A Symmetric Framelet ToolBox*

Ivan Selesnick

Polytechnic University
Brooklyn, NY 11201, USA
selesi@poly.edu
718 260-3416

September 2, 2004

1 Introduction

The symmetric wavelet tight frame described in the paper

> I. W. Selesnick and A. Farras Abdelnour. Symmetric wavelet tight frames with two generators. *Applied and Computational Harmonic Analysis*, 17(2):211-225, September 2004. (Special Issue: Frames in Harmonic Analysis, Part II.)

has been implemented in MATLAB with symmetric boundary conditions. The implementation satisfies Parseval's energy property. These notes describe the implementation of the transform and verifies its properties.

We call this expansive transform the *double-density discrete wavelet transform* (DDWT) because it has exactly twice the number of wavelet coefficients as the critically-sampled DWT.

We use cell arrays in MATLAB to store the wavelet coefficients because it simplifies the programs and it also makes it easy to access the wavelet coefficients corresponding to a

*Research supported by ONR grant N000140310217

specific scale. A cell array can be used to store a set of vectors, each vector having possibly different lengths. Because there are a different number of wavelet coefficients at each scale, a cell array is convenient for organizing the wavelet coefficients.

2 The Filters

The analysis filters are stored as the cell array `af`, and the synthesis filters are stored as the cell array `sf`. Because the frame is a tight frame, the synthesis filters are the time-reversed versions of the analysis filters. The filter `af{1}` is the symmetric even-length lowpass filter, the filter `af{2}` is the symmetric even-length bandpass filter, and the filter `af{3}` is the anti-symmetric even-length highpass filter.

Checking Perfect Reconstruction Conditions

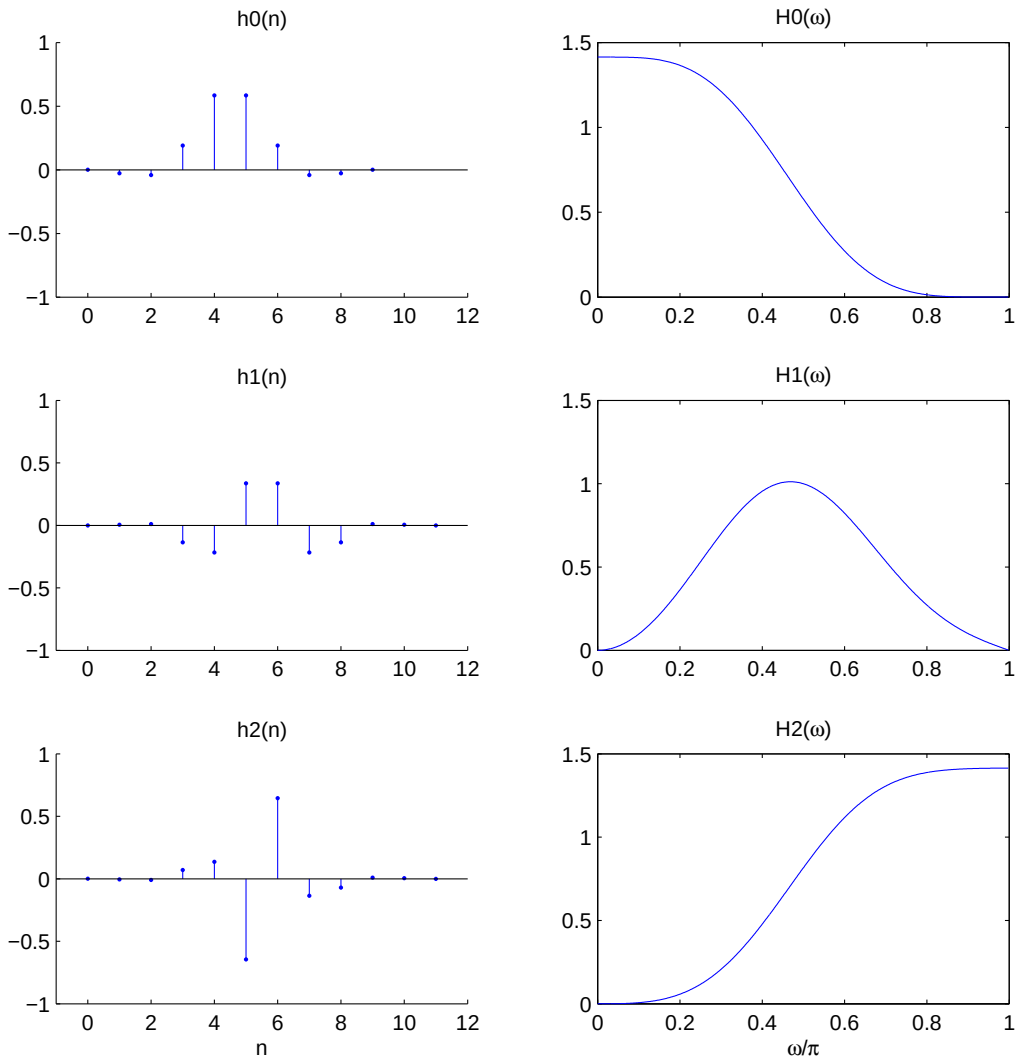
We can check the perfect reconstruction (PR) conditions using the following code fragment. The perfect reconstruction conditions are equations (3) and (4) in the paper.

```
% check perfect reconstruction conditons

[af, sf] = filters1;
h0 = [af{1}; 0; 0];
h1 = af{2};
h2 = af{3};

N = length(h0);
n = [0:N-1]';

g0 = h0(N-n);
g1 = h1(N-n);
g2 = h2(N-n);

Eq3 = conv(h0,g0) + conv(h1,g1) + conv(h2,g2);

s = (-1).^n;
f0 = h0 .* s;
f1 = h1 .* s;
f2 = h2 .* s;

Eq4 = conv(f0,g0) + conv(f1,g1) + conv(f2,g2);

[Eq3 Eq4]
```

