the nonzero thickness of depicted line segments. Figure 2(b) shows what we get if we sample uniformly at 256 points; increase this number further, and the plots start to fill up with connecting line segments. Sadly, given the natural physical limitations of plotters and

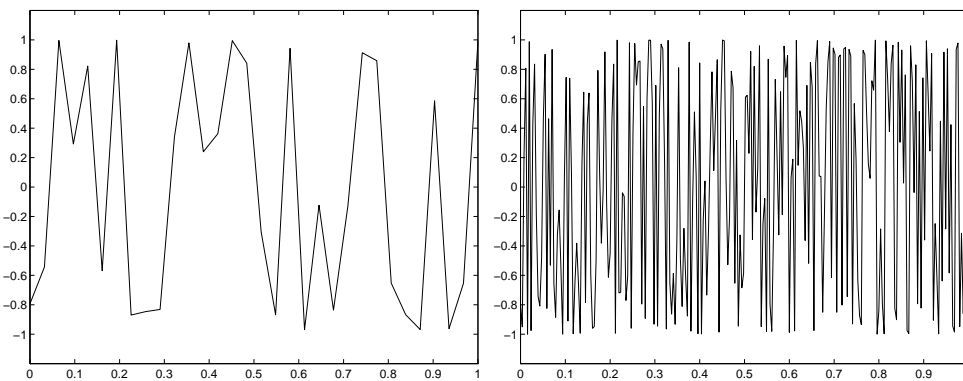


FIGURE 2

Plots of $y = \sin(e^{2x+9})$ using 32 and 256 points respectively

printers, there is little hope in this life of getting an accurate graph of $y = \sin(e^{2x+9})$ on $[0, 1]$, and we hereby admit defeat.

Of course, linear interpolation of sampled points is just one way of plotting: instead of joining the points with line segments, we could use the y -values as step levels for a staircase effect. Figures 3(a) and 3(b) illustrate step function alternatives to Figure 1(a) and Figure 2(a) respectively, namely $y = \sin(15x)$ and $y = \sin(e^{2x+9})$ using the same uniformly sampled points.

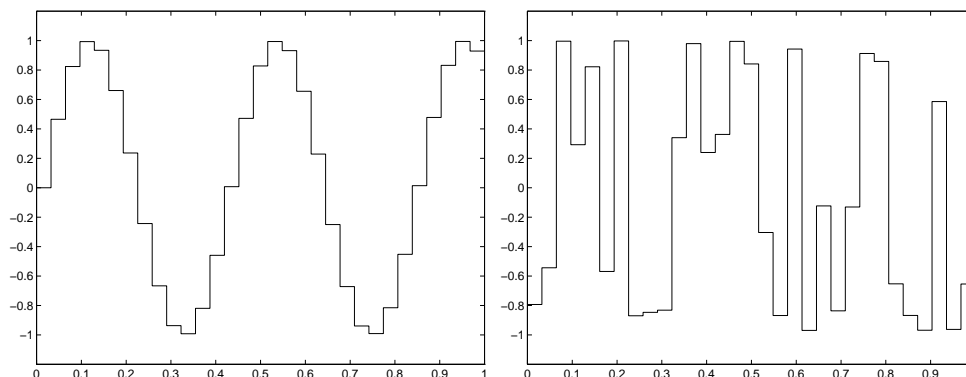


FIGURE 3

Stairs plots of $y = \sin(15x)$ and $y = \sin(e^{2x+9})$ using 32 uniformly sampled points.

Purists may balk at the vertical lines connecting the steps in these pictures, which were generated using *Matlab*'s `stairs` command, but for our purposes these uninvited guests are quite harmless. While these plots leave a lot to be desired, just as the linearly interpolated plots earlier did, the staircase method will lead to better and better approximations of the true graph when more points are used, although a lot more are needed to get away from the jaggies and obtain a continuous effect. (That continuous functions can be approximated on $[0, 1]$ to arbitrary precision by piecewise linear functions, or by step functions, is a simple consequence of their being uniformly continuous on compact intervals [2, 24.4, 24.5].)

There are also questions of data storage and transmission. These become particularly crucial when we explore higher-dimensional analogues of data points in the plane, such as digital images.

The images of Emmy Noether in Figure 4 are derived from two-dimensional arrays of *pixels*—numbers that represent gray levels ranging from black (minimum number) to white (maximum number). These can be thought of as data points (x, y, z) , where z measures the gray level at position (x, y) : we draw a two-dimensional array of small squares, each shaded a constant gray level z according to its position (x, y) in the array. What we really have here are two-dimensional step functions—viewed from above—where the steps are shaded according to their height. (Color images can be dealt with by decomposing into red, green and blue components, and treating each of these like grayscale.)

Figure 4(a) is composed of 256×256 pixels; so it is derived from a matrix of $256^2 = 65536$ pieces of data, each representing a gray level. To produce Figure 4(b) we extracted a 64×64 submatrix from the original 256×256 matrix; the submatrix shows the region around the eyes. This second image requires $64^2 = 4096$ pieces of data to store. Due to the lower resolution it is noticeably more “blocky;” we can



FIGURE 4
Emmy Noether—in person and up close

explicitly see the steps which make it up. Both images use $256 = 2^8$ levels of gray, and as such are called *8-bit images*.

Clearly, it requires a lot of data to represent an image in this way, and that leads to practical problems. For one thing, a standard 1.44MB high-density floppy disc can only accommodate a handful of large, high-quality, color images. Furthermore, image files are time-consuming to transmit, as anybody who has viewed pictures on the World Wide Web can attest. In the images of Emmy Noether, there are regions of little or no variation. Our goal is to take advantage of these somehow, and come up with a more economical way to store the matrices that represent the images.

3. SCHEMING

Here we get down to business and describe an elementary wavelet scheme for transforming, and ultimately compressing, digital data. Whether these data represent samples of a function, a matrix of gray levels, or something else entirely, has no bearing on the scheme itself. While wavelets are behind the ideas presented, we defer any further mention of the “W” word until the next section. Readers who wish to duplicate the results and pictures found here can proceed directly to **Averaging and Differencing with Matrices** upon reading this section.

After we describe the basics of the scheme, and look at some examples, we explain what we really mean by compression. A key ingredient is the standard technique for storing large sparse matrices in terms of their nonzero entries—values and locations only—rather than in matrix form.

As motivation, we first consider the images in Figure 5, which use only two shades of gray. How much information is required to store the first one? If we assume that we have black unless specified otherwise, we need only say where the white is, so it seems reasonable to claim that two pieces of information suffice. If the image is a $4 = 2^2$ pixel image, we could store the facts that pixels $(1, 2)$ and $(2, 1)$ are white. But what if the image is, say, a $65536 = 256^2$ pixel image, which just happens to be composed of large black and white blocks? We will show how to use two pieces of

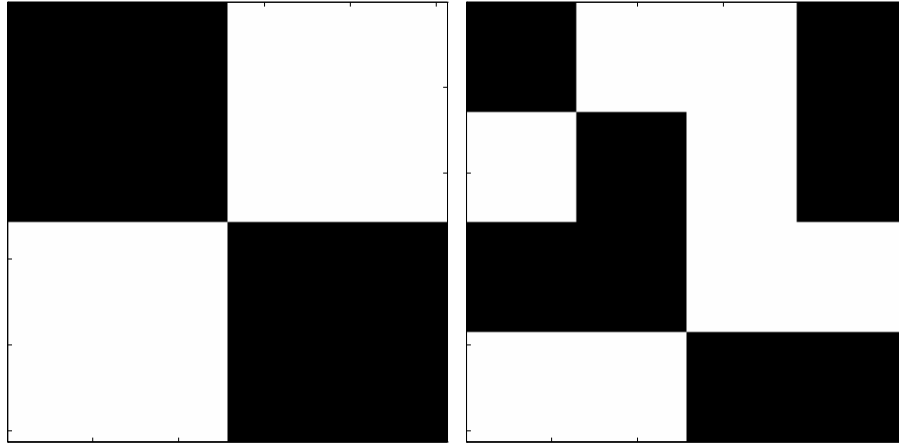


FIGURE 5

How many pieces of information are needed to store these simple images?

information to store the image in this case too; the principle works regardless of the actual resolution.

