MATLAB Implementation of W-Matrix Multiresolution Analyses

Man Kam Kwong*

Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439-4844
E-mail: kwong@mcs.anl.gov

Abstract

We present a MATLAB toolbox on multiresolution analysis based on the W-transform introduced by Kwong and Tang. The toolbox contains basic commands to perform forward and inverse transforms on finite 1D and 2D signals of arbitrary length, to perform multiresolution analysis of given signals to a specified number of levels, to visualize the wavelet decomposition, and to do compression. Examples of numerical experiments are also discussed.

1 Introduction

In [3], Kwong and Tang introduced the concept of W-matrices and used them to construct nonorthogonal multiresolution analyses applicable to finite signals of arbitrary length. We refer the readers to Chui [1] and Daubechies [2] and the bibliographies therein for the classical theory of wavelets. Multiresolution analysis is popularized by Mallat [4].

*This work was supported by the Office of Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38.

A W-matrix is generated by a pair of suitably chosen vectors. The family of W-matrices of different sizes generated by the same pair of vectors defines a W-matrix transform (or, more briefly, W-transform) on finite signals represented by vectors. It is simply the multiplication of the vector by a matrix of the appropriate size in the family, followed by splitting up the product vector into its odd-component and even-component vectors. A multiresolution analysis refers to repeated applications of the W-transform to the odd-component output signal. See Section 3 and [3] for more details.

The most important property of a W-matrix, which is crucial to its practical value, is that both the matrix and its inverse have only a small number of nonzero elements in each row and each column.

It was demonstrated in [3] that a particularly useful special case is the quadratic spline (QS) matrix. It is of order 4: each row and column has at most four nonzero elements. By varying six parameters, one can obtain other W-transforms, including the well-known Daubechies D_4 transform of order 4. One may optimize to choose the best transform for a given particular application. W-matrices of any order are possible, but only those of order 4 are covered in this paper.

In the rest of the paper, we present a MATLAB toolbox of the general W-transform of order 4. Our goal is to write highly versatile and user-friendly commands that are suitable for carrying out both interactive and batch experiments. To this end, we design the commands to be able to provide default values to as many input arguments as possible, and to take different actions according to the form of the input arguments. See Section 3.1.

MATLAB toolboxes based on classical orthogonal wavelets are available in the public domain. Examples are TeachWave (David L. Donoho), WaveletTools (Jeffrey C. Kantor), and WavBox (Carl Taswell). They, of course, do not have our new transforms.

Section 2 is an overview of the available commands. It serves also as a tutorial. Section 3 gives detail explanations of the coding of each command. Section 4 presents examples of M-files used for experimentation. Section 5 contains information on how to obtain the toolbox.

2 Overview of Commands

We assume that the readers are familiar with MATLAB, both as an interactive package and as a programming language, and that the M-files of the commands discussed in this paper have been properly installed (see Section 5). The best way to proceed is to invoke a MATLAB session and try the following hands-on tutorial.

First let us generate a sample signal to experiment with.

```
>> t = 0:0.01:1;
>> x = t .* sin(20*t);
```

The first line generates a linear vector `t`, useful as a shorthand in the next line to construct `x` (or additional sample signals in the future). Another interesting sample signal is `x = t .* cos(10*t) .* sin(20*t)`.

The command to invoke a W-transform is `kwt`. In its simplest form

```
>> y = kwt(x);
```

transforms `x` into a new vector `y` using the default QS transform. To see what `kwt` does to `x`, we graph both the input and output vectors using

```
>> plot(x), plot(y)
```

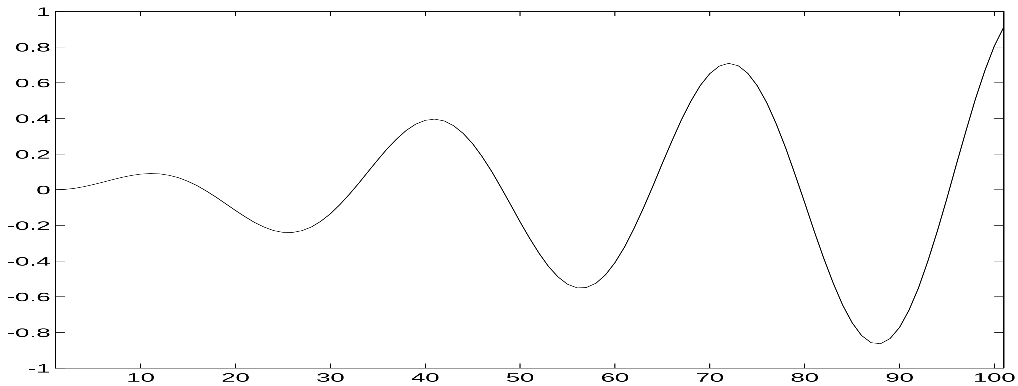

Figure 1. Plot of original signal `x = t sin(20t)`

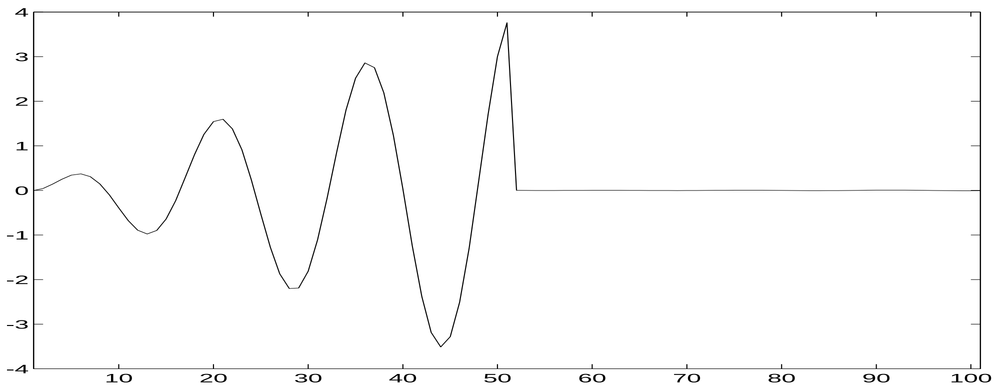

Figure 2. Plot of the QS transform of `x`

One sees clearly that `y` is made up of two parts. Its first half resembles the original signal `x`, while the second half consists of relatively small components. Indeed, we can ask `kwt` to split the output signal into two parts using the following form of the command:

```
>> [y1, y2] = kwt(x);
```

The commands

```
>> length(x), length(y), length(y1), length(y2)
```

give the lengths of `x`, `y`, `y1`, and `y2`; they are 101, 101, 51, and 50, respectively. It is a general rule that the input and output vectors are equal in length. The split output vectors are each half the length of the input vector if it is even; otherwise, `y1` has one more component than `y2`.

The versatility of the command `kwt` lies in its ability to handle correctly input signals of different formats: row or column vectors as well as matrices. In the last case, a two-dimensional W-transform (once in the horizontal direction and once in the vertical direction) is performed to produce four submatrices. If one has a matrix L representing a grayscale image, such as the popular Lena (intensity range: 0–255), one should try

```
>> image(kwt(L)/16), axis('image'), colormap(gray(256))
```

The inverse W-transform is one of the following:

```
>> ikwt(y)
>> ikwt(y1,y2)
```

One should verify that each of these is practically (within the accuracy of numerical error) identical to the original vector x.

Lossy compression of a signal is achieved by discarding small components of y2, or by quantizing. The command

```
>> cy2 = largesta(10, y2);
```

produces a new vector that retains the largest (in absolute value) ten components of y2 and truncates the rest to 0. The actual number of components retained may be different from 10 if more than one component ties for the tenth place. The vector `cy = [y1, cy2]` is now a (lossily) compressed representation of the original signal x. It has only 61 nonzero components. It takes, however, more than 61 data units to record cy, since both the location and magnitude of the nonzero elements of y2 need to be stored. Furthermore, the magnitude of the components of $\mathbf{y}_1$ is no longer within the range assigned to the original signal and thus may require more bits to represent. With more advanced techniques, the result can usually be stored by using just a few extra units in addition to the 61 units. The compression ratio achieved is thus approximately 100/65.