The result of running this program is the following, which verifies that these filters do satisfy the PR conditions.

```
                0                   0
                0                   0
 0.00000000000000   0.00000000000000
-0.00000000000000   0.00000000000000
-0.00000000000000  -0.00000000000000
-0.00000000000000  -0.00000000000000
 0.00000000000001   0.00000000000001
-0.00000000000007   0.00000000000000
-0.00000000000000   0.00000000000001
 0.00000000000036  -0.00000000000000
-0.00000000000002  -0.00000000000000
 1.99999999999940   0.00000000000000
-0.00000000000002   0.00000000000000
 0.00000000000036  -0.00000000000000
-0.00000000000000  -0.00000000000001
-0.00000000000007   0.00000000000000
 0.00000000000001  -0.00000000000001
-0.00000000000000  -0.00000000000000
-0.00000000000000   0.00000000000000
-0.00000000000000   0.00000000000000
 0.00000000000000  -0.00000000000000
                0                   0
                0                   0
```

We can create the following plots of the impulse responses and frequency responses of the three filters with the MATLAB code contained in the accompanying file `check.m`.



3 Implementing the Analysis and Synthesis Filter Banks

The basic building block of the double-density DWT is the analysis and synthesis filter banks. The analysis filter bank is implemented with the program `afb`. The program returns three subband signals obtained by filtering the input signal with each of three filters and down-sampling by two. The three subband signals are called `lo`, `hi1`, and `hi2` in the `afb` program.