Next consider the more complex image in Figure 5(b). If we treat this as a $16 = 4^2$ pixel image, we see eight white and eight black blocks, so we could argue that eight pieces of information suffice to specify this arrangement. But we can do better if we also use the fact that several of the white blocks are adjacent to each other. We will see presently that only five pieces of information are needed, even if the image is at a greater resolution than is apparent. (For a hint as to why five might be enough, consider the top left quarter of this array as a copy of the first arrangement.)

We now move on to our main goal: describing how to transform arrays of data to a form in which regions of “low activity” in the original become easy to locate in the transformed version. Since matrices consist of neatly stacked rows of numbers, we begin with strings of data. Our method will have immediate application to plotting $y = f(x)$ type functions, as we can identify uniformly sampled functions with data strings.

Consider a string of eight pieces of data. This could, for instance, be uniform samples of a function, or a row of an 8×8 pixel image. In order to avoid fractions below, we use these specially cooked-up numbers:

64 48 16 32 56 56 48 24

We process these in several stages, in a manner commonly referred to as *averaging and differencing*, which we will explain in a moment. Successive rows of the table show the starting, intermediate, and final results.

64	48	16	32	56	56	48	24
56	24	56	36	8	-8	0	12
40	46	16	10	8	-8	0	12
43	-3	16	10	8	-8	0	12

The first row is our original data string, which we can think of as four pairs of numbers. The first four numbers in the second row are the averages of those pairs. Similarly, the first two numbers in the third row are the averages of those four averages, taken two at a time, and the first entry in the fourth and last row is the average of the preceding two computed averages.

The remaining numbers, shown in bold, measure deviations from the various averages. The first four bold entries, in the second half of the second row, are the result of subtracting the first four averages from the first elements of the pairs that gave rise to them: subtracting 56, 24, 56, 36 from 64, 16, 56, 48, element by element, yields **8, -8, 0, 12**. These are called *detail coefficients*; they are repeated in each subsequent row of the table. The third and fourth entries in the third row are obtained by subtracting the first and second entries in that row from the first elements of the pairs that start row two: subtracting 40, 46 from 56, 56, element by element, yields **16, 10**. These two new detail coefficients are also repeated in each subsequent row of the table. Finally, the second entry in the last row, **-3**, is the detail coefficient obtained by subtracting the overall average, 43, from the 40 that starts row three.

It is not hard to see that the last average computed is also the overall average of the original eight numbers. This has no effect on the shape of (any plot of) these data: it merely anchors the data vertically. The seven detail coefficients are what really determines the shape.

We have transformed our original string of eight numbers into a new string of eight numbers. The transformation process is, moreover, reversible: we can work back from any row in the table to the previous row—and hence to the first row—by means of appropriate additions and subtractions. In other words, we have lost nothing by transforming our string. *What have we gained? The opportunity to fiddle with the “mostly detail” version!* If we alter the transformed version, by taking advantage of regions of low activity, and use this doctored version to work back up the table, we obtain an approximation to the original data. If we are lucky, this approximation may be visually close to the original.

Our string has one detail coefficient of 0, due to the the adjacent 56’s in the original string; this is one region of low activity. The next smallest detail coefficient (in magnitude) is the **-3**. Let’s reset that to zero, putting 43, **0, 16, 10, 8, -8, 0, 12** in the last row of a blank table, and work our way back up by adding and subtracting as indicated above. The completed table looks like this:

67	51	19	35	53	53	45	21
59	27	53	33	8	-8	0	12
43	43	16	10	8	-8	0	12
43	0	16	10	8	-8	0	12

The first row in this table is our approximation to the original data. In Figure 6(a) we plot the original and the approximation, the latter using dashed lines; for reasons which will be clear later, we have plotted the string as y -values against eight equally spaced x -values in $[0, 1]$. While the differences are discernible, many observers would be hard-pressed to distinguish the plots if seen one at a time.

In Figure 6(b) we plot the original against the approximation (59 59 27 27 53 53 45 21), obtained by the above procedure after dropping two more detail coefficients, namely the **-8** and the **8**. Considering how few data (only five numbers) this approximation is based on, it’s surprisingly good.

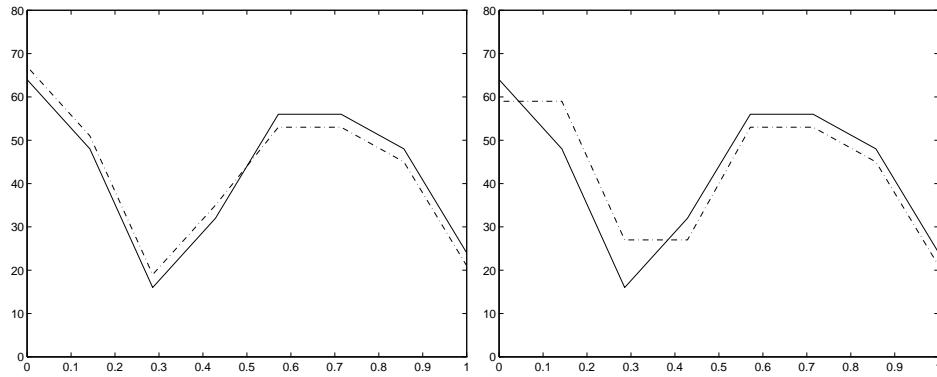


FIGURE 6

Eights pieces of data versus approximations based on six and four detail coefficients respectively

Before we go on, we note that the process can be generalized to strings of any length. We can always pad at the end, say with zeros, until they have length equal to a power of two.

To appreciate the full potential of this scheme, we must think big. Starting with a string of length $256 = 2^8$, eight applications of averaging and differencing yield a string with one overall average and 255 detail coefficients. We can then fiddle with this and work back to an approximation of the original.

In general the compression scheme works like this: Start with a data string, and a fixed nonnegative *threshold* value ϵ . Transform the string as above, and decree that any detail coefficient whose magnitude is less than or equal to ϵ will be reset to zero. Hopefully, this leads to a relatively sparse string (one with a high proportion of zeros), which is thus compressible when it comes to storage. This process is called *lossless compression* when no information is lost (e.g., if $\epsilon = 0$); otherwise it's referred to as *lossy compression* (in which case $\epsilon > 0$). In the former case we can get our original string back. In the latter we can build an approximation of it based on the altered version of the transformed string. The surprise is that we can throw out a sizable proportion of the detail coefficients, and still get decent results.

Let's try this for $y = e^{-10x} \sin(100x)$ on $[0, 1]$, which has a large region of relatively low activity. The plots in Figure 7 are based on 32 and 256 uniformly sampled points.