Alternatively, the command

```
>> cy2 = largesta(0.2,y2);
```

retains the largest (in absolute value) two-tenths of the components of $\mathbf{y}_2$. The compression ratio can be raised by lowering the 0.2 threshold, the first input argument to largesta.

Another tool for compression is the command

```
>> quant(y2,5)
```

which quantizes the components of y2 by using a quantization interval of length 5.

A much higher compression ratio is achievable by repeating the compression process on `y1`, once or several more times, leading naturally to the method of multiresolution analysis. In theory, one can continue the process to as many levels as possible until the most recently obtained `y1` is of length 2. In practice, however, after a few levels, further compression produces either intolerable errors or very little additional benefit. To obtain a three-level multiresolution analysis on `x` is simple:

```
>> z = wma(x,3);
```

When `x` is a row vector, the structure of `z` is `[y31 y32 y22 y12]`, where `y31` denotes the third-level `y1` and so on.

Since a multiresolution analysis produces a variable number of wavelets, it is not feasible to ask `wma` to output the separate wavelets automatically. We provide another function to carve out specified wavelets from the output of `wma`. For 1D signals,

```
>> w = maw(z,2);
```

gives the second-level multiresolution analysis wavelet from `z`, obtained above by using `wma`. Similarly,

```
>> v = maw(z,3,1);
```

gives the `y1` vector at the top level of a three-level multiresolution analysis.

For 2D signals, the third input argument specifies which of the four submatrices is to be carved out. One can also call the interactive m-file `wmai` (which is not a function) to do the multiresolution analysis. MATLAB then prompts for the signal and the number of levels.

```
Enter signal to be analyzed: L
Enter number of levels     : 3
```

The results of the three-level analysis are stored in the signals `WL11`, `WL12`, `WL21`, etc.

For visualization,

```
>> wplot(z,3)
```

plots the components `y31, y32, y22, y12` of `z` to give the familiar third-level wavelet coefficient diagram. Since the wavelet coefficients are usually too small to show up significantly on the graph, we use the command

```
>> wplot(z,3,0.01)
```

to magnify the second-level wavelet coefficients $100 = 1/0.01$ times, the third-level coefficients 100^2 times, and so on.

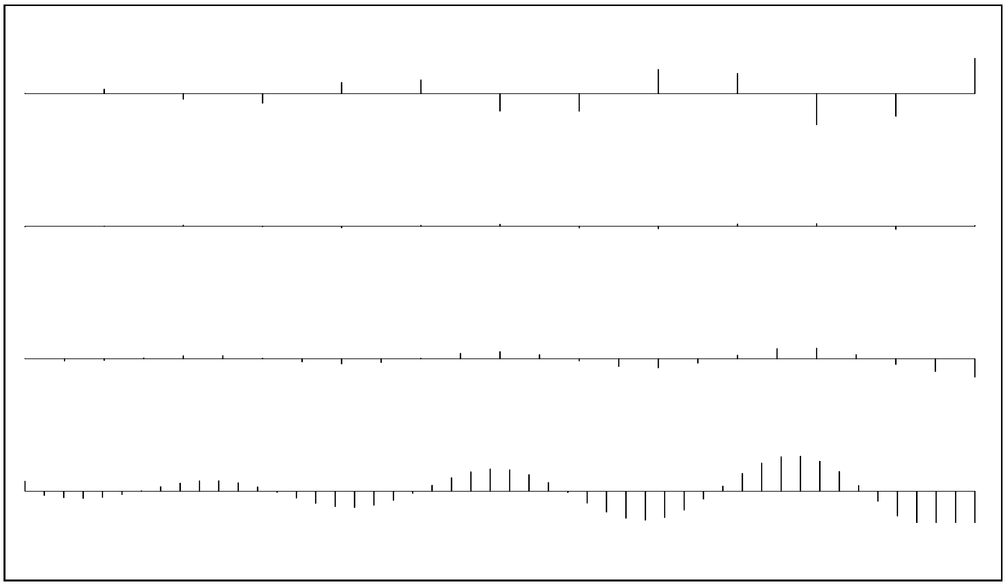

Figure 3. Plot of multiresolution analysis wavelet components

Inverse multiresolution analysis is as expected:

```
>> X = iwma(z,3);
```

The multilevel wavelet representation of the signal must be assembled into a single vector or matrix, such as `z`, before `iwma` is called.

The default W-transform used by `kwt` and `wma` is controlled by a global variable `kw` set by the startup file `startup.m`. Typing `>> kw` reveals the default value `[1 3 3 1]/4`, comprising the coefficients in the 2-scale dilation equation for the wavelet associated with the QS transform. In [3], the QS transform is defined using `[1 3 3 1]`, but we discover that, in practice, it is more convenient to use one-fourth of the vector. This only affects the transform by a scaling. By setting `kw` to other values, such as

```
>> kw = [1, 2, 2, 1]
```

one can choose a different W-transform. The particular choice `kw = kwdau` (`kwdau` is another vector automatically set by the startup file `startup.m`) gives the Daubechies transform of order 4. Our implementation of the Daubechies transform differs from the conventional one in the way it treats the endpoint components; see [3]. Unlike `kwqs` and `kw1`, the vector `kwdau` has six components.

```
kwdau = [0.4830 0.8365 0.2241 -0.1294 0.2679 -3.7321]
```

A second method to call a specific W-transform, without altering the default value of `kw`, is to supply the parameter vector as an additional argument to the function invoked. The commands

```
>> y = kwt(x,kwdu);
>> y = wma(x,kwdu,3);
```

use the Daubechies transform but retain the QS transform as the default for future calls.

The decomposition effected by a W-transform is in general not orthogonal. In [3], it is shown that for the QS transform this is not a real drawback in practice. For most reasonable signals, the error incurred by discarding small components of $\mathbf{y}_2$ is on the same order of magnitude as the optimal compression obtainable by using orthogonal decomposition. For the perfectionist, we provide an *orthogonal compensation* procedure that can further improve the compression performance. Let `e` be the part of `y2` to be discarded, for instance,

```
>> e = y2 - cy1;
```

where `cy1 = largesta(0.1, y2)` as defined before. Then

```
>> cy1 = oc(y1,e);
```

produces a vector `cy1` to be used in place of `y1`, in such a way that the pair `(cy1, cy2)` gives a better approximation to `x` than the pair `(y1, cy2)`.

The commands mentioned above provide an adequate set of building blocks for writing more complicated M-files for experimentation. Examples given in Section 4 include M-files to compare two W-transforms, to compress 2D images, and to search for an optimal W-transform. Other commands in the toolbox include `sf`, which constructs the scaling function; `wf`, which constructs the wavelet function, and `sdil`, which solves a 2-scale dilation equation.

3 The MATLAB Programs

In this section, we present the MATLAB codes of the commands given in Section 2. The algorithms are explained in detail for the benefit of those who need to adapt the codes to other languages.

Each command is contained in a separate MATLAB M-file. We assume that the readers are familiar with the rudiments of MATLAB programming. The commands have not been written in the most robust form; rigorous checking for correct formats of input arguments is lacking in most cases. Comments and feedback are appreciated.

When listing our M-files, we leave out the comments, except those pertaining to the usage of the command. For reference, we have added line numbers.

3.1 User-Friendliness

In this subsection, we discuss our goal and standard for writing user-friendly MATLAB commands. Our objective is to help readers understand the design of the programs presented below. Fortunately, the MATLAB language is powerful enough to allow us to implement most of what we wanted.