In the implementation, there are a couple of details that must be taken into account. First, the symmetric extension at the boundaries of the signal requires that the input signal be symmetrically extended before the filtering is performed. While periodic extension at the

boundaries is easier to implement than symmetric extension, periodic extensions give rise to undesirable artifacts at the boundaries of the reconstructed signal when the wavelet coefficients are processed. Since our filters are symmetric, we can perform symmetric extension. (Symmetric extension of the boundaries requires that the filters be symmetric.) It turns out that symmetric extension for the symmetric double-density DWT filters is not as straightforward as it is for the symmetric biorthogonal DWT. That is because the even-length symmetric biorthogonal filters are symmetric and anti-symmetric about the same point. However, the filters for the double-density DWT are symmetric about different points. It turns out that for the double-density DWT with symmetric extension, the bandpass and highpass subbands will have a different number of samples each. If the input signal `x` has length N (with N even) then one would expect that each of the three subband signals will be of length $N/2$. When implementing the symmetric extension, the lowpass subband signal will have length $N/2$, however, the bandpass subband signal will have length $N/2 + 1$ and the highpass subband signal will have length $N/2 - 1$. So the total number of subband coefficients is $3\,N/2$ as expected.

Second, in order to have Parseval's energy relation, it is necessary that the end-points of the bandpass subband signal be normalized (divided) by $\sqrt{2}$. In the synthesis filter bank, this normalization is removed by multiplying those endpoints by $\sqrt{2}$. This normalization is not needed to have the perfect reconstruction property, it is only needed to satisfy Parseval's energy relation exactly. (That the energy of the signal can be computed by the sum of the energies of the subband signals.).

The synthesis filter bank is implemented with the program `sfb`.

We can check the perfect reconstruction conditions using the following MATLAB code fragment:

```
>> [af, sf] = filters1;
>> x = rand(1,64);
>> [lo, hi1, hi2] = afb(x, af);
>> y = sfb(lo, hi1, hi2, sf);
>> err = x - y;
>> max(abs(err))

ans =

     2.909894547542535e-13
```

