# Wavelet Toolbox™

Reference

**R**2014**a**

*Michel Misiti*
*Yves Misiti*
*Georges Oppenheim*
*Jean-Michel Poggi*
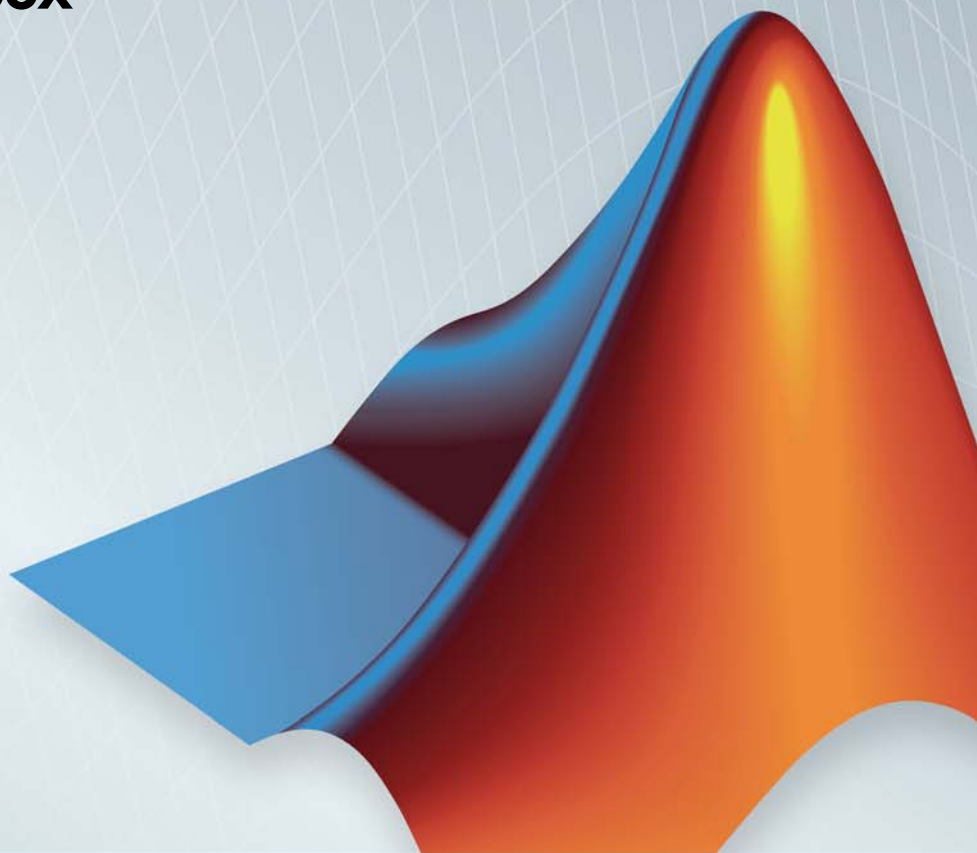
# MATLAB®

MathWorks®

## How to Contact MathWorks

| | |
|---|---|
| www.mathworks.com | Web |
| comp.soft-sys.matlab | Newsgroup |
| www.mathworks.com/contact_TS.html | Technical Support |

| | |
|---|---|
| suggest@mathworks.com | Product enhancement suggestions |
| bugs@mathworks.com | Bug reports |
| doc@mathworks.com | Documentation error reports |
| service@mathworks.com | Order status, license renewals, passcodes |
| info@mathworks.com | Sales, pricing, and general information |

508-647-7000 (Phone)

508-647-7001 (Fax)

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

*Wavelet Toolbox™ Reference*

**Trademarks**

**Patents**

**Revision History**

# Contents

**1**

# Function Reference

addlift
allnodes
appcoef
appcoef2
bestlevt
besttree
biorfilt
biorwavf
bswfun
centfrq
cfs2wpt
cgauwavf
chgwdeccfs
cmddenoise
cmorwavf
coifwavf
conofinf
cwt
cwtft
cwtftinfo
cwtftinfo2
cwtft2
cwtext
dbaux
dbwavf
ddencmp
dddtree
dddtreecfs

**Purpose**      Add lifting steps to lifting scheme

**Syntax**       *LSN* = addlift(*LS*,*ELS*)
                 *LSN* = addlift(*LS*,*ELS*,'begin')
                 *LSN* = addlift(*LS*,*ELS*,'end')
                 addfilt(*LS*,*ELS*)

**Description**  *LSN* = addlift(*LS*,*ELS*) returns the new lifting scheme *LSN* obtained by appending the elementary lifting step *ELS* to the lifting scheme *LS*.

*LSN* = addlift(*LS*,*ELS*,'begin') prepends the specified elementary lifting step.

*ELS* is either a cell array (see lsinfo)

{TYPEVAL, COEFS, MAX_DEG}

or a structure (see liftfilt)

struct('type',TYPEVAL,'value',LPVAL)

with

LPVAL = laurpoly(COEFS, MAX_DEG)

*LSN* = addlift(*LS*,*ELS*,'end') is equivalent to addfilt(*LS*,*ELS*).

If *ELS* is a sequence of elementary lifting steps, stored in a cell array or an array of structures, then each of the elementary lifting steps is added to *LS*.

For more information about lifting schemes, see lsinfo.

**Examples**     % Start from the Haar wavelet and get the
                 % corresponding lifting scheme.
                 lshaar = liftwave('haar');

                 % Visualize the obtained lifting scheme.
                 displs(lshaar);

                 lshaar = {...

```
'd'               [ -1.00000000]  [0]
'p'               [  0.50000000]  [0]
[  1.41421356]  [  0.70710678]  []
};

% Add a primal ELS to the lifting scheme.
els = {'p',[-0.125 0.125],0};
lsnew = addlift(lshaar,els);
displs(lsnew);

lsnew = {...
'd'               [ -1.00000000]                [0]
'p'               [  0.50000000]                [0]
'p'               [ -0.12500000  0.12500000]  [0]
[  1.41421356]  [  0.70710678]                []
};
```

**See Also**    liftfilt

**Purpose**    Tree nodes

**Syntax**    
```
N = allnodes(T)
N = allnodes(T,'deppos')
```

**Description**    `allnodes` is a tree management utility that returns one of two node descriptions: either indices, or depths and positions.

The nodes are numbered from left to right and from top to bottom. The root index is 0.

`N = allnodes(T)` returns the indices of all the nodes of the tree *T* in column vector *N*.

`N = allnodes(T,'deppos')` returns the depths and positions of all the nodes in matrix *N*.

`N(i,1)` is the depth and `N(i,2)` the position of the node `i`.

**Return Nodes of Wavelet Packet Tree**    This example shows how to obtain the depth-position and linear indices of a wavelet packet tree.

Load the noisy Doppler signal and obtain the wavelet packet decomposition down to the level 4 using the `'db2'` wavelet.

```
load noisdopp;
T = wpdec(noisdopp,4,'db2');
```

Obtain the depth-position indices.

```
DepthPosition = allnodes(T,'deppos');
```

Obtain the corresponding linear indices.

```
LinearIndices = allnodes(T);
```

Display the correspondance in a table.

```
table(DepthPosition,LinearIndices)
```

# allnodes

```
ans =

    DepthPosition    LinearIndices
    _____    _____

    0     0               0
    1     0               1
    1     1               2
    2     0               3
    2     1               4
    2     2               5
    2     3               6
    3     0               7
    3     1               8
    3     2               9
    3     3              10
    3     4              11
    3     5              12
    3     6              13
    3     7              14
    4     0              15
    4     1              16
    4     2              17
    4     3              18
    4     4              19
    4     5              20
    4     6              21
    4     7              22
    4     8              23
    4     9              24
    4    10              25
    4    11              26
    4    12              27
    4    13              28
    4    14              29
    4    15              30
```

# appcoef

| | |
|---|---|
| **Purpose** | 1-D approximation coefficients |
| **Syntax** | A = appcoef(C,L,'*wname*',N) |
| | A = appcoef(C,L,'*wname*') |
| | A = appcoef(C,L,Lo_R,Hi_R) |
| | A = appcoef(C,L,Lo_R,Hi_R,N) |

**Description**    appcoef is a one-dimensional wavelet analysis function.

appcoef computes the approximation coefficients of a one-dimensional signal.