The ability to allow some input arguments to be omitted and to assume default values greatly simplifies the syntax of the commands when used in common situations. This feature is especially helpful to beginners.

Two techniques are employed to assign default values. The global variable `kw` is used to designate the default W-transform used by `kwt`, `ikwt`, `wma`, etc. A user can interactively change the default by assigning new values to `kw`. Since many global variables can be dangerous, however, we refrain from using more. Instead, for other default arguments, such as the number of levels used by `wma` and the quantization level used by `quant`, the default values are built into the programs themselves. A user cannot conveniently change a default value without modifying the M-file. We believe that such modification is not difficult to do, and we have supplied easy-to-follow instructions in the M-files using comments.

If possible, we make a command accept an argument in any reasonable format. For instance, `kwt` does not fuss if the input signal `x` is a row or a column vector and will produce a row or column output vector accordingly.

In a similar vein, we attempt to make a command more flexible in accommodating the order of the input arguments. For instance, `wma(4,x)` and `wma(x,4)` are both permitted and produce the same answer. Another example is `largesta(4,y)` and `largesta(y,4)`.

We also try to pack more functionality into a single command. The actual action and output of a command can be different depending on the format of the input or output arguments. Some examples follow.

- Most toolboxes have different commands for 1D and 2D transforms. Our command `kwt` will perform a 1D or a 2D transform according to whether the input signal is 1D or 2D. Also `kwt` will separate out the wavelet vectors/matrices if there are two output arguments for 1D signals or four output arguments for 2D signals.
- The command `ikwt(y1,y2)` or `ikwt(y1,y2,y3,y4)` first assembles the input vector/matrices into a single vector/matrix before performing the inverse W-transform.
- The command `largesta(r,y)` will extract the largest `r` components of `y` if `r` is greater than 1 but the largest fraction `r` of the total number of components if `r` is less than 1.
- The command `quant(y,10)` quantizes the components of `y` to the nearest 10, 20, etc., while `quant(y,'10')` quantizes the components of `y` into approximately 10 levels.

Error-checking is part of being user-friendly. We repeat our earlier remark that we have not spent sufficient effort on perfecting this aspect of our commands (we will do our best when time permits). We believe that the commands are sufficiently easy to use that few users will really need extensive error-checking.

3.2 The Startup File

```
   %%% startup.m

 1 global kw
 2 kwqs = [ 1 3 3 1 ]/4;
 3 kw1 = [ 1 2 2 1 ];
 4 kw =kwqs;
 5 dr = sqrt(3);
 6 dau = [1 dr 2*dr-3 dr-2];
 7 ndau = dau/norm(dau);
 8 wdau = ndau(4:-1:1).*[1 -1 1 -1];
 9 kwdau = [ ndau ndau(3)/wdau(3) ndau(1)/wdau(1)];
10 clear dr dau ndau wdau
```

The commands in this file are to be appended to those in one's startup file, which usually contains path setting and other routine startup commands.

These commands declare a global variable `kw`; define three variables `kwqs`, `kwdau`, and `kw1`; and set `kw` initially to `kwqs`. The global variable `kw` contains the parameter vector that determines the default W-transform used by the commands `kwt`, `wma`, and their families. The variables `kwqs`, `kwdau`, and `kw1` are parameter vectors for the QS, Daubechies, and a third sample transform, respectively. A general parameter vector has six components, as explained in Section 3.3. The fourth and fifth components, when omitted, take the default values 1 and -1, respectively. When one has developed her own favorite W-transforms, their corresponding parameter vectors can be added to the list, simply following the format of line 2 or 3 to define more variables. Line 10 clears the temporary variables used only to construct `kwdau`.

If these variable names conflict with ones normally used in one's own workspace, the reader can change his own notations or modify all the M-files in this paper accordingly.

3.3 The W-transform `kwt`

According to the general theory outlined in [3], an arbitrary vector $h = [h_1, h_2, h_3, h_4]$ and two constants c and d are first chosen. The vector

$$g = [g_1, g_2, g_3, g_4] = [h_1/c, h_2/c, h_3/d, h_4/d] \tag{1}$$

together with h then form a basic pair of vectors that are used to construct the W-matrices (of even and odd sizes, respectively):

$$\mathbf{W} = \begin{pmatrix} g_1+g_2 & g_3 & g_4 & & & & & \\ h_1+h_2 & h_3 & h_4 & & & & & \\ & g_1 & g_2 & g_3 & g_4 & & & \\ & h_1 & h_2 & h_3 & h_4 & & & \\ & & & g_1 & g_2 & g_3 & g_4 & \\ & & & h_1 & h_2 & h_3 & h_4 & \\ & & & & & & \ddots & \\ & & & & & g_1 & g_2 & g_3+g_4 \\ & & & & & h_1 & h_2 & h_3+h_4 \end{pmatrix} \tag{2}$$

and

$$\mathbf{W} = \begin{pmatrix} g_1+g_2 & g_3 & g_4 & & & & & \\ h_1+h_2 & h_3 & h_4 & & & & & \\ & g_1 & g_2 & g_3 & g_4 & & & \\ & h_1 & h_2 & h_3 & h_4 & & & \\ & & & g_1 & g_2 & g_3 & g_4 & \\ & & & h_1 & h_2 & h_3 & h_4 & \\ & & & & & & \ddots & \\ & & & & & g_1 & g_2 & g_3 & g_4 \\ & & & & & h_1 & h_2 & h_3 & h_4 \\ & & & & & & & h_1 & h_2+h_3+h_4 \end{pmatrix}. \tag{3}$$

The corresponding W-transform `kwt(x)` of a column vector `x` is computed by choosing a $\mathbf{W}$ of the appropriate size, forming the matrix product $\mathbf{W}\mathtt{x}$, and then separating out the odd and even components of the resulting vector.

$$\mathtt{y1} = \text{odd components of } \mathbf{W}\mathtt{x} \tag{4}$$

$$\mathtt{y2} = \text{even components of } \mathbf{W}\mathtt{x} \tag{5}$$

We define kwt(x) to be either the pair y1 and y2 or the vector obtained by appending y2 to y1.

For a row vector x, kwt(x) is defined as the transpose of the transform of the transpose of x. For a matrix x, its 2D W-transform is obtained by first applying the 1D W-transform to each column of x and then applying the same W-transform to each row of the resulting matrix.

Most researchers in the wavelet community are, however, more familiar with another vector, which comprises the coefficients of the dilation equation and is related to h by