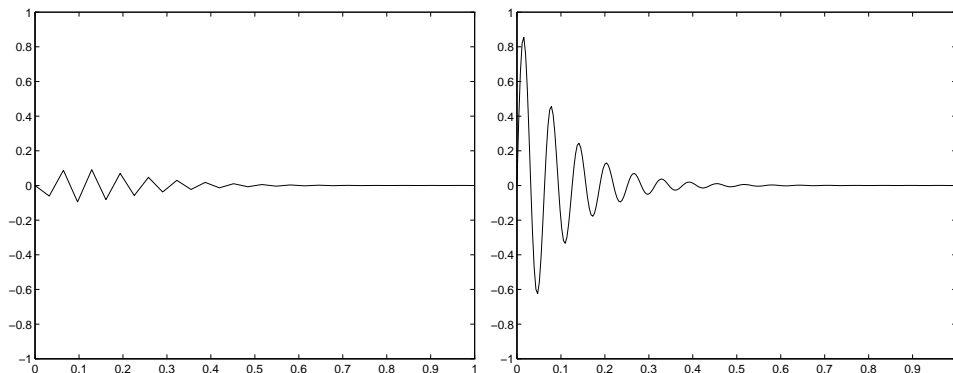


FIGURE 7

Plots of $y = e^{-10x} \sin(100x)$ using 32 and 256 uniformly sampled points, respectively

As Figure 7(b) illustrates, half of the points plotted are essentially wasted. Consider the string of 256 y -values used to derive this plot, which range from -0.6246 to 0.8548 . After eight rounds of averaging and differencing, we get a transformed string which ranges from -0.2766 to 0.4660 . Dumping all detail coefficients less than or equal to 0.04 in magnitude, we get an altered transformed string with 32 nonzero entries. From this sparse string we build an approximation of the original string, which is plotted in Figure 8(a). Despite its limitations, this does a better job than Figure 7(a) of conveying the flavor of the actual graph. Figure 8(b) shows the even better picture obtained when we reduce the cut-off threshold to 0.01 , in which case the altered transformed string has 70 nonzero values.

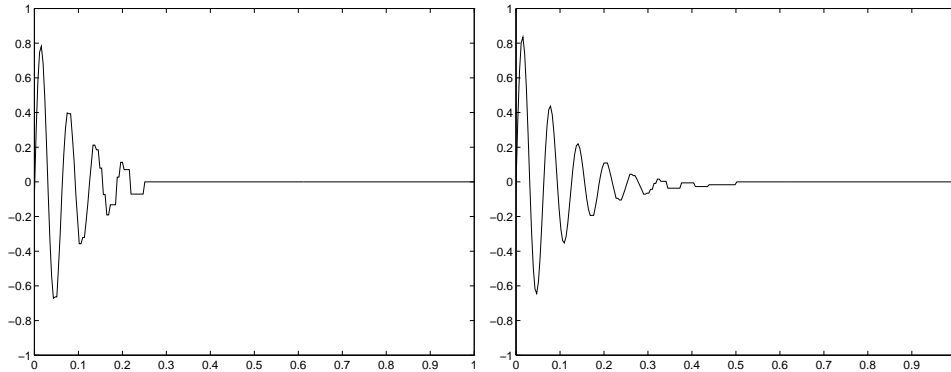


FIGURE 8

Approximations to $y = e^{-10x} \sin(100x)$ using 32 and 70 detail coefficients respectively

The reason why such low thresholds (relative to the range of values) give good results here, using few detail coefficients, is that this function's pulse is rather weak in half of the interval of interest. Using 70 detail coefficients out of 256 gives a "compression ratio" of around 3.5 : 1.

There is a subtlety here worth highlighting: the plots in Figure 8 were generated from just 32 and 70 nonzero numbers, respectively, in sparse strings of length 256 (the doctored transformed strings). However, the plots themselves used all 256 (mostly nonzero) numbers obtained from those strings by reversing our averaging and differencing process. The lossy compression comes into play once we note that it takes significantly less space to store sparse strings of length 256—with only 32 or 70 nonzero entries—than arbitrary strings of length 256.

The approximation technique just outlined has shortcomings as an adaptive plotting scheme—shortcomings that were apparent as early as our first efforts in Figure 6. Most obviously, modest-sized data sets such as those we have been considering lead to thresholded strings of data that produce unacceptably jagged plots. This is because thresholding often yields data strings with constant stretches (horizontal steps) followed by dramatic leaps or drops (steep segments). Perhaps surprisingly, regions of lower activity produce the worst "jaggies." A less obvious problem, which Figure 6(a) illustrates, is that the range of y -values in the approximation may exceed the range of the original y -values. In **Wavelet Details** we will mention smoother schemes that largely avoid these problems.

We explored the above transformation technique in some detail because we can repeat it for image data sets with almost no extra work. What's more, we get better results, since realistic images consist of much larger data sets, in which steps have to

be quite extreme to produce visible blockiness (the higher-dimensional analogue of jaggedness).

We could simply concatenate the rows to obtain one long string, but then we wouldn't be able to exploit natural correlations between adjacent rows of real-world image matrices. Instead, we treat each row as a string and process as above, obtaining an intermediate matrix, and then apply exactly the same transformations to the columns of this matrix to obtain a final row and column transformed matrix.

Specifically, to apply the scheme to a 256×256 matrix, we simply do the averaging and differencing eight times on each row separately, and then eight times on the columns of the resulting matrix. Averaging and differencing columns can also be achieved by transposing the row-transformed matrix, doing row transformations to the result of that transposition, and transposing back. The final result is a new 256×256 matrix, with one overall average pixel value in the top left hand corner, and an awful lot of detail elements. Regions of little variation in the original image manifest themselves as numerous small or zero elements in the transformed matrix, and the thresholding principle described earlier above can be used to effect lossy image compression.

First, let's go back to the simple images in Figure 5. Suppose both are 256×256 pixel images, composed of 128×128 and 64×64 monochromatic sub-blocks respectively. If black pixels match up with matrix entries of 0, and white ones with 1, then performing eight row and then eight column transformations on the matrices corresponding to the images, we obtain matrices that are extremely sparse. The only nonzero entries are bunched up in these 4×4 submatrices in their respective upper left-hand corners:

$$\begin{pmatrix} \frac{1}{2} & 0 & 0 & 0 \\ 0 & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad \begin{pmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{4} \\ 0 & 0 & 0 & \frac{1}{4} \\ 0 & 0 & -\frac{1}{2} & 0 \\ 0 & -\frac{1}{2} & 0 & 0 \end{pmatrix}.$$

Thus the first transformed matrix has only two nonzero entries—whereas the second has five. Storing these matrices efficiently leads to a form of *lossless compression*. The original images can be reconstructed exactly from these smaller data sets.

Now we move on to *lossy compression*. Applying our thresholding scheme to images with only a few gray levels, such as those in Figure 5, is guaranteed to produce poor results, because the averaging process introduces numbers which, if altered and transformed back to image form, correspond to gray levels that were not originally present.

Consider the 8-bit image Noetherian image in Figure 4(a), which contains a great deal of black; in fact, black accounts for 20% of the pixels. When we apply eight row and eight column transformations, we obtain a matrix 30% of whose entries are zero; an increase that can be attributed to the other areas of little variation in the original. For appropriate choices of ϵ —depending on the range of numbers in the matrix used to represent the gray levels of the original image—we get the compressed images in Figure 9. Note the concentration of small blocks near the hairline and collar line, and in the facial features, illustrating the adaptiveness of this scheme. The extreme blockiness of these images is due to the nature of averaging and differencing, which is equivalent to working with certain step functions, as we will see in the next section.