A = appcoef(C,L,'*wname*',N) computes the approximation coefficients at level *N* using the wavelet decomposition structure [C,L] (see wavedec for more information).

'*wname*' is a string containing the wavelet name. Level *N* must be an integer such that $0 \leq N \leq$ length(L)-2.

A = appcoef(C,L,'*wname*') extracts the approximation coefficients at the last level: length(*L*)-2.

Instead of giving the wavelet name, you can give the filters.

For A = appcoef(C,L,Lo_R,Hi_R) or A = appcoef(C,L,Lo_R,Hi_R,N), *Lo_R* is the reconstruction low-pass filter and *Hi_R* is the reconstruction high-pass filter (see wfilters for more information).

**Level 3 Approximation Coefficients**

This example shows how to extract the level 3 approximation coefficients.

Load the signal consisting of electricity usage data.

```
load leleccum;
sig = leleccum(1:3920);
```

Obtain the DWT down to level 5 with the 'sym4' wavelet.

```
[C,L] = wavedec(sig,5,'sym4');
```

Extract the level-3 approximation coefficients. Plot the original signal and the approximation coefficients.

```
Lev = 3;
a3 = appcoef(C,L,'sym4',Lev);
subplot(2,1,1)
plot(sig); title('Original Signal');
subplot(2,1,2)
plot(a3); title('Level-3 Approximation Coefficients');
```

Original Signal



Level-3 Approximation Coefficients
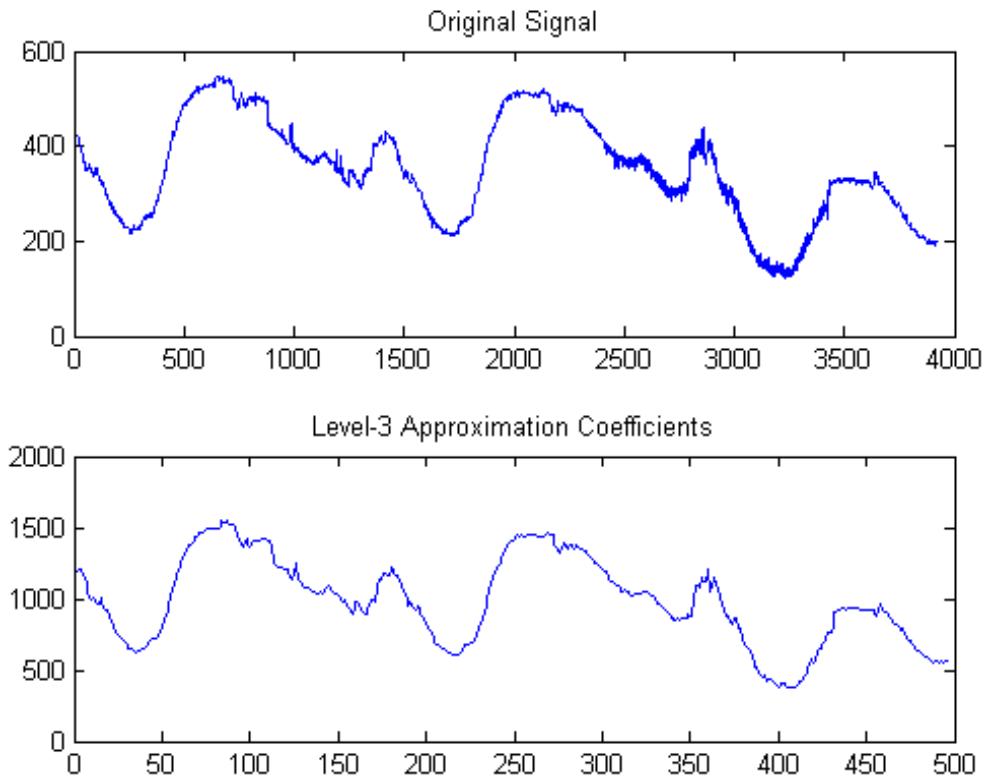
You can substitute any value from 1 to 5 for Lev to obtain the approximation coefficients for the corresponding level.

**Algorithms**
The input vectors *C* and *L* contain all the information about the signal decomposition.

Let NMAX = length(L)-2; then C = [A(NMAX) D(NMAX) ... D(1)] where *A* and the *D* are vectors.

If N = NMAX, then a simple extraction is done; otherwise, appcoef computes iteratively the approximation coefficients using the inverse wavelet transform.

**See Also**    detcoef | wavedec

# appcoef2

**Purpose**  2-D approximation coefficients

**Syntax**
```
A = appcoef2(C,S,'wname',N)
A = appcoef2(C,S,'wname')
A = appcoef2(C,S,Lo_R,Hi_R)
A = appcoef2(C,S,Lo_R,Hi_R,N)
```

**Description**  appcoef2 is a two-dimensional wavelet analysis function. It computes the approximation coefficients of a two-dimensional signal. The syntaxes allow you to give the wavelet name or the filters as inputs.

A = appcoef2(C,S,'wname',N) computes the approximation coefficients at level *N* using the wavelet decomposition structure [*C*,*S*] (see wavedec2 for more information).

'wname' is a string containing the wavelet name. Level *N* must be an integer such that $0 \leq N \leq$ size(S,1)-2.

A = appcoef2(C,S,'wname') extracts the approximation coefficients at the last level: size(S,1)-2.

A = appcoef2(C,S,Lo_R,Hi_R) or A = appcoef2(C,S,Lo_R,Hi_R,N), *Lo_R* is the reconstruction low-pass filter and *Hi_R* is the reconstruction high-pass filter (see wfilters for more information).

**Tips**  If C and S are obtained from an indexed image analysis or a truecolor image analysis, A is an m-by-n matrix or an m-by-n-by-3 array, respectively.

For more information on image formats, see the image and imfinfo reference pages.

**Examples**
```
% The current extension mode is zero-padding (see dwtmode).

% Load original image.
load woman;

% X contains the loaded image.
```

```
% Perform decomposition at level 2
% of X using db1.
[c,s] = wavedec2(X,2,'db1');
sizex = size(X)
sizex =
    256 256


sizec = size(c)

sizec =
     1      65536
    val_s = s

val_s =
    64      64
    64      64
   128     128
   256     256

% Extract approximation coefficients
% at level 2.
ca2 = appcoef2(c,s,'db1',2);
sizeca2 = size(ca2)

sizeca2 =
    64      64

% Compute approximation coefficients
% at level 1.
ca1 = appcoef2(c,s,'db1',1);
sizeca1 = size(ca1)

sizeca1 =
   128     128
```

**Algorithms**     The algorithm is built on the same principle as appcoef.

# appcoef2

**See Also**     detcoef2 | wavedec2

**Purpose**       Best level tree wavelet packet analysis

**Syntax**        T = bestlevt(T)
                  [T,E] = bestlevt(T)

**Description**   bestlevt is a one- or two-dimensional wavelet packet analysis function.

                  bestlevt computes the optimal complete subtree of an initial tree with respect to an entropy type criterion. The resulting complete tree may be of smaller depth than the initial one.

                  T = bestlevt(T) computes the modified wavelet packet tree *T* corresponding to the best level tree decomposition.