$$\overline{g} = [h_4, -h_3, h_2, -h_1]. \tag{6}$$

We program kwt to use an input argument k, obtained by appending the constants c and d (if they are different from the default values of 1 and -1, respectively) to $\overline{g}$, to designate the choice of the W-transform. When k is omitted, its value is taken from the global variable kw. Initially, the default parameter vector is kw = [1 3 3 1]/4, set by the startup.m file. Although any choice of k will lead to a workable transform, only a carefully chosen one will result in a transform appropriate for compression.

```
%        y    = kwt(x)            default W-transform
%   [y1, y2] = kwt(x)             1D row/column x
%   [y1, y2, y3, y4] = kwt(x)     2D matrix x
%        y    = kwt(x,k)          general W-transform
%                                   determined by k

function [y, yw, y3, y4] = kwt(x,k)

if nargin == 1
  global kw
  if exist('kw') ~= 1, k = [1 3 3 1]/4; else k = kw; end
end

ss=size(x);
if ss(1) == 1, x = x(:);
elseif ss(2) > 1
  if nargout == 4, [y, yw, y3, y4] = kwt2(x,k);
  else y = kwt2(x,k); end
  return
end
```

```
13  if length(k) == 4, c=1; d=-1;
14  elseif length(k) == 5, c=k(5); d=-1;
15  elseif length(k) == 6, c=k(5); d = k(6);
16  else   k = [1 3 3 1]/4; c=1; d=-1;
17  end

18  C=-k(4)/c; D=k(3)/c; A=-k(2)/d; B=k(1)/d;

19  s=size(x);
20  n=floor(s(1)/2);
21  x1=x(1:2:s(1),:);
22  x2=x(2:2:s(1),:);

23  if s(1) == 2*n
24  y=[ (C+D)*x1(1,:)+A*x2(1,:)+B*x1(2,:)
25       C*x2(1:n-2,:)+D*x1(2:n-1,:)+A*x2(2:n-1,:)+B*x1(3:n,:)
26       C*x2(n-1,:)+D*x1(n,:)+(A+B)*x2(n,:) ];
27  yw=[ c*(C+D)*x1(1,:)+d*(A*x2(1,:)+B*x1(2,:))
28       c*(C*x2(1:n-2,:)+D*x1(2:n-1,:))+d*(A*x2(2:n-1,:) ...
29       +B*x1(3:n,:))
30       c*(C*x2(n-1,:)+D*x1(n,:))+d*(A+B)*x2(n,:) ];

31  else
32  y=[ (C+D)*x1(1,:)+A*x2(1,:)+B*x1(2,:)
33       C*x2(1:n-1,:)+D*x1(2:n,:)+A*x2(2:n,:)+B*x1(3:n+1,:)
34       C*x2(n,:)+(A+B+D)*x1(n+1,:) ];
35  yw=[ c*(C+D)*x1(1,:)+d*(A*x2(1,:)+B*x1(2,:))
36       c*(C*x2(1:n-1,:)+D*x1(2:n,:))+d*(A*x2(2:n,:) ...
37       +B*x1(3:n+1,:)) ];
38  end

39  nargout <= 1
40    if ss(1) == 1, y = [y' yw'];
41    else y = [y; yw]; end
42  elseif ss(1) == 1,
43    y = y'; yw = yw';
44  end
```

Line 1 declares `kwt` to be a function that has at most two input arguments and at most four output arguments. The case of having four output arguments occurs when `x` is a matrix and the user wants the 2D transform to be separated out into four submatrices.

Lines 2–5 check whether the argument `k` is present. If not, its value is then taken from the global variable `kw`. To take care of the case in which the global variable `kw` may have been inadvertently erased, line 4 sets `kw` to the default QS transform.

Lines 6–12 examine the format of `x` and choose the various paths. Line 7 changes `x` to a column vector if it is input as a row vector. (Lines 42 and 45 convert the output back into a row vector to match the input format.) Line 8 checks whether `x` is a 2D matrix. If it is, the work is delegated to the M-file `kwt2.m`. A user has no need to use `kwt2` directly; it is provided merely to make the coding easier to understand. We omit the listing of `kwt2.m`.

Lines 13–17 check the format of the parameter vector `k,` supplying the default values for c and d if necessary.

Line 18 computes the components of $\overline{g}$ `= [ C, D, A, B ]`. The names used to denote the components have been retained from previous versions of the program.

Lines 21 and 22 extract the odd and even components of `x`. These are used to simplify the computation of the W-transform.

Line 23 checks whether the length of `x` is even. If so, lines 24–30 compute the transform. Note that in the implementation of the W-transform, we do not use matrix multiplication. Lines 32–37 compute the transform for odd-length `x`.

Finally, lines 39–44 put the output in the correct format according to the number of output arguments requested and the format of the input signal.

3.4 The Inverse W-transform `ikwt`

```
%  y = ikwt(x)              default inverse W-transform
%  y = ikwt(x1,x2)          1D x1, x2
%  y = ikwt(x1,x2,x3,x4)    2D matrices
%  y = ikwt(x,k)            general inverse W-transform

function y=ikwt(x1,x2,x3,x4,k)

ss=size(x1);

if nargin == 1
  if ss(1) == 1, x = x1(:);
  elseif ss(2) > 1 & nargin == 1, y = ikwt2(x1); return
  else x = x1;
  end
elseif nargin == 2
  if min(ss) > 1
    y = ikwt2(x1,x2); return
  elseif size(x2,1) == 1 & size(x2,2) < 7 & size(x2,2) > 3
    x = x1(:); k = x2;
  else x = [x1(:); x2(:)];
  end
elseif nargin == 3
  x = [x1(:); x2(:)]; k = x3;
elseif nargin == 4
  y = ikwt2([x1 x2; x3 x4]); return
elseif nargin == 5
  y = ikwt2([x1 x2; x3 x4],k); return
end

if exist('k') ~=1
  global kw
  if exist('kw') ~= 1, k = [1 3 3 1]/4; else k = kw; end
end

if length(k) == 4, c=1; d=-1;
elseif length(k) == 5, c=k(5); d=-1;
elseif length(k) == 6, c=k(5); d = k(6);
else   k = [1 3 3 1]/4; c=1; d=-1;
end
```

```
C=-k(4)/c; D=k(3)/c; A=-k(2)/d; B=k(1)/d;
DD = (A*D-B*C);

s=size(x); n=floor(s(1)/2);

if s(1) == n*2
  x1=x(1:n,:);
  x2=x(n+1:s(1),:);
  y1=[ (d*x1(1,:)-x2(1,:))/(C+D)
       (C*(c*x1(1:n-1,:)-x2(1:n-1,:))...
       +A*(d*x1(2:n,:)-x2(2:n,:)))/DD ]/(d-c);
  y2=[ (D*(-c*x1(1:n-1,:)+x2(1:n-1,:))...
       +B*(-d*x1(2:n,:)+x2(2:n,:)))/DD
       (-c*x1(n,:)+x2(n,:))/(A+B) ]/(d-c);
  y=zeros(s);
  y(1:2:2*n,:)=y1; y(2:2:s(1),:)=y2;

else
  x1=x(1:n+1,:);
  x2=x(n+2:s(1),:);
  y1=[ (d*x1(1,:)-x2(1,:))/(C+D)
       (C*(c*x1(1:n-1,:)-x2(1:n-1,:))...
       +A*(d*x1(2:n,:)-x2(2:n,:)))/DD ]/(d-c);
  DDD = A*D+A^2+A*B-B*C;
  y1=[y1; (C*(c*x1(n,:)-x2(n,:))/(d-c)+A*x1(n+1,:))/DDD];
  y2=(D*(-c*x1(1:n-1,:)+x2(1:n-1,:))...
       +B*(-d*x1(2:n,:)+x2(2:n,:)))/DD/(d-c);
  y2=[y2; (-(D+A+B)*(c*x1(n,:)-x2(n,:))/(d-c)...
       -B*x1(n+1,:))/DDD];
  y=zeros(s);
  y(1:2:s(1),:)=y1; y(2:2:s(1),:)=y2;
end

if ss(1) == 1, y = y'; end
```

A remarkable and useful property of a W-matrix is that its inverse is also generated by a pair of vectors of length 4. Furthermore, all W-matrices of even, or odd, sizes have the same structure; they differ only in the number of pairs of basic vectors used.

Using MAPLE, we are able to obtain the formula for the inverse matrix of a general W-matrix. The algorithm for computing inverse W-transform consists of merging `y1` and `y2` by interlacing their components and then multiply by the inverse W-matrix.

To make `ikwt` more user-friendly, we allow the input of 1D signals either as two separate vectors `y1` and `y2` or as one single vector obtained by appending `y2` to `y1`, and the input of 2D images either as four separates matrices or as one single matrix. The parameter vector `k` may be supplied as an argument or omitted (`kw` is used as default). Ambiguity arises and *CAUTION* must be exercised when there are two input arguments each of length between 4 and 6, since the second argument can be interpreted either as `y2` or as `k`. In such a case, the latter interpretation prevails.