The first image uses 6558 out of $256^2 = 65536$ (actually 65535) coefficients, and the second only 1320. In a sense, we could claim compression ratios of 10:1 and 50:1,

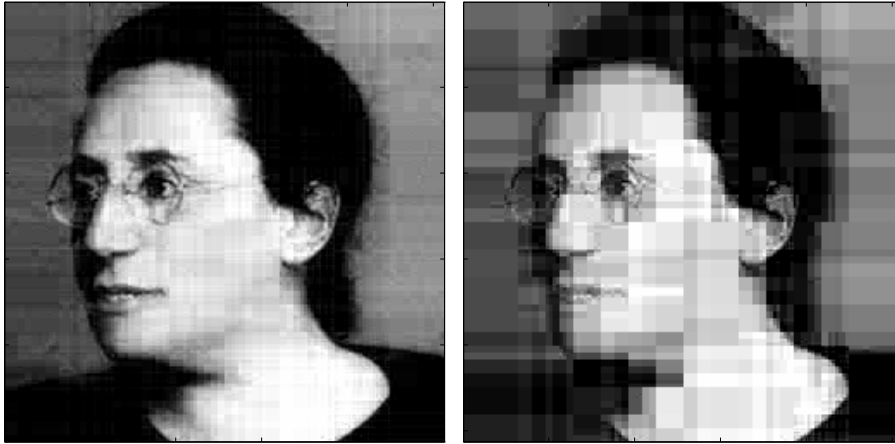


FIGURE 9

Noetherian compression—using 10% and 2% of the detail coefficients respectively

respectively, but in view of the fact that the original matrix has only 45870 nonzero elements, a more realistic claim might be ratios of 7:1 and 35:1, respectively. Indeed, ratios very close to these turn up when we check how many bytes *Matlab* needs to store the sparse forms of these matrices, whether within a *Matlab* session workspace or in external data files. However we compute compression ratios, it's impressive that the images are recognizable at all, considering how little information was used to generate them.

A modification of the above approach, known as *normalization*, that will likely seem unmotivated for now, yields significantly better results: In the “averaging and differencing” process, divide by $\sqrt{2}$ instead of 2 (so that a pair a and b is processed to yield $(a + b)/\sqrt{2}$ and $(a - b)/\sqrt{2}$). Perhaps unexpectedly, this leads to compressed images that are more acceptable to the human eye than those above. Figure 10(a) shows



FIGURE 10

Normalized compression—using 2% and 1% of the detail coefficients respectively

the normalized compressed version of Figure 9(b); both images use 2% of the coefficients. Figure 10(b) shows a normalized compressed image that is visually comparable to—if not better than—Figure 9(b), but uses only 1% of the coefficients.

We freely admit that compression ratio computation is a rather delicate matter, but the compression schemes we have outlined are surely worthwhile, no matter how one computes these ratios.

We remark that for random matrices, whose entries are interpreted as representing gray levels, there is no hope of compression at all. The transformed versions tend to have no nonzero entries to speak of, and thresholding leads to approximations which look unacceptably non-random.

We summarize the central idea of the compression scheme: *Data that exhibit some sort of structure can be efficiently stored in equivalent form as sparse matrices; specifically, in “transformed and sparse” form for lossless compression, and in “transformed, thresholded and sparse” form for lossy compression.* To view the data, or an approximation of it, one simply “expands” to non-sparse form and applies the inverse transformation.

(At this point, readers may skip to **Averaging and Differencing with Matrices** if they wish. There we describe one matrix multiplication implementation of the compression scheme just discussed.)

4. WAVELETS

Our principal aim here is to put our earlier discussions on a firmer mathematical foundation, and to acquaint the reader with some of the standard concepts and notations used in the general study of wavelets. What *are* wavelets, anyway? Before we try to answer this question, we present an alternative vector space description of our discrete, 8-member data sets.

First we identify data strings with a certain class of step functions. A string of length k is identified with the step function on $[0, 1]$ which (potentially) changes at $k-1$ equally spaced x -values and uses the string entries as its step heights. For instance, the string of y -values arising from uniformly sampling $\sin(15x)$ 32 times in $[0, 1]$ is identified with the step function plotted in Figure 3(a). These step functions can in turn can be thought of as linear combinations of dyadically dilated and translated unit step functions on $[0, 1)$. We now explain this in some detail.

Consider the *Haar scaling function*:

$$\phi(x) := \begin{cases} 1 & \text{on } [0, 1) \\ 0 & \text{elsewhere.} \end{cases}$$

Note that ϕ satisfies a *scaling equation* of the form $\phi(x) = \sum_{i \in \mathbb{Z}} c_i \phi(2x - i)$, where in our case the only nonzero c_i 's are $c_0 = c_1 = 1$, i.e., $\phi(x) = \phi(2x) + \phi(2x - 1)$.

For each $0 \leq i \leq 2^3 - 1$, we get an induced (dyadically) dilated and translated scaling function

$$\phi_i^3(x) = \phi(2^3 x - i).$$

These eight functions form a basis for the vector space \mathcal{V}^3 of piecewise constant functions on $[0, 1)$ with possible breaks at $\frac{1}{8}, \frac{2}{8}, \frac{3}{8}, \dots, \frac{7}{8}$. Note that ϕ_0^3 is 1 on $[0, \frac{1}{8})$ only, ϕ_1^3 is 1 on $[\frac{1}{8}, \frac{2}{8})$ only, ϕ_2^3 is 1 on $[\frac{2}{8}, \frac{3}{8})$ only, and so on. Figure 11 shows three of these basis functions together with a typical element of \mathcal{V}^3 . Actually, the last plot in Figure 11 shows the rather special element

$$64\phi_0^3 + 48\phi_1^3 + 16\phi_2^3 + 32\phi_3^3 + 56\phi_4^3 + 56\phi_5^3 + 48\phi_6^3 + 24\phi_7^3 \in \mathcal{V}^3,$$

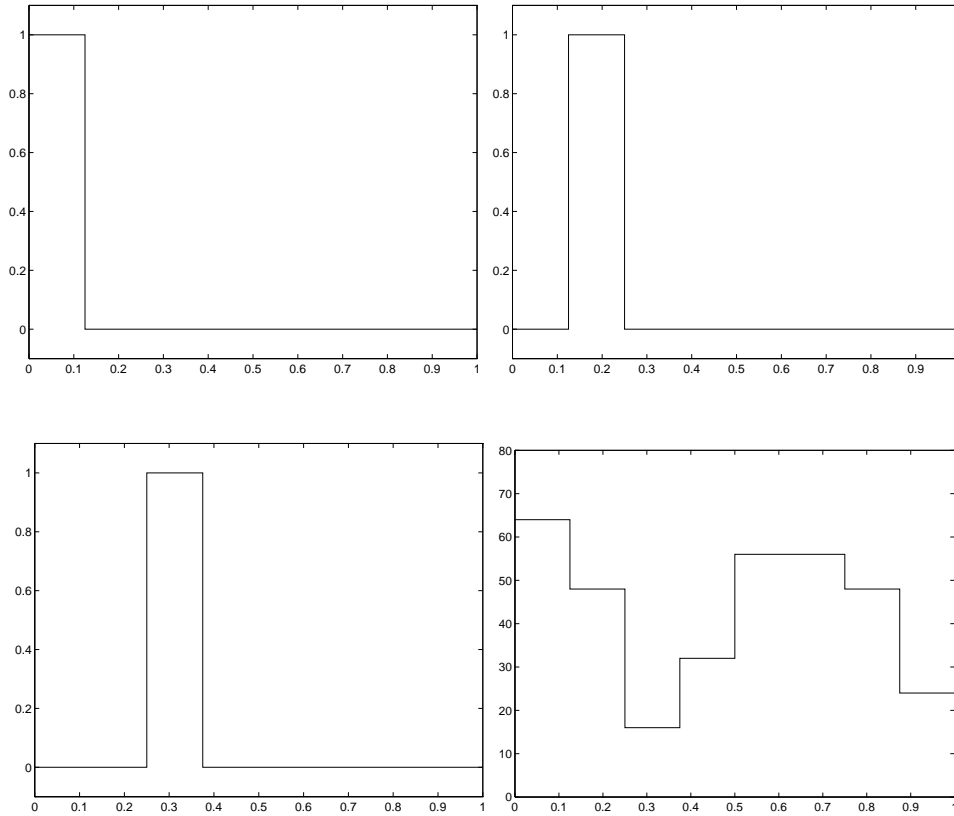


FIGURE 12

The first three of the eight basis functions ϕ_i^3 ($0 \leq i \leq 7$) and an element of \mathcal{V}^3