                  [T,E] = bestlevt(T) computes the best level tree *T*, and in addition, the best entropy value *E*.

                  The optimal entropy of the node, whose index is j-1, is E(j).

**Examples**      % The current extension mode is zero-padding (see dwtmode).

                  % Load signal.
                  load noisdopp;
                  x = noisdopp;

                  % Decompose x at depth 3 with db1 wavelet, using default
                  % entropy (shannon).
                  wpt = wpdec(x,3,'db1');

                  % Decompose the packet [3 0].
                  wpt = wpsplt(wpt,[3 0]);

                  % Plot wavelet packet tree wpt.
                  plot(wpt)
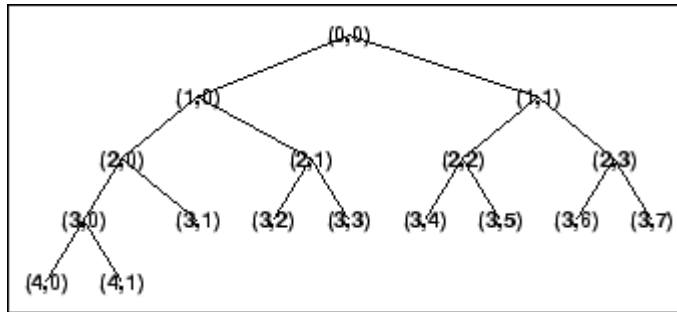
```
% Compute best level tree.
blt = bestlevt(wpt);

% Plot best level tree blt.
plot(blt)
```
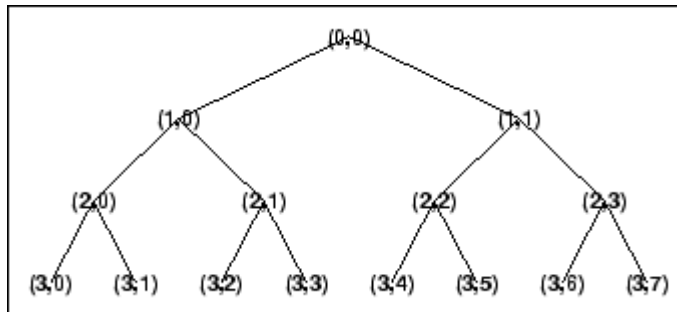


**Algorithms**    See besttree algorithm section. The only difference is that the optimal tree is searched among the complete subtrees of the initial tree, instead of among all the binary subtrees.

**See Also**    besttree | wenergy | wpdec | wpdec2

**Purpose**        Best tree wavelet packet analysis

**Syntax**
```
T = besttree(T)
[T,E] = besttree(T)
[T,E,N] = besttree(T)
```

**Description**    besttree is a one- or two-dimensional wavelet packet analysis function that computes the optimal subtree of an initial tree with respect to an entropy type criterion. The resulting tree may be much smaller than the initial one.

Following the organization of the wavelet packets library, it is natural to count the decompositions issued from a given orthogonal wavelet.

A signal of length $N = 2^L$ can be expanded in α different ways, where α is the number of binary subtrees of a complete binary tree of depth $L$.

As a result, we can conclude that $α \geq 2^{N/2}$ (for more information, see the Mallat's book given in References at page 323).

This number may be very large, and since explicit enumeration is generally intractable, it is interesting to find an optimal decomposition with respect to a convenient criterion, computable by an efficient algorithm. We are looking for a minimum of the criterion.

T = besttree(T) computes the best tree T corresponding to the best entropy value.

[T,E] = besttree(T) computes the best tree T and, in addition, the best entropy value E.

The optimal entropy of the node, whose index is j-1, is E(j).

[T,E,N] = besttree(T) computes the best tree T, the best entropy value E and, in addition, the vector N containing the indices of the merged nodes.

# besttree

**Best
Wavelet
Packet
Tree**

This example shows to obtain the optimal wavelet packet tree based on an entropy criterion.

Load the noisy Doppler signal. Obtain the wavelet packet tree down to level 4 with the `'sym4'` wavelet. Use the periodic extension mode.

```
dwtmode('per');
load noisdopp;
T = wpdec(noisdopp,4,'sym4');
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!  WARNING: Change DWT Extension Mode  !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

****************************************
**  DWT Extension Mode: Periodization  **
****************************************
```

Obtain the best wavelet packet tree and plot the result.

```
BstTree = besttree(T);
plot(BstTree)
```
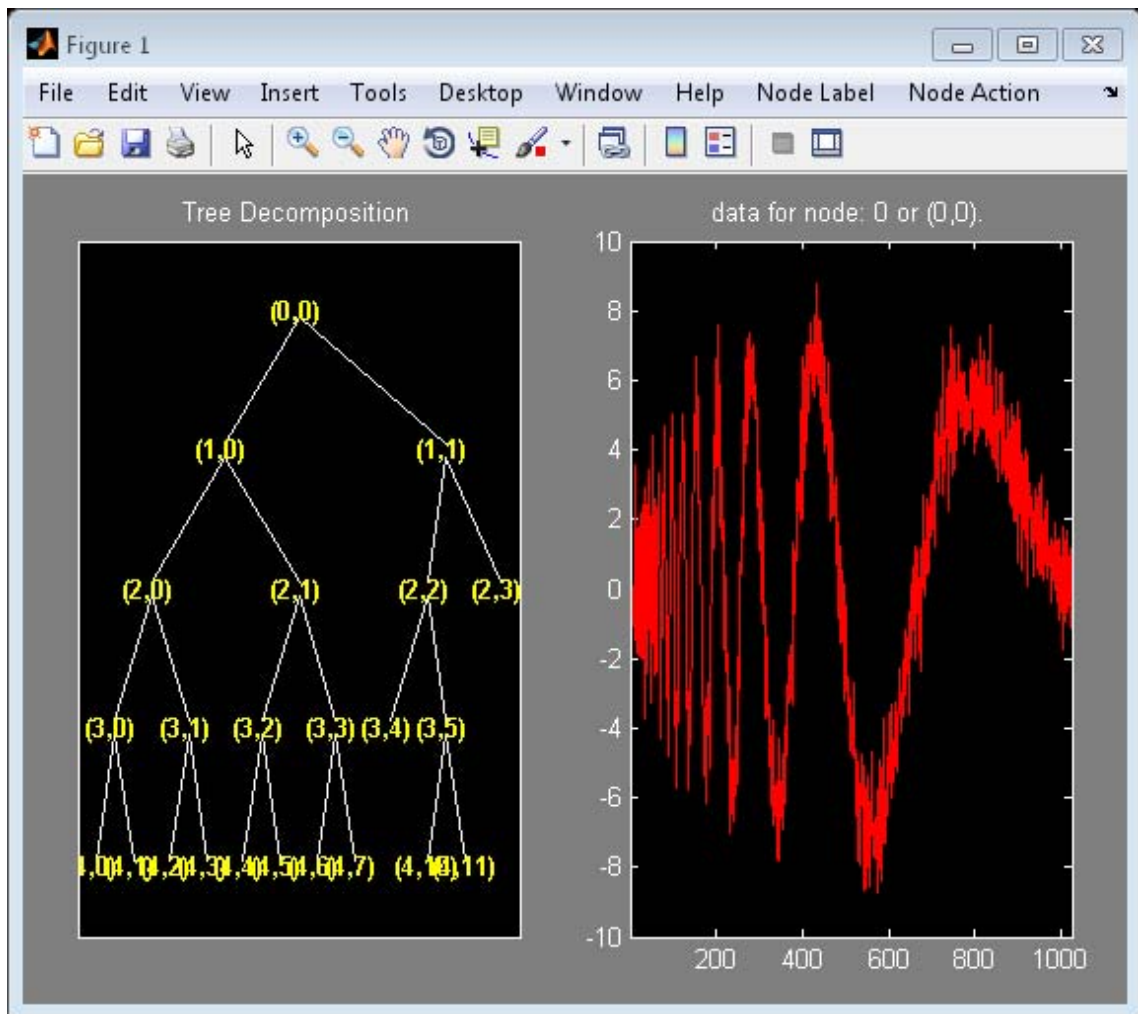
Return the DWT extension mode to the default value.

```
dwtmode('sym');
```