Lines 3–21 examine the number of input arguments and choose the appropriate path, in particular, delegating the work to `ikwt2` if there is indication that the input signal is a matrix. The listing of `ikwt2` is omitted.

Lines 22–32 check the existence and format of `k` and extract from it the coefficients used in the inverse transform.

Lines 34–44 perform the inverse transform on even-lengthed signals and lines 46–58 on odd-lengthed signals. The formulas used are slightly more complicated than the corresponding ones for the forward transform. This is because the inverse matrix has a factor the reciprocal of the determinant of the W-matrix and components in both the first two rows and the last two rows have rather involved expressions. Other than that, the formulas are straightforward. Line 60 puts the output in a row vector if the input signal is a row vector.

3.5 The W-Multiresolution Analysis `wma`

The command `wma` carries out a multiresolution analysis on a signal `x` up to `n` levels using the parameter vector `k`. As usual, the default `k` is taken from `kw`. The default `n` is 3. The arguments `n` and `k` can be input in either order.

```
    %  y = wma(x)         default W-multiresol. anal., n = 3
    %  y = wma(x,n)       n-level analysis
    %  y = wma(x,n,k)     general W-multiresol. anal.
    %
    %  [y, ind] = wma(x)  ind can be used to separate out the
    %                       wavelet vectors

 1  function [y, ind] = wma(x,n,k)

 2  if nargin == 1, n = 3;
 3  elseif nargin ==2, if length(n) > 1, k = n; n = 3; end
 4  else  if length(k)==1, tmp = n; n = k; k = tmp; end
 5  end

 6  if exist('k') ~=1
 7    global kw,
 8    if exist('kw') ~= 1, k = [1 3 3 1]/4; else k = kw; end
 9  end

10  ss = size(x); ind = ss;

11  for ii = 1:n
12    y(1:ss(1),1:ss(2)) = kwt(x,k);
13    ss = ceil(ss/2); ind = [ss; ind];
14    x = y(1:ss(1),1:ss(2));
15  end
```

Line 1 indicates that the function has an optional output `ind`, which gives the indices marking the ends of the various segments of the transformed signal at all the levels. For instance, if `ind` is the matrix

```
[ 1 13; 1 26; 1 51; 1 101 ],
```

then the first-level `y2` is made up of the 52-nd to the 101-st components, the second-level `y2` is of the 27-th to the 51-st components, the third-level `y2` of the 14-th to the 26-th components, and finally the third-level `y1` of the 1-st to the 13-th components. That the first column of `ind` are all 1 indicates that the signal is one dimensional. If `x` is a matrix, the first column of `ind` will give the corresponding indices in the vertical direction.

Lines 2–5 check the number and format of the input arguments, figuring out which argument is `n` and which is `k` and supplying the default value for `n` if needed. Lines 6–9 supply the default value for `k` if needed.

Lines 11–16 form the main loop that calls `kwt` to perform the transform and then find the size and signal to be transformed at the next level.

3.6 The Inverse W-Multiresolution Analysis `iwma`

The inverse W-multiresolution analysis is similar to the forward analysis except that `ikwt` is invoked instead of `kwt`, and the highest level is dealt with first. The arguments `n` and `k` can be input in either order.

The various wavelet vectors/submatrices must be first assembled into one single signal `x` before input into the function.

```
   %  y = iwma(x)       default inverse W-multiresol. transf.
   %                      n = 3
   %  y = iwma(x,n)     n-level default inverse W-ma
   %  y = iwma(x,n,k)   general inverse W-ma

function y = iwma(x,n,k)

if nargin == 1, n = 3;
elseif nargin == 2, if length(n) > 1, k = n; n = 3; end
else  if length(k)==1, tmp = n; n = k; k = tmp; end
end

if exist('k') ~=1
  global kw,
  if exist('kw') ~= 1, k = [1 3 3 1]/4; else k = kw; end
end

ss(n,:) = size(x);
for ii = n-1:-1:1
  ss(ii,:) = ceil(ss(ii+1,:)/2);
end

for ii = 1:n
  x(1:ss(ii,1),1:ss(ii,2)) = ...
     ikwt(x(1:ss(ii,1),1:ss(ii,2)),k);
end
y = x;
```

3.7 Extracting Multiresolution Analysis Wavelets `maw`

```
    %  z = maw(y,j)      j-th level wavelet vector for 1D y
    %  z = maw(y,j,1)    (top) j-th level y1 vector for 1D y
    %
    %  z = maw(y,j,h)    h-th submatrix (out of 4) at j-th level
    %                      for 2D y

 1  function z = maw(y,j,h)

 2  ind = size(y);

 3  if min(ind) == 1
 4    ind = length(y);
 5    for i = 1:j-1, ind = ceil(ind/2); end
 6    if nargin == 2, z = y(ceil(ind/2)+1:ind);
 7    else z = y(1:ceil(ind/2)); end
 8  else

 9    for i = 1:j-1, ind = ceil(ind/2); end
10    ind0 = ceil(ind/2);
11    if ~exist('h'), h = 1; end

12    if h == 1,     z = y(1:ind0(1),1:ind0(2));
13    elseif h == 2, z = y(1:ind0(1),ind0(2)+1:ind(2));
14    elseif h == 3, z = y(ind0(1)+1:ind(1),1:ind0(2));
15    else z = y(ind0(1)+1:ind(1),ind0(2)+1:ind(2));
16    end
17  end
```

For one-dimensional signals, we need only one index to specify the level at which to extract the wavelet. Thus `maw(y,3)` gives the level 3 wavelet. Supply a second index if one requires the top-level `y1` vector.

For two-dimensional signals, we need one index to specify the level and another index to specify which of the four submatrices is needed.

Lines 3–7 handle the 1D case and lines 8–17 the 2D case.

3.8 Interactive Multiresolution Analysis wmai

```
%%% wmai.m

SIGNAL = input('Enter signal to be analyzed: ','s');
LEVELS = input('Enter the number of levels : ');
if length(LEVELS) == 0, LEVELS = 3; end
SIZES = 0; eval([ 'SIZES = size(' SIGNAL ');' ])
if min(SIZES) == 1,
  eval([ '[W' SIGNAL '11, W' SIGNAL '12] = ...
       kwt(' SIGNAL ');' ] )
else
  eval([ '[W' SIGNAL '11, W' SIGNAL '12, W' SIGNAL '13,  ...
       W' SIGNAL '14] = kwt(' SIGNAL ');' ] )
end
for ii = 2:LEVELS
  II = int2str(ii);
  if min(SIZES) == 1,
    eval([ '[W' SIGNAL II '1, W' SIGNAL II '2] = ...
         kwt(W' SIGNAL int2str(ii-1) '1);' ] )
  else
    eval([ '[W' SIGNAL II '1, W' SIGNAL II '2, ...
         W' SIGNAL II '3, W' SIGNAL II '4] = ...
         kwt(W' SIGNAL int2str(ii-1) '1);' ] )
  end
end
```

The MATLAB command `eval` is used to concatenate strings to construct variable names for the wavelet vector/matrix.

A disadvantage of using a script, such as `wmai`, instead of a function is that a few variables, such as `SIGNAL` and `LEVELS`, will be left dangling in the workspace.

Beware of conflicting variable names.

