

Wavelets

The word ‘multiscale’ can mean many things. However, in this book we are generally concerned with the representation of objects at a set of scales and then manipulating these representations at several scales simultaneously.

One main aim of this book is to explain the role of wavelet methods *in statistics*, and so the current chapter is necessarily a rather brief introduction to wavelets. More mathematical (and authoritative) accounts can be found in Daubechies (1992), Meyer (1993b), Chui (1997), Mallat (1998), Burrus et al. (1997), and Walter and Shen (2001). A useful article that charts the history of wavelets is Jawerth and Sweldens (1994). The book by Heil and Walnut (2006) contains many important early papers concerning wavelet theory.

Statisticians also have reason to be proud. Yates (1937) introduced a fast computational algorithm for the (hand) analysis of observations taken in a factorial experiment. In modern times, this algorithm might be called a ‘generalized FFT’, but it is also equivalent to a Haar wavelet packet transform, which we will learn about later in Section 2.11. So statisticians have been ‘doing’ wavelets, and wavelet packets, since at least 1937!

2.1 Multiscale Transforms

2.1.1 A multiscale analysis of a sequence

Before we attempt formal definitions of wavelets and the wavelet transform we shall provide a gentle introduction to the main ideas of multiscale analysis. The simple description we give next explains the main features of a wavelet transform.

As many problems in statistics arise as a sequence of data observations, we choose to consider the wavelet analysis of sequences rather than functions, although we will examine the wavelet transform of functions later. Another reason is that we want to use **R** to illustrate our discussion, and **R** naturally handles discrete sequences (vectors).

We begin with discrete sequence (vector) of data: $y = (y_1, y_2, \dots, y_n)$, where each y_i is a real number and i is an integer ranging from one to n . For our illustration, we assume that the length of our sequence n is a power of two, $n = 2^J$, for some integer $J \geq 0$. Setting $n = 2^J$ should not be seen as an absolute limitation as the description below can be modified for other n . We call a sequence where $n = 2^J$ a *dyadic* one.

The following description explains how we extract multiscale ‘information’ from the vector y . The key information we extract is the ‘detail’ in the sequence at different scales and different locations. Informally, by ‘detail’ we mean ‘degree of difference’ or (even more roughly) ‘variation’ of the observations of the vector at the given scale and location.

The first step in obtaining the detail we require is

$$d_k = y_{2k} - y_{2k-1}, \quad (2.1)$$

for $k = 1, \dots, n/2$. So, for example, $d_1 = y_2 - y_1$, $d_2 = y_4 - y_3$, and so on. Operation (2.1) extracts ‘detail’ in that if y_{2k} is very similar to y_{2k-1} , then the coefficient d_k will be very small. If $y_{2k} = y_{2k-1}$ then the d_k will be exactly zero. This seemingly trivial point becomes extremely important later on. If y_{2k} is very different from y_{2k-1} , then the coefficient d_k will be very large.

Hence, the sequence d_k encodes the difference between successive pairs of observations in the original y vector. However, $\{d_k\}_{k=1}^{n/2}$ is *not* the more conventional *first difference vector* (**diff** in \mathbb{R}). Specifically, differences such as $y_3 - y_2$ are missing from the $\{d_k\}$ sequence. The $\{d_k\}$ sequence encodes the difference or *detail* at locations (approximately) $(2k + 2k - 1)/2 = 2k - 1/2$.

We mentioned above that we wished to obtain ‘detail’ at several different scales and locations. Clearly the $\{d_k\}$ sequence gives us information at several different locations. However, each d_k only gives us information about a particular y_{2k} and its *immediate* neighbour. Since there are no closer neighbours, the sequence $\{d_k\}$ gives us information at and around those points y_{2k} at the *finest* possible scale of detail. How can we obtain information at coarser scales? The next step will begin to do this for us.

The next step is extremely similar to the previous one except the subtraction in (2.1) is replaced by a summation:

$$c_k = y_{2k} + y_{2k-1} \quad (2.2)$$

for $k = 1, \dots, n/2$. This time the sequence $\{c_k\}_{k=1}^{n/2}$ is a set of scaled local averages (scaled because we failed to divide by two, which a proper mean would require), and the information in $\{c_k\}$ is a coarsening of that in the original y vector. Indeed, the operation that turns $\{y_i\}$ into $\{c_k\}$ is similar to a moving average smoothing operation, except, as with the differencing above, we average non-overlapping consecutive pairs. Contrast this to regular moving averages, which average over overlapping consecutive pairs.

An important point to notice is that each c_k contains information originating from both y_{2k} and y_{2k-1} . In other words, it includes information from

two adjacent observations. If we now wished to obtain coarser *detail* than contained in $\{d_k\}$, then we could compare two adjacent c_k .

Before we do this, we need to introduce some further notation. We first introduced finest-scale detail d_k . Now we are about to introduce coarser-scale detail. Later, we will go on to introduce detail at successively coarser scales. Hence, we need some way of keeping track of the scale of the detail. We do this by introducing another subscript, j (which some authors represent by a superscript). The original sequence y consisted of 2^J observations. The finest-level detail $\{d_k\}$ consists of $n/2 = 2^{J-1}$ observations, so the extra subscript we choose for the finest-level detail is $j = J - 1$ and we now refer to the d_k as $d_{J-1,k}$. Sometimes the comma is omitted when the identity and context of the coefficients is clear, i.e., $d_{j,k}$. Thus, the finest-level averages, or smooths, c_k are renamed to become $c_{J-1,k}$.

To obtain the next coarsest detail we repeat the operation of (2.1) to the finest-level averages, $c_{J-1,k}$ as follows:

$$d_{J-2,\ell} = c_{J-1,2\ell} - c_{J-1,2\ell-1}, \quad (2.3)$$

this time for $\ell = 1, \dots, n/4$. Again, $d_{J-2,\ell}$ encodes the difference, or detail present, between the coefficients $c_{J-1,2\ell}$ and $c_{J-1,2\ell-1}$ in *exactly the same way* as for the finer-detail coefficient in (2.1). From a quick glance of (2.3) it does not immediately appear that $d_{J-2,\ell}$ is at a different scale from $d_{J-1,k}$. However, writing the $c_{J-1,\cdot}$ in terms of their constituent parts as defined by (2.2), gives

$$d_{J-2,\ell} = (y_{4\ell} + y_{4\ell-1}) - (y_{4\ell-2} + y_{4\ell-3}) \quad (2.4)$$

for the same ℓ as in (2.3). For example, if $\ell = 1$, we have $d_{J-2,1} = (y_4 + y_3) - (y_2 + y_1)$. It should be clear now that $d_{J-2,\ell}$ is a set of differences of components that are averages of two original data points. Hence, they can be thought of as ‘scale two’ differences, whereas the $d_{J-1,k}$ could be thought of as ‘scale one’ differences. This is our first encounter with multiscale: we have differences that exist at two different scales.

Scale/level terminology. At this point, we feel the need to issue a warning over terminology. In the literature the words ‘scale’, ‘level’, and occasionally ‘resolution’ are sometimes used interchangeably. In this book, we strive to use ‘level’ for the integral quantity j and ‘scale’ is taken to be the quantity 2^j (or 2^{-j}). However, depending on the context, we sometimes use scale to mean level. With the notation in this book j larger (positive) corresponds to finer scales, j smaller to coarser scales.

Now nothing can stop us! We can repeat the averaging Formula (2.2) on the $c_{J-1,k}$ themselves to obtain

$$c_{J-2,\ell} = c_{J-1,2\ell} + c_{J-1,2\ell-1} \quad (2.5)$$

for $\ell = 1, \dots, n/4$. Writing (2.5) in terms of the original vector y for $\ell = 1$ gives $c_{J-2,1} = (y_2 + y_1) + (y_4 + y_3) = y_1 + y_2 + y_3 + y_4$: the local mean of the first four observations without the $\frac{1}{4}$ —again $c_{J-2,\ell}$ is a kind of moving average.

By repeating procedures (2.1) and (2.2) we can continue to produce both detail and smoothed coefficients at progressively coarser scales. Note that the actual scale increases by a factor of two each time and the number of coefficients at each scale decreases by a factor of two. The latter point also tells us when the algorithm stops: when only one c coefficient is produced. This happens when there is only $2^0 = 1$ coefficient, and hence this final coefficient must have level index $j = 0$ (and be $c_{0,1}$).

Figure 2.1 shows the (2.1) and (2.2) operations in block diagram form. These kinds of diagrams are used extensively in the literature and are useful for showing the main features of multiscale algorithms. Figure 2.1 shows the generic step of our multiscale algorithm above. Essentially an input vector $c_j = (c_{j,1}, c_{j,2}, \dots, c_{j,m})$ is transformed into two output vectors c_{j-1} and d_{j-1} by the above mathematical operations. Since Figure 2.1 depicts the ‘generic

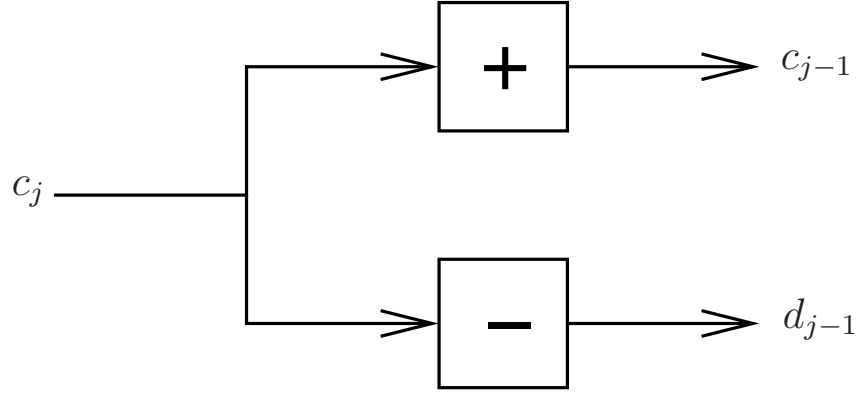


Fig. 2.1. Generic step in ‘multiscale transform’. The input *vector*, c_j , is transformed into two output vectors, c_{j-1} and d_{j-1} , by the addition and subtraction operations defined in Equations (2.1) and (2.2) for $j = J, \dots, 1$.

step’, the figure also implicitly indicates that the output c_{j-1} will get fed into an identical copy of the block diagram to produce vectors c_{j-2} and d_{j-2} and so on. Figure 2.1 does not show that the initial input to the ‘multiscale algorithm’ is the input vector y , although it could be that $c^J = y$. Also, the figure does not clearly indicate that the length of c_{j-1} (and d_{j-1}) is half the length of c_j , and so, in total, *the number of output elements of the step is identical to the number of input elements*.

Example 2.1. Suppose that we begin with the following sequence of numbers: $y = (y_1, \dots, y_n) = (1, 1, 7, 9, 2, 8, 8, 6)$. Since there are eight elements of y , we have $n = 8$ and hence $J = 3$ since $2^3 = 8$. First apply Formula (2.1) and simply subtract the first number from the second as follows: $d_{2,1} = y_2 - y_1 = 1 - 1 = 0$. For the remaining d coefficients at level $j = 2$ we obtain $d_{2,2} = y_4 - y_3 = 9 - 7 =$

2, $d_{2,3} = y_6 - y_5 = 8 - 2 = 6$ and finally $d_{2,4} = y_8 - y_7 = 6 - 8 = -2$. As promised there are $2^{J-1} = n/2 = 4$ coefficients at level 2.

For the ‘local average’, we perform the same operations as before but replace the subtraction by addition. Thus, $c_{2,1} = y_2 + y_1 = 1 + 1 = 2$ and for the others $c_{2,2} = 9 + 7 = 16$, $c_{2,3} = 8 + 2 = 10$, and $c_{2,4} = 6 + 8 = 14$.

Notice how we started off with eight y_i and we have produced four $d_{2,\cdot}$ coefficients and four $c_{2,\cdot}$ coefficients. Hence, we produced as many output coefficients as input data. It is useful to write down these computations in a graphical form such as that depicted by Figure 2.2. The organization of

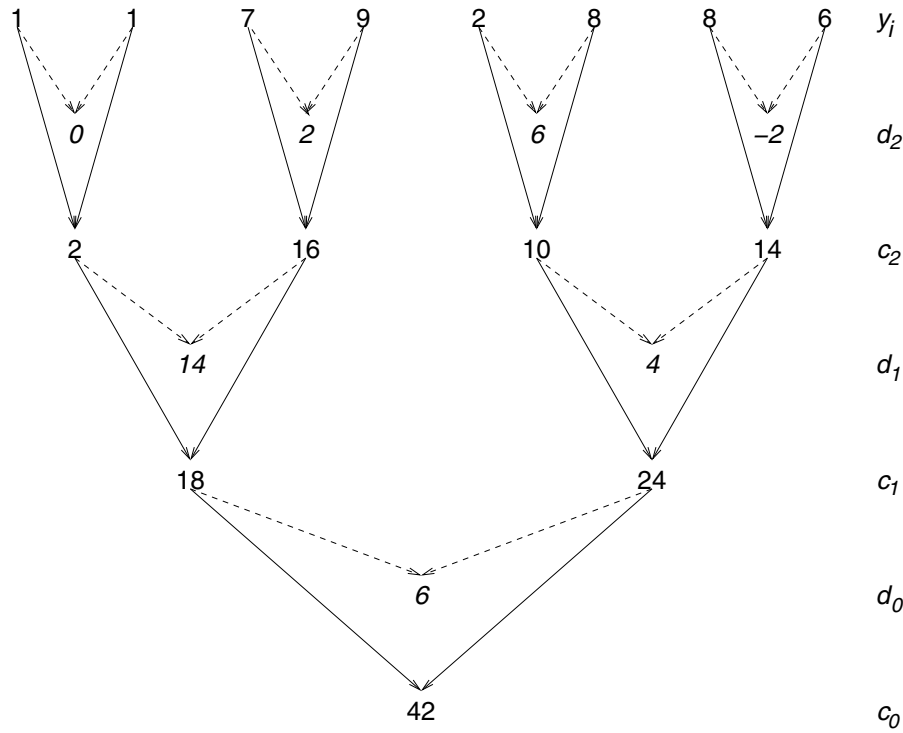


Fig. 2.2. Graphical depiction of a multiscale transform. The *dotted arrows* depict a subtraction and *numbers in italics* the corresponding detail coefficient $d_{j,k}$. The *solid arrows* indicate addition, and *numbers set in the upright font* correspond to the $c_{j,k}$.

coefficients in Figure 2.2 can be visualized as an inverted pyramid (many numbers at the top, one number at the bottom, and steadily decreasing from top to bottom). The algorithm that we described above is an example of a *pyramid* algorithm.

The derived coefficients in Figure 2.2 all provide information about the original sequence in a scale/location fashion. For example, the final 42

indicates that the sum of the *whole* original sequence is 42. The 18 indicates that the sum of the first four elements of the sequence is 18. The 4 indicates that the sum of the last quarter of the data minus the sum of the third quarter is four. In this last example we are essentially saying that the consecutive difference in the ‘scale two’ information in the third and last quarters is four.

So far we have avoided using the word *wavelet* in our description of the multiscale algorithm above. However, the $d_{j,k}$ ‘detail’ coefficients are *wavelet* coefficients and the $c_{j,k}$ coefficients are known as *father wavelet* or *scaling function* coefficients. The algorithm that we have derived is one kind of (discrete) wavelet transform (DWT), and the general pyramid algorithm for wavelets is due to Mallat (1989b). The wavelets underlying the transform above are called *Haar wavelets* after Haar (1910). Welcome to Wavelets!

Inverse. The original sequence can be exactly reconstructed by using only the wavelet coefficients $d_{j,k}$ and the last c_{00} . For example, the inverse formulae to the simple ones in (2.3) and (2.5) are

$$c_{j-1,2k} = (c_{j-2,k} + d_{j-2,k})/2 \quad (2.6)$$

and

$$c_{j-1,2k-1} = (c_{j-2,k} - d_{j-2,k})/2. \quad (2.7)$$

Section 2.7.4 gives a full description of the inverse discrete wavelet transform.

Sparsity. A key property of wavelet coefficient sequences is that they are often sparse. For example, suppose we started with the input sequence $(1, 1, 1, 1, 2, 2, 2, 2)$. If we processed this sequence with the algorithm depicted by Figure 2.2, then *all* of the wavelet coefficients at scales one and two would be *exactly* zero. The only non-zero coefficient would be $d_0 = -4$. Hence, the wavelet coefficients are an extremely sparse set. This behaviour is characteristic of wavelets: piecewise smooth functions have sparse representations. The vector we chose was actually piecewise constant, an extreme example of piecewise smooth. The sparsity is a consequence of the unconditional basis property of wavelets briefly discussed in the previous chapter and also of the vanishing moments property of wavelets to be discussed in Section 2.4.

Energy. In the example above the input sequence was $(1, 1, 7, 9, 2, 8, 8, 6)$. This input sequence can be thought to possess an ‘energy’ or norm as defined by $\|y\|^2 = \sum_{i=1}^8 y_i^2$. (See Section B.1.3 for a definition of norm.) Here, the norm of the input sequence is $1+1+49+4+64+64+36 = 219$. The transform wavelet coefficients are (from finest to coarsest) $(0, 2, 6, -2, 14, 4, 6, 42)$. What is the norm of the wavelet coefficients? It is $0+4+36+4+196+16+36+1764 = 2056$. Hence the norm, or energy, of the output sequence is much larger than that of the input. We would like a transform where the ‘output energy’ is the same as the input. We address this in the next section.

2.1.2 Discrete Haar wavelets

To address the ‘energy’ problem at the end of the last example, let us think about how we might change Formulae (2.1) and (2.2) so as to conserve energy.

Suppose we introduce a multiplier α as follows. Thus (2.1) becomes

$$d_k = \alpha(y_{2k} - y_{2k-1}), \quad (2.8)$$

and similarly (2.2) becomes

$$c_k = \alpha(y_{2k} + y_{2k-1}). \quad (2.9)$$

Thus, with this mini transform the input (y_{2k}, y_{2k-1}) is transformed into the output (d_k, c_k) and the (squared) norm of the output is

$$\begin{aligned} d_k^2 + c_k^2 &= \alpha^2(y_{2k}^2 - 2y_{2k}y_{2k-1} + y_{2k-1}^2) + \alpha^2(y_{2k}^2 + 2y_{2k}y_{2k-1} + y_{2k-1}^2) \\ &= 2\alpha^2(y_{2k}^2 + y_{2k-1}^2), \end{aligned} \quad (2.10)$$

where $y_{2k}^2 + y_{2k-1}^2$ is the (squared) norm of the input coefficients. Hence, if we wish the norm of the output to equal the norm of the input, then we should arrange for $2\alpha^2 = 1$ and hence we should set $\alpha = 2^{-1/2}$. With this normalization the formula for the discrete wavelet coefficients is

$$d_k = (y_{2k} - y_{2k-1})/\sqrt{2}, \quad (2.11)$$

and similarly for the father wavelet coefficient c_k . Mostly we keep this normalization throughout, although it is sometimes convenient to use other normalizations. For example, see the normalization for the Haar-Fisz transform in Section 6.4.6.

We can rewrite (2.11) in the following way:

$$d_k = g_0 y_{2k} + g_1 y_{2k-1}, \quad (2.12)$$

where $g_0 = 2^{-1/2}$ and $g_1 = -2^{-1/2}$, or in the more general form:

$$d_k = \sum_{\ell=-\infty}^{\infty} g_\ell y_{2k-\ell}, \quad (2.13)$$

where

$$g_\ell = \begin{cases} 2^{-1/2} & \text{for } \ell = 0, \\ -2^{-1/2} & \text{for } \ell = 1, \\ 0 & \text{otherwise.} \end{cases} \quad (2.14)$$

Equation (2.13) is similar to a filtering operation with filter coefficients of $\{g_\ell\}_{\ell=-\infty}^{\infty}$.

Example 2.2. If we repeat Example 2.1 with the new normalization, then $d_{2,1} = (y_2 - y_1)/\sqrt{2} = (1 - 1)/\sqrt{2} = 0$, and then for the remaining d coefficients at scale $j = 2$ we obtain $d_{2,2} = (y_4 - y_3)/\sqrt{2} = (9 - 7)/\sqrt{2} = \sqrt{2}$, $d_{2,3} = (y_6 - y_5)/\sqrt{2} = (8 - 2)/\sqrt{2} = 3\sqrt{2}$, and, finally, $d_{2,4} = (y_8 - y_7)/\sqrt{2} = (6 - 8)/\sqrt{2} = -\sqrt{2}$.

Also, $c_{2,1} = (y_2 + y_1)/\sqrt{2} = (1 + 1)/\sqrt{2} = \sqrt{2}$ and for the others $c_{2,2} = (9 + 7)/\sqrt{2} = 8\sqrt{2}$, $c_{2,3} = (8 + 2)/\sqrt{2} = 5\sqrt{2}$, and $c_{2,4} = (6 + 8)/\sqrt{2} = 7\sqrt{2}$.

The $c_{2,k}$ permit us to find the $d_{1,\ell}$ and $c_{1,\ell}$ as follows: $d_{1,1} = (c_{2,2} - c_{2,1})/\sqrt{2} = (8\sqrt{2} - \sqrt{2})/\sqrt{2} = 7$, $d_{1,2} = (c_{2,4} - c_{2,3})/\sqrt{2} = (7\sqrt{2} - 5\sqrt{2})/\sqrt{2} = 2$, and similarly $c_{1,1} = 9$, $c_{1,2} = 12$.

Finally, $d_{0,1} = (c_{1,2} - c_{1,1})/\sqrt{2} = (12 - 9)/\sqrt{2} = 3\sqrt{2}/2$ and $c_{0,1} = (12 + 9)/\sqrt{2} = 21\sqrt{2}/2$.

Example 2.3. Let us perform the transform described in Example 2.2 in **WaveThresh**. First, start R and load the **WaveThresh** library by the command

```
> library("WaveThresh")
```

and now create the vector that contains our input to the transform:

```
> y <- c(1,1,7,9,2,8,8,6)
```

The function to perform the discrete wavelet transform in **WaveThresh** is called **wd**. So let us perform that transform and store the answers in an object called **ywd**:

```
> ywd <- wd(y, filter.number=1, family="DaubExPhase")
```

The **wd** call here supplies two extra arguments: the **filter.number** and **family** arguments that specify the type of wavelet that is used for the transform. Here, the values **filter.number=1** and **family="DaubExPhase"** specify Haar wavelets (we will see why these argument names are used later).