# besttree

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!  WARNING: Change DWT Extension Mode  !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

********************************************************
**   DWT Extension Mode: Symmetrization (half-point)  **
********************************************************
```

**Algorithms**

Consider the one-dimensional case. Starting with the root node, the best tree is calculated using the following scheme. A node N is split into two nodes N1 and N2 if and only if the sum of the entropy of N1 and N2 is lower than the entropy of N. This is a local criterion based only on the information available at the node N.

Several entropy type criteria can be used (see `wenergy` for more information). If the entropy function is an additive function along the wavelet packet coefficients, this algorithm leads to the best tree.

Starting from an initial tree T and using the merging side of this algorithm, we obtain the best tree among all the binary subtrees of T.

**References**

Coifman, R.R.; M.V. Wickerhauser (1992), "Entropy-based algorithms for best basis selection," *IEEE Trans. on Inf. Theory*, vol. 38, 2, pp. 713–718.

Mallat, S. (1998), *A wavelet tour of signal processing*, Academic Press.

**See Also**

`bestlevt` | `wenergy` | `wpcoef` | `wpdec` | `wpdec2` | `wprcoef`

**How To**

- "Reconstructing a Signal Approximation from a Node"

**Purpose**     Biorthogonal wavelet filter set

**Syntax**     [Lo_D,Hi_D,Lo_R,Hi_R] = biorfilt(*DF*,*RF*)
[Lo_D1,Hi_D1,Lo_R1,Hi_R1,Lo_D2,Hi_D2,Lo_R2,Hi_R2]
= biorfilt(*DF*,*RF*,
    '*8*')

**Description**     The biorfilt command returns either four or eight filters associated
with biorthogonal wavelets.

[Lo_D,Hi_D,Lo_R,Hi_R] = biorfilt(*DF*,*RF*) computes four filters
associated with the biorthogonal wavelet specified by decomposition
filter *DF* and reconstruction filter *RF*. These filters are

| Lo_D | Decomposition low-pass filter |
|------|-------------------------------|
| Hi_D | Decomposition high-pass filter |
| Lo_R | Reconstruction low-pass filter |
| Hi_R | Reconstruction high-pass filter |

[Lo_D1,Hi_D1,Lo_R1,Hi_R1,Lo_D2,Hi_D2,Lo_R2,Hi_R2] =
biorfilt(*DF*,*RF*, '*8*') returns eight filters, the first four associated
with the decomposition wavelet, and the last four associated with the
reconstruction wavelet.

It is well known in the subband filtering community that if the same
FIR filters are used for reconstruction and decomposition, then
symmetry and exact reconstruction are incompatible (except with the
Haar wavelet). Therefore, with biorthogonal filters, two wavelets are
introduced instead of just one:

One wavelet, $\tilde{\psi}$ , is used in the analysis, and the coefficients of a signal
*s* are

$$\tilde{c}_{j,k} = \int s(x)\tilde{\psi}_{j,k}(x)dx$$

The other wavelet, ψ, is used in the synthesis:

$$s = \sum_{j,k} \tilde{c}_{j,k} \psi_{j,k}$$

Furthermore, the two wavelets are related by duality in the following sense:

$$\int \tilde{\psi}_{j,k}(x)\psi_{j',k'}(x)dx = 0 \text{ as soon as } j \neq j' \text{ or } k \neq k' \text{ and}$$

$$\int \tilde{\phi}_{0,k}(x)\phi_{0,k'}(x)dx = 0 \text{ as soon as } k \neq k'.$$

It becomes apparent, as A. Cohen pointed out in his thesis (p. 110), that "the useful properties for analysis (e.g., oscillations, null moments) can be concentrated in the $\tilde{\psi}$ function; whereas, the interesting properties for synthesis (regularity) are assigned to the ψ function. The separation of these two tasks proves very useful."

$\tilde{\psi}$ and ψ can have very different regularity properties, ψ being more regular than $\tilde{\psi}$.

The $\tilde{\psi}$, ψ, $\tilde{\phi}$ and φ functions are zero outside a segment.

**Biorthogonal Filters and Transfer Functions**

This example shows how to obtain the decompositition (analysis) and reconstruction (synthesis) filters for the `'bior3.5'` wavelet.

Determine the two scaling and wavelet filters associated with the `'bior3.5'` wavelet.

```
[Rf,Df] = biorwavf('bior3.5');
[LoD,HiD,LoR,HiR] = biorfilt(Df,Rf);
```

Plot the filter impulse responses.

```
subplot(221); stem(LoD);
title('Dec. low-pass filter bior3.5');
subplot(222); stem(HiD);
title('Dec. high-pass filter bior3.5');
subplot(223); stem(LoR);
title('Rec. low-pass filter bior3.5');
```

```
subplot(224); stem(HiR);
title('Rec. high-pass filter bior3.5');
```
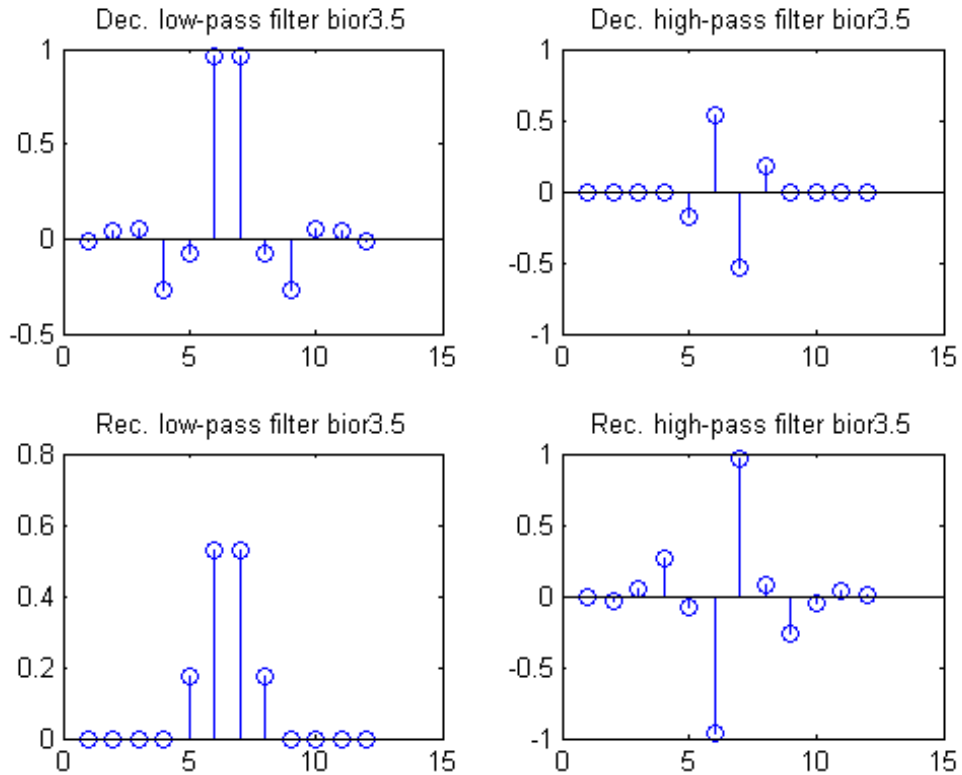


Demonstrate that autocorrelations at even lags are only zero for dual pairs of filters. Examine the autocorrelation sequence for the lowpass decomposition filter.