3.9 Visualization `wplot` and `lbar`

Using `wplot`, one can visualize the top-level `y1` and the `y2` of all levels in the traditional plot (Figure 3) used by many wavelet researchers. Built into `wplot` is the same procedure used by `maw` to carve out the various segments from the input signal (lines 13–16). Each segment is then plotted in a subplot using the subroutine `lbar`.

```
    % wplot(x)        plot 3-level multiresol. anal.
    % wplot(x,n)      plot n-level multiresol. anal.
    % wplot(x,n,r)    use r < 1 as magnification factor
    % wplot(x,n,r,c)  use color c

 1  function wplot(x,n,r,c)

 2  if exist('n') ~= 1, n = 3; r = 1;
 3  elseif exist('r') ~=1, if n > 1, n = ceil(n); r = 1;
 4                         else r = n; n = 3; end
 5  else if n > 1, n = ceil(n);
 6       else tmp = r; r = n; n = max(2,tmp); end
 7  end
 8  if exist('c') ~= 1, c = 'y'; end

 9  ind = length(x);
10  mx = min(min(x),0); Mx = max(max(x),0);
11  figure(gcf)

12  for ii = n+1:-1:1
13    if ii > 2, xx = x(ceil(ind/2)+1:ind)/r^(ii-2);
14    elseif ii > 1, xx = x(ceil(ind/2)+1:ind);
15    else xx = x(1:ind); end
16    ind = ceil(ind/2);

17    h = subplot(n+1,1,ii);
18    lbar(xx,c)
19    axis([ 1 length(xx) mx Mx ])
20    axis off
21    rect = get(h,'pos');
22    set(h,'pos',[ rect(1) rect(2)-(n+1-ii)*rect(4)/10 ...
23        rect(3) rect(4) ])
24  end
```

Lines 2–8 check the input arguments. The argument `n` specifies the number of levels (defaults to 3) and argument `r` is a magnification factor used for lower level wavelets; `n` and `r` can be input in either order. The argument `c` specifies the color (defaults to yellow) of the plot and can be input only after `n` and `r`.

Line 10 determines the lower and upper limits on the vertical axis used by each subplot. Line 11 raises the current figure to the foreground.

Lines 17–20 do the plotting and adjust the axis limits. Lines 21–23 move the subplots closer together for aesthetic reasons.

By convention, a wavelet plot is constructed by drawing a vertical line at each integral point on the horizontal with height proportional to the appropriate component of the wavelet vector. In essence, it is a bar graph in which each bar is a vertical line. MATLAB does not seem to have a command to draw such a plot, and we therefore write our own. Expecting it to be of interest in other contexts, we make the command `lbar` slightly more general. It can take two optional arguments, one to specify the range to be labeled on the horizontal axis, and another to specify the color of the graph.

```
    % lbar(y)        plot y as line-bar graph
    % lbar(x,y)      plot y VS x as line-bar graph
    % lbar(x,y,c)    use color c

 1  function lbar(x,y,c)

 2  n = length(x);

 3  if nargin == 1, y = x(:); x = (1:n)'; c = 'y';
 4  elseif nargin == 2,
 5    if length(y) == 1, c = y; y = x(:); x = (1:n)';
 6    else x = x(:); y = y(:); c = 'y'; end
 7  end

 8  n = 3*n;
 9  xx = zeros(n,1); yy = xx;
10  xx(1:3:n) = x; xx(2:3:n) = x; xx(3:3:n) = x;
11  yy(2:3:n) = y;
12  plot(xx,yy,c)
```

3.10 Compression Utilities `largesta` and `quant`

```
%   y      = largesta(n,x)   n > 1, largest (absolute value)
%                                   n components of x
%  [y, N] = largesta(r,x)   r < 1, largest fraction r
%                            N = number of components retained

function [ y, N ] = largesta(n,x)

if length(n) > 1, tmp = n; n = x(1); x = tmp; end
ax = abs(x);
M = max(max(ax));
a = 0; b = M; n1 = prod(size(x(:))); n2 = nnz(ax == M);

if  n == 1, y = x; n = n1; return
elseif n <= 0, y = 0*x; n = 0; return
elseif n < 1, n = min(n1-1,round(n1*n));
end

while n1 ~= n & n2 ~= n & n1 ~= n2 & b-a > 0.000001*M
  c = (a+b)/2;
  nc = nnz(ax >= c);
  if nc > n, a = c; n1 = nc;
  else       b = c; n2 = nc;
  end
end
if n1 == n
  small = find(ax < b);
  % y = x.*(ax > b);
else
  small = find(ax < c);
  % y = x.*(ax > c);
end

y = x;
y(small) = zeros(size(small));
N = nnz(y);
```

The command `largesta` retains the largest (in absolute values) `n` elements of the matrix `x` and truncates the rest to 0. If `n` is larger than 1, it is used as the actual count of elements to be retained. If `n` is less than or equal to 1, it is interpreted as the fraction of the total number of elements to be retained. The number of elements actually retained may differ from the requested number. The algorithm uses a bisection procedure. A sister command `largest` retains the largest (in algebraic value) elements.

The command `quant(x,n)` quantizes the components to take values in the set $\{0, \pm n, \pm 2n, \pm 3n, \cdots\}$. If `n` is a string containing a number, quant will produce approximately `n` quantization levels. If `n` is not specified, approximately ten quantization levels will be used.

```
%    y    = quant(x)      quantize x (approximately 10 levels)
%    y    = quant(x,q)    use q as quantization interval
%  [y, Q] = quant(x,'n')  use approximately n levels
%                         Q = actual quantization interval

function [y, q] = quant(x,q)

if nargin == 1
  q = '10';
elseif length(x) == 1 | isstr(x)
  tmp = x; x = q; q = tmp;
end

if isstr(q)
  eval([ 'q = ' q ';' ])
  r = (max(max(x)) - min(min(x)))/q*1.5;
  p = floor(log(r)/log(10));
  q = floor(r/10^p)*10^p;
end

y = round(x/q)*q;
```

In principle, neither `largesta` nor `quant` is the ultimate utility for automating compression. The ideal utility can automatically determine the cutoff threshold, such as the article `n` or `r` to `largesta` and the article `q` to `quant`, based upon a specified compression ratio or compression performance.

3.11 Orthogonal Compensation oc

The W-transform associated with a W-matrix W can be summarized in the equations

$$\mathtt{y} = \mathbf{W}\mathtt{x}, \quad \mathtt{x} = \mathbf{W}^{-1}\mathtt{y}. \tag{7}$$

The odd and even components of $\mathtt{y}$ form the pair of vectors

$$\mathtt{y1} = [y_{11}, y_{12}, \cdots]', \quad \mathtt{y2} = [y_{21}, y_{22}, \cdots]'. \tag{8}$$

Let us denote the columns of the matrix $\mathbf{W}^{-1}$ as

$$\left[\begin{array}{ccccc} \mathbf{v}_1 & \mathbf{w}_1 & \mathbf{v}_2 & \mathbf{w}_2 & \cdots \end{array} \right]. \tag{9}$$

Then the second equation in (7) has the equivalent form

$$\mathtt{x} = (y_{11}\mathbf{v}_1 + y_{12}\mathbf{v}_2 + \cdots) + (y_{21}\mathbf{w}_1 + y_{22}\mathbf{w}_2 + \cdots). \tag{10}$$

This equation suggests that the W-transform can be interpreted as the decomposition of $\mathtt{x}$ along the subspaces G and H, spanned by $\mathbf{v}$ and $\mathbf{w}$, respectively.

In the analogous interpretation of the Haar and Daubechies D_4 transforms, the linear subspaces G and H are orthogonal to each other. In addition, the one-dimensional subspaces generated by all the $\mathbf{v}_i$ and $\mathbf{w}_i$ are mutually orthogonal. When some of the components in $\mathtt{y2}$ are discarded, the compressed vector is then the unique signal, in the space spanned by the remaining base vectors, that best approximates the original signal.

For a general W-transform, G and H are not necessarily orthogonal. This fact seems to argue against the use of general W-transforms. In practice, a reasonable signal (one that is not wildly oscillating or badly degraded by noise) usually has such small coefficients in the H subspace decomposition that even if we do not take additional steps to optimize the approximation, the error incurred in simply discarding them is negligible. This is true, in particular, for the QS transform.