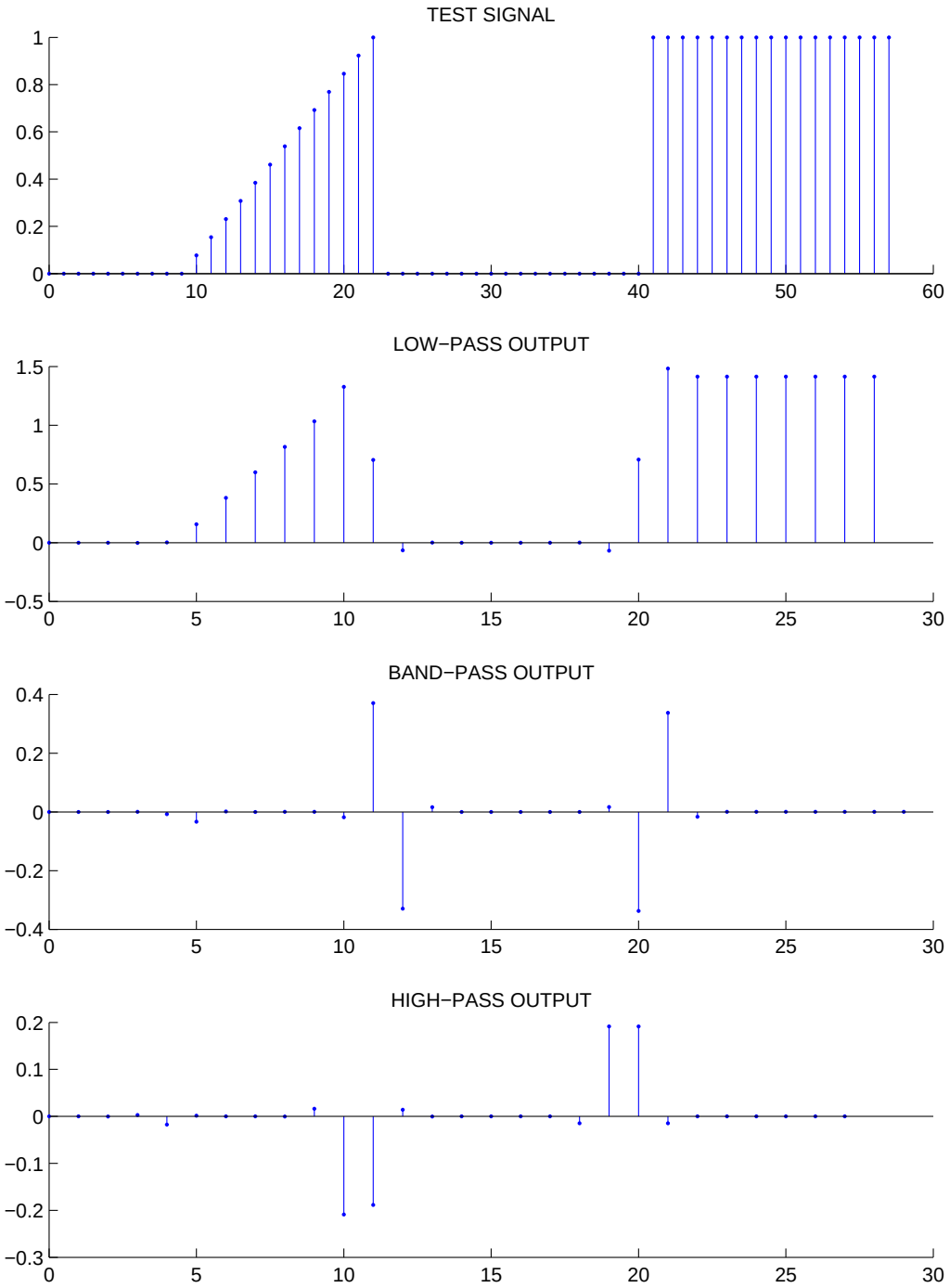
Parseval's Energy Relation

We can check Parseval's energy relation with the following MATLAB code fragment:

```
>> x*x'

ans =

  20.52511996979790

>> lo*lo' + hi1*hi1' + hi2*hi2'

ans =

  20.52511996979641
```

This illustrates that the total energy of the three subband signals equals the energy of the input signal.

For a simple test signal the following figure illustrates the three subband signals. Notice that the bandpass and highpass subbands signals equal zero over the constant and ramp segments of the test signal, except for neighborhoods around the discontinuities. This illustrates the vanishing moment properties of the filters.


TEST SIGNAL
LOW−PASS OUTPUT
BAND−PASS OUTPUT
HIGH−PASS OUTPUT

4 Main Program

The main program for computing the double-density DWT is `ddwt`. The program is very simple. It just calls the analysis filter bank program `afb` iteratively — each time reassigning the input signal `x` as the lowpass subband signal and storing the bandpass and highpass subband signals in a cell array.

To clarify the way in which the wavelet coefficients are organized in the cell array, lets take a 3-level DDWT of a length 128 signal:

```
>> [af, sf] = filters1;
>> x = rand(1,128);
>> w = ddwt(x,3,af)

w =

    {1x2 cell}    {1x2 cell}    {1x2 cell}    [1x16 double]
```

The cell `w{1}` contains the wavelet coefficients coming from the first level.

```
>> w{1}

ans =

    [1x65 double]    [1x63 double]
```

The vector `w{1}{1}` contains the 65 wavelet coefficients coming from the bandpass subband at the first level, and `w{1}{2}` contains the 63 wavelet coefficients coming from the highpass subband at the first level. Similarly:

```
>> w{2}

ans =

    [1x33 double]    [1x31 double]

>> w{3}

ans =
```

```
    [1x17 double]    [1x15 double]
```