The **ywd** object returned by the **wd** call is a *composite* object (or list object). That is, **ywd** contains many different *components* all giving some useful information about the wavelet transform that was performed. The names of the components can be displayed by using the **names** command as follows:

```
> names(ywd)
[1] "C"      "D"      "nlevels" "fl.dbase" "filter"
[6] "type"   "bc"     "date"
```

For example, if one wishes to know what filter produced a particular wavelet decomposition object, then one can type

```
> ywd$filter
```

and see the output

```
$H
[1] 0.7071068 0.7071068
```

```
$G
NULL
```

```
$name
```



```

[1] "Haar wavelet"

$family
[1] "DaubExPhase"

$filter.number
[1] 1

```

which contains information about the wavelet used for the transform. Another interesting component of the `ywd$filter` object is the `H` component, which is equal to the vector $(2^{-1/2}, 2^{-1/2})$. This vector is the one involved in the filtering operation, analogous to that in (2.13), that produces the c_k , in other words:

$$c_k = \sum_{\ell=-\infty}^{\infty} h_{\ell} y_{2k-\ell}, \quad (2.15)$$

where

$$h_{\ell} = \begin{cases} 2^{-1/2} & \text{for } \ell = 0, \\ 2^{-1/2} & \text{for } \ell = 1, \\ 0 & \text{otherwise.} \end{cases} \quad (2.16)$$

Possibly the most important information contained within the wavelet decomposition object `ywd` are the wavelet coefficients. They are stored in the `D` component of the object, and they can be accessed directly if desired (see the Help page of `wd` to discover how, and in what order, the coefficients are stored). However, the coefficients are stored in a manner that is efficient for computers, but less convenient for human interpretation. Hence, `WaveThresh` provides a function, called `accessD`, to extract the coefficients from the `ywd` object in a readable form.

Suppose we wished to extract the finest-level coefficients. From Example 2.2 these coefficients are $(d_{2,1}, d_{2,2}, d_{2,3}, d_{2,4}) = (0, \sqrt{2}, 3\sqrt{2}, -\sqrt{2})$. We can obtain the same answer by accessing level two coefficients from the `ywd` object as follows:

```

> accessD(ywd, level=2)
[1] 0.000000 -1.414214 -4.242641 1.414214

```

The answer looks correct except the numbers are the negative of what they should be. Why is this? The answer is that `WaveThresh` uses the filter $g_0 = -2^{-1/2}$ and $g_1 = 2^{-1/2}$ instead of the one shown in (2.13). However, this raises a good point: for this kind of analysis one can use filter coefficients themselves or their negation, and/or one can use the reversed set of filter coefficients. In all these circumstances, one still obtains the same kind of analysis.

Other resolution levels in the wavelet decomposition object can be obtained using the `accessD` function with the `levels` arguments set to one and

zero. The $c_{j,k}$ father wavelet coefficients can be extracted using the `accessC` command, which has an analogous mode of operation.

It is often useful to obtain a picture of the wavelet coefficients. This can be achieved in `WaveThresh` by merely plotting the coefficients as follows:

```
> plot(ywd)
```

which produces a plot like the one in Figure 2.3.

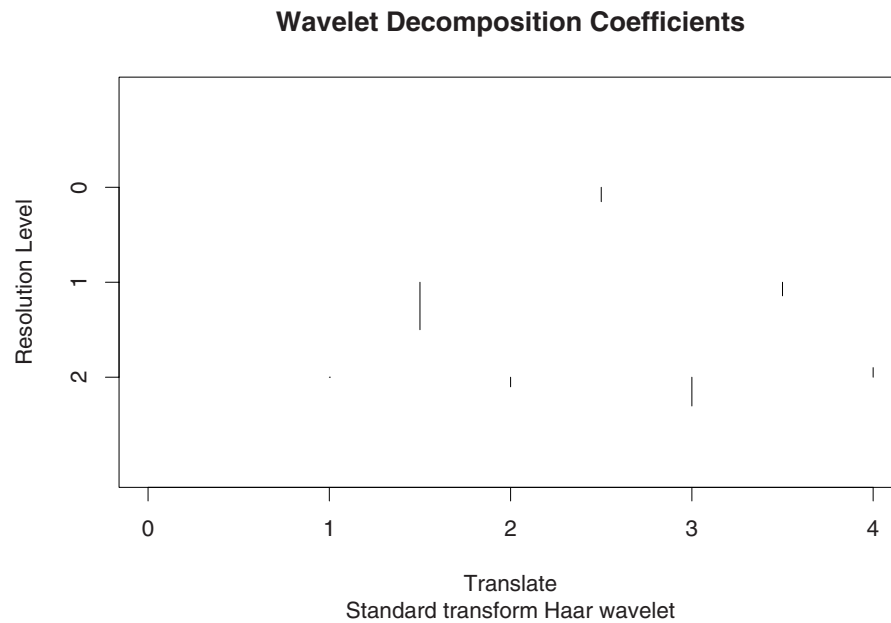


Fig. 2.3. Wavelet coefficient plot of `ywd`. The coefficients $d_{j,k}$ are plotted with the finest-scale coefficients at the *bottom* of the plot, and the coarsest at the *top*. The level is indicated by the *left-hand axis*. The value of the coefficient is displayed by a *vertical mark* located along an imaginary horizontal line centred at each level. Thus, the three marks located at resolution level 2 correspond to the three non-zero coefficients $d_{2,2}$, $d_{2,3}$, and $d_{2,4}$. Note that the zero $d_{2,1}$ is not plotted. The k , or location parameter, of each $d_{j,k}$ wavelet coefficient is labelled ‘Translate’, and the horizontal positions of the coefficients indicate the approximate position in the original sequence from which the coefficient is derived. Produced by `f.wav1()`.

Other interesting information about the `ywd` object can be obtained by simply typing the name of the object. For example:

```
> ywd
Class 'wd' : Discrete Wavelet Transform Object:
  ~~      : List with 8 components with names
```

```

C D nlevels fl.dbase filter type bc date

$C and $D are LONG coefficient vectors

Created on : Mon Dec  4 22:27:11 2006
Type of decomposition: wavelet

summary(.):
-----
Levels: 3
Length of original: 8
Filter was: Haar wavelet
Boundary handling: periodic
Transform type: wavelet
Date: Mon Dec  4 22:27:11 2006

```

This output provides a wealth of information the details of which are explained in the `WaveThresh` Help page for `wd`.

2.1.3 Matrix representation

The example in the previous sections, and depicted in Figure 2.2, takes a vector input, $y = (1, 1, 7, 9, 2, 8, 8, 6)$, and produces a set of output coefficients that can be represented as a vector:

$$d = (21\sqrt{2}/2, 0, -\sqrt{2}, -3\sqrt{2}, \sqrt{2}, -7, -2, 3\sqrt{2}/2),$$

as calculated at the end of Example 2.2. Since the output has been computed from the input using a series of simple additions, subtractions, and constant scalings, it is no surprise that one can compute the output from the input using a matrix multiplication. Indeed, if one defines the matrix

$$W = \begin{bmatrix} \sqrt{2}/4 & \sqrt{2}/4 & \sqrt{2}/4 & \sqrt{2}/4 & \sqrt{2}/4 & \sqrt{2}/4 & \sqrt{2}/4 & \sqrt{2}/4 \\ 1/\sqrt{2} & -1/\sqrt{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/\sqrt{2} & -1/\sqrt{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/\sqrt{2} & -1/\sqrt{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/\sqrt{2} & -1/\sqrt{2} \\ 1/2 & 1/2 & -1/2 & -1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 1/2 & -1/2 & -1/2 \\ \sqrt{2}/4 & \sqrt{2}/4 & \sqrt{2}/4 & \sqrt{2}/4 & -\sqrt{2}/4 & -\sqrt{2}/4 & -\sqrt{2}/4 & -\sqrt{2}/4 \end{bmatrix}, \quad (2.17)$$

it is easy to check that $d = Wx$. It is instructive to see the structure of the previous equations contained within the matrix. Another point of interest is in the three ‘wavelet vectors’ at different scales that are ‘stored’ within the matrix, for example, $(1/\sqrt{2}, -1/\sqrt{2})$ in rows two through five, $(1/2, 1/2, -1/2, -1/2)$ in rows six and seven, and $(1, 1, 1, 1, -1, -1, -1, -1)/2\sqrt{2}$ in the last row.

The reader can check that W is an orthogonal matrix in that

$$W^T W = W W^T = I. \quad (2.18)$$

One can ‘see’ this by taking any row and multiplying component-wise by any other row and summing the result (the inner product of any two rows) and obtaining zero for different rows or one for the same row. (See Section B.1.3 for a definition of inner product.)

Since W is an orthogonal matrix it follows that

$$\|d\|^2 = d^T d = (Wy)^T Wy = y^T (W^T W) y = y^T y = \|y\|^2, \quad (2.19)$$

in other words, the length of the output vector d is the same as that of the input vector y and (2.19) is Parseval’s relation.

Not all wavelets are orthogonal and there are uses for non-orthogonal wavelets. For example, with non-orthogonal wavelets it is possible to adjust the relative resolution in time and scale (e.g. more time resolution whilst sacrificing frequency resolution), see Shensa (1996) for example. Most of the wavelets we will consider in this book are orthogonal, although sometimes we shall use collections which do not form orthogonal systems, for example, the non-decimated wavelet transform described in Section 2.9.

The operation $d = Wy$ carries out the wavelet transform using a matrix multiplication operation rather than the pyramidal technique we described earlier in Sections 2.1.1 and 2.1.2. If y was a vector containing a dyadic number, $n = 2^J$, of entries and hence W was of dimension $n \times n$, then the computational effort in performing the Wy operation is $\mathcal{O}(n^2)$ (the effort for multiplying the first row of W by y is n multiplications and $n - 1$ additions, roughly n ‘operations’. Repeating this for each of the n rows of W results in n^2 operations in total). See Section B.1.9 for a definition of \mathcal{O} .

The pyramidal algorithm of earlier sections produces the same wavelet coefficients as the matrix multiplication, but some consideration shows that it produces them in $\mathcal{O}(n)$ operations. Each coefficient is produced with one operation and coefficients are cascaded into each other in an efficient way so that the n coefficients that are produced take only $\mathcal{O}(n)$ operations. This result is quite remarkable and places the pyramid algorithm firmly into the class of ‘fast algorithms’ and capable of ‘real-time’ operation. As we will see later, the pyramid algorithm applies to a wide variety of wavelets, and hence one of the advertised benefits of wavelets is that they possess fast wavelet transforms.

The pyramidal wavelet transform is an example of a fast algorithm with calculations carefully organized to obtain efficient operation. It is also the case that only $\mathcal{O}(n)$ memory locations are required for the pyramidal execution as the two inputs can be completely replaced by a father and mother wavelet coefficient at each step, and then the father used in subsequent processing, as in Figure 2.2, for example. Another well-known example of a ‘fast algorithm’ is the fast Fourier transform (or FFT), which computes the discrete Fourier

transform in $\mathcal{O}(n \log n)$ operations. Wavelets have been promoted as being ‘faster than the FFT’, but one must realize that the discrete wavelet and Fourier transforms compute quite different transforms. Here, $\log n$ is small for even quite large n .

WaveThresh contains functionality to produce the matrix representations of various wavelet transforms. Although the key wavelet transformation functions in **WaveThresh**, like **wd**, use pyramidal algorithms for efficiency, it is sometimes useful to be able to obtain a wavelet transform matrix. To produce the matrix W shown in (2.17) use the command **GenW** as follows:

```
> W1 <-t(GenW(filter.number=1, family="DaubExPhase"))
```

Then examining **W1** gives

```
> W1
      [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.3535534 0.3535534 0.3535534 0.3535534 0.3535534
[2,] 0.7071068 -0.7071068 0.0000000 0.0000000 0.0000000
[3,] 0.0000000 0.0000000 0.7071068 -0.7071068 0.0000000
[4,] 0.0000000 0.0000000 0.0000000 0.0000000 0.7071068
[5,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
[6,] 0.5000000 0.5000000 -0.5000000 -0.5000000 0.0000000
[7,] 0.0000000 0.0000000 0.0000000 0.0000000 0.5000000
[8,] 0.3535534 0.3535534 0.3535534 0.3535534 -0.3535534
      [,6]      [,7]      [,8]
[1,] 0.3535534 0.3535534 0.3535534
[2,] 0.0000000 0.0000000 0.0000000
[3,] 0.0000000 0.0000000 0.0000000
[4,] -0.7071068 0.0000000 0.0000000
[5,] 0.0000000 0.7071068 -0.7071068
[6,] 0.0000000 0.0000000 0.0000000
[7,] 0.5000000 -0.5000000 -0.5000000
[8,] -0.3535534 -0.3535534 -0.3535534
```

which is the same as W given in (2.17) except in a rounded floating-point representation. Matrices for different n can be computed by changing the **n** argument to **GenW** and different wavelets can also be specified. See later for details on wavelet specification in **WaveThresh**.

One can verify the orthogonality of W using **WaveThresh**. For example:

```
> W1 %*% t(W1)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,] 1 0 0 0 0 0 0 0
[2,] 0 1 0 0 0 0 0 0
[3,] 0 0 1 0 0 0 0 0
[4,] 0 0 0 1 0 0 0 0
[5,] 0 0 0 0 1 0 0 0
[6,] 0 0 0 0 0 1 0 0
```

[7,]	0	0	0	0	0	0	1	0
[8,]	0	0	0	0	0	0	0	1

2.2 Haar Wavelets (on Functions)

2.2.1 Scaling and translation notation

First, we introduce a useful notation. Given any function $p(x)$, on $x \in \mathbb{R}$ say, we can form the (dyadically) scaled and translated version $p_{j,k}(x)$ defined by

$$p_{j,k}(x) = 2^{j/2} p(2^j x - k) \quad (2.20)$$

for all $x \in \mathbb{R}$ and where j, k are integers. Note that if the function $p(x)$ is ‘concentrated’ around zero, then $p_{j,k}(x)$ is concentrated around $2^{-j}k$. The $2^{j/2}$ factor ensures that $p_{j,k}(x)$ has the same norm as $p(x)$. In other words

$$\begin{aligned} \|p_{j,k}(x)\|^2 &= \int_{-\infty}^{\infty} p_{j,k}^2(x) dx \\ &= \int_{-\infty}^{\infty} 2^j p^2(2^j x - k) dx \\ &= \int_{-\infty}^{\infty} p^2(y) dy = \|p\|^2, \end{aligned} \quad (2.21)$$

where the substitution $y = 2^j x - k$ is made at (2.21).

2.2.2 Fine-scale approximations

More mathematical works introduce wavelets that operate on functions rather than discrete sequences. So, let us suppose that we have a function $f(x)$ defined on the interval $x \in [0, 1]$. It is perfectly possible to extend the following ideas to other intervals, the whole line \mathbb{R} , or d -dimensional Euclidean space.

Obviously, with a discrete sequence, the finest resolution that one can achieve is that of the sequence itself and, for Haar wavelets, the finest-scale wavelet coefficients involve pairs of these sequence values. For Haar, involving any more than pairs automatically means a larger-scale Haar wavelet. Also, recall that the Haar DWT progresses from finer to coarser scales.

With complete knowledge of a function, $f(x)$, one can, in principle, investigate it at any scale that one desires. So, typically, to initiate the Haar wavelet transform we need to choose a fixed finest scale from which to start. This fixed-scale consideration actually produces a discrete sequence, and further processing of *only the sequence* can produce all subsequent information at coarser scales (although it could, of course, be obtained from the function). We have not answered the question about how to obtain such a discrete sequence from a function. This is an important consideration and there are

many ways to do it; see Section 2.7.3 for two suggestions. However, until then suppose that such a sequence, derived from $f(x)$, is available.

In the discrete case the finest-scale wavelet coefficients involved subtracting one element from its neighbour in consecutive pairs of sequence values. For the Haar wavelet transform on functions we derive a similar notion which involves subtracting integrals of the function over consecutive pairs of intervals.

Another way of looking at this is to start with a fine-scale local averaging of the function. First define the *Haar father wavelet* at scale 2^J by $\phi(2^J x)$, where

$$\phi(x) = \begin{cases} 1, & x \in [0, 1], \\ 0 & \text{otherwise.} \end{cases} \quad (2.22)$$

Then define the finest-level (scale 2^J) father wavelet coefficients to be

$$c_{J,k} = \int_0^1 f(x) 2^{J/2} \phi(2^J x - k) dx, \quad (2.23)$$

or, using our scaling/translation notation, (2.23) becomes

$$c_{J,k} = \int_0^1 f(x) \phi_{J,k}(x) dx = \langle f, \phi_{J,k} \rangle, \quad (2.24)$$

the latter representation using an inner product notation.

At this point, it is worth explaining what the $c_{J,k}$ represent. To do this we should explore what the $\phi_{J,k}(x)$ functions look like. Using (2.20) and (2.22) it can be seen that

$$\phi_{J,k}(x) = \begin{cases} 2^{J/2} & x \in [2^{-J}k, 2^{-J}(k+1)], \\ 0 & \text{otherwise.} \end{cases} \quad (2.25)$$

That is, the function $\phi_{J,k}(x)$ is constant over the interval $I_{J,k} = [2^{-J}k, 2^{-J}(k+1)]$ and zero elsewhere. If the function $f(x)$ is defined on $[0, 1]$, then the range of k where $I_{J,k}$ overlaps $[0, 1]$ is from 0 to $2^J - 1$. Thus, the coefficient $c_{J,k}$ is just the integral of $f(x)$ on the interval $I_{J,k}$ (and proportional to the *local average* of $f(x)$ over the interval $I_{J,k}$).

In fact, the set of coefficients $\{c_{J,k}\}_{k=0}^{2^J-1}$ and the associated Haar father wavelets at that scale define an approximation $f_J(x)$ to $f(x)$ defined by

$$f_J(x) = \sum_{k=0}^{2^J-1} c_{J,k} \phi_{J,k}(x). \quad (2.26)$$

Figure 2.4 illustrates (2.26) for three different values of J . Plot a in Figure 2.4 shows a section of some real inductance plethysmography data collected by the Department of Anaesthesia at the Bristol Royal Infirmary which was first presented and described in Nason (1996). Essentially, this time series reflects changes in voltage, as a patient breathes, taken from a measuring device

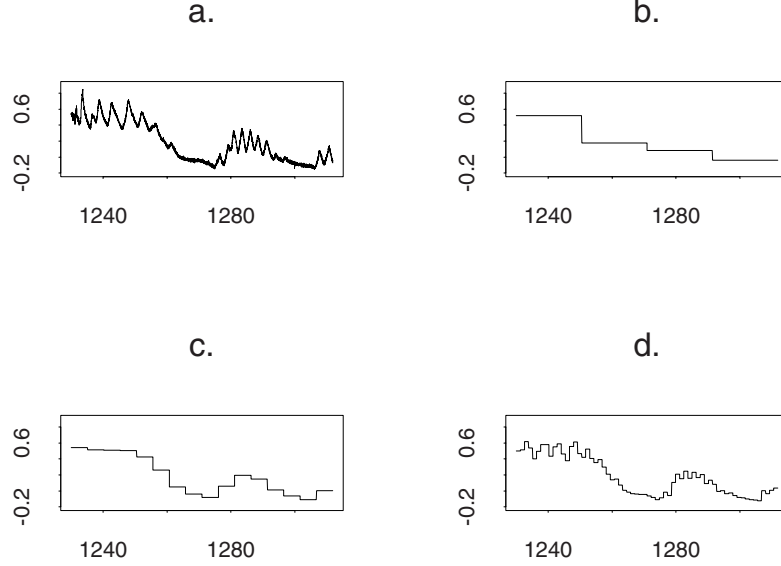


Fig. 2.4. Section of inductance plethysmography data from `WaveThresh` (a), (b) projected onto Haar father wavelet spaces $J = 2$, (c) $J = 4$, and (d) $J = 6$. In each plot the horizontal label is time in seconds, and the vertical axis is millivolts.

encapsulated in a belt worn by the patient. Plots b, c, and d in Figure 2.4 show Haar father wavelet approximations at levels $J = 2, 4$ and 6 . The original data sequence is of length 4096 , which corresponds to level $J = 12$. These Haar approximations are reminiscent of the *staircase approximation* useful (for example) in measure theory for proving, among other things, the monotone convergence theorem, see Williams (1991) or Kingman and Taylor (1966).

2.2.3 Computing coarser-scale c from-finer scale ones

Up to now, there is nothing special about J . We could compute the local average over these dyadic intervals $I_{j,k}$ for any j and k . An interesting situation occurs if one considers how to compute the integral of $f(x)$ over $I_{J-1,k}$ —that is the interval that is twice the width of $I_{J,k}$ and contains the intervals $I_{J,2k}$ and $I_{J,2k+1}$. It turns out that we can rewrite $c_{J-1,k}$ in terms of $c_{J,2k}$ and $c_{J,2k+1}$ as follows:

$$\begin{aligned}
c_{J-1,k} &= \int_{2^{-(J-1)}k}^{2^{-(J-1)}(k+1)} f(x) \phi_{J-1,k}(x) dx \\
&= 2^{-1/2} \int_{2^{-J}2k}^{2^{-J}(2k+2)} f(x) 2^{J/2} \phi(2^{J-1}x - k) dx \quad (2.27)
\end{aligned}$$

$$\begin{aligned}
&= 2^{-1/2} \left\{ \int_{2^{-J}2k}^{2^{-J}(2k+1)} f(x) 2^{J/2} \phi(2^J x - 2k) dx \right. \\
&\quad \left. + \int_{2^{-J}(2k+1)}^{2^{-J}(2k+2)} f(x) 2^{J/2} \phi(2^J x - (2k+1)) dx \right\} \quad (2.28)
\end{aligned}$$

$$\begin{aligned}
&= 2^{-1/2} \left\{ \int_{2^{-J}2k}^{2^{-J}(2k+1)} f(x) \phi_{J,2k}(x) dx \right. \\
&\quad \left. + \int_{2^{-J}(2k+1)}^{2^{-J}(2k+2)} f(x) \phi_{J,2k+1}(x) dx \right\} \\
&= 2^{-1/2} (c_{J,2k} + c_{J,2k+1}). \quad (2.29)
\end{aligned}$$

The key step in the above argument is the transition from the scale $J-1$ in (2.27) to scale J in (2.28). This step can happen because, for Haar wavelets,

$$\phi(y) = \phi(2y) + \phi(2y-1). \quad (2.30)$$

This equation is depicted graphically by Figure 2.5 and shows how $\phi(y)$ is exactly composed of two side-by-side rescalings of the original. Equation (2.30) is a special case of a more general relationship between father wavelets taken at adjacent dyadic scales. The formula for general wavelets is (2.47). It is an important equation and is known as the *dilation equation*, *two-scale relation*, or the *scaling equation* for father wavelets and it is an example of a *refinement equation*. Using this two-scale relation it is easy to see how (2.27) turns into (2.28) by setting $y = 2^{J-1}x - k$ and then we have

$$\phi(2^{J-1}x - k) = \phi(2^J x - 2k) + \phi(2^J x - 2k - 1). \quad (2.31)$$

A key point here is that to compute $c_{J-1,k}$ one does not necessarily need access to the function and apply the integration given in (2.24). One needs only the values $c_{J,2k}$ and $c_{J,2k+1}$ and to apply the simple Formula (2.29).