which is just another way of thinking of our earlier data string

$$64 \ 48 \ 16 \ 32 \ 56 \ 56 \ 48 \ 24.$$

In contrast to the piecewise linear plot in Figure 6, we now have a step function representation of our data string. Similarly, any string of length eight can be identified with an element of \mathcal{V}^3 . We can describe the averaging and differencing scheme from the last section in terms of this version of data strings, but first we need some more vector spaces. As above, the four functions ϕ_i^2 defined by

$$\phi_i^2(x) := \phi(2^2x - i),$$

for $0 \leq i \leq 2^2 - 1$, form a basis for the vector space \mathcal{V}^2 of piecewise constant functions on $[0, 1)$ with possible breaks at $\frac{1}{4}, \frac{2}{4}, \frac{3}{4}$; the two functions ϕ_i^1 defined by

$$\phi_i^1(x) := \phi(2^1x - i),$$

for $0 \leq i \leq 2^1 - 1$, form a basis for the vector space \mathcal{V}^1 of piecewise constant functions on $[0, 1)$ with a possible break at $\frac{1}{2}$; and $\phi_0^0 := \phi$ itself is a basis for the vector space \mathcal{V}^0 of constant functions on $[0, 1)$. Note that $\mathcal{V}^0 \subset \mathcal{V}^1 \subset \mathcal{V}^2 \subset \mathcal{V}^3$.

We can identify the various averages derived in **Scheming** with elements of these new vector spaces, by treating these averages as lower-resolution versions of the original string. Specifically, we match up 56, 24, 56, 36 with $56\phi_0^2 + 24\phi_1^2 + 56\phi_2^2 + 36\phi_3^2$, then 40, 46 with $40\phi_0^1 + 46\phi_1^1$, and finally 43 with $43\phi_0^0 = 43\phi$.

It only remains to find a new interpretation for the detail coefficients. This is where the wavelets finally enter the picture—fasten your seatbelts! Consider the inner product

$$\langle f, g \rangle := \int_0^1 f(t)g(t)dt$$

defined on \mathcal{V}^3 ; two functions are orthogonal if and only if their product on $[0, 1]$ encloses equal areas on each side of the horizontal axis. For each $j = 0, 1, 2$, we define the *wavelet space* \mathcal{W}^j to be the orthogonal complement of \mathcal{V}^j in \mathcal{V}^{j+1} , so that we get the (orthogonal) direct sum decomposition:

$$\mathcal{V}^{j+1} = \mathcal{V}^j \oplus \mathcal{W}^j.$$

We have

$$\mathcal{V}^3 = \mathcal{V}^2 \oplus \mathcal{W}^2 = \mathcal{V}^1 \oplus \mathcal{W}^1 \oplus \mathcal{W}^2 = \mathcal{V}^0 \oplus \mathcal{W}^0 \oplus \mathcal{W}^1 \oplus \mathcal{W}^2.$$

Each \mathcal{W}^j has a natural basis $\{\chi_i^j : 0 \leq i \leq 2^j - 1\}$ which we will describe in a moment, and expressing step functions in \mathcal{V}^3 in terms of these new bases brings us to the various detail coefficients we encountered before, which will henceforth be known as *wavelet coefficients*.

The *mother Haar wavelet* is defined by

$$\chi(x) := \begin{cases} 1 & \text{on } [0, \frac{1}{2}) \\ -1 & \text{on } [\frac{1}{2}, 1) \\ 0 & \text{elsewhere.} \end{cases}$$

(Equivalently, we could have defined $\chi(x) := \phi(2x) - \phi(2x - 1)$.) Notice that $\{\chi\}$ is a basis for \mathcal{W}^0 since χ is clearly orthogonal to ϕ . The four functions

$$\chi_i^2(x) := \chi(2^2x - i),$$

for $0 \leq i \leq 2^2 - 1$, form a basis for \mathcal{W}^2 , because, on the one hand, they are orthogonal to the corresponding functions ϕ_i^2 ($0 \leq i \leq 3$) which form a basis for the subspace \mathcal{V}^2 of \mathcal{V}^3 , and, on the other hand, they are visibly orthogonal to each other (see Figure 12).

Similarly, the two functions χ_i^1 defined by

$$\chi_i^1(x) := \chi(2^1x - i),$$

for $0 \leq i \leq 2^1 - 1$, form a basis for \mathcal{W}^1 .

In present notation, the three steps in the averaging and differencing transformation in the preceding section correspond to the following chain of identities:

$$\begin{aligned} & 64\phi_0^3 + 48\phi_1^3 + 16\phi_2^3 + 32\phi_3^3 + 56\phi_4^3 + 56\phi_5^3 + 48\phi_6^3 + 24\phi_7^3 \\ &= 56\phi_0^2 + 24\phi_1^2 + 56\phi_2^2 + 36\phi_3^2 + 8\chi_0^2 - 8\chi_1^2 + 0\chi_2^2 + 12\chi_3^2 \\ &= 40\phi_0^1 + 46\phi_1^1 + 16\chi_0^1 + 10\chi_1^1 + 8\chi_0^2 - 8\chi_1^2 + 0\chi_2^2 + 12\chi_3^2 \\ &= 43\phi_0^0 - 3\chi_0^0 + 16\chi_0^1 + 10\chi_1^1 + 8\chi_0^2 - 8\chi_1^2 + 0\chi_2^2 + 12\chi_3^2 \end{aligned}$$

The final, fully-transformed version, consists of one overall average and seven wavelet coefficients; this is simply a decomposition with respect to a very special basis.

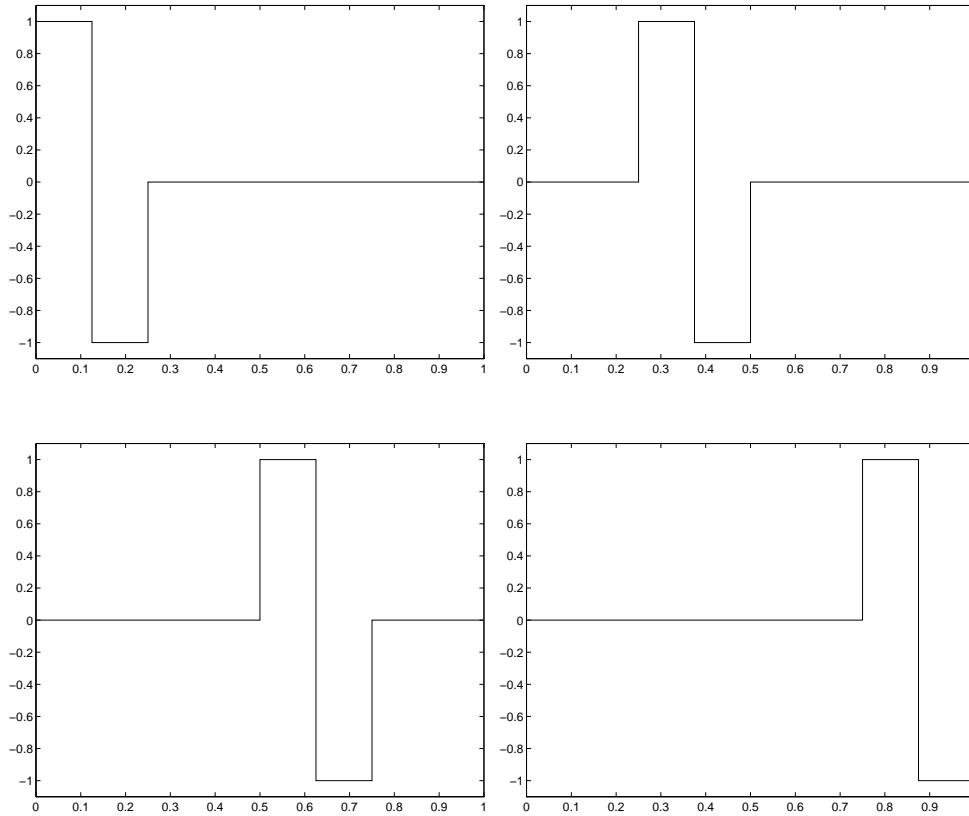


FIGURE 12
The four wavelets χ_i^2 ($0 \leq i \leq 3$), which form a basis for \mathcal{W}^2