Parseval's Energy Relation

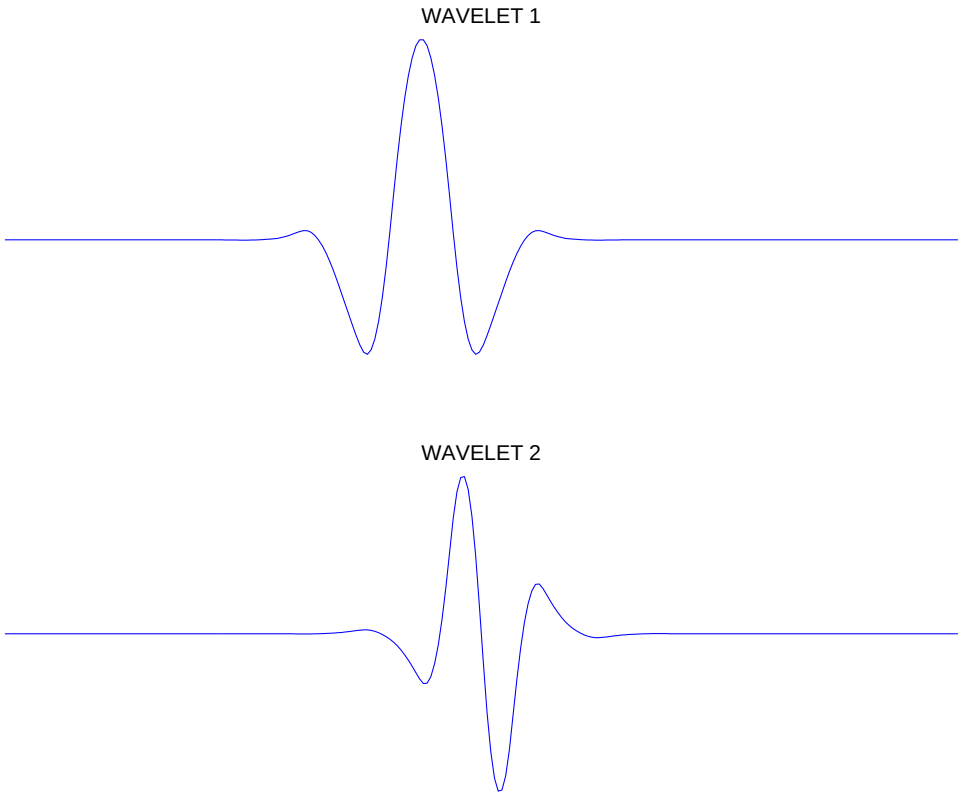
We can verify that the energy of the input signal and the total energy of the wavelet coefficients are equal using the following MATLAB code fragment.

```
>> [af, sf] = filters1;
>> x = rand(1,128);
>> E1 = x*x'

E1 =

   41.3498

>> w = ddwt(x,3,af);
>> E2 = sum(w{4}.^2);
>> for j = 1:3, for i = 1:2, E2 = E2 + sum(w{j}{i}.^2); end, end, E2

E2 =

   41.3498

>> E1 - E2

ans =

   8.1641e-12
```

The energy of the signal, `E1`, and the total energy in the wavelet domain, `E2`, are the same.

The program for the inverse double-density DWT is `ddwti`.

Using the program `ddwti` we can plot the wavelets by setting all the wavelet coefficients to zero, except for one of them at a time. Then, computing the inverse DDWT gives the corresponding wavelet.