Moreover, if one wishes to compute values of $c_{J-2,\ell}$ right down to $c_{0,m}$ (for some ℓ, m), i.e., c at coarser scales still, then one needs only values of c from the next finest scale and the integration in (2.24) is not required. Of course, the computation in (2.29) is *precisely* the one in the discrete wavelet transform that we discussed in Section 2.1.2, and hence computation of *all* the coarser-scale father wavelet coefficients from a given scale 2^J is a fast and efficient algorithm.

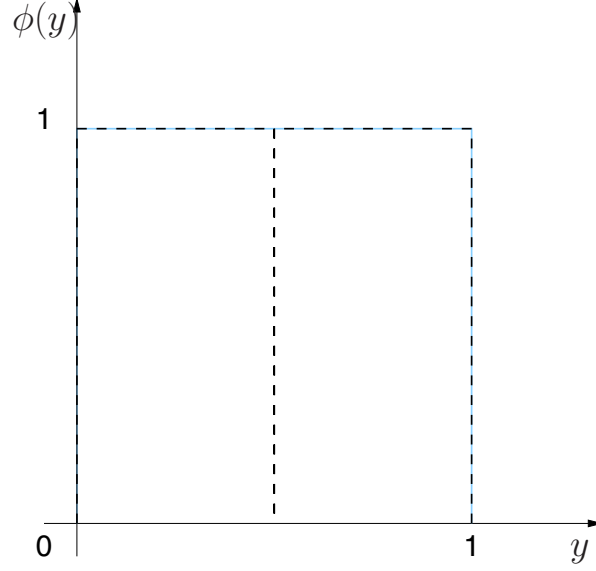


Fig. 2.5. Solid grey line is plot of $\phi(y)$ versus y . Two black dashed lines are $\phi(2y)$ and $\phi(2y - 1)$ to the left and right respectively.

2.2.4 The difference between scale approximations — wavelets

Suppose we have two Haar approximations of the same function but at two different scale levels. For definiteness suppose we have $f_0(x)$ and $f_1(x)$, the two coarsest approximations (actually approximation is probably not a good term here if the function f is at all wiggly since coarse representations will not resemble the original). The former, $f_0(x)$, is just a constant function $c_{00}\phi(x)$, a multiple of the father wavelet. The approximation $f_1(x)$ is of the form (2.26), which simplifies here to

$$f_1(x) = c_{1,0}\phi_{1,0}(x) + c_{1,1}\phi_{1,1}(x) = c_{1,0}2^{1/2}\phi(2x) + c_{1,1}2^{1/2}\phi(2x - 1). \quad (2.32)$$

What is the difference between $f_0(x)$ and $f_1(x)$? The difference is the ‘detail’ lost in going from a finer representation, f_1 , to a coarser one, f_0 . Mathematically:

$$\begin{aligned} f_1(x) - f_0(x) &= c_{0,0}\phi(x) - 2^{1/2} \{c_{1,0}\phi(2x) + c_{1,1}\phi(2x - 1)\} \\ &= c_{0,0} \{\phi(2x) + \phi(2x - 1)\} \\ &\quad - 2^{1/2} \{c_{1,0}\phi(2x) + c_{1,1}\phi(2x - 1)\}, \end{aligned} \quad (2.33)$$

using (2.30). Hence

$$f_1(x) - f_0(x) = (c_{0,0} - 2^{1/2}c_{1,0})\phi(2x) + (c_{0,0} - 2^{1/2}c_{1,1})\phi(2x - 1), \quad (2.34)$$

and since (2.29) implies $c_{0,0} = (c_{1,0} + c_{1,1})/\sqrt{2}$, we have

$$f_1(x) - f_0(x) = \{(c_{1,1} - c_{1,0})\phi(2x) + (c_{1,0} - c_{1,1})\phi(2x - 1)\} / \sqrt{2}. \quad (2.35)$$

Now suppose we define

$$d_{0,0} = (c_{1,1} - c_{1,0}) / \sqrt{2}, \quad (2.36)$$

so that the difference becomes

$$f_1(x) - f_0(x) = d_{0,0} \{\phi(2x) - \phi(2x - 1)\}. \quad (2.37)$$

At this point, it is useful to define the *Haar mother wavelet* defined by

$$\begin{aligned} \psi(x) &= \phi(2x) - \phi(2x - 1) \\ &= \begin{cases} 1 & \text{if } x \in [0, \frac{1}{2}), \\ -1 & \text{if } x \in [\frac{1}{2}, 1), \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \quad (2.38)$$

Then the difference between two approximations at scales one and zero is given by substituting $\psi(x)$ into (2.37), to obtain

$$f_1(x) - f_0(x) = d_{0,0}\psi(x). \quad (2.39)$$

Another way of looking at this is to rearrange (2.39) to obtain

$$f_1(x) = c_{0,0}\phi(x) + d_{0,0}\psi(x). \quad (2.40)$$

In other words, the finer approximation at level 1 can be obtained from the coarser approximation at level 0 *plus* the detail encapsulated in $d_{0,0}$. This can be generalized and works at all levels (simply imagine making everything described above operate at a finer scale and stacking those smaller mother and father wavelets next to each other) and one can obtain

$$\begin{aligned} f_{j+1}(x) &= f_j(x) + \sum_{k=0}^{2^j-1} d_{j,k}\psi_{j,k}(x) \\ &= \sum_{k=0}^{2^j-1} c_{j,k}\phi_{j,k}(x) + \sum_{k=0}^{2^j-1} d_{j,k}\psi_{j,k}(x). \end{aligned} \quad (2.41)$$

A Haar father wavelet approximation at finer scale $j+1$ can be obtained using the equivalent approximation at scale j plus the details stored in $\{d_{j,k}\}_{k=0}^{2^j-1}$.

2.2.5 Link between Haar wavelet transform and discrete version

Recall Formulae (2.29) and (2.36)

$$\begin{aligned} c_{0,0} &= (c_{1,1} + c_{1,0}) / \sqrt{2}, \\ d_{0,0} &= (c_{1,1} - c_{1,0}) / \sqrt{2}. \end{aligned} \quad (2.42)$$

These show that, given the finer sequence $(c_{1,0}, c_{1,1})$, it is possible to obtain the coarser-scale mother and father wavelet coefficients without reference to either the actual mother and father wavelet functions themselves (i.e., $\psi(x), \phi(x)$) or the original function $f(x)$. This again generalizes to all scales. Once the finest-scale coefficients $\{c_{J,k}\}_{k=0}^{2^J-1}$ are acquired, all the coarser-scale father and mother wavelet coefficients can be obtained using the discrete wavelet transform described in Section 2.1.2. Precise formulae for obtaining coarser scales from finer, for all scales, are given by (2.91).

2.2.6 The discrete wavelet transform coefficient structure

Given a sequence y_1, \dots, y_n , where $n = 2^J$, the *discrete wavelet transform* produces a vector of coefficients as described above consisting of the last, most coarse, father wavelet coefficient $c_{0,0}$ and the wavelet coefficients $d_{j,k}$ for $j = 0, \dots, J-1$ and $k = 0, \dots, 2^j - 1$.

2.2.7 Some discrete Haar wavelet transform examples

We now show two examples of computing and plotting Haar wavelet coefficients. The two functions we choose are the Blocks and Doppler test functions introduced by Donoho and Johnstone (1994b) and further discussed in Section 3.4. These functions can be produced using the `DJ.EX` function in `WaveThresh`. The plots of the Blocks and Doppler functions, and the wavelet coefficients are shown in Figures 2.6 and 2.7. The code that produced Figure 2.7 in `WaveThresh` was as follows:

```
> yy <- DJ.EX()$doppler

> yywd <- wd(yy, filter.number=1, family="DaubExPhase")

> x <- 1:1024

> oldpar <- par(mfrow=c(2,2))

> plot(x, yy, type="l", xlab="x", ylab="Doppler")
> plot(x, yy, type="l", xlab="x", ylab="Doppler")

> plot(yywd, main="")
> plot(yywd, scaling="by.level", main="")

> par(oldpar)
```

The code for Figure 2.6 is similar but `Blocks` replaces `Doppler`.

The coefficients plotted in the bottom rows of Figures 2.6 and 2.7 are the same in each picture. The difference is that the coefficients in the bottom

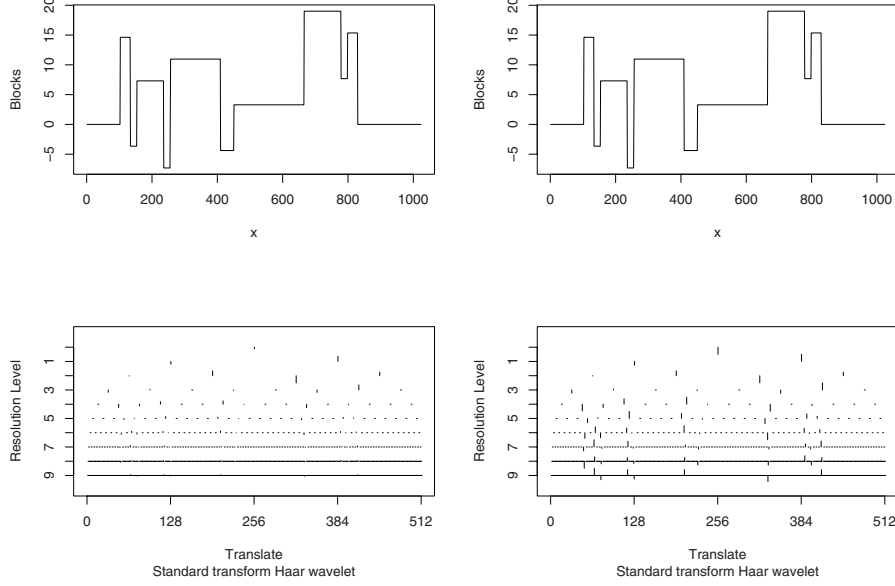


Fig. 2.6. *Top row: left and right: identical copies of the Blocks function. Bottom left: Haar discrete wavelet coefficients, $d_{j,k}$, of Blocks function (see discussion around Figure 2.3 for description of coefficient layout). All coefficients plotted to same scale and hence different coefficients are comparable. Bottom right: as left but with coefficients at each level plotted according to a scale that varies according to level. Thus, coefficient size at different levels cannot be compared. The ones at coarse levels are actually bigger. Produced by `f.wav13()`.*

left subplot of each are all plotted to the same scale, whereas the ones in the right are plotted with a different scale for each level (by scale here we mean the relative height of the small vertical lines that represent the coefficient values, not the resolution level, j , of the coefficients.) In both pictures it can be seen that as the level increases, to finer scales, the coefficients get progressively smaller (in absolute size). The decay rate of wavelet coefficients is mathematically related to the smoothness of the function under consideration, see Daubechies (1992, Section 2.9), Mallat and Hwang (1992), and Antoniadis and Gijbels (2002).

Three other features can be picked up from these wavelet coefficient plots. In Figure 2.6 the discontinuities in the Blocks function appear clearly as the large coefficients. Where there is a discontinuity a large coefficient appears at a nearby time location, with the exception of the coarser scales where there is not necessarily any coefficient located near to the discontinuities. The other point to note about Figure 2.6 is that many coefficients are *exactly* zero. This is because, in Haar terms, two neighbours, identical in value, were subtracted as in (2.42) to give an exact zero; and this happens at coarser scales too. One

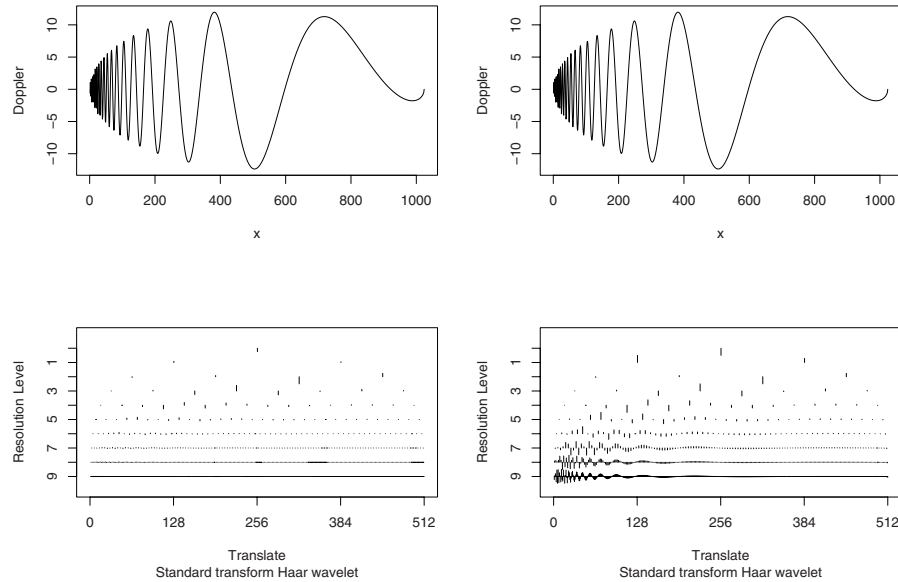


Fig. 2.7. As Figure 2.6 but applied to the Doppler function. Produced by `f.wav14()`.

can examine the coefficients more directly. For example, looking at the first 15 coefficients at level eight gives

```
> accessD(wd(DJ.EX())$blocks), level=8)[1:15]
[1] 9.471238e-17 -3.005645e-16 1.729031e-15 -1.773625e-16
[5] 1.149976e-16 -3.110585e-17 4.289763e-18 -1.270489e-19
[9] -1.362097e-20 0.000000e+00 0.000000e+00 0.000000e+00
[13] 0.000000e+00 0.000000e+00 0.000000e+00
```

Many of these are exactly zero. The ones that are extremely small (e.g. the first 9.47×10^{-17}) are only non-zero because the floating-point rounding error can be considered to be exactly zero for practical purposes. Figure 2.6 is a direct illustration of the *sparsity* of a wavelet representation of a function as few of the wavelet coefficients are non-zero. This turns out to happen for a wide range of signals decomposed with the right kind of wavelets. Such a property is of great use for compression purposes, see e.g. Taubman and Marcellin (2001), and for statistical nonparametric regression, which we will elaborate on in Chapter 3.

Finally, in Figure 2.7, in the bottom right subplot, the oscillatory nature of the Doppler signal clearly shows up in the coefficients, especially at the finer scales. In particular, it can be seen that there is a relationship between the local frequency of oscillation in the Doppler signal and where interesting behaviour in the wavelet coefficients turns up. Specifically, large variation in the fine-scale coefficients occurs at the beginning of the set of coefficients. The ‘fine-scale’ coefficients correspond to identification of ‘high-frequency’

information, and this ties in with the high frequencies in Doppler near the start of the signal. However, large variation in coarser-level coefficients starts much later, which ties in with the lower-frequency part of the Doppler signal. Hence, the coefficients here are a kind of ‘time-frequency’ display of the varying frequency information contained within the Doppler signal. At a given time-scale location, (j, k) , pair, the size of the coefficients gives information on how much oscillatory power there is *locally* at that scale. From such a plot one can clearly appreciate that there is a direct, but reciprocal, relationship between scale and frequency (e.g. small scale is equivalent to high frequency, and *vice versa*). The reader will not then be surprised to learn that these kinds of coefficient plots, and developments thereof, are useful for time series analysis and modelling. We will elaborate on this in Chapter 5.

2.3 Multiresolution Analysis

This section gives a brief and simple account of multiresolution analysis, which is the theoretical framework around which wavelets are built. This section will concentrate on introducing and explaining concepts. We shall quote some results without proof. Full, comprehensive, and mathematical accounts can be found in several texts such as Mallat (1989a,b), Meyer (1993b), and Daubechies (1988, 1992).

The previous sections were prescient in the sense that we began our discussion with a vector of data and, first, produced a set of detail coefficients and a set of smooth coefficients (by differencing and averaging in pairs). It can be appreciated that a function that has reasonable non-zero ‘fine-scale’ coefficients potentially possesses a more intricate structure than one whose ‘fine-scale’ coefficients are very small or zero. Further, one could envisage beginning with a low-resolution function and then progressively adding finer detail by inventing a new layer of detail coefficients and working back to the sequence that would have produced them (actually the inverse wavelet transform).

2.3.1 Multiresolution analysis

These kinds of considerations lead us on to ‘scale spaces’ of functions. Informally, we might define the space V_j as the space (collection) of functions with detail up to some finest scale of resolution. These spaces could possibly contain functions with less detail, but there would be some absolute maximum level of detail. Here larger j would indicate V_j containing functions with finer and finer scales. Hence, one would expect that if a function was in V_j , then it must also be in V_ℓ for $\ell > j$. Mathematically this is expressed as $V_j \subset V_\ell$ for $\ell > j$. This means that the spaces form a ladder:

$$\cdots \subset V_{-2} \subset V_{-1} \subset V_0 \subset V_1 \subset V_2 \subset \cdots . \quad (2.43)$$

As j becomes large and positive we include more and more functions of increasingly finer resolution. Eventually, as j tends to infinity we want to include all functions: mathematically this means that the union of all the V_j spaces is equivalent to the whole function space we are interested in. As j becomes large and negative we include fewer and fewer functions, and detail is progressively lost. As j tends to negative infinity the intersection of all the spaces is just the zero function.

The previous section using Haar wavelets was also instructive as it clearly showed that the detail added at level $j + 1$ is somehow twice as fine as the detail added at level j . Hence, this means that if $f(x)$ is a member of V_j , then $f(2x)$ (which is the same function but varies twice as rapidly as $f(x)$) should belong to V_{j+1} . We refer to this as *interscale linkage*. Also, if we take a function $f(x)$ and shift it along the line, say by an integral amount k , to form $f(x - k)$, then we do not change its level of resolution. Thus, if $f(x)$ is a member of V_0 , then so is $f(x - k)$.

Finally, we have not said much about the contents of any of these V_j spaces. Since the Haar father wavelet function $\phi(x)$ seemed to be the key function in the previous sections for building up functions at various levels of detail, we shall say that $\phi(x)$ is an element of V_0 and go further to assume that $\{\phi(x - k)\}_k$ is an orthonormal basis for V_0 . Hence, because of interscale linkage we can say that

$$\{\phi_{j,k}(x)\}_{k \in \mathbb{Z}} \text{ forms an orthonormal basis for } V_j. \quad (2.44)$$

The conditions listed above form the basis for a *multiresolution analysis* (MRA) of a space of functions. The challenge for wavelet design and development is to find such $\phi(x)$ that can satisfy these conditions for a MRA, and sometimes possess other properties, to be useful in various circumstances.

2.3.2 Projection notation

Daubechies (1988) introduced a projection operator P_j that projects a function into the space V_j . Since $\{\phi_{j,k}(x)\}_k$ is a basis for V_j , the projection can be written as

$$f_j(x) = \sum_{k \in \mathbb{Z}} c_{j,k} \phi_{j,k}(x) = P_j f \quad (2.45)$$

for some coefficients $\{c_{j,k}\}_k$. We saw this representation previously in (2.26) applying to just Haar wavelets. Here, it is valid for more general father wavelet functions, but the result is similar. Informally, $P_j f$ can be thought of as the ‘explanation’ of the function f using just the father wavelets at level j , or, in slightly more statistical terms, the ‘best fitting model’ of a linear combination of $\phi_{j,k}(x)$ to $f(x)$ (although this is a serious abuse of terminology because (2.45) is a mathematical representation and not a stochastic one).

The orthogonality of the basis means that the coefficients can be computed by

$$c_{j,k} = \int_{-\infty}^{\infty} f(x) \phi_{j,k}(x) dx = \langle f, \phi_{j,k} \rangle, \quad (2.46)$$

where \langle, \rangle is the usual inner product operator, see Appendix B.1.3.

2.3.3 The dilation equation and wavelet construction

From the ladder of subspaces in (2.43) space V_0 is a subspace of V_1 . Since $\{\phi_{1k}(x)\}$ is a basis for V_1 , and $\phi(x) \in V_0$, we must be able to write

$$\phi(x) = \sum_{n \in \mathbb{Z}} h_n \phi_{1n}(x). \quad (2.47)$$

This equation is called the *dilation equation*, and it is the generalization of (2.30). The dilation equation is fundamental in the theory of wavelets as its solution enables one to begin building a *general* MRA, not just for Haar wavelets.

However, for Haar wavelets, if one compares (2.47) and (2.30), one can see that the h_n for Haar must be $h_0 = h_1 = 1/\sqrt{2}$.

The dilation equation controls how the scaling functions relate to each other for two consecutive scales. In (2.30) the father wavelet can be constructed by adding two double-scale versions of itself placed next to each other. The general dilation equation in (2.47) says that $\phi(x)$ is constructed by a linear combination, h_n , of double-scale versions of itself. Daubechies (1992) provides a key result that establishes the existence and construction of the wavelets

Theorem 1 (Daubechies (1992), p.135) *If $\{V_j\}_{j \in \mathbb{Z}}$ with ϕ form a multiresolution analysis of $L^2(\mathbb{R})$, then there exists an associated orthonormal wavelet basis $\{\psi_{j,k}(x) : j, k \in \mathbb{Z}\}$ for $L^2(\mathbb{R})$ such that for $j \in \mathbb{Z}$*

$$P_{j+1}f = P_jf + \sum_k \langle f, \psi_{j,k} \rangle \psi_{j,k}(x). \quad (2.48)$$

One possibility for the construction of the wavelet $\psi(x)$ is

$$\hat{\psi}(\omega) = e^{i\omega/2} \overline{m_0(\omega/2 + \pi)} \hat{\phi}(\omega/2), \quad (2.49)$$

where $\hat{\psi}$ and $\hat{\phi}$ are the Fourier transforms of ψ and ϕ respectively and where

$$m_0(\omega) = \frac{1}{\sqrt{2}} \sum_n h_n e^{-in\omega}, \quad (2.50)$$

or equivalently

$$\psi(x) = \sum_n (-1)^{n-1} \overline{h_{-n-1}} \phi_{1,n}(x). \quad (2.51)$$

The function $\psi(x)$ is known as the mother wavelet. The coefficient in (2.51) is important as it expresses how the wavelet is to be constructed in terms of the (next) finer-scale father wavelet coefficients. This set of coefficients has its own notation:

$$g_n = (-1)^{n-1} h_{1-n}. \quad (2.52)$$

For Haar wavelets, using the values of h_n from before gives us $g_0 = -1/\sqrt{2}$ and $g_1 = 1/\sqrt{2}$.