```
npad = 2*length(LoD)-1;
LoDxcr = fftshift(ifft(abs(fft(LoD,npad)).^2));
lags = -floor(npad/2):floor(npad/2);
```

```
figure;
stem(lags,LoDxcr,'markerfacecolor',[0 0 1])
set(gca,'xtick',-10:2:10)
```
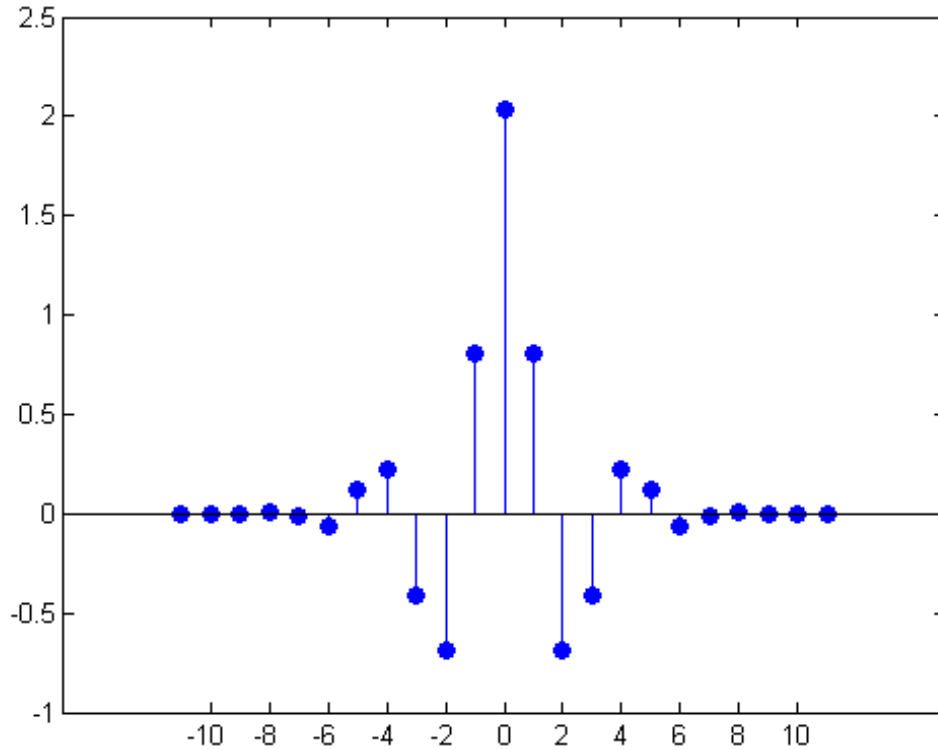


Examine the cross correlation sequence for the lowpass decomposition and synthesis filters. Compare the result with the preceding figure.

```
npad = 2*length(LoD)-1;
xcr = fftshift(ifft(fft(LoD,npad).*conj(fft(LoR,npad))));
lags = -floor(npad/2):floor(npad/2);
```

```
stem(lags,xcr,'markerfacecolor',[0 0 1])
set(gca,'xtick',-10:2:10)
```
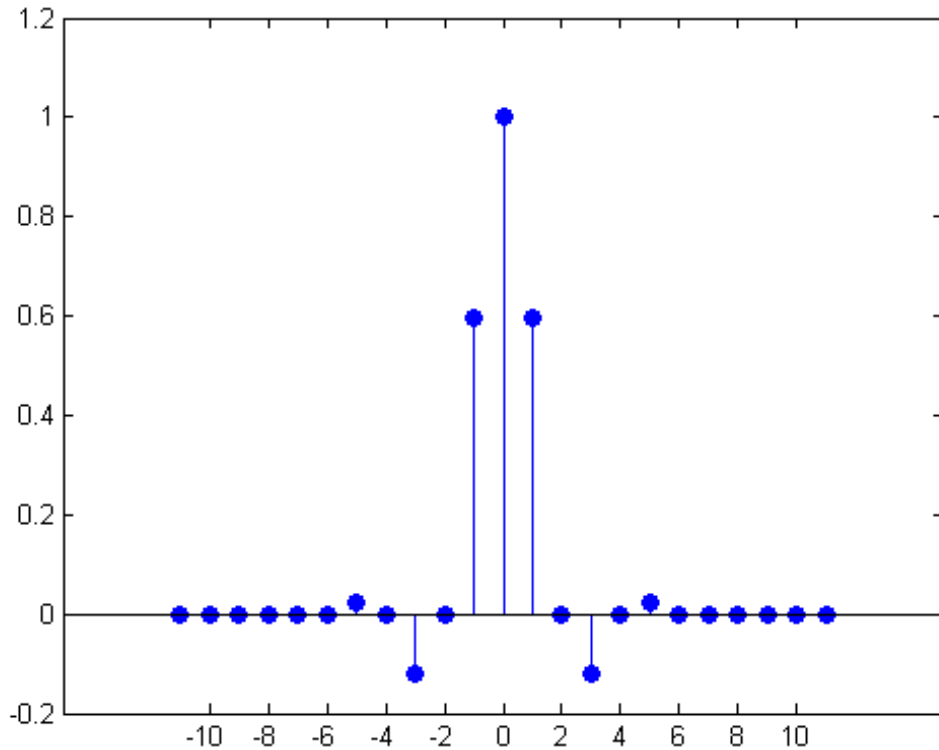


Compare the transfer functions of the analysis and synthesis scaling and wavelet filters

```
dftLoD = fft(LoD,64);
dftLoD = dftLoD(1:length(dftLoD)/2+1);
dftHiD= fft(HiD,64);
dftHiD = dftHiD(1:length(dftHiD)/2+1);
```

```
dftLoR = fft(LoR,64);
dftLoR = dftLoR(1:length(dftLoR)/2+1);
dftHiR = fft(HiR,64);
dftHiR = dftHiR(1:length(dftHiR)/2+1);
df = (2*pi)/64;
freqvec = 0:df:pi;