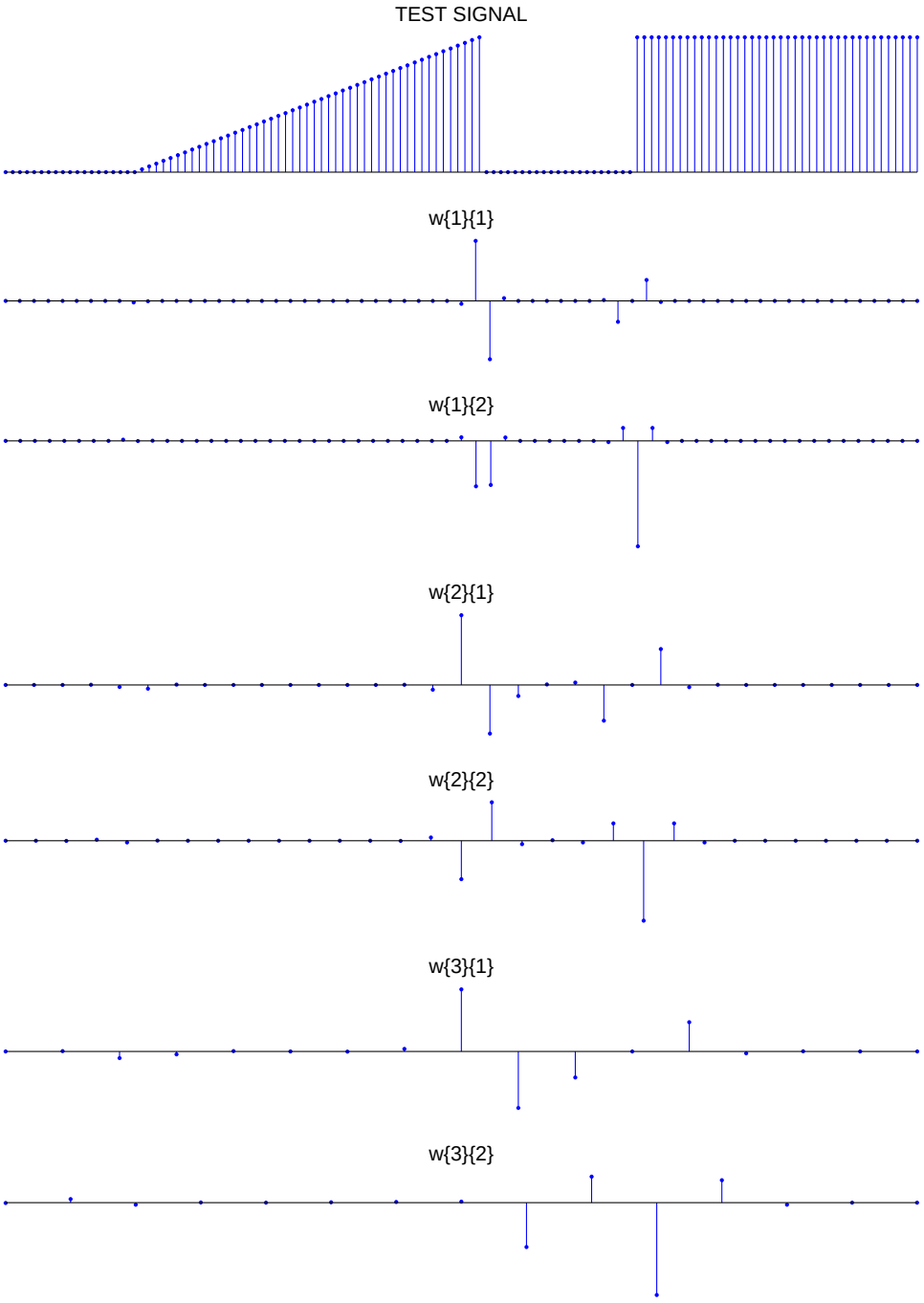



```
% COMPUTE AND DISPLAY WAVELETS
[af, sf] = filters1;
x = zeros(1,256);           % all zero signal
w = ddwt(x,4,af);           % all zero wavelet coefficients
w{4}{1}(8) = 1;             % set a single wavelet coeff to 1
y1 = ddwti(w,4,sf);         % compute the inverse DDWT
w = ddwt(x,4,af);           % all zero wavelet coefficients
w{4}{2}(8) = 1;             % set a different single wavelet coeff to 1
y2 = ddwti(w,4,sf);         % compute the inverse DDWT
figure(1)                   % display wavelets
clf
subplot(2,1,1)
plot(y1)
axis tight off
title('WAVELET 1')
subplot(2,1,2)
plot(y2)
axis tight off
title('WAVELET 2')
orient portrait
print -depsc PlotWavelets
```

In the program `check.m` accompanying this document, the wavelets that coincide with the boundary of the signal are also shown, to illustrate the symmetric extension.