Daubechies' Theorem 1 also makes clear that, from (2.48), the difference between two projections $(P_{j+1} - P_j)f$ can be expressed as a linear combination of wavelets. Indeed, the space characterized by the orthonormal basis of wavelets $\{\psi_{j,k}(x)\}_k$ is usually denoted W_j and characterizes the detail lost in going from P_{j+1} to P_j .

The representations given in (2.41) (Haar wavelets) and (2.48) (general wavelets) can be telescoped to give a fine-scale representation of a function:

$$f(x) = \sum_{k \in \mathbb{Z}} c_{j_0,k} \phi_{j_0,k}(x) + \sum_{j=j_0}^{\infty} \sum_{k \in \mathbb{Z}} d_{j,k} \psi_{j,k}(x). \quad (2.53)$$

This useful representation says that a general function $f(x)$ can be represented as a 'smooth' or 'kernel-like' part involving the $\phi_{j_0,k}$ and a set of detail representations $\sum_{k \in \mathbb{Z}} d_{j,k} \psi_{j,k}(x)$ accumulating information at a set of scales j ranging from j_0 to infinity. One can think of the first set of terms of (2.53), $\phi_{j_0,k}$, representing the 'average' or 'overall' level of function and the rest representing the detail. The $\phi(x)$ functions are not unlike many kernel functions often found in statistics—especially in kernel density estimation or kernel regression. However, the father wavelets, $\phi(x)$, tend to be used differently in that for wavelets the 'bandwidth' is 2^{j_0} with j_0 chosen on an integral scale, whereas the usual kernel bandwidth is chosen to be some positive real number. It is possible to mix the ideas of 'wavelet level' and 'kernel bandwidth' and come up with a more general representation, such as (4.16), that combines the strengths of kernels and wavelets, see Hall and Patil (1995), and Hall and Nason (1997). We will discuss this more in Section 4.7

2.4 Vanishing Moments

Wavelets can possess a number of vanishing moments: a function $\psi \in L^2(\mathbb{R})$ is said to have m *vanishing moments* if it satisfies

$$\int x^\ell \psi(x) dx = 0, \quad (2.54)$$

for $\ell = 0, \dots, m-1$ (under certain technical conditions).

Vanishing moments are important because if a wavelet has m vanishing moments, then all wavelet coefficients of any polynomial of degree m or less

will be exactly zero. Thus, if one has a function that is quite smooth and only interrupted by the occasional discontinuity or other singularity, then the wavelet coefficients ‘on the smooth parts’ will be very small or even zero if the behaviour at that point is polynomial of a certain order or less.

This property has important consequences for data compression. If the object to be compressed is mostly smooth, then the wavelet transform of the object will be sparse in the sense that many wavelet coefficients will be exactly zero (and hence their values do not need to be stored or compressed). The non-zero coefficients are those that encode the discontinuities or non-smooth parts. However, the idea is that for a ‘mostly smooth’ object there will be few non-zero coefficients to compress further.

Similar remarks apply to many statistical estimation problems. Taking the wavelet transform of an object is often advantageous as it results in a sparse representation of that object. Having only a few non-zero coefficients means that there are few coefficients that actually need to be estimated. In terms of information, it is better to have n pieces of data to estimate a *few* coefficients rather than n pieces of data to estimate n coefficients!

The `wvmoments` function in `WaveThresh` calculates the moments of wavelets numerically.

2.5 WaveThresh Wavelets (and What Some Look Like)

2.5.1 Daubechies’ compactly supported wavelets

One of the most important achievements in wavelet theory was the construction of orthogonal wavelets that were compactly supported but were smoother than Haar wavelets. Daubechies (1988) constructed such wavelets by an ingenious solution of the dilation equation (2.47) that resulted in a family of orthonormal wavelets (several families actually). Each member of each family is indexed by a number N , which refers to the number of vanishing moments (although in some references N denotes the length of h_n , which is twice the number of vanishing moments). `WaveThresh` contains two families of Daubechies wavelets which, in the package at least, are called the *least-asymmetric* and *extremal-phase* wavelets respectively. The least-asymmetric wavelets are sometimes known as *symmlets*. Real-valued compact orthonormal wavelets cannot be symmetric or antisymmetric (unless it is the Haar wavelet, see Daubechies (1992, Theorem 8.1.4)) and the least-asymmetric family is a choice that tries to minimize the degree of asymmetry. A deft discussion of the degree of asymmetry (or, more technically, departure from phase linearity) and the phase properties of wavelet filters can be found in Percival and Walden (2000, pp. 108–116). However, both compactly supported complex-valued and biorthogonal wavelets can be symmetric, see Sections 2.5.2 and 2.6.5.

The key quantity for performing fast wavelet transforms is the sequence of filter coefficients $\{h_n\}$. In **WaveThresh**, the **wd** function has access to the filter coefficients of various families through the **filter.select** function. In **WaveThresh**, the ‘extremal-phase’ family has vanishing moments ranging from one (Haar) to ten and the ‘least-asymmetric’ has them from four to ten. Wavelets in these families possess members with higher numbers of vanishing moments, but they are not stored within **WaveThresh**.

For example, to see the filter coefficients, $\{h_n\}$, for Haar wavelets, we examine the wavelet with **filter.number=1** and **family="DaubExPhase"** as follows:

```
> filter.select(filter.number=1, family="DaubExPhase")
$H
[1] 0.7071068 0.7071068

$G
NULL

$name
[1] "Haar wavelet"

$family
[1] "DaubExPhase"

$filter.number
[1] 1
```

The actual coefficients are stored in the **\$H** component as an approximation to the vector $(1/\sqrt{2}, 1/\sqrt{2})$, as noted before. As another example, we choose the wavelet with **filter.number=4** and **family="DaubLeAsymm"** by:

```
> filter.select(filter.number=4, family="DaubLeAsymm")
$H
[1] -0.07576571 -0.02963553 0.49761867 0.80373875
[5] 0.29785780 -0.09921954 -0.01260397 0.03222310

$G
NULL

$name
[1] "Daub cmpct on least asym N=4"

$family
[1] "DaubLeAsymm"

$filter.number
[1] 4
```

The length of the vector $\$H$ is eight, twice the number of vanishing moments.

It is easy to draw pictures of wavelets within **WaveThresh**. The following **draw.default** commands produced the pictures of wavelets and their scaling functions shown in Figure 2.8:

```
> oldpar<-par(mfrow=c(2,1))#To plot one fig above the other

> draw.default(filter.number=4, family="DaubExPhase",
+             enhance=FALSE, main="a.")

> draw.default(filter.number=4, family="DaubExPhase",
+             enhance=FALSE, scaling.function=TRUE, main="b.")

> par(oldpar)
```

The **draw.default** function is the default method for the generic **draw** function. The generic function, **draw()**, can be used directly on objects produced by other functions such as **wd** so as to produce a picture of the wavelet that resulted in a particular wavelet decomposition. The picture of the $N = 10$ ‘least-asymmetric’ wavelet shown in Figure 2.9 can be produced with similar commands, but using the arguments **filter.number=10** and **family="DaubExPhase"**.

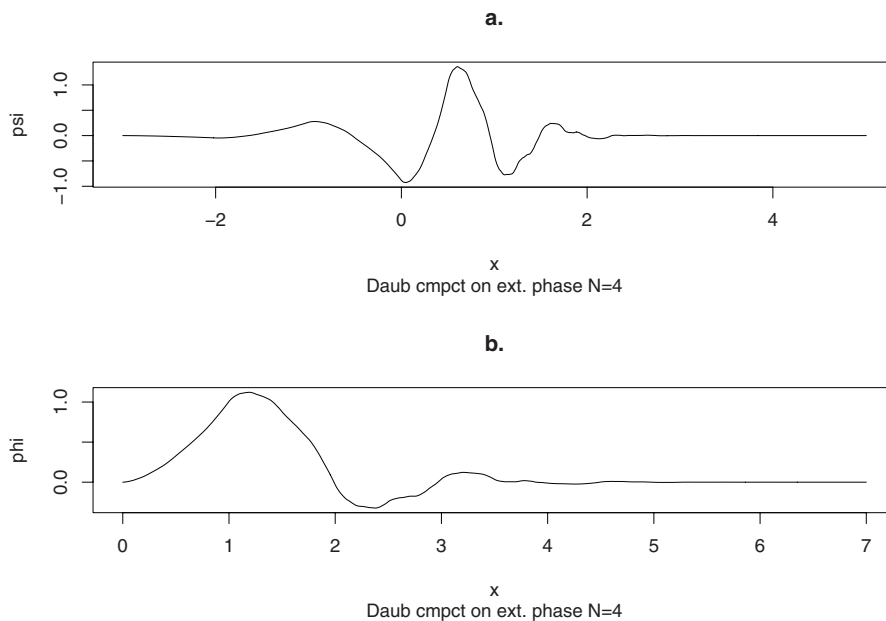


Fig. 2.8. Daubechies ‘extremal-phase’ wavelet with four vanishing moments: (a) mother wavelet and (b) father wavelet. Produced by **f.wav2()**.

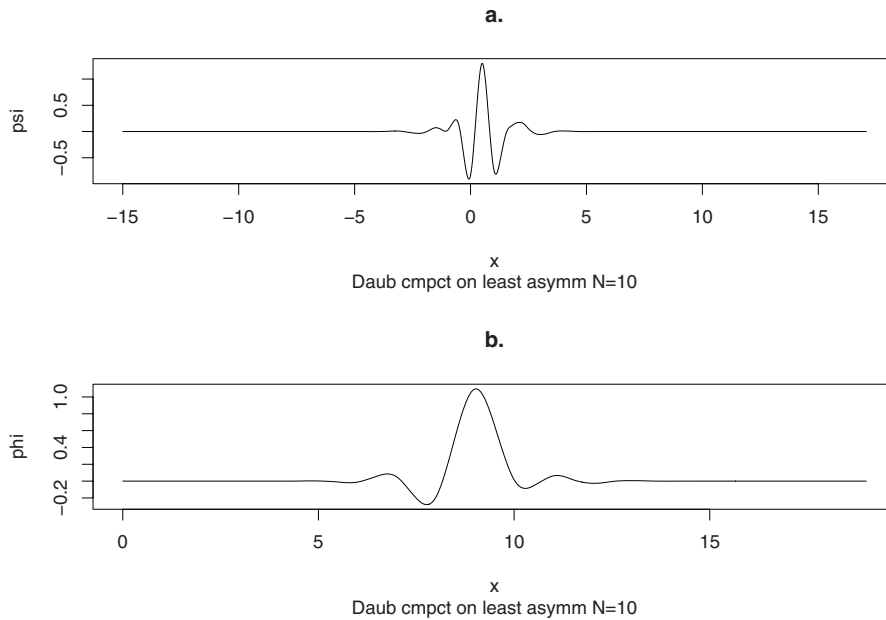


Fig. 2.9. Daubechies ‘least-asymmetric’ wavelet with ten vanishing moments: (a) mother wavelet, and (b) father wavelet. Produced by `f.wav3()`.

One can also use `GenW` to produce the wavelet transform matrix associated with a Daubechies’ wavelet. For example, for the Daubechies’ extremal-phase wavelet with two vanishing moments, the associated 8×8 matrix can be produced using the command

```
> W2 <- t(GenW(n=8, filter.number=3, family="DaubExPhase"))
```

and looks like

```
> W2
      [,1]      [,2]      [,3]      [,4]
[1,] 0.35355339 0.35355339 0.35355339 0.35355339
[2,] 0.80689151 -0.33267055 0.00000000 0.00000000
[3,] -0.13501102 -0.45987750 0.80689151 -0.33267055
[4,] 0.03522629 0.08544127 -0.13501102 -0.45987750
[5,] 0.00000000 0.00000000 0.03522629 0.08544127
[6,] 0.08019599 0.73683030 0.34431765 -0.32938217
[7,] -0.23056099 -0.04589588 -0.19395265 -0.36155225
[8,] -0.38061458 -0.02274768 0.21973837 0.55347099
      [,5]      [,6]      [,7]      [,8]
[1,] 0.35355339 0.35355339 0.35355339 0.35355339
[2,] 0.03522629 0.08544127 -0.13501102 -0.45987750
[3,] 0.00000000 0.00000000 0.03522629 0.08544127
```

```

[4,]  0.80689151 -0.33267055  0.00000000  0.00000000
[5,] -0.13501102 -0.45987750  0.80689151 -0.33267055
[6,] -0.23056099 -0.04589588 -0.19395265 -0.36155225
[7,]  0.08019599  0.73683030  0.34431765 -0.32938217
[8,]  0.38061458  0.02274768 -0.21973837 -0.55347099

```

2.5.2 Complex-valued Daubechies' wavelets

Complex-valued Daubechies' wavelets (CVDW) are described in detail by Lina and Mayrand (1995). For a given number of N vanishing moments there are 2^{N-1} possible solutions to the equations that define the Daubechies' wavelets, but not all are distinct. When $N = 3$, there are four solutions but only two are distinct: two give the real extremal-phase wavelet and the remaining two are a complex-valued conjugate pair. This $N = 3$ complex-valued wavelet was also derived and illustrated by Lawton (1993) via 'zero-flipping'. Lawton further noted that, apart from the Haar wavelet, the only compactly supported wavelets which are symmetric are CVDWs with an odd number of vanishing moments (other, asymmetric complex-valued wavelets are possible for higher N). The wavelet transform matrix, W , still exists for these complex-valued wavelets but the matrix is now unitary (the complex-valued version of orthogonal), i.e. it satisfies $W\bar{W}^T = \bar{W}^TW = I$, where $\bar{\cdot}$ denotes complex conjugation.

Currently neither **GenW** nor **draw** can produce matrices or pictures of complex-valued wavelets (although it would be not too difficult to modify them to do so). Figure 2.10 shows pictures of the $N = 3$ real- and complex-valued wavelets.

In **WaveThresh**, the complex-valued wavelet transform is carried out using the usual **wd** function but specifying the **family** option to be "LinaMayrand" and using a slightly different specification for the **filter.number** argument. For example, for these wavelets with five vanishing moments there are four different wavelets which can be used by supplying one of the numbers 5.1, 5.2, 5.3, or 5.4 as the **filter.select** argument. Many standard **WaveThresh** functions for processing wavelet coefficients are still available for complex-valued transforms. For example, the **plot** (or, more precisely, the **plot.wd** function) function by default plots the modulus of the complex-valued coefficient at each location. Arguments can be specified using the **aspect** argument to plot the real part, or imaginary part, or argument, or almost any real-valued function of the coefficient.

We show how complex-valued wavelets can be used for denoising purposes, including some **WaveThresh** examples, in Section 3.14.

2.6 Other Wavelets

There exist many other wavelets and associated multiresolution analyses. Here, we give a quick glimpse of the 'wavelet zoo'! We refer the reader to

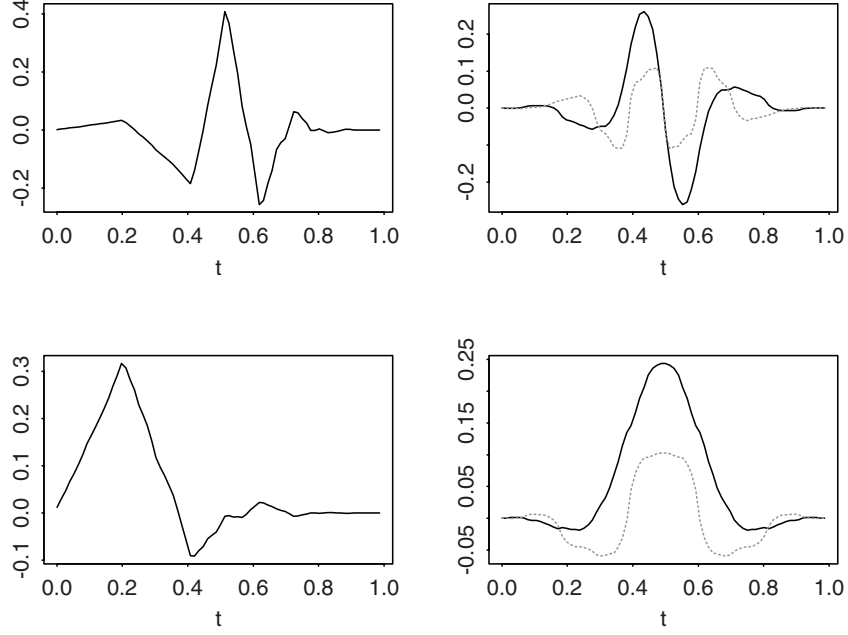


Fig. 2.10. The wavelets (*top*) and scaling functions (*bottom*) for Daubechies' wavelet $N = 3$ (*left*) and complex Daubechies' wavelet equivalent (*right*). The real part is drawn as a *solid black line* and the imaginary part as a *dotted line*.

the comprehensive books by Daubechies (1992) and Chui (1997) for further details on each of the following wavelets.

2.6.1 Shannon wavelet

The Haar scaling function, or father wavelet, given in (2.22) is localized in the x (time or space) domain in that it is compactly supported (i.e., is only non-zero on the interval $[0, 1]$). Its Fourier transform is given by

$$\hat{\phi}(\omega) = (2\pi)^{-1/2} e^{-i\omega/2} \text{sinc}(\omega/2), \quad (2.55)$$

where

$$\text{sinc}(\omega) = \begin{cases} \frac{\sin \omega}{\omega} & \text{for } \omega \neq 0, \\ 1 & \text{for } \omega = 0. \end{cases} \quad (2.56)$$

The sinc function is also known as the Shannon sampling function and is much used in signal processing.

Note that $\hat{\phi}(\omega)$ has a decay like $|\omega|^{-1}$. So the Haar mother wavelet is compactly supported in the x domain but with support over the whole of the real line in the frequency domain with a decay of $|\omega|^{-1}$.

For the Shannon wavelet, it is the other way around. The wavelet is compactly supported in the Fourier domain and has a decay like $|x|^{-1}$ in the time domain. Chui (1997, 3.1.5) defines the Shannon father wavelet to be

$$\phi_S(x) = \text{sinc}(\pi x). \quad (2.57)$$

The associated mother wavelet is given by Chui (1997, 4.2.4):

$$\psi_S(x) = \frac{\sin 2\pi x - \cos \pi x}{\pi(x - 1/2)}. \quad (2.58)$$

Both ϕ_S and ψ_S are supported over \mathbb{R} . The Fourier transform of ψ_S is given in Chui (1997, 4.2.6) by

$$\hat{\psi}_S(\omega) = -e^{-i\omega/2} I_{[-2\pi, -\pi) \cup (\pi, 2\pi]}(\omega), \quad (2.59)$$

in other words compactly supported on $(\pi, 2\pi]$ and its reflection in the origin.

The Shannon wavelet is not that different from the Littlewood–Paley wavelet given in Daubechies (1992, p. 115) by

$$\psi(x) = (\pi x)^{-1} (\sin 2\pi x - \sin \pi x). \quad (2.60)$$

In statistics the Shannon wavelet seems to be rarely used, certainly in practical applications. In a sense, it is the Fourier equivalent of the Haar wavelet, and hence certain paedagogical statements about wavelets could be made equally about Shannon as about Haar. However, since Haar is easier to convey in the time domain (and possibly because it is older), it is usually Haar that is used. However, the Shannon wavelet is occasionally used in statistics in a theoretical setting. For example, Chui (1997) remarks that Daubechies wavelets, with very high numbers of vanishing moments, ‘imitate’ the Shannon wavelet, which can be useful in understanding the behaviour of those higher-order wavelets in, for example, estimation of the spectral properties of wavelet-based stochastic processes, see Section 5.3.5.

2.6.2 Meyer wavelet

The Meyer wavelet, see Daubechies (1992, p. 116), has a similar Fourier transform to the Shannon wavelet but with the ‘sharp corners’ of its compact support purposely smoothed out, which results in a wavelet with faster decay. Meyer wavelets are used extensively in the analysis of statistical inverse problems. Such problems are often expressed as convolution problems which are considerably simplified by application of the Fourier transform, and the compact support of the Meyer wavelet in that domain provides computational benefits. See Kolaczyk (1994, 1996), who first introduced these ideas. For an important recent work that combines fast Fourier and wavelet transforms, and a comprehensive overview of the area see Johnstone et al. (2004). We discuss statistical inverse problems further in Section 4.9.

2.6.3 Spline wavelets

Chui (1997) provides a comprehensive introduction to wavelet theory and to spline wavelets. In particular, Chui (1997) defines the first-order cardinal B -spline by the Haar father wavelet defined in (2.22):

$$N_1(x) = \phi(x). \quad (2.61)$$

The m th-order cardinal B -spline, $m \geq 2$, is defined by the following recursive convolution:

$$\begin{aligned} N_m(x) &= \int_{-\infty}^{\infty} N_{m-1}(x-u)N_1(u) du \\ &= \int_0^1 N_{m-1}(x-u) du, \end{aligned} \quad (2.62)$$

in view of the definition of N_1 .