Our earlier dumping of the smallest detail coefficients, to effect a good approximation to the original data, boils down to setting some of the wavelet coefficients to zero. In our first compression example, we approximated

$$64\phi_0^3 + 48\phi_1^3 + 16\phi_2^3 + 32\phi_3^3 + 56\phi_4^3 + 56\phi_5^3 + 48\phi_6^3 + 24\phi_6^3 \\ = 43\phi_0^0 - 3\chi_0^0 + 16\chi_0^1 + 10\chi_1^1 + 8\chi_0^2 - 8\chi_1^2 + 0\chi_2^2 + 12\chi_3^2$$

by the element $43\phi_0^0 + 0\chi_0^0 + 16\chi_0^1 + 10\chi_1^1 + 8\chi_0^2 - 8\chi_1^2 + 0\chi_2^2 + 12\chi_3^2$. These are illustrated in the stairs plots in Figure 13. As in the zig-zag plots in Figure 6(a), the two data strings are difficult to tell apart visually.

These ideas can be extended in the obvious way: For each nonnegative integer j , let \mathcal{V}^j be the vector space of piecewise constant functions on $[0, 1)$ with possible breaks at $\frac{1}{2^j}, \frac{2}{2^j}, \frac{3}{2^j}, \dots, \frac{2^j-1}{2^j}$. Then the 2^j functions ϕ_i^j defined by $\phi_i^j(x) := \phi(2^j x - i)$, $0 \leq i \leq 2^j - 1$, form a basis for \mathcal{V}^j . We thus get an infinite ascending chain¹ of vector spaces $\mathcal{V}^0 \subset \mathcal{V}^1 \subset \mathcal{V}^2 \subset \dots \subset \mathcal{V}^j \subset \mathcal{V}^{j+1} \subset \dots$, each of which is an inner product space with respect to the inner product $\langle f, g \rangle := \int_0^1 f(t)g(t)dt$. The wavelet space \mathcal{W}^j is then defined to be the orthogonal complement of \mathcal{V}^j in \mathcal{V}^{j+1} . The functions

$$\chi_i^j(x) := \chi(2^j x - i),$$

¹Emmy Noether's presence in these pages might prompt one to ask whether this chain stops! Ideally, no, but in practice, yes: for sampled signals there is a limit to the resolution that can be attained.

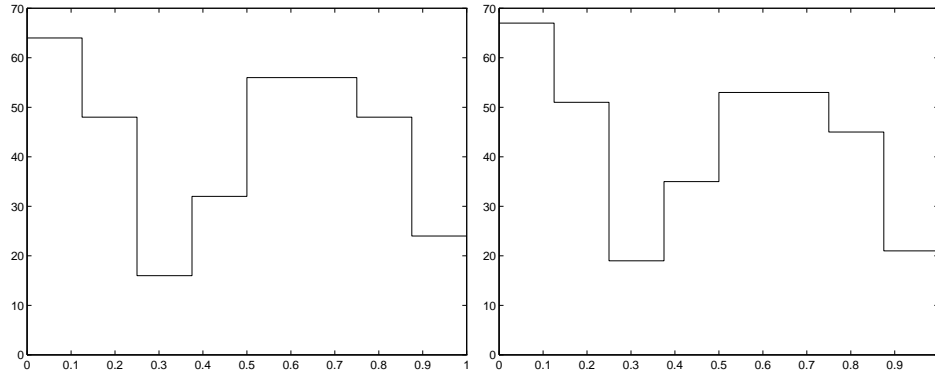


FIGURE 1

Spot the difference—a step function and an approximation of it

for $0 \leq i \leq 2^j - 1$, form a basis for \mathcal{W}^j . For any j we have

$$\begin{aligned} \mathcal{V}^j &= \mathcal{V}^{j-1} \oplus \mathcal{W}^{j-1} = \mathcal{V}^{j-2} \oplus \mathcal{W}^{j-2} \oplus \mathcal{W}^{j-1} = \dots \\ &= \mathcal{V}^0 \oplus \mathcal{W}^0 \oplus \mathcal{W}^1 \oplus \dots \oplus \mathcal{W}^{j-2} \oplus \mathcal{W}^{j-1} \end{aligned}$$

Working with strings of length 256 (such as when approximating plots of functions sampled uniformly at 2^8 points) is thus equivalent to working in the larger space \mathcal{V}^8 and using the identity:

$$\mathcal{V}^8 = \mathcal{V}^0 \oplus \mathcal{W}^0 \oplus \mathcal{W}^1 \oplus \dots \oplus \mathcal{W}^6 \oplus \mathcal{W}^7$$

There are two-dimensional analogs of these ideas, based on products of dilated and translated versions of univariate scaling functions and mother wavelets, which provide a theoretical framework for the digital image representation and compression ideas from the last section. Details can be found in [15, 16, 17, 8, 7, 11].

5. AVERAGING AND DIFFERENCING WITH MATRICES

Here we give a natural matrix formulation of the averaging and differencing technique explained in **Scheming**. We provide enough details to allow the curious reader to use a standard computer algebra package, such as *Matlab*, to reproduce the pictures in this article. The *Matlab M-files* we used are available from <http://www.spelman.edu/~colm>. Matrix multiplication is not necessarily the most efficient approach here; for large data sets there are better ways to effect the transformations.

Let A_1 , A_2 , and A_3 , respectively, denote the following matrices:

$$\left(\begin{array}{cccccccc} \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} \end{array} \right), \left(\begin{array}{cccccccc} \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & -\frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array} \right),$$

$$\begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

The three-stage transformation from (64 48 16 32 56 56 48 24) to (43 -3 16 10 8 -8 0 12) can be thought of in terms of these matrix equations:

$$\begin{aligned} (56\ 24\ 56\ 36\ \mathbf{8}\ -\mathbf{8}\ \mathbf{0}\ \mathbf{12}) &= (64\ 48\ 16\ 32\ 56\ 56\ 48\ 24)A_1, \\ (40\ 46\ \mathbf{16}\ \mathbf{10}\ \mathbf{8}\ -\mathbf{8}\ \mathbf{0}\ \mathbf{12}) &= (56\ 24\ 56\ 36\ \mathbf{8}\ -\mathbf{8}\ \mathbf{0}\ \mathbf{12})A_2, \\ (43\ -\mathbf{3}\ \mathbf{16}\ \mathbf{10}\ \mathbf{8}\ -\mathbf{8}\ \mathbf{0}\ \mathbf{12}) &= (40\ 46\ \mathbf{16}\ \mathbf{10}\ \mathbf{8}\ -\mathbf{8}\ \mathbf{0}\ \mathbf{12})A_3. \end{aligned}$$

or, equivalently, this single equation:

$$(43\ -\mathbf{3}\ \mathbf{16}\ \mathbf{10}\ \mathbf{8}\ -\mathbf{8}\ \mathbf{0}\ \mathbf{12}) = (64\ 48\ 16\ 32\ 56\ 56\ 48\ 24)A_1A_2A_3.$$