subplot(211); plot(freqvec,abs(dftLoD),freqvec,abs(dftHiD),'r');
axis tight;
title('Transfer modulus for dec. filters')
subplot(212); plot(freqvec,abs(dftLoR),freqvec,abs(dftHiR),'r');
axis tight;
title('Transfer modulus for rec. filters')
```

Transfer modulus for dec. filters

Transfer modulus for rec. filters
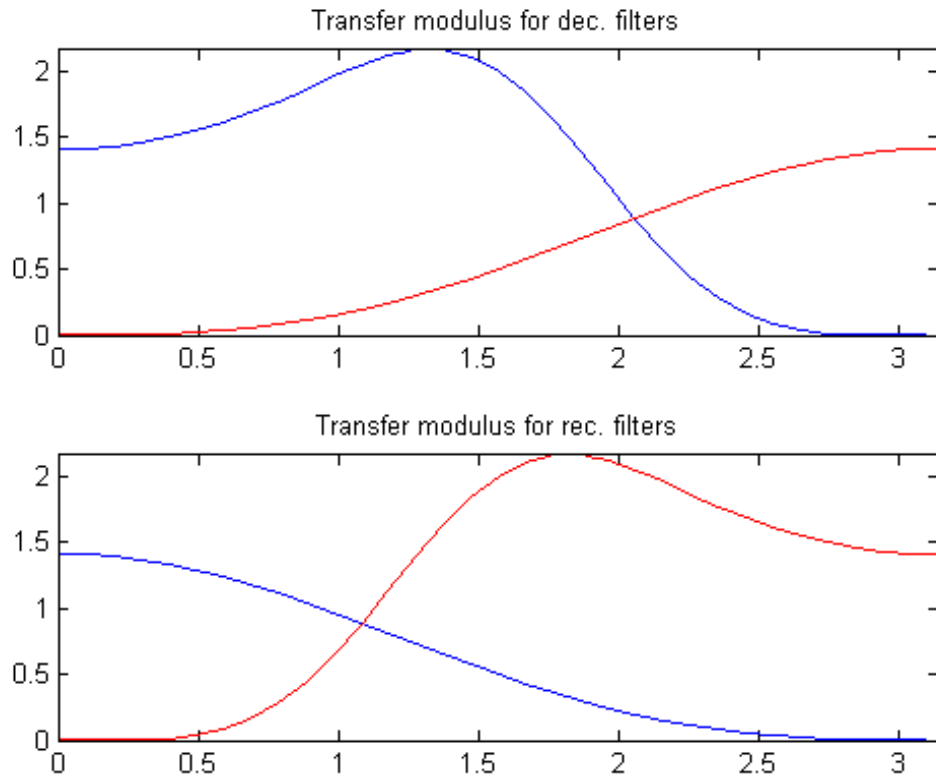
**References**    Cohen, A. (1992), "Ondelettes, analyses multirésolution et traitement numérique du signal," *Ph. D. Thesis*, University of Paris IX, DAUPHINE.

Daubechies, I. (1992), *Ten lectures on wavelets*, CBMS-NSF conference series in applied mathematics. SIAM Ed.

**See Also**    biorwavf | orthfilt

# biorwavf

| | |
|---|---|
| **Purpose** | Biorthogonal spline wavelet filter |
| **Syntax** | [RF,DF] = biorwavf(W) |

**Description**    [RF,DF] = biorwavf(W) returns the reconstruction (synthesis) and decomposition (analysis) filters associated with the biorthogonal wavelet specified by the string W.

W = 'biorNr.Nd' where possible values for *Nr* and *Nd* are

```
Nr = 1          Nd = 1 , 3 or 5
Nr = 2          Nd = 2 , 4 , 6 or 8
Nr = 3          Nd = 1 , 3 , 5 , 7 or 9
Nr = 4          Nd = 4
Nr = 5          Nd = 5
Nr = 6          Nd = 8
```

The output arguments are filters.

- RF is the reconstruction filter.
- DF is the decomposition filter.

**Examples**

```
% Set spline biorthogonal wavelet name.
wname = 'bior2.2';

% Compute the two corresponding scaling filters.
% rf is the reconstruction scaling filter.
% df is the decomposition scaling filter.
[rf,rd] = biorwavf(wname)

rf =
    0.2500   0.5000   0.2500

df =
```

```
        -0.1250    0.2500    0.7500    0.2500 -0.1250
```

**See Also**       biorfilt | waveinfo

# bswfun

| | |
|---|---|
| **Purpose** | Biorthogonal scaling and wavelet functions |
| **Syntax** | `[PHIS,PSIS,PHIA,PSIA,XVAL] = bswfun(LoD,HiD,LoR,HiR)`<br>`bswfun(LoD,HiD,LoR,HiR,ITER)`<br>`bswfun(LoD,HiD,LoR,HiR,'plot')`<br>`bswfun(LoD,HiD,LoR,HiR,ITER,'plot')`<br>`bswfun(LoD,HiD,LoR,HiR,'plot',ITER)` |
| **Description** | `[PHIS,PSIS,PHIA,PSIA,XVAL] = bswfun(LoD,HiD,LoR,HiR)` returns approximations on the grid XVAL of the two pairs of scaling function and wavelet (PHIA,PSIA), (PHIS,PSIS) associated with the two pairs of filters (LoD,HiD), (LoR,HiR).<br><br>`bswfun(LoD,HiD,LoR,HiR,ITER)` computes the two pairs of scaling and wavelet functions using ITER iterations.<br><br>`bswfun(LoD,HiD,LoR,HiR,'plot')` or<br>`bswfun(LoD,HiD,LoR,HiR,ITER,'plot')` or<br>`bswfun(LoD,HiD,LoR,HiR,'plot',ITER)` computes and plots the functions. |

**Examples**

```
% Start from the Cohen-Daubechies-Feauveau wavelet
% and get the corresponding lifting scheme.
lscdf = liftwave('cdf3.1');

% Visualize the obtained lifting scheme.
displs(lscdf);

lscdf = {...
'p'              [ -0.33333333]                 [-1]
'd'              [ -0.37500000 -1.12500000]     [1]
'p'              [  0.44444444]                 [0]
[  2.12132034]   [  0.47140452]                 []
};

% Transform the lifting scheme to biorthogonal
% filters quadruplet.
```

```
[LoD,HiD,LoR,HiR] = ls2filt(lscdf);

% Visualize the two pairs of scaling and wavelet
% functions.
bswfun(LoD,HiD,LoR,HiR,'plot');
```
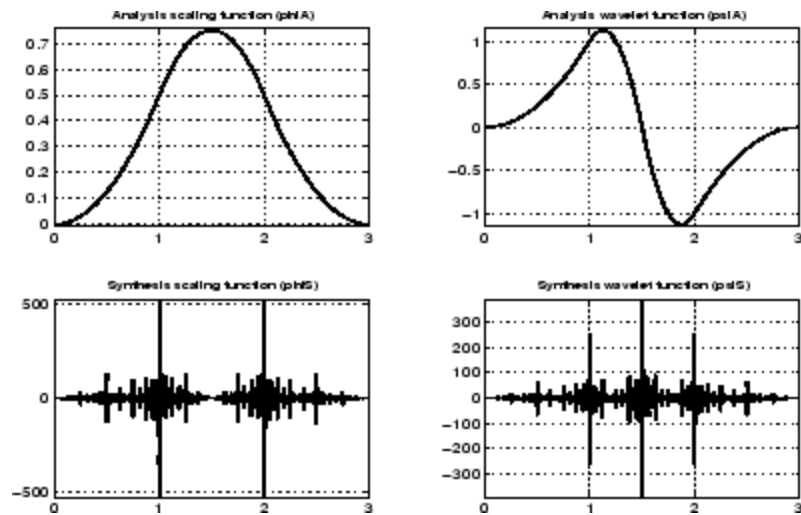


**Algorithms**    This function uses the cascade algorithm.

**See Also**    wavefun

# centfrq

| | |
|---|---|
| **Purpose** | Wavelet center frequency |
| **Syntax** | FREQ = centfrq('*wname*') |
| | FREQ = centfrq('*wname*',ITER) |
| | [FREQ,XVAL,RECFREQ] = centfrq('*wname*',ITER,'plot') |

**Description**    FREQ = centfrq('*wname*') returns the center frequency in herz of the wavelet function, '*wname*' (see wavefun for more information).

For FREQ = centfrq('*wname*',ITER), ITER is the number of iterations performed by the function wavefun, which is used to compute the wavelet.

[FREQ,XVAL,RECFREQ] = centfrq('*wname*',ITER,'plot') returns, in addition, the associated center frequency based approximation RECFREQ on the $2^{ITER}$ points grid XVAL and plots the wavelet function and RECFREQ.

**Examples**
```
% Example 1: a real wavelet
wname = 'db2';