We give below the method of orthogonal compensation to enhance the approximation when discarding some of the components of $\mathtt{y2}$. Let $\mathbf{d}$ be the vector to be discarded. It is likely to be a partial sum of the expression in the second pair of parentheses in (10). We decompose $\mathbf{d}$ into a linear combination of the vectors $\mathbf{v}_i$ and an error vector that is orthogonal to G:

$$\mathbf{d} = (a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots) + \mathbf{e}. \tag{11}$$

After determining a_i, they are added to the corresponding y_{1i}, so that the actual part that is discarded is $\mathbf{e}$, which is orthogonal to G. To this end, we take inner products of $\mathbf{d}$ with each of $\mathbf{v}_i$. One can easily verify that a_i is the solution to the tridiagonal system of linear equations

$$\begin{pmatrix} \alpha_{11} & \alpha_{12} & & & \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & & \\ & \alpha_{32} & \alpha_{33} & \alpha_{34} & \\ & & & \ddots & \\ & & & & \ddots \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ \vdots \end{pmatrix} = \begin{pmatrix} <\mathbf{d},\mathbf{v}_1> \\ <\mathbf{d},\mathbf{v}_2> \\ <\mathbf{d},\mathbf{v}_3> \\ \vdots \\ \vdots \end{pmatrix}, \tag{12}$$

where $\alpha_{ij} =< \mathbf{v}_i, \mathbf{v}_j >$. The matrix in the equation is square, with width equal to the length of `y1`.

```
% y = oc(y1,ye)      orthogonal compensation, default W-tranf.
% y = oc(y1,ye,k)    general W-transform

function y = oc(y1,ye,k)

if nargin == 2
  global kw
  if exist('kw') ~= 1, k = [1 3 3 1]/4; else k = kw; end
end

if length(k) == 4, c=1; d=-1;
elseif length(k) == 5, c=k(5); d=-1;
elseif length(k) == 6, c=k(5); d = k(6);
else   k = [1 3 3 1]/4; c=1; d=-1;
end

k1=k(1); k2=k(2); k3=k(3); k4=k(4);
h1=-k1/d; h2=-k2/d; h3=-k3/c; h4=-k4/c;
```

```
s1 = size(y1); se = size(ye);
y1 = y1(:); ye = ye(:);
m = length(y1); n = length(ye);

A = k1*k3+k2*k4;
B = k1^2+k2^2+k3^2+k4^2;
C = k3*k1+k4*k2;
B1 = (k2*k3-k1*k4)^2/(k2-k1)^2 +k3^2+k4^2;
Bn = k1^2+k2^2+(k2*k3-k1*k4)^2/(k4-k3)^2;

D = k1*h3+k2*h4;
E = k1*h1+k2*h2+k3*h3+k4*h4;
F = k3*h1+k4*h2;
E1 = -((k2*k3-k1*k4)/(k2-k1)/c)^2*d+k3*h3+k4*h4;

if m == n
  En = k1*h1+k2*h2-((k2*k3-k1*k4)/(k4-k3)/d)^2*c;
  z = ([ 0;D*ye(1:n-1) ] ...
       + [ E1*ye(1);E*ye(2:n-1);En*ye(n) ] ...
       + [ F*ye(2:n);0 ]);
  GH = toeplitz([ B A zeros(1,n-2) ],...
       [ B C zeros(1,n-2) ]);
  GH(1,1) = B1;  GH(n,n) = Bn;

else
  r = -h2*h3+h1*h4;
  r = (r/(r-h1*h2+h2^2))^2;
  Dn = (c-d)*r*( (h1-h2+h3)*h1 - h2*k4 );
  En = k1*h1+k2*h2-r*c*( (h1-h2+h3)^2 + (k4)^2 );
  z = ([ 0;D*ye(1:n-1);Dn*ye(n) ] ...
       + [ E1*ye(1);E*ye(2:n-1);En*ye(n);0 ] ...
       + [ F*ye(2:n);0;0 ]);
  GH = toeplitz([ B A zeros(1,n-1) ],...
       [ B C zeros(1,n-1) ]);
  GH(1,1) = B1;
  GH(n,n) = k1^2+k2^2+r*c^2*( (h1-h2+h3)^2 + (k4)^2 );
  GH(n+1,n+1) = r*(d-c)^2*(h1^2+h2^2);
  GH(n,n+1) = -c*Dn;
  GH(n+1,n) = GH(n,n+1);
end

y = y1+GH\z;
if s1(1) ==1, y = y'; end
```

4 Examples of Numerical Experiments

An example not covered in this section is in the program obtainable by `ftp` as described in Section 5. The M-file `ldemo.m` gives the data and explanation of a 10-fold compression of Lena. With such a high compression ratio, distortion is inevitable. But the restored image contains fewer offensive visual artifacts such as blocking effects. This suggests that W-transforms may be an alternative to DCT in achieving high ratio compression.

4.1 `compare`

Our first experiment compares the performance of two W-transforms in compressing a sample signal `x`. The signal `x` is transformed into two vectors `y1` and `y2`. An approximate signal is restore by retaining only some of the components of `y2`. The discrepancies between the original and the approximate signals are then computed using the L^2 norm. To prepare for the experiment, one chooses a signal `x` and the W-transforms and then invokes `compare`.

>> x = *DEFINITION OF SIGNAL*
```
>> kw = [ 1 3 3 1 ]/4; kww = kwdau; compare
```

A sample output is as follows.

```
          W1OC        W1         W2

  # Coef             errors

     2   0.001031   0.001107   0.015413
     5   0.000928   0.001014   0.013358
     7   0.000818   0.000882   0.012025
    ..   ........   ........   ........
    35   0.000185   0.000207   0.001881
    37   0.000171   0.000191   0.001533
    40   0.000131   0.000148   0.000991
    ..   ........   ........   ........
```

The first column gives the number of components of `y2` retained (these numbers can be controlled by changing the variable `RR`). The second column gives the errors when the signal is compressed using the first W-transform with orthogonal compensation. The third and fourth columns give the errors when the signal is compressed using the two W-transforms without orthogonal compensation. The sample signal is the same signal `x` used in Section 2, and the W-transforms compared are the QS and D_4 transforms. As seen in the above example, retaining just 2 components of `y2` in the QS transform is better than retaining 37 components in the D_4 transform.

The listing of the m-file follows.

```
%%% compare.m

if ~exist('RR'), RR = 0.05:0.05:0.5; end

normx = norm(x);
[y1 y2] = kwt(x);
[d1 d2] = kwt(x,kww);

disp(' ')
disp('           W1OC        W1          W2')
disp(' ')
disp('  # Coef             errors')
disp(' ')

for r=RR
  cy2 = largesta(r,y2); cx = ikwt(y1,cy2);     cE=cx-x;
  oy1 = oc(y1,y2-cy2);  ox = ikwt(oy1,cy2);    oE=ox-x;
  cd2 = largesta(r,d2); dx = ikwt(d1,cd2,kww); dE=dx-x;

  fprintf('   %3d   %f   %f   %f\n',...
  nnz(cy2),norm(oE)/normx,norm(cE)/normx,norm(dE)/normx)
end
```

4.2 Decompression `decp`

The decompression function first compresses the W-transformed output matrices `Y1`, `Y2`, `Y3`, and `Y4` (obtained from an earlier W-transform), either by discarding small components or by quantizing the detail submatrices (according to whether the input argument `r` is less than or greater than 1), and then reconstructs an approximate image. The optional output arguments `cr` and `nz` give the *compression ratio* and *number of nonzero elements retained.* These terms do not have the precise meaning as commonly used and are provided for reference only.

```
function [X, cr, nz]= decp(Y1,Y2,Y3,Y4,r)

s = size(Y1);

if nargin == 1, Y2 = zeros(s); Y3 = Y2; Y4 = Y2;
elseif nargin ==5 & r <= 1 ,
                    Y = [ 0*Y1 Y2; Y3 Y4 ];
                    Y = largesta(r,Y);
                    Y(s) = Y1;
elseif nargin == 5, Y2 = quant(Y2,r);
                    Y3 = quant(Y3,r);
                    Y4 = quant(Y4,4*r);
  nz = prod(size(Y1))+nnz(Y2)+nnz(Y3)+nnz(Y4);
  Y = [Y1 Y2;Y3 Y4];
end

X = ikwt2(Y);
cr = 4*s(1)*s(2)/nz;
```