On taking Fourier transforms since convolutions turn into products, (2.63) turns into:

$$\hat{N}_m(\omega) = \hat{N}_{m-1}(\omega)\hat{N}_1(\omega) = \dots = \hat{N}_1^m(\omega). \quad (2.63)$$

What is the Fourier transform of $N_1(x)$? We could use (2.55), but it is more useful at this point to take the Fourier transform of both sides of the two-scale Equation (2.30), which in cardinal B -spline notation is

$$N_1(x) = N_1(2x) + N_1(2x-1), \quad (2.64)$$

and taking Fourier transforms gives

$$\begin{aligned} \hat{N}_1(\omega) &= (2\pi)^{-1/2} \left\{ \int N_1(2x)e^{-i\omega x} dx + \int N_1(2x-1)e^{-i\omega x} dx \right\} \\ &= \frac{1}{2}(2\pi)^{-1/2} \left\{ \int N_1(y)e^{-iy\omega/2} dy + \int N_1(y)e^{-i(y+1)\omega/2} dy \right\} \\ &= \frac{1}{2}(1 + e^{-i\omega/2})\hat{N}_1(\omega/2), \end{aligned} \quad (2.65)$$

by substituting $y = 2x$ and $y = 2x-1$ in the integrals on line 1 of (2.65). Hence using (2.63) and (2.65) together implies that

$$\hat{N}_m(\omega) = \left(\frac{1 + e^{-i\omega/2}}{2} \right)^m \hat{N}_m(\omega/2). \quad (2.66)$$

Chui (1997) shows that (2.66) translates to the following formula in the x domain:

$$N_m(x) = 2^{-m+1} \sum_{k=0}^m \binom{m}{k} N_m(2x-k), \quad (2.67)$$

and this formula defines the two-scale relation for the m th-order cardinal B -spline. For example, for $m = 2$ the two-scale relation (2.67) becomes

$$N_2(x) = 2^{-1} \{N_2(2x) + 2N_2(2x - 1) + N_2(2x - 2)\}. \quad (2.68)$$

In view of (2.63) the cardinal B -splines are compactly supported and, using two-scale relations such as (2.67), they can be used as scaling functions to start a multiresolution analysis. The m th-order cardinal spline B -wavelet can be generated by

$$\psi_m(x) = \sum_{k=0}^{3m-2} q_k N_m(2x - k), \quad (2.69)$$

where

$$q_k = \frac{(-1)^k}{2^{m-1}} \sum_{\ell=0}^m \binom{m}{\ell} N_{2m}(k - \ell + 1), \quad (2.70)$$

see formulae (5.2.25) and (5.2.24) respectively in Chui (1997). Hence since the cardinal B -splines are compactly supported, the cardinal spline B -wavelet is also compactly supported. However, these spline wavelets are not orthogonal functions, which makes them less attractive for some applications such as nonparametric regression.

The cardinal spline B -wavelets can be orthogonalized according to an ‘orthogonalization trick’, see Daubechies (1992, p. 147) for details. These orthogonalized wavelets are known as the Battle–Lemarié wavelets. Strömberg wavelets are also a kind of orthogonal spline wavelet with similar properties to Battle–Lemarié wavelets, see Daubechies (1992, p. 116) or Chui (1997, p. 75) for further details.

2.6.4 Coiflets

Coiflets have similar properties to Daubechies wavelets except the scaling function is also chosen so that it has vanishing moments. In other words, the scaling function satisfies (2.54) with ϕ instead of ψ and for moments $\ell = 1, \dots, m$. Note $\ell = 0$ is not possible since for all scaling functions we must have $\int \phi(x) dx \neq 0$. Coiflets are named in honour of R. Coifman, who first requested them, see Daubechies (1992, Section 8.2) for more details.

2.6.5 Biorthogonal wavelets

In what we have seen up to now a wavelet, $\psi(x)$, typically performs both an *analysis* and a *synthesis* role. The analysis role means that the wavelet coefficients of a function $f(x)$ can be discovered by

$$d_{j,k} = \int f(x) \psi_{j,k}(x) dx. \quad (2.71)$$

Further, the same wavelet can be used to form the *synthesis* of the function as in (2.41). With *biorthogonal wavelets* two functions are used, the analyzing wavelet, $\psi(x)$, and its *dual*, the synthesizing wavelet $\tilde{\psi}(x)$. In regular Euclidean space with an orthogonal basis, one can read off the coefficients of the components of a vector simply by looking at the projection onto the (orthogonal) basis elements. For a non-orthogonal basis, one constructs a dual basis with each dual basis element orthogonal to a corresponding original basis element and the projection onto the dual can ‘read off’ the coefficients necessary for synthesis. Put mathematically this means that $\langle \psi_{j,k}, \tilde{\psi}_{\ell,m} \rangle = \delta_{j,\ell} \delta_{k,m}$, see Jawerth and Sweldens (1994).

Filtering systems (filter banks) predating wavelets were known in the signal processing literature, see, e.g., Nguyen and Vaidyanathan (1989), and Vetterli and Herley (1992). For a tutorial introduction to filter banks see Vaidyanathan (1990). The connections to wavelets and development of compactly supported wavelets are described by Cohen et al. (1992).

2.7 The General (Fast) Discrete Wavelet Transform

2.7.1 The forward transform

In Section 2.2.3 we explained how to compute coarser-scale Haar wavelet coefficients. In this section, we will explain how this works for more general wavelet coefficients defined in Section 2.3.

Suppose we have a function $f(x) \in L^2(\mathbb{R})$. How can we obtain coarser-level father wavelet coefficients from finer ones, say, level $J-1$ from J ? To see this, recall that the father wavelet coefficients of $f(x)$ at level $J-1$ are given by

$$c_{J-1,k} = \int_{\mathbb{R}} f(x) \phi_{J-1,k}(x) dx, \quad (2.72)$$

since $\{\phi_{J-1,k}(x)\}_k$ is an orthonormal basis for V_{J-1} .

We now need an expression for $\phi_{J-1,k}(x)$ in terms of $\phi_{J,\ell}(x)$ and use the dilation equation (2.47) for this:

$$\begin{aligned} \phi_{J-1,k}(x) &= 2^{(J-1)/2} \phi(2^{J-1}x - k) \\ &= 2^{(J-1)/2} \sum_n h_n \phi_{1,n}(2^{J-1}x - k) \\ &= 2^{(J-1)/2} \sum_n h_n 2^{1/2} \phi\{2(2^{J-1}x - k) - n\} \\ &= 2^{J/2} \sum_n h_n \phi(2^Jx - 2k - n) \\ &= \sum_n h_n \phi_{J,n+2k}(x). \end{aligned} \quad (2.73)$$

In fact, (2.47) is a special case of (2.73) with $J = 1$ and $k = 0$.

Now let us substitute (2.73) into (2.72) to obtain

$$\begin{aligned}
 c_{J-1,k} &= \int_{\mathbb{R}} f(x) \sum_n h_n \phi_{J,n+2k}(x) dx \\
 &= \sum_n h_n \int_{\mathbb{R}} f(x) \phi_{J,n+2k}(x) dx \\
 &= \sum_n h_n c_{J,n+2k},
 \end{aligned} \tag{2.74}$$

or, with a little rearrangement, in its usual form:

$$c_{J-1,k} = \sum_n h_{n-2k} c_{J,n}. \tag{2.75}$$

An equation to obtain *wavelet* coefficients at scale $J-1$ from father wavelet coefficients at scale J can be developed in a similar way. Instead of using the scaling function dilation equation, we use the analogous Equation (2.51) in (2.73), and then after some working we obtain

$$d_{J-1,k} = \sum_n g_{n-2k} c_{J,n}. \tag{2.76}$$

Note that (2.75) and (2.76) hold for any scale j replacing J for $j = 1, \dots, J$.

2.7.2 Filtering, dyadic decimation, downsampling

The operations described by Equations (2.75) and (2.76) can be thought of in another way. For example, we can achieve the same result as (2.75) by first *filtering* the sequence $\{c_{J,n}\}$ with the filter $\{h_n\}$ to obtain

$$c_{J-1,k}^* = \sum_n h_{n-k} c_{J,n}. \tag{2.77}$$

This is a standard convolution operation. Then we could pick ‘every other one’ to obtain $c_{J-1,k} = c_{J-1,2k}^*$. This latter operation is known as *dyadic decimation* or *downsampling* by an integer factor of 2. Here, we borrow the notation of Nason and Silverman (1995) and define the (even) dyadic decimation operator \mathcal{D}_0 by

$$(\mathcal{D}_0 x)_\ell = x_{2\ell}, \tag{2.78}$$

for some sequence $\{x_i\}$.

Hence the operations described by Formulae (2.75) and (2.76) can be written more succinctly as

$$c_{J-1} = \mathcal{D}_0 \mathcal{H} c_J \text{ and } d_{J-1} = \mathcal{D}_0 \mathcal{G} c_J, \tag{2.79}$$

where \mathcal{H} and \mathcal{G} denote the regular filtering operation, e.g. (2.77). In (2.79) we have denoted the input and outputs to these operations using a more efficient vector notation, c_J, c_{J-1}, d_{J-1} , rather than sequences.

Nason and Silverman (1995) note that the *whole* set of discrete wavelet transform (coefficients) can be expressed as

$$d_j = \mathcal{D}_0 \mathcal{G} (\mathcal{D}_0 \mathcal{H})^{J-j-1} c_J, \quad (2.80)$$

for $j = 0, \dots, J-1$ and similarly for the father wavelet coefficients:

$$c_j = (\mathcal{D}_0 \mathcal{H})^{J-j} c_J, \quad (2.81)$$

for the same range of j . Remember d_j and c_j here are vectors of length 2^j (for periodized wavelet transforms).

This vector/operator notation is useful, particularly because the computational units $\mathcal{D}_0 \mathcal{G}$ and $\mathcal{D}_0 \mathcal{H}$ can be compartmentalized in a computer program for easy deployment and robust checking. However, the notation is mathematically liberating and of great use when developing more complex algorithms such as the non-decimated wavelet transform, the wavelet packet transform, or combinations of these. Specifically, one might have wondered why we chose ‘even’ dyadic decimation, i.e. picked out each even element x_{2j} rather than the odd indexed ones, x_{2j+1} . This is a good question, and the ‘solution’ is the non-decimated transform which we describe in Section 2.9. Wavelet packets we describe in Section 2.11 and non-decimated wavelet packets in Section 2.12.

2.7.3 Obtaining the initial fine-scale father coefficients

In much of the above, and more precisely at the beginning of Section 2.7.1, we mentioned several times that the wavelet transform is initiated from a set of ‘finest-scale’ father wavelet coefficients, $\{c_{J,k}\}_{k \in \mathbb{Z}}$. Where do these mysterious finest-scale coefficients come from? We outline two approaches.

A *deterministic approach* is described in Daubechies (1992, Chapter 5, Note 12). Suppose the information about our function comes to us as samples, i.e. our information about a function f comes to us in terms of function values at a set of integers: $f(n)$, $n \in \mathbb{Z}$. Suppose that we wish to find the father coefficients of that $f \in V_0$ (‘information’ orthogonal to V_0 cannot be recovered; whether *your* actual f completely lies in V_0 is another matter).

Now, since $f \in V_0$, we have

$$f(x) = \sum_k \langle f, \phi_{0,k} \rangle \phi_{0,k}(x), \quad (2.82)$$

where $\langle \cdot, \cdot \rangle$ indicates the inner product, again see Appendix B.1.3. Therefore

$$f(n) = \sum_k \langle f, \phi_{0,k} \rangle \phi(n-k). \quad (2.83)$$

Applying the discrete Fourier transform (Appendix B.1.7) to both sides of (2.83) gives

$$\begin{aligned}
\sum_n f(n)e^{-i\omega n} &= \sum_k \langle f, \phi_{0,k} \rangle \sum_n \phi(n-k)e^{-i\omega n} \\
&= \sum_k \langle f, \phi_{0,k} \rangle \sum_m \phi(m)e^{-i\omega(m+k)} \\
&= \left\{ \sum_k \langle f, \phi_{0,k} \rangle e^{-i\omega k} \right\} \left\{ \sum_m \phi(m)e^{-i\omega m} \right\} \\
&= \Phi(\omega) \sum_k \langle f, \phi_{0,k} \rangle e^{-i\omega k}, \tag{2.84}
\end{aligned}$$

where $\Phi(\omega) = \sum_m \phi(m)e^{-i\omega m}$ is the discrete Fourier transform of $\{\phi_m(x)\}_m$.

Our objective is to obtain the coefficients $c_{0k} = \langle f, \phi_{0,k} \rangle$. To do this, rearrange (2.84) and introduce notation $F(\omega)$, to obtain

$$\sum_k \langle f, \phi_{0,k} \rangle e^{-i\omega k} = \Phi^{-1}(\omega) \sum_n f(n)e^{-i\omega n} = F(\omega). \tag{2.85}$$

Hence taking the inverse Fourier transform of (2.85) gives

$$\begin{aligned}
\langle f, \phi_{0,k} \rangle &= (2\pi)^{-1} \int_0^{2\pi} F(\omega) e^{i\omega k} d\omega \\
&= (2\pi)^{-1} \int_0^{2\pi} \sum_n f(n) e^{-i\omega(n-k)} \Phi^{-1}(\omega) d\omega \\
&= \sum_n f(n) (2\pi)^{-1} \int_0^{2\pi} e^{-i\omega(n-k)} \Phi^{-1}(\omega) d\omega \\
&= \sum_n a_{n-k} f(n), \tag{2.86}
\end{aligned}$$

where $a_m = (2\pi)^{-1} \int_0^{2\pi} e^{-i\omega m} \Phi^{-1}(\omega) d\omega$.

For example, for the Daubechies' 'extremal-phase' wavelet with two vanishing moments we have $\phi(0) \approx 0.01$, $\phi(1) \approx 1.36$, $\phi(2) \approx -0.36$, and $\phi(n) = 0, n \neq 0, 1, 2$. This can be checked by drawing a picture of this scaling function. For example, using the `WaveThresh` function:

```
> draw.default(filter.number=2, family="DaubExPhase",
+   scaling.function=TRUE)
```

Hence denoting $\phi(n)$ by ϕ_n to save space

$$\Phi(\omega) = \sum_m \phi(m)e^{-i\omega m} \approx \phi_0 + \phi_1 e^{-i\omega} + \phi_2 e^{-2i\omega}, \tag{2.87}$$

and

$$\begin{aligned}
|\Phi(\omega)|^2 &= \phi_0^2 + \phi_1^2 + \phi_2^2 + 2(\phi_0\phi_1 + \phi_1\phi_2)\cos\omega + 2\phi_0\phi_2\cos(2\omega) \\
&= \phi_1^2 + 2\phi_1\phi_2\cos\omega,
\end{aligned} \tag{2.88}$$

which is *very* approximately a constant. Here, $a_m = \text{const} \times \delta_{0,m}$ for some constant and $\langle f, \phi_{0,k} \rangle \approx \text{const} \times f(k)$. So, one might *claim* that one only needs to initialize the wavelet transform using the original function samples. However, it can be seen that the above results in a massive approximation, which is prone to error. Taking the V_0 scaling function coefficients to be the samples is known as the ‘wavelet crime’, as coined by Strang and Nguyen (1996). The crime can properly be avoided by computing $\Phi(\omega)$ and using more accurate a_m .

A stochastic approach. A somewhat more familiar approach can be adopted in statistical situations. For example, in density estimation, one might be interested in collecting independent observations, X_1, \dots, X_n , from some, unknown, probability density $f(x)$. The scaling function coefficients of f are given by

$$\langle f, \phi_{j,k} \rangle = \int f(x)\phi_{j,k}(x) dx = \mathbb{E}[\phi_{j,k}(X)]. \tag{2.89}$$

Then an unbiased estimator of $\langle f, \phi_{j,k} \rangle$ is given by the equivalent sample quantity, i.e.

$$\widehat{\langle f, \phi_{j,k} \rangle} = n^{-1} \sum_{i=1}^n \phi_{j,k}(X_i). \tag{2.90}$$

The values $\phi_{j,k}(X_i)$ can be computed efficiently using the algorithm given in Daubechies and Lagarias (1992). Further details on this algorithm and its use in density estimation can be found in Herrick et al. (2001).

2.7.4 Inverse discrete wavelet transform

In Section 2.2.5, Formula (2.42) showed how to obtain coarser father and mother wavelet coefficients from father coefficients at the next finer scale. These formulae are more usually written for a general scale as something like

$$\begin{aligned}
c_{j-1,k} &= (c_{j,2k} + c_{j,2k+1})/\sqrt{2}, \\
d_{j-1,k} &= (c_{j,2k} - c_{j,2k+1})/\sqrt{2}.
\end{aligned} \tag{2.91}$$

Now suppose our problem is how to invert this operation: i.e. given the $c_{j-1,k}, d_{j-1,k}$, how do we obtain the $c_{j,2k}$ and $c_{j,2k+1}$? One can solve the equations in (2.91) and obtain the following formulae:

$$\begin{aligned}
c_{j,2k} &= (c_{j-1,k} + d_{j-1,k})/\sqrt{2}, \\
c_{j,2k+1} &= (c_{j-1,k} - d_{j-1,k})/\sqrt{2}.
\end{aligned} \tag{2.92}$$

The interesting thing about (2.92) is that the form of the inverse relationship is *exactly* the same as the forward relationship in (2.91).

For general wavelets Mallat (1989b) shows that the inversion relation is given by

$$c_{j,n} = \sum_k h_{n-2k} c_{j-1,k} + \sum_k g_{n-2k} d_{j-1,k}, \quad (2.93)$$

where h_n, g_n are known as the quadrature mirror filters defined by (2.47) and (2.52). Again, the filters used for computing the inverse transform are the same as those that computed the forward one.

Earlier, in Section 2.1.3, Equation (2.17) displayed the matrix representation of the Haar wavelet transform. We also remarked in that section that the matrix was orthogonal in that $W^T W = I$. This implies that the inverse transform to the Haar wavelet transform is just W^T . For example, the transpose of (2.17) is

$$W^T = \begin{bmatrix} \sqrt{2}/4 & 1/\sqrt{2} & 0 & 0 & 0 & 1/2 & 0 & \sqrt{2}/4 \\ \sqrt{2}/4 & -1/\sqrt{2} & 0 & 0 & 0 & 1/2 & 0 & \sqrt{2}/4 \\ \sqrt{2}/4 & 0 & 1/\sqrt{2} & 0 & 0 & -1/2 & 0 & \sqrt{2}/4 \\ \sqrt{2}/4 & 0 & -1/\sqrt{2} & 0 & 0 & -1/2 & 0 & \sqrt{2}/4 \\ \sqrt{2}/4 & 0 & 0 & 1/\sqrt{2} & 0 & 0 & 1/2 & -\sqrt{2}/4 \\ \sqrt{2}/4 & 0 & 0 & -1/\sqrt{2} & 0 & 0 & 1/2 & -\sqrt{2}/4 \\ \sqrt{2}/4 & 0 & 0 & 0 & 1/\sqrt{2} & 0 & -1/2 & -\sqrt{2}/4 \\ \sqrt{2}/4 & 0 & 0 & 0 & -1/\sqrt{2} & 0 & -1/2 & -\sqrt{2}/4 \end{bmatrix}. \quad (2.94)$$

Example 2.4. Let us continue Example 2.3, where we computed the discrete Haar wavelet transform on vector \mathbf{y} to produce the object \mathbf{ywd} . The inverse transform is performed using the `wr` function as follows:

```
> yinv <- wr(ywd)
```

and if we examine the contents of the inverse transformed vector we obtain

```
> yinv
[1] 1 1 7 9 2 8 8 6
```

So \mathbf{yinv} is precisely the same as \mathbf{y} , which is exactly what we planned.

2.8 Boundary Conditions

One nice feature of Haar wavelets is that one does not need to think about computing coefficients near ‘boundaries’. If one has a dyadic sequence, then the Haar filters transform that sequence in pairs to produce another dyadic sequence, which can then be processed again in the same way. For more general Daubechies wavelets, one has to treat the issue of boundaries more carefully.

For example, let us examine again the simplest compactly supported Daubechies’ wavelet (apart from Haar). The detail filter associated with

this wavelet has four elements, which we have already denoted in (2.52) by $\{g_k\}_{k=0}^3$. (It is, approximately, $(0.482, -0.837, 0.224, -0.129)$, and can be produced by the `filter.select` function in `WaveThresh`.)

Suppose we have the dyadic data vector x_0, \dots, x_{31} . Then the ‘first’ coefficient will be $\sum_{k=0}^3 g_k x_k$. Due to even dyadic decimation the next coefficient will be $\sum_{k=0}^3 g_k x_{k+2}$. The operation can be viewed as a window of four g_k consecutive coefficients initially coinciding with the first four elements of $\{x_k\}$ but then skipping two elements ‘to the right’ each time.

However, one could also wonder what happens when the window also skips to the left, i.e. $\sum_{k=0}^3 g_k x_{k-2}$. Initially, this seems promising as x_0, x_1 are covered when $k = 2, 3$. However, what are x_{-2}, x_{-1} when $k = 0, 1$? Although it probably does not seem to matter very much here as we are only ‘missing’ two observations (x_{-1}, x_{-2}) , the problem becomes more ‘serious’ for longer filters corresponding to smoother Daubechies’ wavelets with a larger number of vanishing moments (for example, with ten vanishing moments the filter is of length 20. So, again we could have x_{-1}, x_{-2} ‘missing’ but still could potentially make use of the information in x_0, \dots, x_{17}).

An obvious way of coping with this boundary ‘problem’ is to artificially extend the boundary in some way. In the examples discussed above this consists of artificially providing the ‘missing’ observations. `WaveThresh` implements two types of boundary extension for some routines: periodic and symmetric end reflection. The function `wd` possesses both options, but many other functions just have the periodic extension. Periodic extension is sometimes also known as being equivalent to using periodized wavelets (for the discrete case).

For a function f defined on the compact interval, say, $[0, 1]$, then periodic extension assumes that $f(-x) = f(1 - x)$. That is information to the ‘left’ of the domain of definition is actually obtained from the right-hand end of the function. The formula works for both ends of the function, i.e., $f(-0.2) = f(0.8)$ and $f(1.2) = f(0.2)$. Symmetric end reflection assumes $f(-x) = f(x)$ and $f(1 + x) = f(1 - x)$ for $x \in [0, 1]$. To give an example, in the example above x_{-1}, x_{-2} would actually be set to x_{31} and x_{30} respectively for periodic extension and x_1 and x_2 respectively for symmetric end reflection. In `WaveThresh`, these two options are selected using the `bc="periodic"` or `bc="symmetric"` arguments.

In the above we have talked about adapting the data so as to handle boundaries. The other possibility is to leave the data alone and to modify the wavelets themselves. In terms of mathematical wavelets the problem of boundaries occurs when the wavelet, at a coarse scale, is too big, or too big and too near the edge (or over the edge) compared with the interval that the data are defined upon. One solution is to modify the wavelets that overlap the edge by replacing them with special ‘edge’ wavelets that retain the orthogonality of the system.