% Compute the center frequency and display
% the wavelet function and the associated
% center frequency based approximation.
iter = 8;
cfreq = centfrq(wname,8,'plot')

cfreq =
    0.6667
```

Wavelet db2 (in bold) and Center frequency based approximation



Period: 1.5; Cent. Freq: 0.66667
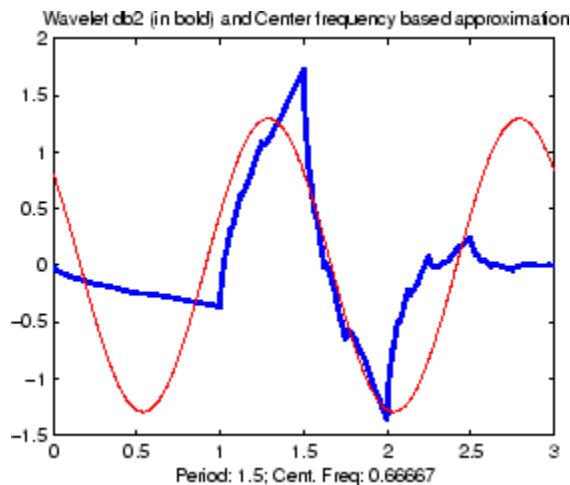
```
% Example 2: a complex wavelet
wname = 'cgau6';

% Compute the center frequency and display
% the wavelet function and the associated
% center frequency based approximation.
cfreq = centfrq(wname,8,'plot')

cfreq =

    0.6000
```

Wavelet cgau6 (in bold) and Center frequency based approximation
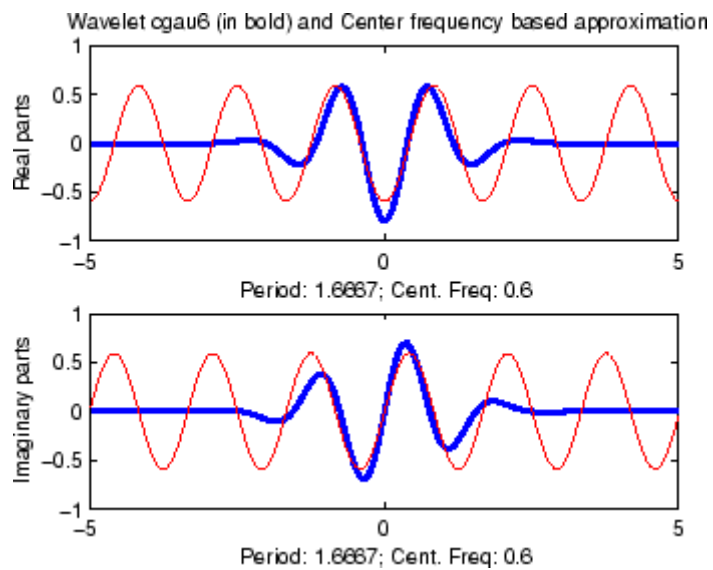
**See Also**      `scal2frq | wavefun`

**Purpose**     Wavelet packet tree construction from coefficients

**Syntax**

**Description**     CFS2WPT builds a wavelet packet tree (T) and the related analyzed signal or image (X) using the following input information:

*WNAME*: name of the wavelet used for the analysis

*SIZE_OF_DATA*: size of the analyzed signal or image

*TN_OF_TREE*: vector containing the terminal node indices of the tree

*ORDER*: 2 for a signal or 4 for an image

*CFS*: coefficients used to reconstruct the original signal or image. *CFS* is optional. When CFS2WPT is used without the *CFS* input parameter, the wavelet packet tree structure (T) is generated, but all the tree coefficients are null (including X).

**Build Wavelet Packet Tree**

This example shows how to build a wavelet packet tree in two ways: 1.) By filling the wavelet packet tree with coefficients, and 2.) By creating the wavelet packet tree and using write

Load an image and obtain the wavelet packet decomposition down to level 2 with the 'sym4' wavelet.

```
load detail;
imagesc(X); colormap gray; title('Original Image');
Tr = wpdec2(X,2,'sym4');
```

Original Image
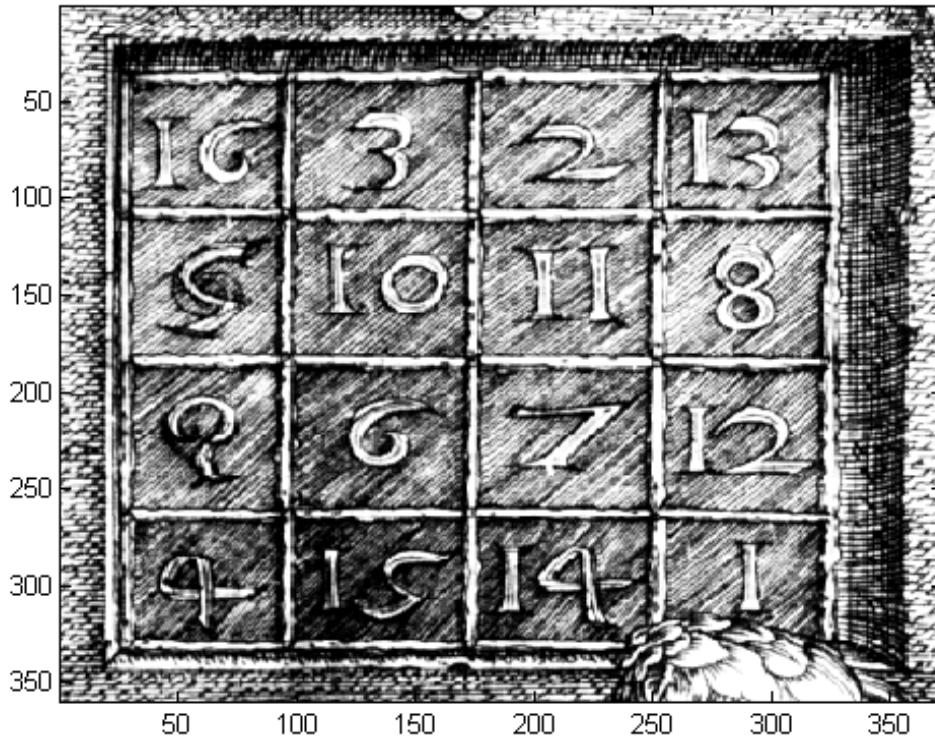


Read the coefficients from the wavelet packet tree. Add $N(0, 40^2)$ noise to the coefficients and plot the new wavelet packet tree.