4.3 Image Compression `imcomp`

```
%%% imcomp.m

if exist('IMCOMPR') ~= 1
if ~exist('r3'), r3 = [ .01 .05 .1 ]; end
fprintf('current r3 = ')
fprintf(' %f ',r3)
tr3 = input('Enter new r3: ','s');
if length(tr3) > 0
  eval([ 'tr3 = [' tr3 '];']);
  if length(tr3) == 2
    eval([ 'r3(round(' num2str(tr3(1)) ')) = '...
           num2str(tr3(2)) ';'])
  else r3 = tr3;
  end
end
else
  r3 = IMCOMPR;
end

kwold = kw;
kw = kwqs;

LEVELS = length(r3);
if LEVELS == 4
[cWL31 cr4 nz4] = decp(WL41,WL42,WL43,WL44,r3(4));
nz4 = nz4 - prod(size(WL41));
disp([ 'Level 4 decompression done   nz = ' int2str(nz4) ])
else
cWL31 = WL31;
end

[cWL21 cr3 nz3] = decp(cWL31,WL32,WL33,WL34,r3(3));
N3 = prod(size(WL31)); nz3 = nz3 - N3;
disp([ 'Level 3 decompression done   nz = ' int2str(nz3) ])

[cWL11 cr2 nz2] = decp(cWL21,WL22,WL23,WL24,r3(2));
nz2 = nz2 - prod(size(cWL21));
disp([ 'Level 2 decompression done   nz = ' int2str(nz2) ])
```

```
[cL cr1 nz1] = decp(cWL11,WL12,WL13,WL14,r3(1));
NN = prod(size(cWL11)); nz1 = nz1 - NN;
disp([ 'Level 1 decompression done   nz = ' int2str(nz1) ])

if r3(1) < 1
  if LEVELS == 4
    nz = nz4+nz3+nz2+nz1+prod(size(WL41));
  else
    nz = nz3+nz2+nz1+prod(size(WL31));
  end

cr = prod(size(L))/nz;
fprintf('\nCompression Ratio: %d/%d = %f\n',prod(size(L)),nz,cr)

TITLE = [ 'Compressed Image, CR = ' num2str(cr) '(' ...
  num2str(r3(1)) ', ' num2str(r3(2)) ', ' num2str(r3(3)) ];
XLABEL = [ int2str(nz1) '    ' int2str(nz2) '    ' ...
          int2str(nz3) ];

if LEVELS == 3
  TITLE = [ TITLE ')' ];
else
  TITLE = [ TITLE ', ' num2str(r3(4)) ')' ];
  XLABEL = [ XLABEL '    ' int2str(nz4) ];
end

else
  cr = 8*NN/(N3*10 + nz3*(22-log(r3(3))/log(2)) +...
       nz2*(18-log(r3(2))/log(2)) + nz1*(14-log(r3(1))/log(2)));
  fprintf('\ncr = %f\n',cr)

TITLE = [ 'Quantized Image, cr = ' num2str(cr) '  (' ...
  int2str(r3(1)) ', ' int2str(r3(2)) ', ' int2str(r3(3)) ')' ];
XLABEL = [ int2str(nz1) '    ' int2str(nz2) '    ' int2str(nz3) ];
end

image(cL), colormap(gray(256))
axis image
title(TITLE)
xlabel(XLABEL)
```

Our third experiment uses a 256-level grayscale image stored in a matrix `L`. The command `wmai` should be used first to produce a three- or four-level multiresolution analysis of `L`. One then invokes `incomp`. One will be prompted to input `r,` comprising three or four values to be used as the fractions of components of the detail matrices to be retained in each of the levels. Experience shows that the first-level details are less important, however, hence, only a relatively small fraction of the components need be kept to give a satisfactory approximate image. The suggested default is `r = 0.01, 0.05, 0.1` using three levels.

The compression ratio displayed by the program is actually based on the ratio of the size of the original image to the number of nonzero elements retained in the compressed representation, adjusted by the qunatization level. The QS transform, for one, leads to transformed matrices with elements much larger than the original matrix. Hence, a precise bit rate count of the compressed data has to be performed to give an accurate compression ratio. On the other hand, we have not used entropy compression, such as Huffman coding or arithmetic coding, that can further increase the compression ratio.

4.4 Finding the Optimal W-transform `opt`

Our fourth experiment compresses a signal `x` by taking the W-transform specified by `kww = [ 1 r r 1 ]` and then discarding all except the largest (in absolute value) 4 components of the resulting `y2`. The value of `r` is varied (as specified by the vector `RRopt`) and the corresponding errors printed.

```
normx = norm(x);
if ~exist('RRopt'), RRopt = 2.5:0.05:3.5; end
  disp(' ')
  disp(' k vector used is [ 1 r r 1 ]')
  disp(' ')
  disp('    r          WOC          W')
  disp(' ')
for r=RRopt
  kww = [1 r r 1];
  [y1 y2] = kwt(x,kww);
  cy2 = largesta(4,y2);
  cx = ikwt(y1,cy2,kww);
  cE=cx-x;
  oy1 = oc(y1,y2-cy2,kww);
  ox = ikwt(oy1,cy2,kww);
  oE=ox-x;
  fprintf('%f   %f   %f\n',r,norm(oE)/normx,norm(cE)/normx)
end
```

The following output shows that the optimal `r` is about 2.95.

```
    r         WOC         W
........  ........  ........
2.850000  0.002690  0.002926
2.900000  0.001450  0.001571
2.950000  0.000277  0.000294
3.000000  0.000984  0.001060
3.050000  0.002113  0.002270
........  ........  ........
```

To determine a more accurate result, change `RRopt` to `2.90:0.01:3.0.` and invoke `opt` again.

5 `ftp` Information

As more experiments are performed, new commands and M-files will certainly be created, and we will add these to our repertoire in future. The toolbox also contains several general purpose graphics/image utilities not covered in this paper, such as `mag` (and `shk`) which magnifies (shrinks) the current figure window. The most up-to-date toolbox can be obtained by `ftp` at `info.mcs.anl.gov`. After logging in, change directory to `pub/W-transfrom` and `get` the files `wtransf1.ps`, `wtransf2.ps`, and `wtransf.tar.Z`.

The first file is [3], the second file is this paper, and the third file contains the toolbox. On a Unix machine, use the commands

```
% uncompress wtransf.tar.Z
% tar xvf wtransf.tar
```

to install the toolbox.

Please send comments and suggestions to `kwong@mcs.anl.gov`.

References

[1] Chui, C. K., *An Introduction to Wavelets*, Academic Press, 1992.

[2] Daubechies, I., *Ten Lectures on Wavelets*, CBMS-NSF Series Appl. Math., SIAM, 1991.

[3] Kwong, Man Kam, and Tang, Peter P. T., *W-matrices and Nonorthogonal Multiresolution Analysis of Finite Signalsof Arbitrary Length*, Argonne National Laboratory Preprint Series MCS-P449-0794, 1994.

[4] Mallat, S. G., *A theory for multiresolution signal decomposition: The wavelet representation,* IEEE Trans. on Pattern Analysis and Machine Intelligence 11 (1989), 674–693.

[5] Taswell, C., and McGill, K. C., *Wavelet transform algorithms for finite-duration discrete-time signals,* Numerical Analysis Project Manuscript NA-91-07, Department of Computer Science, Stanford University, 1991.