So it all boils down to linear algebra! Since the columns of the A_i 's are evidently orthogonal to each other with respect to the standard dot product, each of these matrices is invertible. The inverses are even easy to write down—after all they simply reverse the three averaging and differencing steps. In any case, we can recover the original string from the transformed version by the operation:

$$(64\ 48\ 16\ 32\ 56\ 56\ 48\ 24) = (43\ -\mathbf{3}\ \mathbf{16}\ \mathbf{10}\ \mathbf{8}\ -\mathbf{8}\ \mathbf{0}\ \mathbf{12})A_3^{-1}A_2^{-1}A_1^{-1}.$$

It is a routine matter to construct the corresponding $2^r \times 2^r$ matrices A_1, A_2, \dots, A_r needed to work with strings of length 2^r , and to write down the corresponding equations. For simplicity we write W in place of the product $A_1A_2 \dots A_r$ from now on. As mentioned earlier, there is no loss of generality in assuming that each string's length is a power of 2.

For two-dimensional image matrices, we do the same row transformations to each row, followed by corresponding column transformations. The beauty of the string transformation approach is that the equations relating the “before” and “after” strings are valid applied to an image matrix and its row-transformed form. If P is a $2^r \times 2^r$ image matrix then the equations $Q = PW$ and $P = QW^{-1}$ express the relationships between P and its row-transformed image Q . To handle column transformations, we repeat the steps above with a few transposes (denoted by $'$) thrown in. Putting everything together gives the following equations, which express the relationship between the original P and the *row-and-column-transformed* image T :

$$T = ((PW)'W)' = W'PW \quad \text{and} \quad P = ((T')W^{-1})'W^{-1} = (W^{-1})'TW^{-1}.$$

One smart shortcut we can take is to replace all of the $\pm\frac{1}{2}$'s in the matrices A_j with $\pm\frac{1}{\sqrt{2}}$'s: this is equivalent to the non-intuitive “averaging” mentioned at the end of the last section. The columns of each matrix A_j then form an orthonormal set. Consequently the same is true of the matrix W , which speeds up the reconstruction process, since the matrix inverses are simply transposes. There is more than mere speed at stake here: as we already saw in Figure 10, this normalization also leads to

compressed images that are more acceptable to the human eye. (In the language and notation introduced in **Wavelets**, this is equivalent to normalizing the Haar scaling and wavelets functions, so that we use

$$\phi_i^j(x) = 2^{\frac{j}{2}}\phi(2^jx - i) \quad \text{and} \quad \chi_i^j(x) = 2^{\frac{j}{2}}\chi(2^jx - i),$$

(for $0 \leq i \leq 2^j - 1$) as bases for \mathcal{V}^j and \mathcal{W}^j , respectively.)

In matrix terms, the image compression scheme works like this: Start with P , and compute $T = W'PW$, which (we hope) will be somewhat sparse. Choose a threshold value ϵ , and replace by zero any entries of T whose absolute value is less than or equal to ϵ . Denote the resulting doctored matrix by D ; this is sparse by design, and thus easier to store and transmit than P . To reconstruct an image from D , compute $R = (W^{-1})'DW^{-1}$.

Lossless compression is the case where $D = T$ (e.g., if $\epsilon = 0$) so that $R = P$. Otherwise we have *lossy compression*, in which case the goal is to pick ϵ carefully, so as to balance the conflicting requirements of storage (the more zeros in D , the better) and visual acceptability of the reconstruction R .

6. WAVELETS ON THE WORLD WIDE WEB

In the case of real-time image retrieval, such as grabbing images on the World Wide Web, the compression technique we have discussed allows for a type of progressive image transmission: When an image P is requested electronically, a wavelet-encoded version T is brought out of storage, and bits of information about it are sent “over the wires,” starting with the overall average and the larger wavelet coefficients, and working down to the smallest wavelet coefficients.

As this information is received by the user, it is used to display a reconstruction of P , starting with a very crude approximation of the image that, rapidly updated and refined, looks noticeably better as more wavelet coefficients are used. For instance, the images in Figures 9 and 10 could form stages in a progressive transmission. Eventually (assuming the user has deemed this picture worth waiting for) all of the wavelet coefficients will have been transmitted and a perfect copy of P displayed. If the user loses interest or patience along the way, she can easily halt the process and move on to some more pressing task, such as learning Fourier analysis.

7. WAVELET DETAILS

In attempting to make this introduction to wavelets as easy and painless as possible, we may have suggested that the subject is neither deep nor profound: nothing could be further from the truth. Here we try to put the Haar wavelets, which were used in image processing as far back as the 1970s [14], in context, and hint at the recent generalizations which have generated so much interest in the mathematical community and elsewhere.

A wide variety of wavelets is available to decompose, analyze, and synthesize both discrete and continuous data. In general, a wavelet is any function whose dilations and translations form a Riesz basis for the function space $\mathcal{L}^2(\mathbb{R})$ (the set of square integrable functions on the real line). For simplicity, we ignore normalization considerations. We also assume that all functions are real-valued.

Most wavelets are derived from a corresponding *scaling function*, namely a function ϕ satisfying a *scaling equation* $\phi(x) = \sum_{i \in \mathbb{Z}} c_i \phi(2x - i)$. Given such a function, we

define \mathcal{V}^0 to be the closure of the linear span of the set of integer translates $\phi_i^0(x) := \phi(x - i)$, $i \in \mathbb{Z}$, of $\phi(x)$, and then for each $j \in \mathbb{Z}$ take \mathcal{V}^j to be the closure of the linear span of the set of dilated and translated functions $\phi_i^j(x) := \phi(2^j x - i)$, $i \in \mathbb{Z}$. A *multiresolution analysis* (MRA) is said to exist when the induced doubly infinite collection of vector spaces $\dots \subset \mathcal{V}^{-2} \subset \mathcal{V}^{-1} \subset \mathcal{V}^0 \subset \mathcal{V}^1 \subset \mathcal{V}^2 \subset \dots$ satisfies three criteria:

1. $f(x) \in \mathcal{V}^j \Leftrightarrow f(2^{-j}x) \in \mathcal{V}^0, \forall j \in \mathbb{Z}$
2. $\bigcap_{i \in \mathbb{Z}} \mathcal{V}^i = \{0\}$
3. $\overline{\bigcup_{i \in \mathbb{Z}} \mathcal{V}^i} = \mathcal{L}^2(\mathbb{R})$.

Once a MRA is in place, it is an easy matter to define the corresponding *mother wavelet*:

$$\chi(x) := \sum_{i \in \mathbb{Z}} (-1)^i c_{1-i} \phi(2x - i),$$

where $\phi(x) = \sum_{i \in \mathbb{Z}} c_i \phi(2x - i)$. This wavelet turns out to have zero integral over the whole real line.

One way to generalize the Haar scaling function (which is a first order B-spline) and wavelet is as follows: for any $k \in \mathbb{N}$, the k th order B-spline (which can be thought of as the convolution of the Haar scaling function with itself $k - 1$ times) satisfies the scaling equation $\phi(x) = \sum_{i=0}^k 2^{-k+1} \binom{k}{i} \phi(2x - i)$. This yields an MRA, and hence a wavelet in the manner just described [6, Chapter 5], [18]. These *spline wavelets* are compactly supported and have $k - 2$ continuous derivatives, but only in the Haar case do we get orthogonality between members of the induced family of translated and dilated functions.