The solutions above either wrap the function around on itself (as much as is necessary) for periodized wavelets or reflect the function in its boundaries. The other possibility is to modify the wavelet so that it always remains on

the original data domain of definition. This wavelet modification underlies the procedure known as ‘wavelets on the interval’ due to Cohen et al. (1993). This procedure produces wavelet coefficients at progressively coarser scales but does not borrow information from periodization or reflection. In **WaveThresh** the ‘wavelets on the interval’ method is implemented within the basic wavelet transform function, **wd**, using the **bc="interval"** option.

2.9 Non-decimated Wavelets

2.9.1 The ϵ -decimated wavelet transform

Section 2.7.2 described the basic forward discrete wavelet transform step as a filtering by \mathcal{H} followed by a dyadic decimation step \mathcal{D}_0 . Recall that the dyadic decimation step, \mathcal{D}_0 , essentially picked every even element from a vector. The question was raised there about why, for example, was not every *odd* element picked from the filtered vector instead? The answer is that it could be. For example, we could define the odd dyadic decimation operator \mathcal{D}_1 by

$$(\mathcal{D}_1 x)_\ell = x_{2\ell+1}, \quad (2.95)$$

and then the j th level mother and father wavelet coefficients would be obtained by the same formulae as in (2.80) and (2.81), but replacing \mathcal{D}_0 by \mathcal{D}_1 . As Nason and Silverman (1995) point out, this is merely a selection of a different orthogonal basis to the one defined by (2.80) and (2.81).

Nason and Silverman (1995) further point out that, at each level, one could choose either to use \mathcal{D}_0 or \mathcal{D}_1 , and a particular orthogonal basis could be labelled using the zeroes or ones implicit in the choice of particular \mathcal{D}_0 or \mathcal{D}_1 at each stage. Hence, a particular basis could be represented by the J -digit binary number $\epsilon = \epsilon_{J-1}\epsilon_{J-2}\cdots\epsilon_0$, where ϵ_j is one if \mathcal{D}_1 was used to produce level j and zero if \mathcal{D}_0 was used. Such a transform is termed the *ϵ -decimated wavelet transform*. Inversion can be handled in a similar way.

Now let us return to the finest scale. It can be easily seen that the effect of \mathcal{D}_1 can be achieved by first cyclically ‘rotating’ the sequence by one position (i.e., making $x_{k+1} = x_k$ and $x_0 = x_{2^J-1}$) and then applying \mathcal{D}_0 , i.e. $\mathcal{D}_1 = \mathcal{D}_0\mathcal{S}$, where \mathcal{S} is the shift operator defined by $(\mathcal{S}x)_j = x_{j+1}$. By an extension of this argument, and using the fact that $\mathcal{S}\mathcal{D}_0 = \mathcal{D}_0\mathcal{S}^2$, and that \mathcal{S} commutes with \mathcal{H} and \mathcal{G} , Nason and Silverman (1995) show that the basis vectors of the ϵ -decimated wavelet transform can be obtained from those of the standard discrete wavelet transform (DWT) by applying a particular shift operator. Hence, they note, the choice of ϵ corresponds to a particular choice of ‘origin’ with respect to which the basis functions are defined.

An important point is, therefore, that the standard DWT is dependent on choice of origin. A shift of the input data can potentially result in a completely different set of wavelet coefficients compared to those of the original data. For some statistical purposes, e.g., nonparametric regression, we probably would

not want our regression method to be sensitive to the choice of origin. Indeed, typically we would prefer our method to be *invariant* to the origin choice, i.e. translation invariant.

2.9.2 The non-decimated wavelet transform (NDWT)

Basic idea. The standard *decimated* DWT is orthogonal and transforms information from one basis to another. The Parseval relation shows that the total energy is conserved after transformation.

However, there are several applications where it might be useful to retain and make use of extra information. For example, in Examples 2.2 on p. 21 coefficient $d_{2,1} = (y_2 - y_1)/\sqrt{2}$ and $d_{2,2} = (y_4 - y_3)/\sqrt{2}$. These first two coefficients encode the difference between (y_1, y_2) and (y_3, y_4) respectively, but what about information that might be contained in the difference between y_2 and y_3 ? The values y_2, y_3 might have quite different values, and hence not forming a difference between these two values might mean we miss something.

Now suppose we follow the recipe for the ϵ -decimated transform given in the previous section. If the original sequence had been rotated cyclically by one position, then we would obtain the sequence (y_8, y_1, \dots, y_7) , and then on taking the Haar wavelet transform as before gives $d_{2,2} = (y_3 - y_2)/\sqrt{2}$. Applying the transform to the cyclically shifted sequence results in wavelet coefficients, as before, but the set that appeared to be ‘missing’ as noted above.

Hence, if we wish to retain more information and not ‘miss out’ potentially interesting differences, we should keep both the original set of wavelet coefficients *and* also the coefficients that resulted after shifting and transformation. However, one can immediately see that keeping extra information destroys the orthogonal structure and the new transformation is redundant. (In particular, one could make use of either the original or the shifted coefficients to reconstruct the original sequence.)

More precisely. The idea of the non-decimated wavelet transform (NDWT) is to retain both the odd and even decimations at each scale and continue to do the same at each subsequent scale. So, start with the input vector (y_1, \dots, y_n) , then apply and retain *both* $\mathcal{D}_0 \mathcal{G}y$ and $\mathcal{D}_1 \mathcal{G}y$ —the odd and even indexed ‘wavelet’ filtered observations. Each of these sequences is of length $n/2$, and so, in total, the number of wavelet coefficients (both decimations) at the finest scale is $2 \times n/2 = n$.

We perform a similar operation to obtain the finest-scale father wavelet coefficients and compute $\mathcal{D}_0 \mathcal{H}y$ ($n/2$ numbers) and $\mathcal{D}_1 \mathcal{H}y$ ($n/2$ numbers). Then for the next level wavelet coefficients we apply *both* $\mathcal{D}_0 \mathcal{G}$ and $\mathcal{D}_1 \mathcal{G}$ to *both* of $\mathcal{D}_0 \mathcal{H}y$ and $\mathcal{D}_1 \mathcal{H}y$. The result of each of these is $n/4$ wavelet coefficients at scale $J-2$. Since there are four sets, the total number of coefficients is n . A flow diagram illustrating the operation of the NDWT is shown in Figure 2.11.

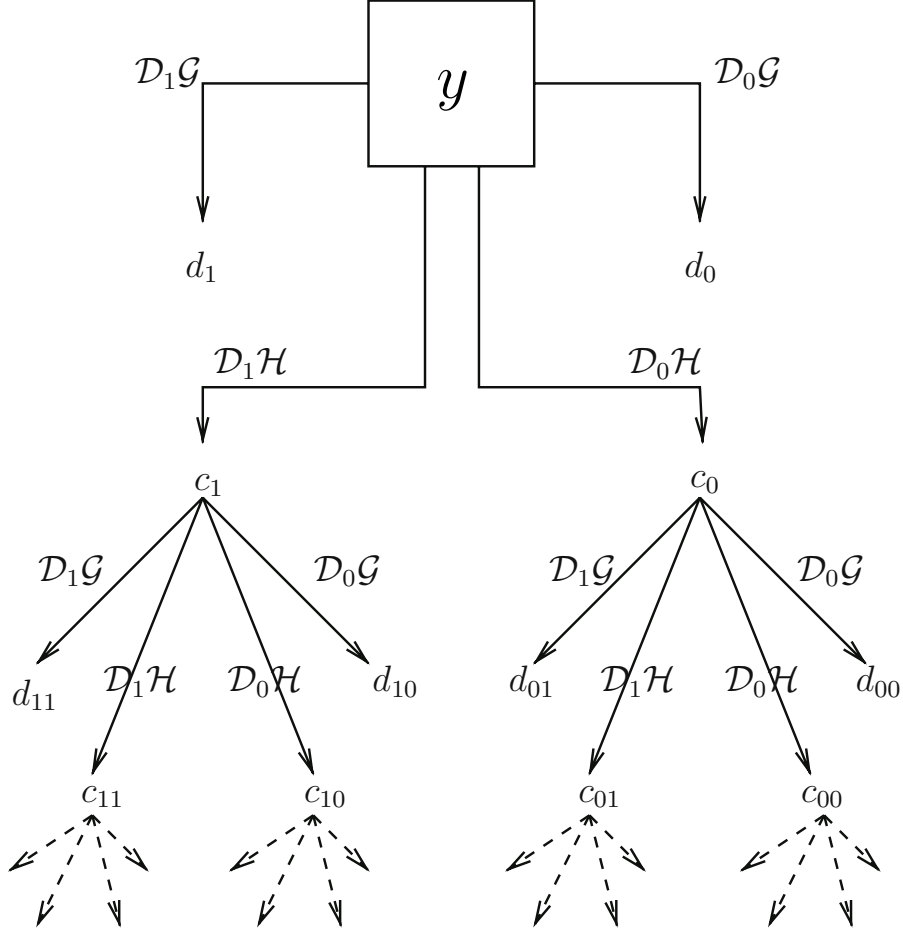


Fig. 2.11. Non-decimated wavelet transform flow diagram. The finest-scale wavelet coefficients are d_0 and d_1 . The next finest scale are $d_{00}, d_{01}, d_{10}, d_{11}$. The coefficients that only have 0 in the subscript correspond to the usual wavelet coefficients.

Continuing in this way, at scale $J - j$ there will be 2^j sets of coefficients each of length $2^{-j}n$ for $j = 1, \dots, J$ (remember $n = 2^J$). For the ‘next’ coarser scale, there will be twice the number of sets of wavelet coefficients that are half the length of the existing ones. Hence, the number of wavelet coefficients at each scale is always $2^{-j}n \times 2^j = n$. Since there are J scales, the total number of coefficients produced by the NDWT is Jn , and since $J = \log_2 n$, the number of coefficients produced is sometimes written as $n \log_2 n$. Since the production of each coefficient requires a fixed number of operations (which depends on the length of the wavelet filter in use), the computational effort required to compute the NDWT is also $\mathcal{O}(n \log_2 n)$. Although not as ‘fast’ as the discrete wavelet transform, which is $\mathcal{O}(n)$, the non-decimated algorithm

is still considered to be a fast algorithm (the $\log_2 n$ is considered almost to be ‘constant’).

We often refer to these ‘sets’ of coefficients as *packets*. These packets are different from the *wavelet packets* described in 2.11, although their method of computation is structurally similar.

Getting rid of the ‘origin-sensitivity’ is a desirable goal, and many authors have introduced the non-decimated ‘technique’ working from many points of view and on many problems. See, for example, Holschneider et al. (1989), Beylkin et al. (1991), Mallat (1991), and Shensa (1992). Also, Pesquet et al. (1996) list several papers that innovate in this area. One of the earliest statistical mentions of the NDWT is known as the *maximal-overlap* wavelet transform developed by Percival and Guttorp (1994); Percival (1995). In the latter work, the utility of the NDWT is demonstrated when attempting to estimate the variance within a time series at different scales. We discuss this further in Section 5.2.2. Coifman and Donoho (1995) introduced a NDWT that produced coefficients as ‘packets’. They considered different ϵ -decimations as ‘cycle spins’ and then used the results of averaging over several (often all) cycle spins as a means for constructing a translation-invariant (TI) regression method. We describe TI-denoising in more detail in Section 3.12.1. Nason and Silverman (1995) highlight the possibility for using non-decimated wavelets for determining the spectrum of a nonstationary or evolving time series. This latter idea was put on a sound theoretical footing by Nason et al. (2000), who introduced *locally stationary wavelet processes*: a class of nonstationary evolving time series constructed from non-decimated discrete wavelets, see Section 5.3.

Note that Nason and Silverman (1995) called the NDWT the ‘stationary’ wavelet transform. This turns out not to be a good name because the NDWT is actually useful for studying nonstationary time series, see Section 5.3. However, some older works occasionally refer to the older name.

2.9.3 Time and packet NDWT orderings

We have already informally mentioned two of the usual ways of presenting, or ordering, non-decimated wavelet coefficients. Let us again return to our simple example of (y_1, y_2, \dots, y_8) . We could simply compute the non-decimated coefficients in *time* order (we omit the $\sqrt{2}$ denominator for clarity):

$$(y_2 - y_1), (y_3 - y_2), (y_4 - y_3), (y_5 - y_4), (y_6 - y_5), (y_7 - y_6), (y_8 - y_7), (y_1 - y_8). \quad (2.96)$$

Or we could make direct use of the flow diagram depicted in Figure 2.11 to see the results of the non-decimated transform (to the first scale) as two packets: $\mathcal{D}_0\mathcal{G}$:

$$(y_2 - y_1), (y_4 - y_3), (y_6 - y_5), (y_8 - y_7), \quad (2.97)$$

or the odd decimation $\mathcal{D}_1\mathcal{G}$ packet as

$$(y_3 - y_2), (y_5 - y_4), (y_7 - y_6), (y_1 - y_8). \quad (2.98)$$

The coefficients contained within (2.96) and both (2.97) and (2.98) are exactly the same; it is merely the orderings that are different. One can continue in either fashion for coarser scales, and this results in a time-ordered NDWT or a packet-ordered one. The time-ordered transform can be achieved via a standard filtering (convolution) operation as noticed by Percival (1995), and hence it is easy to make this work for arbitrary n , not just $n = 2^J$. The packet-ordered transform produces packets as specified by the flow diagram in Figure 2.11.

The time-ordered transform is often useful for time series applications precisely because it is useful to have the coefficients in the same time order as the original data, see Section 5.3. The packet-ordered transform is often useful for nonparametric regression applications as each packet of coefficients corresponds to a particular type of basis element and it is convenient to apply modifications to whole packets and to combine packets flexibly to construct estimators, see Section 3.12.1.

Example 2.5. Let us return again to our simple example. Let $(y_1, \dots, y_n) = (1, 1, 7, 9, 2, 8, 8, 6)$. In `WaveThresh` the time-ordered wavelet transform is carried out using, again, the function `wd` but this time using the argument `type="station"`. For example,

```
> ywdS <- wd(y, filter.number=1, family="DaubExPhase",
+           type="station")
```

computes the NDWT using Haar wavelets. Different wavelets can be selected by supplying values to the `filter.number` and `family` arguments as described in Section 2.5.1.

Recall that in Example 2.3 we computed the (decimated) discrete wavelet transform of `y` and deposited it in the `ywd` object. Recall also that we extracted the finest-scale wavelet coefficients with the command

```
> accessD(ywd, level=2)
[1] 0.000000 -1.414214 -4.242641 1.414214
```

Let us do the same with our non-decimated object stored in `ywdS`:

```
> accessD(ywdS, level=2)
[1] 0.000000 -4.242641 -1.414214 4.949747 -4.242641
[6] 0.000000 1.414214 3.535534
```

As emphasized above, see how the original decimated wavelet coefficients appear at positions 1, 3, 5, 7 of the non-decimated vector—these correspond to the even dyadic decimation operator \mathcal{D}_0 . (Positions 1, 3, 5, 7 are actually odd, but in the C programming language—which much of the low level of `WaveThresh` is written in—the positions are actually 0, 2, 4, 6. C arrays start at 0 and not 1.)

Example 2.6. Now let us apply the packet-ordered transform. This is carried out using the `wst` function:

```
> ywst <- wst(y, filter.number=1, family="DaubExPhase")
```

Let us look again at the finest-scale coefficients:

```
> accessD(ywst, level=2)
[1] 0.000000 -1.414214 -4.242641 1.414214 -4.242641
[6] 4.949747 0.000000 3.535534
```

Thus, like the previous example, the number of coefficients at the finest scale is eight, the same as the length of `y`. However, here the first four coefficients are just the even-decimated wavelet coefficients (the same as the decimated wavelet coefficients from `ywd`) and the second four are the oddly decimated coefficients.

Although we have accessed the finest-scale coefficients using `accessD`, since the coefficients in `ywdS` are packet-ordered, it is more useful to be able to extract packets of coefficients. This extraction can be carried out using the `getpacket` function. For example, to extract the odd-decimated coefficients type:

```
> getpacket(ywst, level=2, index=1)
[1] -4.242641 4.949747 0.000000 3.535534
```

and use `index=0` to obtain the even-decimated coefficients.

What about packets at coarser levels? In Figure 2.11, at the second finest scale ($J - 2$, if $J = 3$ this is level 1), there should be four packets of length 2 which are indexed by binary 00, 01, 10, and 11. These can be obtained by supplying the `level=1` argument and setting the `index` argument to be the base ten equivalent of the binary 00, 01, 10, or 11. For example, to obtain the 10 packet type:

```
> getpacket(ywst, level=1, index=3)
[1] -2.5 -0.5
```

Example 2.7. We have shown above that the time-ordered and packet-ordered NDWTs are equivalent; it is just the orderings that are different. Hence, it should be possible to easily convert one type of object into another. This is indeed the case. For example, one could easily obtain the finest-scale time-ordered coefficients merely by interweaving the two sets of packet-ordered coefficients. Similar weavings operate at different scales, and details can be found in Nason and Sapatinas (2002). In `WaveThresh`, the conversion between one object and another is carried out using the `convert` function. Used on a `wst` class object it produces the `wd` class object and *vice versa*.

For example, if we again look at the finest-scale coefficients of the `ywst` object *after* conversion to a `wd` object, then we should observe the same coefficients as if we applied `accessD` directly to `ywd`. Thus, to check:


```
> accessD(convert(ywst), level=2)
[1] 0.000000 -4.242641 -1.414214 4.949747 -4.242641
[6] 0.000000 1.414214 3.535534
```

which gives the same result as applying `accessD` to `ywd`, as shown in Examples 2.5.

Example 2.8. Let us end this series of examples with a more substantial one. Define the symmetric chirp function by

$$y(x) = \sin(\pi/x),$$

for $x = \epsilon' + (-1, -1 + \delta, -1 + 2\delta, \dots, 1 - 2\delta)$, where $\epsilon' = 10^{-5}$ and $\delta = 1/512$ (essentially x is just a vector ranging from -1 to 1 in increments of $1/512$. The ϵ' is added so that x is never zero. The length of x is 1024). A plot of (x, y) is shown in Figure 2.12. The `WaveThresh` function `simchirp` can be

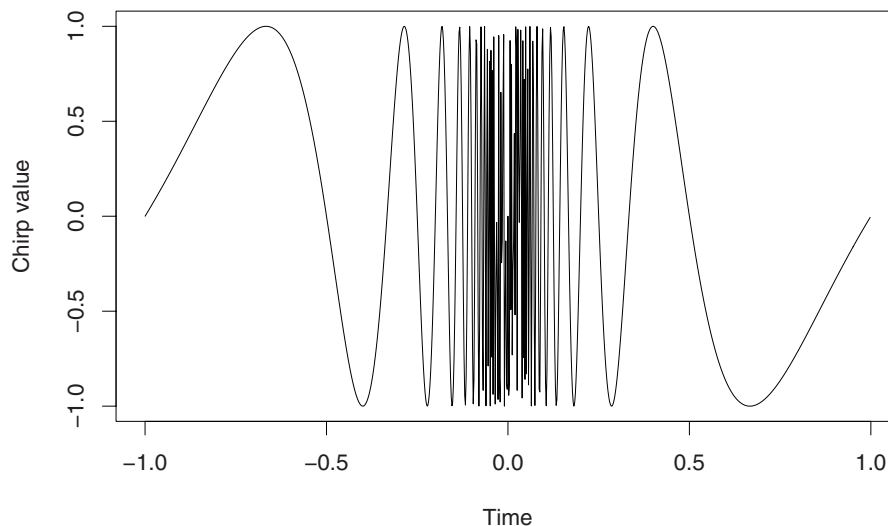


Fig. 2.12. Simulated chirp signal, see text for definition. Produced by `f.wav6()`. (Reproduced with permission from Nason and Silverman (1995).)

used to compute this function and returns an (x, y) vector containing values as follows:

```
> y <- simchirp()

> ywd <- wd(y$y, filter.number=2, family="DaubExPhase")

> plot(ywd, scaling="by.level", main="")
```

These commands also compute the discrete wavelet transform of y using the Daubechies compactly supported extremal-phase wavelet with two vanishing moments and then plot the result which is shown in Figure 2.13. The chirp

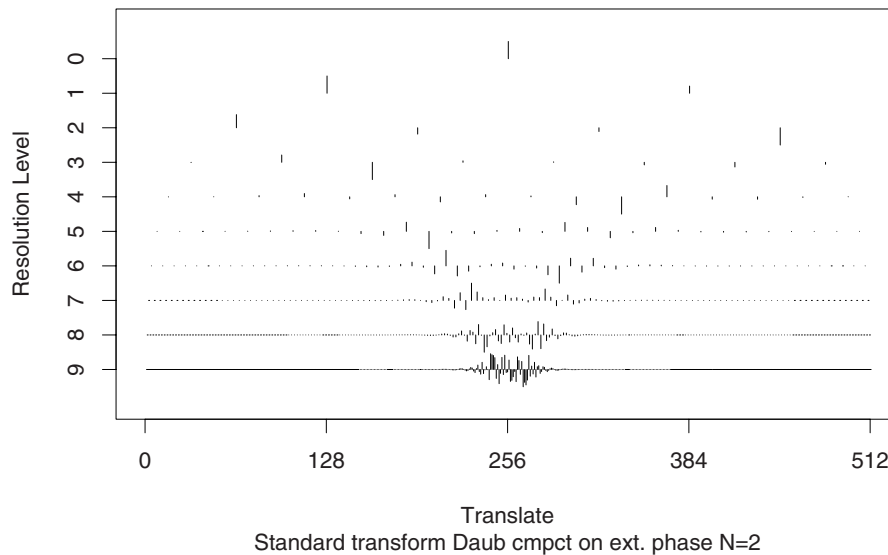


Fig. 2.13. Discrete wavelet coefficients of simulated chirp signal. Produced by `f.wav7()`. (Reproduced with permission from Nason and Silverman (1995).)

nature of the signal can be clearly identified from the wavelet coefficients, especially at the finer scales. However, as the scales get coarser (small resolution level) it is difficult to see any oscillation, which is unfortunate as the chirp contains power at lower frequencies.