A Simple Test Signal

For a simple 128-point test signal, the following figure illustrates the wavelet coefficients of a 3-level double-density DWT.



For the $N = 128$ point signal, the total number of wavelet coefficients is

$$\underbrace{N/2 + N/2}_{\text{level 1}} + \underbrace{N/4 + N/4}_{\text{level 2}} + \underbrace{N/8 + N/8}_{\text{level 3}} + \underbrace{N/8}_{\text{level 3 lowpass}} = 15\,N/8 = 240$$

Program Listing

filters1.m

```
function [af, sf] = filters1

% [h0, h1, h2] = filters1
% Symmetric Filters for the Double-Density Wavelet Transform
% Reference:
% I. W. Selesnick and A. Farras Abdelnour.
% Symmetric wavelet tight frames with two generators.
% Applied and Computational Harmonic Analalysis, 17(2), 2004.

af{1} = [
   0.00069616789827
  -0.02692519074183
  -0.04145457368920
   0.19056483888763
   0.58422553883167
   0.58422553883167
   0.19056483888763
  -0.04145457368920
  -0.02692519074183
   0.00069616789827
];

af{2} = [
  -0.00014203017443
   0.00549320005590
   0.01098019299363
  -0.13644909765612
  -0.21696226276259
   0.33707999754362
   0.33707999754362
  -0.21696226276259
  -0.13644909765612
   0.01098019299363
   0.00549320005590
  -0.00014203017443
];

af{3} = [
   0.00014203017443
  -0.00549320005590
```

```
  -0.00927404236573
   0.07046152309968
   0.13542356651691
  -0.64578354990472
   0.64578354990472
  -0.13542356651691
  -0.07046152309968
   0.00927404236573
   0.00549320005590
  -0.00014203017443
];

sf{1} = af{1}(end:-1:1);
sf{2} = af{2}(end:-1:1);
sf{3} = af{3}(end:-1:1);
```

afb.m

```
function [lo, hi1, hi2] = afb(x, af)

% Analysis filter bank
%
% USAGE:
%    [lo, hi1, hi2] = afb(x, af)
% INPUT:
%    x - N-point vector, with N even
%    af - analysis filters
%    af{1} - lowpass filter (symmetric even-length)
%    af{2} - bandpass filter (symmetric even-length)
%    af{3} - highpass filter (antisymmetric even-length)
% OUTPUT:
%    lo - lowpass output
%    hi1 - bandpass output
%    hi2 - highpass output
% EXAMPLE:
%    [af, sf] = filters1;
%    x = rand(1,64);
%    [lo, hi1, hi2] = afb(x, af);
%    y = sfb(lo, hi1, hi2, sf);
%    err = x - y;
%    max(abs(err))
%
% WAVELET SOFTWARE AT POLYTECHNIC UNIVERSITY, BROOKLYN, NY
% http://taco.poly.edu/WaveletSoftware/

% Ivan Selesnick
% selesi@poly.edu

N = length(x);

h0 = af{1};
h1 = af{2};
h2 = af{3};

L0 = length(h0)/2;
L1 = length(h1)/2;
L2 = length(h2)/2;
```

```
% symmetric extension
A = L0;
xe = [x(A:-1:1) x x(N:-1:N-A+1)];
% lowpass filter
lo = conv(xe, h0);
% extract valid part
lo = lo(2*L0-1+2*[1:N/2]);

% symmetric extension
A = L1;
xe = [x(A:-1:1) x x(N:-1:N-A+1)];
% highpass filter 1
hi1 = conv(xe, h1);
% down-sample and extract valid part
hi1 = hi1(2*L1-2+2*[1:N/2+1]);
% normalize
hi1(1) = hi1(1)/sqrt(2);
hi1(N/2+1) = hi1(N/2+1)/sqrt(2);

% symmetric extension
A = L2;
xe = [x(A:-1:1) x x(N:-1:N-A+1)];
% highpass filter 2
hi2 = conv(xe, h2);
% down-sample and extract valid part
hi2 = hi2(2*L1-2+2*[2:N/2]);
```