While one does not always insist on orthogonality for such basis functions, it is generally considered desirable for wavelets to have compact support, or at least rapid decay, in sharp contrast to the behavior of the sines and cosines which play a central role in Fourier analysis. This renders wavelets ideal for representing non-periodic functions, especially those with spikes or discontinuities. For one thing, fewer basis elements and coefficients are needed to represent such a function when compared with the classical Fourier series expansion.

There are three things to try to juggle here: smoothness, support, and orthogonality. Sadly, we can't have everything: there are no infinitely differentiable orthonormal wavelets which have exponential decay (never mind compact support) [6, Chapter 5]; so some sort of compromise is in order.

The spline wavelet construction above can be modified to yield the so-called Battle-Lemarie wavelets, which have exponential decay, are $k - 2$ times continuously differentiable *and* orthonormal. In 1988, Daubechies made a breakthrough with the construction of compactly supported, orthonormal wavelets with any desired finite degree of smoothness. Her simplest non-trivial example is continuous, and is derived from a continuous scaling function ϕ , which satisfies

$$\phi(x) = \frac{1 + \sqrt{3}}{4} \phi(2x) + \frac{3 + \sqrt{3}}{4} \phi(2x - 1) + \frac{3 - \sqrt{3}}{4} \phi(2x - 2) + \frac{1 - \sqrt{3}}{4} \phi(2x - 3).$$

There are no closed form expressions for these functions: they are studied by means of a careful analysis that starts with taking the Fourier transform of the scaling equation [6, Chapter 6]. (For fixed x , we *can* solve for $\phi(x)$ as the limit of the sequence $\Phi_j(x)$)

defined recursively via: $\Phi_j(x) = \sum_{i \in \mathbb{Z}} c_i \Phi_{j-1}(2x - i)$, where Φ_0 is the Haar scaling function.) Despite our ingrained instincts, which suggest that we try to “solve” any equation set in front of us, in general there is no need to get our hands on the scaling functions or wavelets themselves; in many ways they are best explored using the numbers c_i alone.

These more sophisticated, continuous wavelets, produce smoother, more satisfactory compressed images than the ones that we obtained [15, 16, 17]. Here lies the real potential for progressive image transmission, and perhaps adaptive plotting, too.

For full mathematical treatments, the reader could start with [6], [5], or [13]. Books covering applications as well as theory include [3] and [1]. A gentler survey of the field can be found in [10].

8. CLOSING REMARKS

A major advantage of wavelet over Fourier methods, which we have not touched on at all, is that it with wavelets it is possible simultaneously to localize in space (or time) and frequency. Also, wavelets capture detail at different scales at the same time: the plots of the damped sine curve in Figure 8 and the compressed images of Emmy Noether illustrate how the wavelet details take advantage of the changing nature of the data variation over different regions. See [6], [5], or [13] for further details.

Glassner’s *Principles of Digital Image Synthesis* is an excellent resource, full of helpful pictures, for wavelet basics as they relate to graphics, that also discusses some of the connections with Fourier methods [8, Chapter 6]. Strang and Nguyen [20] treat wavelets from a signal processing perspective.

For an account of a recent adoption of wavelets as a standard for image compression, see [4] or [20]. Another common use of wavelets is to the *denoising* of digital data. There, unlike in the compression we considered, one discards detail coefficients *larger* than a certain threshold (see [7], [20]). There are also wavelet applications to video compression [20]; to medicine (tomography, MRI images, mammography, radiography, and neural networks) [1]; to audio and speech signals [21], [20]; and to partial differential operators and equations [3], [20].

An excellent World Wide Web resource for wavelet matters is The Wavelet Digest <http://www.wavelet.org/wavelet> (email: help@wavelet.org).

Acknowledgements: This research was funded in part by the W.K. Kellogg Foundation, through Spelman College’s Center for Scientific Applications of Mathematics (CSAM). The author would also like to thank CSAM Director Sylvia Bozeman, who first stimulated his interest in wavelets.

Thanks to: Tony DeRose and colleagues at the University of Washington for showing the author (and the world) that wavelet basics can be understood and appreciated without a solid background in Fourier methods; Björn Jawerth of the University of South Carolina for an inspiring workshop at SIAM-SEAS in Charleston in March 1995; Alain Fournier *et al.* for the superb wavelets course at SIGGRAPH ’95; and Rhonda Hughes of Bryn Mawr College for helpful conversations and clarifications.

Matthew Wright of the University of Southampton dispensed sound advice about *Matlab* and other matters. Shun Yan Cheung of Emory University provided invaluable L^AT_EX assistance. The Editor and the referees made numerous helpful suggestions, which led to a better and more readable article.

REFERENCES

- [1] Akram Aldroubi and Michael Unser (editors), *Wavelets in Medicine and Biology*, CRC Pr., Boca Raton FL, 1996.
- [2] Robert G. Bartle, *The Elements of Real Analysis*, 2nd edition, John Wiley & Sons, New York NY, 1976.
- [3] John J. Benedetto and Michael Frazier (editors), *Wavelets; Mathematics and Applications*, CRC Pr., Boca Raton FL, 1996.
- [4] Christopher M. Brislawn, Fingerprints go digital, *AMS Notices* 42 (1995), 1278-1283.

- [5] Charles Chui, *An Introduction to Wavelets*, Academic Pr. , San Diego CA, 1992.
- [6] Ingrid Daubechies, *Ten Lectures on Wavelets*, CBMS 61, SIAM Pr., Philadelphia PA, 1992.
- [7] Ron DeVore, Björn Jawerth and Bradley Lucier, Image compression through wavelet transform coding, *IEEE Trans. Information Theory* 38, 2 (1992), 719-746.
- [8] Andrew S. Glassner, *Principles of Digital Image Synthesis*, Morgan Kaufmann, San Francisco CA, 1995.
- [9] André Heck, *Introduction to Maple*, Springer Verlag, New York NY, 1993.
- [10] Barbara Burke Hubbard, *The World According To Wavelets*, A.K. Peters, Wellesley MA, 1996.
- [11] Björn Jawerth and Wim Sweldens, An overview of wavelet based multiresolution analyses, *SIAM Reviews* 36, 3 (1994), 377-412.
- [12] *Matlab Reference Guide*, The MathWorks, Natick MA, 1992.
- [13] Yves Meyer, *Wavelets: Algorithms and Applications*, SIAM Pr., Philadelphia PA, 1993.
- [14] William Pratt, *Digital Image Processing*, Wiley-Interscience, New York NY, 1978.
- [15] Eric Stollnitz, Tony DeRose and David Salesin, Wavelets for computer graphics: a primer, Part 1, *IEEE Computer Graphics And Applications* 5, 3 (1995), 76-84.
- [16] Eric Stollnitz, Tony DeRose and David Salesin, Wavelets for computer graphics: a primer, part 2, *IEEE Computer Graphics And Applications* 5, 4 (1995), 75-85.
- [17] Eric Stollnitz, Tony DeRose and David Salesin, *Wavelets for Computer Graphics*, Morgan Kaufmann, San Francisco CA, 1996.
- [18] Gilbert Strang, Wavelets and dilation equations: a brief introduction, *SIAM Reviews* 31, 4 (1989), 614-627.
- [19] Gilbert Strang, Wavelets, *American Scientist* 82 (1994), 250-255.
- [20] Gilbert Strang and Truong Nguyen, *Wavelets and Filter Banks*, Wellesley-Cambridge Pr., Wellesley MA, 1996.
- [21] Mladen Victor Wickerhauser, *Adapted Wavelet Analysis from Theory to Software*, A.K. Peters, Wellesley MA, 1994.
- [22] Tom Wickham-Jones, *Mathematica Graphics*, Telos/Springer Verlag, Santa Clara CA, 1994.