The ‘missing’ oscillation turns up in its full glory when one examines a non-decimated DWT of the simulated chirp signal. This is shown in Figure 2.14, which was produced using the following code:

```
> ywd <- wd(y$y, filter.number=2, family="DaubExPhase",
+          type="station")

> plot(ywd, scaling="by.level", main="")
```

The reason the lower-frequency oscillation appears to be missing in the DWT is that the transform has been highly decimated at the lower levels (lower frequencies = coarser scales). In comparing Figure 2.13 with 2.14, one can see why the non-decimated transform is more useful for time series analysis. Although the transform is not orthogonal, and the system is redundant, significant information about the oscillatory behaviour at medium and low frequencies (coarser scales) is retained. The chirp signal is an example of a

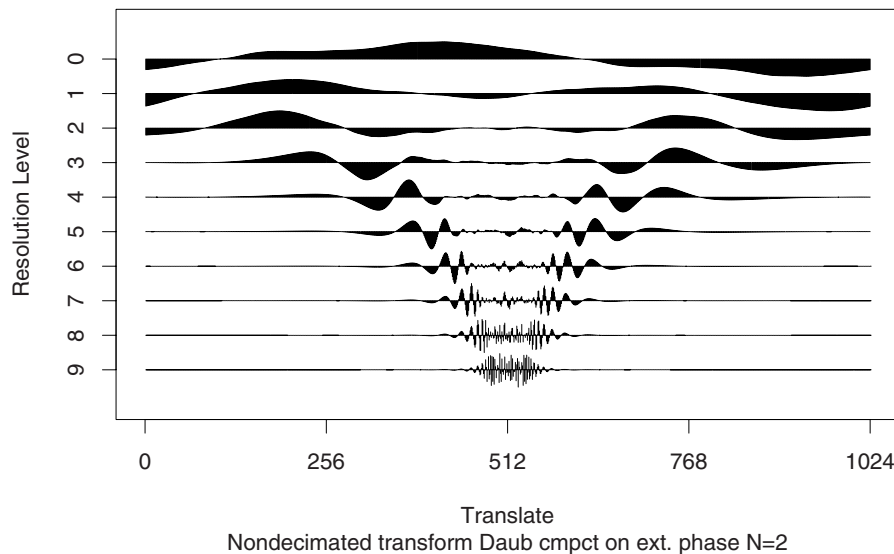


Fig. 2.14. Time-ordered non-decimated wavelet coefficients of simulated chirp signal. Produced by `f.wav8()`. (Reproduced with permission from Nason and Silverman (1995).)

deterministic time series. However, the NDWT is useful in the modelling and analysis of stochastic time series as described further in Chapter 5.

Finally, we also compute and plot the packet-ordered NDWT. This is achieved with the following commands:

```
> ywst <- wst(y$y, filter.number=2, family="DaubExPhase")

> plot(ywst, scaling="by.level", main="")
```

The plot is shown in Figure 2.15. The bottom curve in Figure 2.15 is again just the simulated chirp itself (which can be viewed as finest-scale, data-scale, scaling function coefficients). At the finest detail scale, level nine, there are two packets, the even and oddly decimated coefficients respectively. The packets are separated by a short vertical dotted line. As mentioned above, if one interlaced the coefficients from each packet one at a time, then one would recover the scale level nine coefficients from the time-ordered plot in Figure 2.14. On successively coarser scales the number of packets doubles, but the number of coefficients per packet halves: overall, the number of coefficients remains constant at each level.

2.9.4 Final comments on non-decimated wavelets

To conclude this section on non-decimated wavelets, we refer forward to three sections that take this idea further.

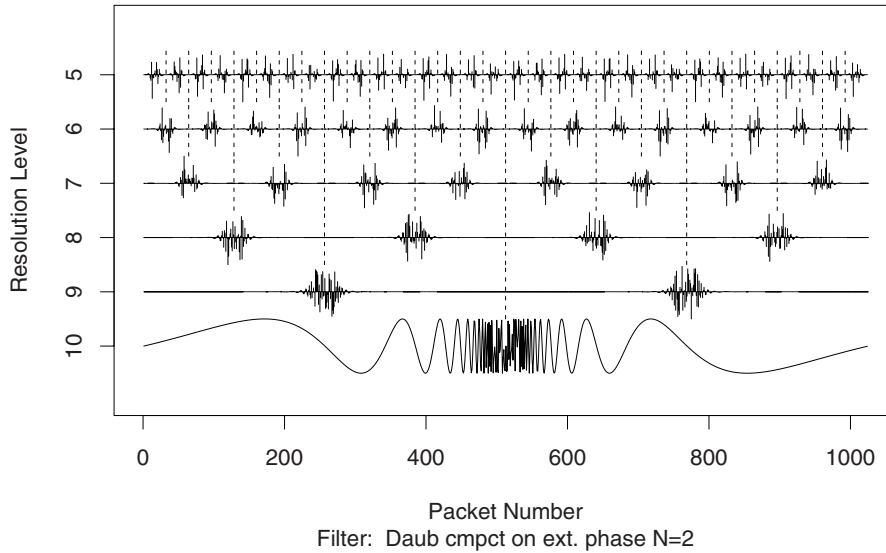


Fig. 2.15. Packet-ordered non-decimated wavelet coefficients of simulated chirp signal. Produced by `f.wav9()`.

1. Section 2.11 describes a generalization of wavelets, called wavelet packets. Wavelet packets can also be extended to produce a non-decimated version, which we describe in Section 2.12.
2. The next chapter explains how the NDWT can be a useful tool for non-parametric regression problems. Section 3.12.1 explains how ϵ -decimated bases can be selected, or how averaging can be carried out over all ϵ -decimated bases in an efficient manner to perform nonparametric regression.
3. Chapter 5 describes how non-decimated wavelets can be used for the modelling and analysis of time series.

Last, we alert the reader to the fact that wavelet transforms computed with different computer packages can sometimes give different results. With decimated transforms the results can be different between packages, although the differences are often minor or trivial and usually due to different wavelet scalings or reflections (e.g., if $\psi(x)$ is a wavelet, then so is $\psi(-x)$). However, with non-decimated transforms the scope for differences increases mainly due to the number of legitimate, but different, ways in which the coefficients can be interwoven.

2.10 Multiple Wavelets

Multiple wavelets are bases with more than one mother and father wavelet. The number of mother wavelets is often denoted by L , and for simplicity of

exposition we concentrate on $L = 2$. In this section we base our exposition on, and borrow notation from, Downie and Silverman (1998), which draws on work on multiple wavelets by Geronimo et al. (1994), Strang and Strela (1994), Strang and Strela (1995) and Xia et al. (1996) and Strela et al. (1999). See Goodman and Lee (1994), Chui and Lian (1996), Rong-Qing et al. (1998) for further insights and references.

An (orthonormal) multiple wavelet basis admits the following representation, which is a multiple version of (2.53):

$$f(x) = \sum_{k \in \mathbb{Z}} C_{J,k}^T \Phi_{J,k}(x) + \sum_{j=1}^J \sum_{k \in \mathbb{Z}} D_{j,k}^T \Psi_{j,k}(x), \quad (2.99)$$

where $C_{J,k} = (c_{J,k,1}, c_{J,k,2})^T$ and $D_{j,k} = (d_{j,k,1}, d_{j,k,2})^T$ are *vector* coefficients of dimension $L = 2$. Also, $\Psi_{j,k}(x) = 2^{j/2} \Psi(2^j x - k)$, similarly for $\Phi_{J,k}(x)$, which is very similar to the usual dilation/translation formula, as for single wavelets in (2.20).

The quantity $\Phi(x)$ is actually a vector function of x given by $\Phi(x) = (\phi_1(x), \phi_2(x))^T$ and $\Psi(x) = (\psi_1(x), \psi_2(x))^T$. The basis functions are orthonormal, i.e.

$$\int \psi_l(2^j x - k) \psi_{l'}(2^{j'} x - k') dx = \delta_{l,l'} \delta_{j,j'} \delta_{k,k'}, \quad (2.100)$$

and the $\phi_1(x)$ and $\phi_2(x)$ are orthonormal to all the wavelets $\psi_l(2^j x - k)$. The vector functions $\Phi(x)$ and $\Psi(x)$ satisfy the following dilation equations, which are similar to the single wavelet ones of (2.47) and (2.51):

$$\Phi(x) = \sum_{k \in \mathbb{Z}} H_k \Phi(2x - k), \quad \Psi(x) = \sum_{k \in \mathbb{Z}} G_k \Psi(2x - k), \quad (2.101)$$

where now H_k and G_k are 2×2 matrices.

The *discrete multiple wavelet transform* (DMWT), as described by Xia et al. (1996), is similar to the discrete wavelet transform given in (2.75) and (2.76) and can be written as

$$C_{j,k} = \sqrt{2} \sum_n H_n C_{j+1,n+2k} \quad \text{and} \quad D_{j,k} = \sqrt{2} \sum_n G_n C_{j+1,n+2k}, \quad (2.102)$$

for $j = 0, \dots, J-1$. Again, the idea is similar to before: obtain coarser-scale wavelet and scaling function coefficients from finer scale ones. The inverse formula is similar to the single wavelet case.

The rationale for multiple wavelet bases as given by Strang and Strela (1995) is that (i) multiple wavelets can be symmetric, (ii) they can possess short support, (iii) they can have higher accuracy, and (iv) can be orthogonal. Strang and Strela (1995) recall Daubechies (1992) to remind us that no single wavelet can possess these four properties simultaneously.

In most statistical work, the multiple wavelet transform has been proposed for denoising of univariate signals. However, there is immediately a problem

with this. The starting (input) coefficients for the DMWT, $\{C_{J,n}\}$, are 2D vectors. Hence, a way has to be found to transform a univariate input sequence into a sequence of 2D vectors. Indeed, such ways have been devised and are called *prefilters*. More on these issues will be discussed in our section on multiple wavelet denoising in Section 3.13.

Example 2.9. Let us continue our previous example and compute the multiple wavelet transform of the chirp signal introduced in Example 2.8. The multiple wavelet code within **WaveThresh** was introduced by Downie (1997). The main functions are: **mwd** for the forward multiple wavelet transform and **mwr** for its inverse. The multiple wavelet transform of the chirp signal can be obtained by the following commands:

```
> y <- simchirp()

> ymwd <- mwd(y$y)

> plot(ymwd, cex=cex)
```

The plot is displayed in Figure 2.16.

2.11 Wavelet Packet Transforms

In Section 2.9 we considered how both odd and even decimation could be applied at each wavelet transform step to obtain the non-decimated wavelet transform. However, for both the decimated and non-decimated transforms the transform cascades by applying filters to the output of a smooth filtering (\mathcal{H}). One might reasonably ask the question: is it possible, and sensible, to apply both filtering operations (\mathcal{H} and \mathcal{G}) to the output after a filtering by either \mathcal{H} or \mathcal{G} ? The answer turns out to be yes, and the resulting coefficients are wavelet packet coefficients.

Section 2.3 explained that a set of orthogonal wavelets $\{\psi_{j,k}(x)\}_{j,k}$ was a *basis* for the space of functions $L^2(\mathbb{R})$. However, it is not the only possible basis. Other bases for such function spaces are orthogonal polynomials and the Fourier basis. Indeed, there are many such bases, and it is possible to organize some of them into collections called *basis libraries*. One such library is the *wavelet packet* library, which we will describe below and is described in detail by Wickerhauser (1994), see also Coifman and Wickerhauser (1992) and Hess-Nielsen and Wickerhauser (1996). Other basis libraries include the local cosine basis library, see Bernardini and Kovačević (1996), and the SLEX library which is useful for time series analyses, see Ombao et al. (2001), Ombao et al. (2002, 2005).

Following the description in Coifman and Wickerhauser (1992) we start from a Daubechies mother and father wavelet, ψ and ϕ , respectively. Let $W_0(x) = \phi(x)$ and $W_1(x) = \psi(x)$. Then define the sequence of functions $\{W_k(x)\}_{k=0}^\infty$ by

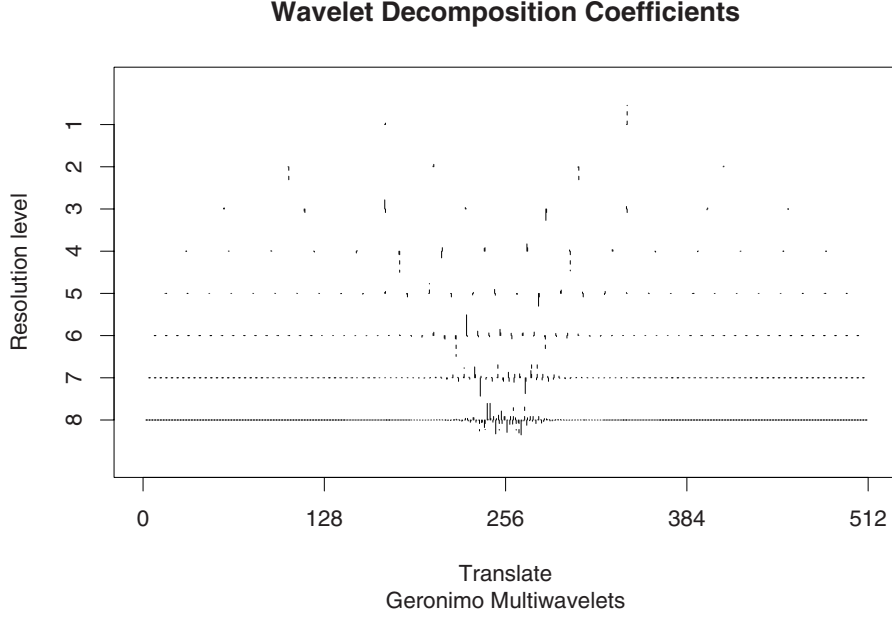


Fig. 2.16. Multiple wavelet transform coefficients of chirp signal. At each time-scale location there are two coefficients: one for each of the wavelets at that location. In `WaveThresh` on a colour display the two different sets of coefficients can be plotted in different colours. Here, as different line styles, so some coefficients are *dashed*, some are *solid*. Produced by `f.wav10()`.

$$\begin{aligned}
 W_{2n}(x) &= \sqrt{2} \sum_k h_k W_n(2x - k), \\
 W_{2n+1}(x) &= \sqrt{2} \sum_k g_k W_n(2x - k).
 \end{aligned}
 \tag{2.103}$$

This definition fulfils the description given above in that both h_k and g_k are applied to $W_0 = \phi$ and both to $W_1 = \psi$ and then both h_k and g_k are applied to the results of these. Coifman and Wickerhauser (1992) define the library of wavelet packet bases to be the collection of orthonormal bases comprised of (dilated and translated versions of W_n) functions of the form $W_n(2^j x - k)$, where $j, k \in \mathbb{Z}$ and $n \in \mathbb{N}$. Here j and k are the scale and translation numbers respectively and n is a new kind of parameter called the number of oscillations. Hence, they conclude that $W_n(2^j - k)$ should be (approximately) centred at $2^j k$, have support size proportional to 2^{-j} and oscillate approximately n times. To form an orthonormal basis they cite the following proposition.

Proposition 1 (Coifman and Wickerhauser (1992)) *Any collection of indices $(j, n, k) \in \mathbb{N} \times \mathbb{N} \times \mathbb{Z}$, such that the intervals $[2^j n, 2^j(n+1))$ form a disjoint cover of $[0, \infty)$ and k ranges over all the integers, corresponds to an orthonormal basis of $L^2(\mathbb{R})$.*

In other words, wavelet packets at different scales but identical locations (or covering locations) cannot be part of the same basis.

The definition of wavelet packets in (2.103) shows how coefficients/basis functions are obtained by repeated application of both the \mathcal{H} and \mathcal{G} filters to the original data. This operation is depicted by Figure 2.17. Figure 2.18

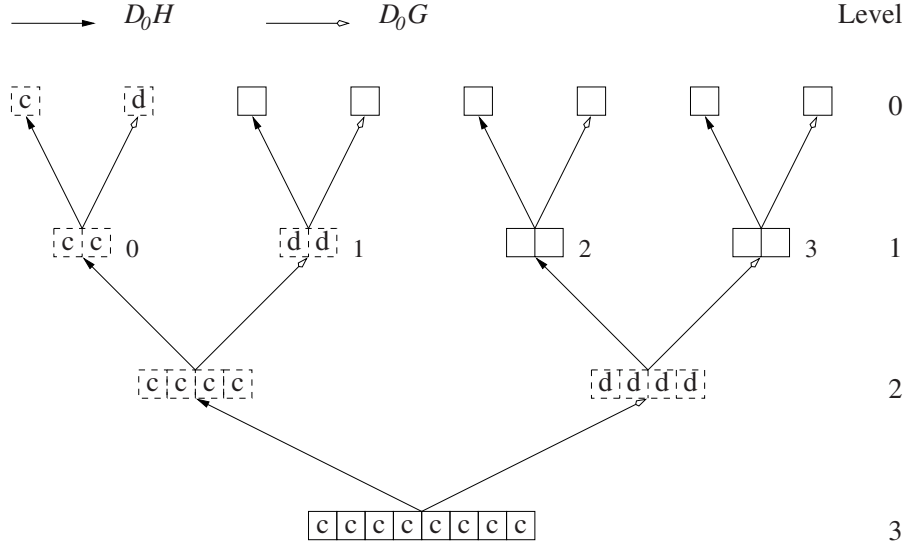


Fig. 2.17. Illustration of wavelet packet transform applied to eight data points (*bottom to top*). The $\mathcal{D}_0\mathcal{H}$, $\mathcal{D}_0\mathcal{G}$ filters carry out the smooth and detail operations as in the regular wavelet transform. The difference is that both are applied recursively to the original data with input at the *bottom* of the picture. The regular wavelet coefficients are labelled 'd' and the regular scaling function coefficients are labelled 'c'. The *arrows* at the *top* of the figure indicate which filter is which. Reproduced with permission from Nason and Sapatinas (2002).

shows examples of four wavelet packet functions.

2.11.1 Best-basis algorithms

This section addresses how we might use a library of bases. In Section 2.9.2 we described the set of non-decimated wavelets and how that formed an overdetermined set of functions from which different bases (the ϵ -decimated basis) could be *selected* or, in a regression procedure, representations with respect to many basis elements could be averaged over, see Section 3.12.1. Hence, the non-decimated wavelets are also a basis library and usage usually depends on *selecting* a basis element or *averaging* over the results of many.

For wavelet packets, selection is the predominant mode of operation. Basis averaging could be considered but has received little attention in the

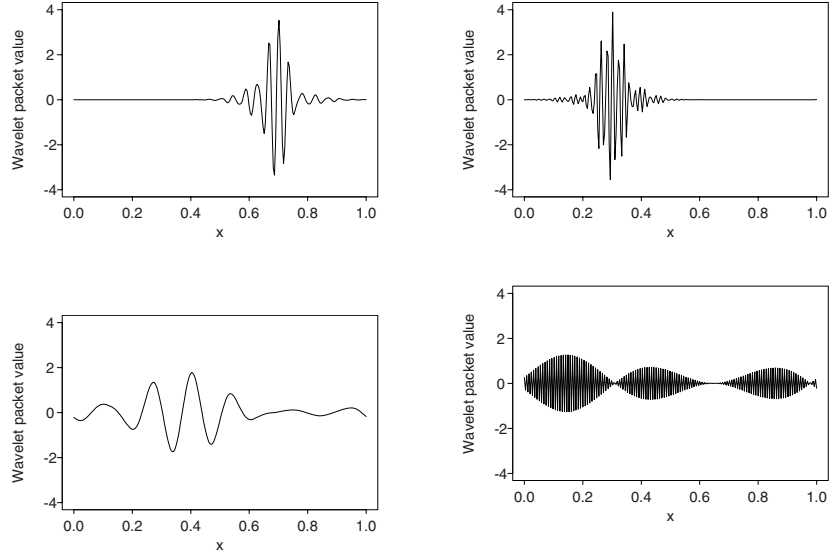


Fig. 2.18. Four wavelet packets derived from Daubechies (1988) least-asymmetric mother wavelet with ten vanishing moments. These four wavelet packets are actually orthogonal and drawn by the `drawwp.default()` function in `WaveThresh`. The vertical scale is exaggerated by ten times. Reproduced with permission from Nason and Sapatinas (2002).

literature. So, for statistical purposes how does selection work? In principle, it is simple for nonparametric regression. One selects a particular wavelet packet basis, obtains a representation of the noisy data with respect to that, thresholds (reduce noise, see Chapter 3), and then inverts the packet transform with respect to that basis. This task can be carried out rapidly using fast algorithms.

However, the whole set of wavelet packet coefficients can be computed rapidly in only $\mathcal{O}(N \log N)$ operations. Hence, an interesting question arises: is it better to select a basis first and then threshold, or is it best to threshold and then select a basis? Again, not much attention has been paid to this problem. For an example of basis selection followed by denoising see Ghugre et al. (2003). However, if the denoising can be done well on all wavelet packet coefficients simultaneously, then it might be better to denoise first and then perform basis selection. The reason for this is that many basis selection techniques are based on the Coifman and Wickerhauser (1992) best-basis algorithm, which is a method that was originally designed to work on deterministic functions. Of course, if the denoising is not good, then the basis selection might not work anyhow. We say a little more on denoising with wavelet packets in Section 3.16.

Coifman–Wickerhauser best-basis method. A possible motivation for the best-basis method is signal compression. That is, can a basis be found that gives the most efficient representation of a signal? Here efficient can roughly be translated into ‘most sparse’. A vector of coefficients is said to be sparse if most of its entries are zero, and only a few are non-zero. The Shannon entropy is suggested as a measure of sparsity. Given a set of basis coefficients $\{v_i\}$, the Shannon entropy can be written as $-\sum |v_i|^2 \log |v_i|^2$. For example, the **WaveThresh** function **Shannon.entropy** computes the Shannon entropy. Suppose we apply it to two vectors: $v^{(1)} = (0, 0, 1)$ and $v^{(2)} = (1, 1, 1)/\sqrt{3}$. Both these vectors have unit norm.

```
> v1 <- c(0,0,1)
> Shannon.entropy(v1)
[1] 0

> v2 <- rep(1/sqrt(3), 3)
> Shannon.entropy(v2)
[1] 1.098612
```

(technically **Shannon.entropy** computes the negative Shannon entropy). These computations suggest that the Shannon entropy is minimized by sparse vectors. Indeed, it can be proved that the ‘most-non-sparse’ vector $v^{(2)}$ maximizes the Shannon entropy. (Here is a proof for a very simple case. The Shannon entropy is more usually computed on probabilities. Suppose we have two probabilities p_1, p_2 and $p_1 + p_2 = 1$ and the (positive) Shannon entropy is $\mathcal{S}_e(\{p_i\}) = \sum_i p_i \log p_i = p_1 \log p_1 + (1 - p_1) \log(1 - p_1)$. Let us find the stationary points: $\partial \mathcal{S}_e / \partial p_1 = \log p_1 - \log(1 - p_1) = 0$, which implies $\log\{p_1/(1 - p_1)\} = 0$, which implies $p_1 = p_2 = 1/2$, which is the least-sparse vector. Differentiating \mathcal{S}_e again verifies a minimum. For the *negative* Shannon entropy it is a maximum. The proof for general dimensionality $\{p_i\}_{i=1}^n$ is not much more difficult.)