sfb.m

```
function y = sfb(lo, hi1, hi2, sf)

% Synthesis filter bank
%
% USAGE:
%    y = sfb(lo, hi1, hi2, sf)
% INPUT:
%    lo - lowpass input
%    hi1 - bandpass input
%    hi2 - highpass input
%    sf - synthesis filters
% OUTPUT:
%    y - output signal
% See also afb
%
% WAVELET SOFTWARE AT POLYTECHNIC UNIVERSITY, BROOKLYN, NY
% http://taco.poly.edu/WaveletSoftware/

N = 2*length(lo);

g0 = sf{1};
g1 = sf{2};
g2 = sf{3};

L0 = length(g0);
L1 = length(g1);
L2 = length(g2);

% symmetric extension
A = L0/2;
lo = [lo(A:-1:1) lo lo(N/2:-1:N/2-A)];
% lowpass filter
lo = up(lo, 2);
lo = conv(lo, g0);
% extract valid part
lo = lo(3*L0/2-1+[1:N]);

% normalize
hi1(1) = sqrt(2)*hi1(1);
hi1(N/2+1) = sqrt(2)*hi1(N/2+1);
```

```
% symmetric extension
A = L1/2;
hi1 = [hi1(A:-1:2) hi1 hi1(N/2:-1:N/2-A)];
% highpass filter
hi1 = up(hi1, 2);
hi1 = conv(hi1, g1);
% extract valid part
hi1 = hi1(3*L1/2-2+[1:N]);

% symmetric extension
A = L2/2;
hi2 = [0 hi2 0];
hi2 = [-hi2(A:-1:2) hi2 -hi2(N/2:-1:N/2-A)];
% highpass filter
hi2 = up(hi2, 2);
hi2 = conv(hi2, g2);
% extract valid part
hi2 = hi2(3*L1/2-2+[1:N]);

% add signals
y = lo + hi1 + hi2;
```

ddwt.m

```
function w = ddwt(x, J, af)

% Double-Density Wavelet Transform
%
% USAGE:
%    w = ddwt(x, J, af)
% INPUT:
%    x - N-point vector with N divisible by 2^J
%    J - number of stages
%    af - analysis filters (even length)
%    af{i} - filter i (i = 1,2)
% OUTPUT:
%    w{j}{i} - wavelet coefficients (j = 1..J, i = 1,2)
%    w{J+1} - scaling coefficients
% EXAMPLE:
%    [af, sf] = filters1;
%    x = rand(1,128);
%    w = ddwt(x,3,af);
%    y = ddwti(w,3,sf);
%    err = x - y;
%    max(abs(err))
%
% WAVELET SOFTWARE AT POLYTECHNIC UNIVERSITY, BROOKLYN, NY
% http://taco.poly.edu/WaveletSoftware/

% Ivan Selesnick
% selesi@poly.edu

for j = 1:J
    [x w{j}{1} w{j}{2}] = afb(x, af);
end
w{J+1} = x;
```

ddwti.m

```
function y = ddwti(w, J, sf)

% Inverse Double-Density Wavelet Transform
%
% USAGE:
%    y = ddwti(w, J, sf)
% INPUT:
%    w - wavelet coefficients
%    J - number of stages
%    sf - synthesis filters
% OUTPUT:
%    y - output signal
% See also ddwt
%
% WAVELET SOFTWARE AT POLYTECHNIC UNIVERSITY, BROOKLYN, NY
% http://taco.poly.edu/WaveletSoftware/

y = w{J+1};
for j = J:-1:1
   y = sfb(y, w{j}{1}, w{j}{2}, sf);
end
```

up.m

```
function y = up(x,M)
% y = up(x,M)
% M-fold up-sampling of a 1-D signal

[r,c] = size(x);
if r > c
   y = zeros(M*r,1);
else
   y = zeros(1,M*c);
end
y(1:M:end) = x;
```