To summarize, the Shannon entropy can be used to measure the sparsity of a vector, and the Coifman–Wickerhauser algorithm searches for the basis that minimizes the overall negative Shannon entropy (actually Coifman and Wickerhauser (1992) is more general than this and admits more general cost functions). Coifman and Wickerhauser (1992) show that the best basis can be obtained by starting from the finest-scale functions and comparing the entropy of that representation by the next coarsest scale packets, and then selecting the one that minimizes the entropy (either the packet or the combination of the two children). Then this operation is applied recursively if required.

2.11.2 WaveThresh example

The wavelet packet transform is implemented in **WaveThresh** by the **wp** function. It takes a dyadic-length vector to transform and requires the **filter.number** and **family** arguments to specify the underlying wavelet

family and number of vanishing moments. For example, suppose we wished to compute the wavelet packet transform of a vector of iid Gaussian random variables. This can be achieved by

```
> z <- rnorm(256)

> zwp <- wp(z, filter.number=2, family="DaubExPhase")

> plot(zwp, color.force=TRUE)
```

This produces the wavelet packet plot shown in Figure 2.19. Let us now replace

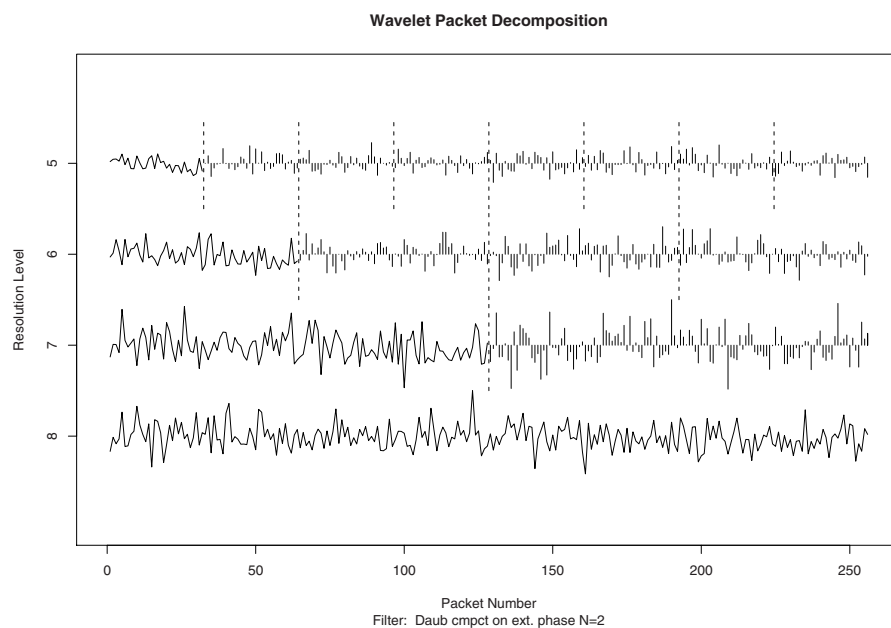


Fig. 2.19. Wavelet packet coefficients of the independent Gaussian sequence z . The time series at the *bottom* of the plot, scale eight, depicts the original data, z . At scales seven through five different wavelet packets are separated by *vertical dotted lines*. The first packet at each scale corresponds to scaling function coefficients, and these have been plotted as a time series rather than a set of small vertical lines (as in previous plots of coefficients). This is because the scaling function coefficients can be thought of as a successive coarsening of the original series and hence are a kind of smooth of the original. The regular wavelet coefficients are always the second packet at each scale. The default plot arguments in `plot.wp` only plot up to scale five and no lower. Produced by `f.wav11()`.

one of the packets in this basis by a very sparse packet. We shall replace the fourth packet (packet 3) at scale six by a packet consisting of all zeroes and a single value of 100. We can investigate the current values of packet (6,3)

(index packet 3 is the fourth at scale six, the others are indexed 0, 1, 2) by again using the generic `getpacket` function:

```
> getpacket(zwp, level=6, index=3)
[1] -1.004520984 2.300091601 -0.765667778 0.614727692
[5] 2.257342407 0.816656404 0.017121135 -0.353660951
[9] 0.959106692 1.227197543 ...
...
[57] 0.183307351 -0.435437120 0.373848181 -0.565281279
[60] -0.746125550 1.118635271 0.773617722 -1.888108807
[64] -0.182469097
```

So, a vector consisting of a single 100 and all others equal to zero is very sparse. Let us create a new wavelet packet object, `zwp2`, which is identical to `zwp` in all respects except it contains the new sparse packet:

```
> zwp2 <- putpacket(zwp, level=6, index=3,
+   packet=c(rep(0,10), 100, rep(0,53)))

> plot(zwp2)
```

This last plot command produces the wavelet packet plot as shown in Figure 2.20. To apply the Coifman–Wickerhauser best-basis algorithm using Shannon entropy we use the `MaNoVe` function (which stands for ‘make node vector’, i.e. select a basis of packet nodes). We can then examine the basis selected merely by typing the name of the node vector:

```
> zwp2.nv <- MaNoVe(zwp2)

> zwp2.nv
Level: 6 Packet: 3
Level: 3 Packet: 5
Level: 3 Packet: 11
Level: 2 Packet: 5
Level: 2 Packet: 12
...
```

As can be seen, (6,3) was selected as a basis element—not surprisingly as it is extremely sparse. The representation can be inverted with respect to the new selected basis contained within `zwp2.nv` by calling `InvBasis(zwp2, zwp2.nv)`. If the inversion is plotted, one sees a very large spike near the beginning of the series. This is the consequence of the ‘super-sparse’ (6,3) packet.

More information on the usage of wavelet packets in statistical problems in regression and time series can be found in Sections 3.16 and 5.5.

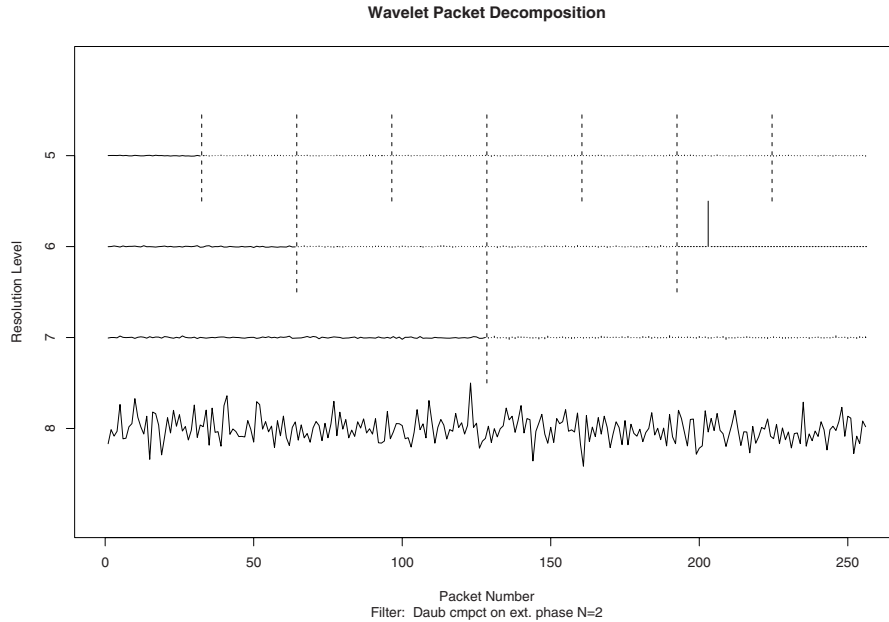


Fig. 2.20. Wavelet packet coefficients of `zwp2`. Apart from packet (6,3), these coefficients are identical to those in Figure 2.19. However, since the plotted coefficient sizes are relative, the duplicated coefficients have been plotted much smaller than those in Figure 2.19 because of the large relative size of the 100 coefficient in the fourth packet at level 6 (which stands out as the tenth coefficient after the start of the fourth packet indicated by the *vertical dotted line*.) Produced by `f.wav12()`.

2.12 Non-decimated Wavelet Packet Transforms

The discrete wavelet transform relied on even dyadic decimation, \mathcal{D}_0 , and the smoothing and detail filters, \mathcal{H} and \mathcal{G} , but iterating on the results of the \mathcal{H} filter only. One generalization of the wavelet transform, the non-decimated transform, pointed out that the odd dyadic decimation operator, \mathcal{D}_1 , was perfectly valid and both could be used at each step of the wavelet transform.

In the previous section another generalization, the wavelet packet transform, showed that iteration of both \mathcal{H} and \mathcal{G} could be applied to the results of *both* of the previous filters, not just \mathcal{H} .

These two generalizations can themselves be combined by recursively applying the four operators $\mathcal{D}_0\mathcal{H}$, $\mathcal{D}_0\mathcal{G}$, $\mathcal{D}_1\mathcal{H}$, and $\mathcal{D}_1\mathcal{G}$. Although this may sound complicated, the result is that we obtain wavelet packets that are non-decimated. Just as non-decimated wavelets are useful for time series analysis, so are non-decimated wavelet packets. See Section 5.6 for further information.

2.13 Multivariate Wavelet Transforms

The extension of wavelet methods to 2D regularly spaced data (images) and to such data in higher dimensions was proposed by Mallat (1989b). A simplified explanation appears in Nason and Silverman (1994). Suppose one has an $n \times n$ matrix x where n is dyadic. In its simplest form one applies both the $\mathcal{D}_0\mathcal{H}$ and $\mathcal{D}_0\mathcal{G}$ operators from (2.79) to the rows of the matrix. This results in two $n \times (n/2)$ matrices, which we will call H and G . Then both operators are again applied but to both the columns of H and G . This results in four matrices HH , GH , HG , and GG each of dimension $(n/2) \times (n/2)$. The matrix HH is the result of applying the ‘averaging’ operator $\mathcal{D}_0\mathcal{H}$ to both rows and columns of x , and this is the set of scaling function coefficients with respect to the 2D scaling function $\Phi(x, y) = \Phi(x)\Phi(y)$. The other matrices GH , HG , and GG create finest-scale wavelet detail in the horizontal, vertical, and ‘diagonal’ directions. This algorithmic step is then repeated by applying the same filtering operations to HH , which generates a new HH , GH , HG , and GG at the next finest scale and then the step is repeated by application to the new HH , and so on (exactly the same as the recursive application of $\mathcal{D}_0\mathcal{H}$ to the c vectors in the 1D transform). The basic algorithmic step for the 2D separable transform is depicted in Figure 2.21.

The transform we have described here is an example of a separable wavelet transform because the 2D scaling function $\Phi(x, y)$ can be separated into the product of two 1D scaling functions $\phi(x)\phi(y)$. The same happens with the wavelets except there are three of them encoding the horizontal, vertical, and diagonal detail $\Psi^H(x, y) = \psi(x)\phi(y)$, $\Psi^V(x, y) = \phi(x)\psi(y)$, and $\Psi^D(x, y) = \psi(x)\psi(y)$. For a more detailed description see Mallat (1998). For nonseparable wavelets see Kovačević and Vetterli (1992) or Li (2005) for a more recent construction and further references.

The 2D transform of an image is shown in Figure 2.22, and the layout of the coefficients is shown in Figure 2.23. The coefficient image was produced with the following commands in `WaveThresh`:

```
#
# Enable access to teddy image
#
> data(teddy)
#
# Setup grey scale for image colors
#
> greycol <- grey((0:255)/255)
#
# Compute wavelet coefficients of teddy image
#
> teddyimwd <- imwd(teddy, filter.number=10)
#
# Compute scaling for coefficient display
```

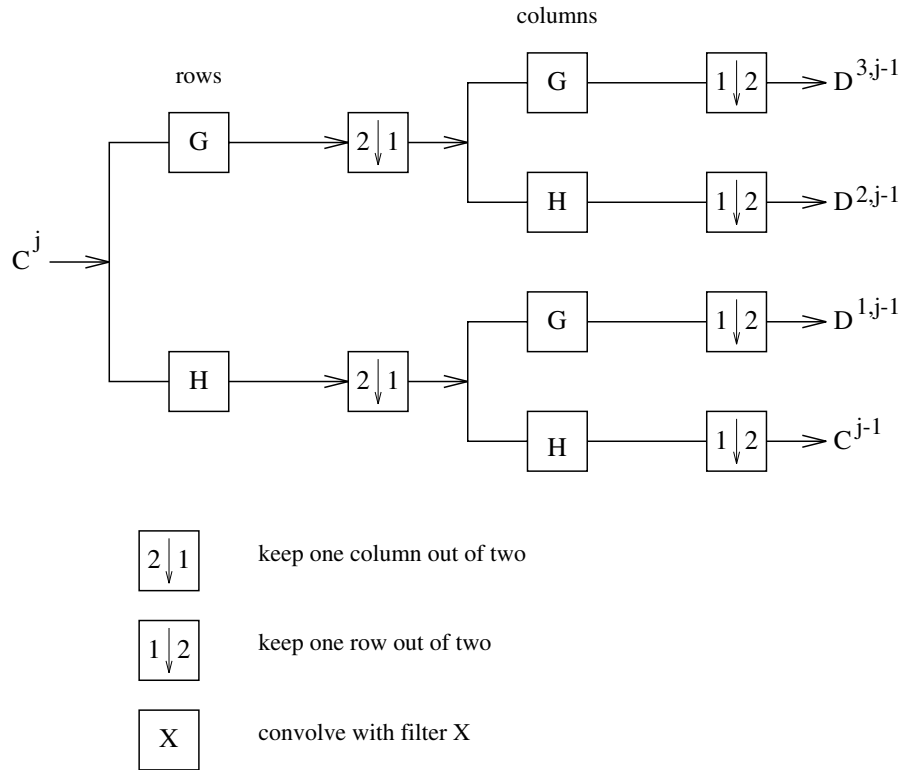


Fig. 2.21. Schematic diagram of the central step of the 2D discrete wavelet transform. The input image on the *left* is at level j and the outputs are the smoothed image C^{j-1} plus horizontal, vertical, and diagonal detail D^1, D^2 , and D^3 . The smoothed image C^{j-1} is fed into an identical step at the next coarsest resolution level. Here $2 \downarrow 1$ and $1 \downarrow 2$ denote dyadic decimation \mathcal{D}_0 in the horizontal and vertical directions. (After Mallat (1989b)).

```
# (just a suggestion)
#
> myt <- function(x) 20+sqrt(x)
#
# Display image of Teddy
#
> plot(teddyimwd, col=greycol, transform=TRUE, tfunction=myt)
```

In both Figures 2.22 and 2.23, the top left block corresponds to the finest detail in the vertical direction, the top right block corresponds to the finest detail in the diagonal direction, and the bottom right block to the horizontal detail (i.e. the GH , GG , and HG coefficients produced by the algorithm mentioned above). These three blocks form an inverted ‘L’ of coefficients at the finest

resolution. The next step of the algorithm produces a similar set of coefficients at the second finest resolution according to the same layout, and so on.



Fig. 2.22. Wavelet coefficients of teddy image shown in Figure 4.6 on p. 142. Produced by `f.wav15()`. (After Mallat (1989b)).

Within `WaveThresh`, the 1D DWT is carried out using the `wd` and `wr` functions, the 2D DWT using the `imwd` and `imwr` functions, and the 3D DWT using the `wd3D` and `wr3D` functions. Both `wd` and `imwd` can perform the time-ordered non-decimated transform, and `wst` and `wst2D` can perform 2D packet-ordered non-decimated transforms.

2.14 Other Topics

The continuous wavelet transform. We have first presented the story of wavelets as a method to extract multiscale information from a sequence, then explained how a set of functions called Haar wavelets can be used to provide a theoretical underpinning to this extraction. Then we demonstrated that the idea could be generalized to wavelets that are smoother than Haar wavelets and, for some applications, more useful. In many mathematical presentations, e.g. Daubechies (1992) (whose development we will follow here), the starting point is the *continuous wavelet transform*, CWT. Here the starting point is a function $f \in L^2(\mathbb{R})$ whose CWT is given by

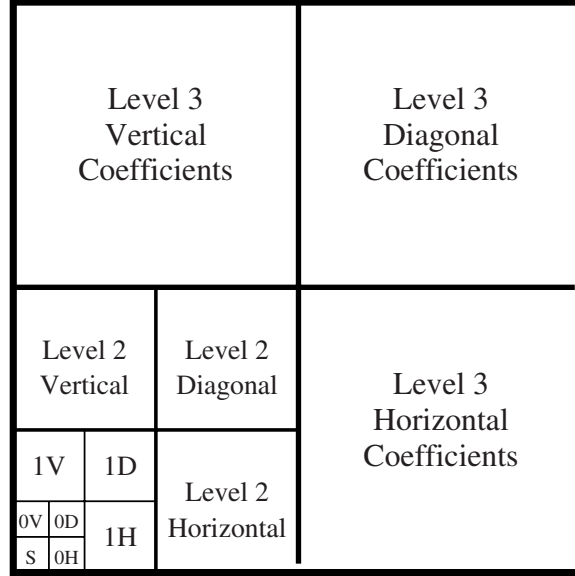


Fig. 2.23. Diagram showing general layout of wavelet coefficients as depicted in Figure 2.22. The plan here stops at the fourth iteration (level 0) whereas the one in Figure 2.22 is the result of nine iterations. (After Mallat (1989b)).

$$F(a, b) = \int_{-\infty}^{\infty} f(x) \psi^{a,b}(x) dx, \quad (2.104)$$

for $a, b \in \mathbb{R}, a \neq 0$, where

$$\psi^{a,b}(x) = |a|^{-1/2} \psi\left(\frac{x-b}{a}\right), \quad (2.105)$$

where $\psi \in L^2(\mathbb{R})$ satisfies a technical admissibility condition which Daubechies (1992) notes “for all practical purposes, [the admissibility condition] is equivalent to the requirement that $\int \psi(x) dx = 0$ ”. The function f can be recovered from its CWT, $F(a, b)$. There are many accounts of the CWT; see, for example, Heil and Walnut (1989), Daubechies (1992), Meyer (1993b), Jawerth and Sweldens (1994), Vidakovic (1999a), and Mallat (1998), to name but a few. As for the DWT above there are many wavelets that can be used for $\psi(x)$ here, for example, the Haar, the Shannon from Section 2.6.1, and the so-called ‘Mexican-hat’ wavelet which is the second derivative of the normal probability density function.

Antoniadis and Gijbels (2002) note that in practical applications the CWT is usually computed on a discrete grid of points, and one of the most popular, but by no means the only, discretizations is to set $a = 2^j$ and $b = k$. Antoniadis and Gijbels (2002) refer to this as the continuous discrete wavelet transform (CDWT) and mention a fast computational algorithm by Abry (1994) which

is equivalent to the non-decimated wavelet transform from Section 2.9. The CWT can be discretized to $a = 2^j$ and $b = k2^j$ to obtain the DWT. In this context the CDWT is often used for jump or singularity detection, such as in Mallat and Hwang (1992) and Antoniadis and Gijbels (2002) and references therein. Torrence and Compo (1998) is an extremely well-written and engaging description of the use of the CWT for the analysis of meteorological time series such as the El Niño Southern Oscillation.

Lifting is a technique that permits the multiscale method to be applied to more general data situations. The wavelet transforms we have described above are limited to data that occur regularly spaced on a grid, and for computational convenience and speed we have also assumed that $n = 2^J$ (although this latter restriction can often be circumvented by clever algorithm modification). What about data that are not regularly spaced? Most regression methods, parametric and nonparametric, can be directly applied to irregularly spaced data, so what about wavelet methods? Many papers have been written that are devoted to enabling wavelets in the irregular case. This body of work is reviewed in Section 4.5 along with an example of use for one of them. Generally, many of them work by ‘transforming’ the irregular data, in some way, so as to fit the regular wavelet transform. Lifting is somewhat different as it can cope directly with the irregular data.

As with wavelets we introduce lifting by reexamining the Haar wavelet transform but presented ‘lifting style’. Suppose we begin with two data points (or scaling function coefficients at some scale), c_1 and c_2 . The usual way of presenting the Haar transform is with equations such as (2.42). However, we could achieve the same result by first carrying the following operation:

$$c_1 \leftarrow (c_1 - c_2)/\sqrt{2}. \quad (2.106)$$

Here \leftarrow is used instead of $=$ to denote that the result of the calculation on the right-hand side of the equation is assigned and overwrites the existing location c_1 . Then taking this *new* value of c_1 we can form

$$c_2 \leftarrow \sqrt{2}c_2 + c_1. \quad (2.107)$$

In lifting, Equation (2.106) is known as the *predict* step and (2.107) is known as the *update* step. The steps can be chained similarly to the Haar transform to produce the full transform. The beauty of lifting is its simplicity. For example, the inverse transformation merely reverses the steps by undoing (2.107) and then undoing (2.106). Many other existing wavelet transforms can be ‘put in lifting form’. The lifting scheme was introduced by Wim Sweldens, see Sweldens (1996, 1997) for example.

A major benefit of lifting is that the idea can be extended to a wider range of data set-ups than described earlier in this book. For example, for irregular data, in several dimensions, it is possible to obtain the detail, or ‘wavelet’ coefficient, for a point in the following way. First, identify the neighbours of such a point and then, using some method, e.g., linear regression on the

neighbours, work out the fitted value of the point. Then the detail is just the fitted value minus the observed value. These multiresolution analyses for irregular data can be used for nonparametric regression purposes but are beyond the scope of the present text. See Jansen et al. (2001), Claypoole et al. (2003), Delouille et al. (2004a,b), Nunes et al. (2006), and the book by Jansen and Oonincx (2005) for further information on lifting in statistics.



<http://www.springer.com/978-0-387-75960-9>

Wavelet Methods in Statistics with R

Nason, G.

2008, X, 259 p., Softcover

ISBN: 978-0-387-75960-9