```
cfs = read(Tr,'allcfs');
noisyCfs = cfs + 40*rand(size(cfs));
noisyT = cfs2wpt('sym4',size(X),tnodes(Tr),4,noisyCfs);
plot(noisyT)
```
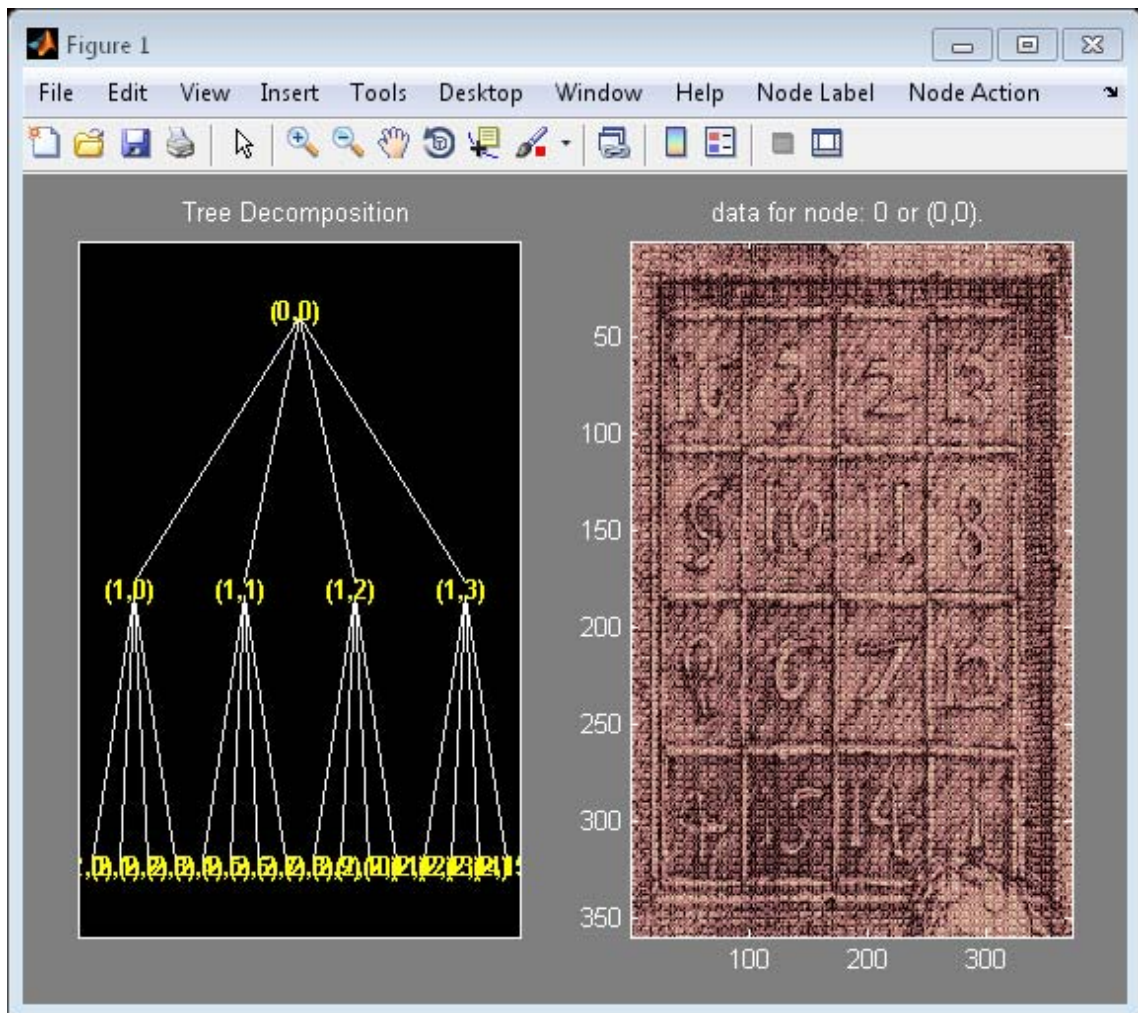
To illustrate building a wavelet packet tree using write, construct an admissible binary wavelet packet tree with terminal nodes [2 3 9 10]. The analyzing wavelet is 'sym4' and the signal length is 1024.

```
tr = cfs2wpt('sym4',[1 1024],[2 3 9 10]',2);
```

Fill terminal nodes [3 9] with $N(0,1)$ coefficients.

```
sN = read(tr,'sizes',[3,9]);
sN3 = sN(1,:); sN9 = sN(2,:);
cfsN3 = randn(sN3);
cfsN9 = randn(sN9);
tr = write(tr,'cfs',3,cfsN3,'cfs',9,cfsN9);
```

Plot the resulting wavelet packet tree and synthesized signal.

```
plot(tr)
```
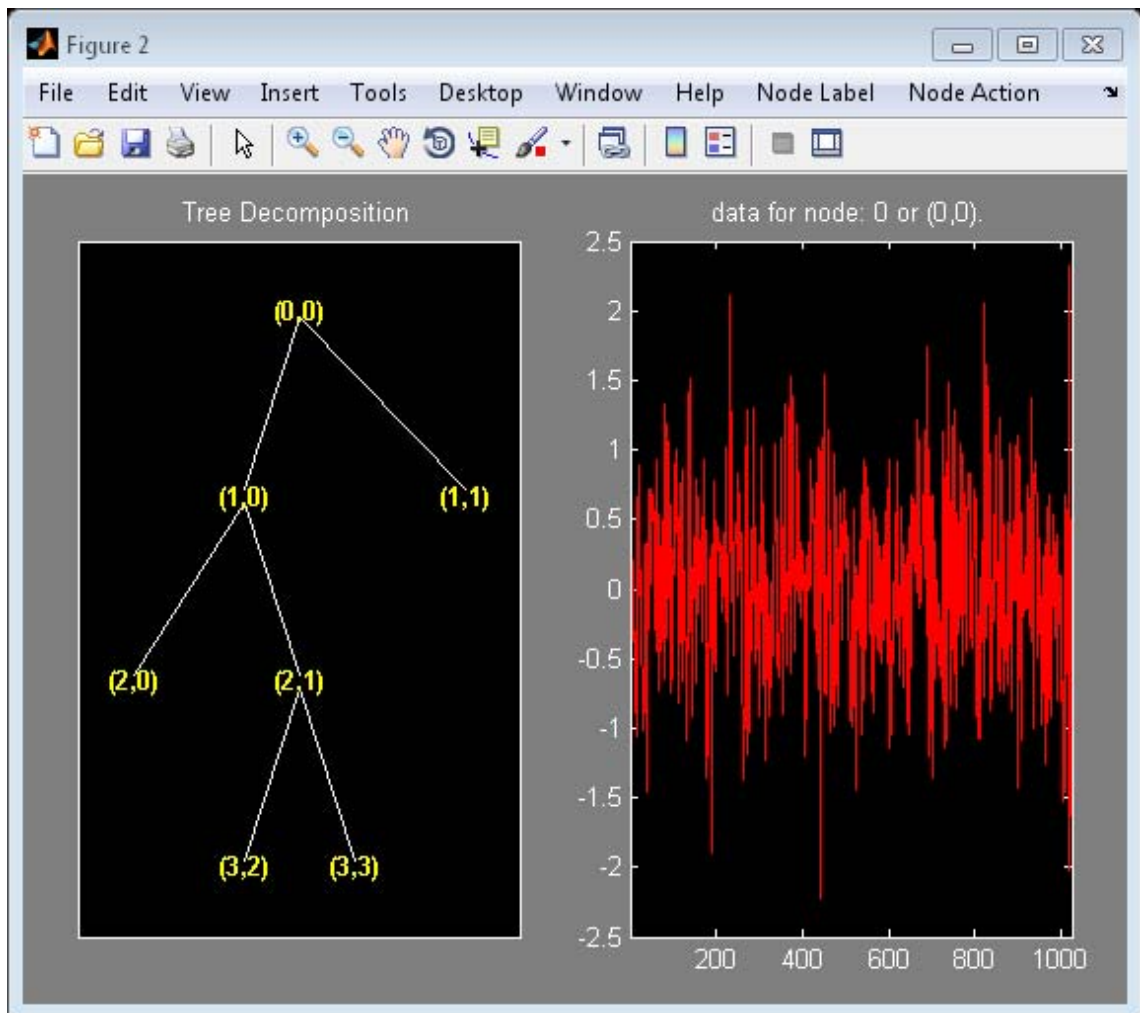
# cgauwavf

**Purpose**      Complex Gaussian wavelet

**Syntax**
```
[PSI,X] = cgauwavf(LB,UB,N,P)
[PSI,X] = cgauwavf(LB,UB,N)
[PSI,X] = cgauwavf(LB,UB,N,1)
```

**Description**   `[PSI,X] = cgauwavf(LB,UB,N,P)` returns values of the P-th derivative of the complex Gaussian function on an N point regular grid for the interval [LB,UB]. Cp is such that the 2-norm of the P-th derivative of *F* is equal to 1.

For P > 8, Symbolic Math Toolbox™ software is required.

Output arguments are the wavelet function PSI computed on the grid X.

`[PSI,X] = cgauwavf(LB,UB,N)` is equivalent to
`[PSI,X] = cgauwavf(LB,UB,N,1)`

These wavelets have an effective support of [-5 5].

**Examples**
```
% Set effective support and grid parameters.
lb = -5; ub = 5; n = 1000;

% Compute complex Gaussian wavelet of order 4.
[psi,x] = cgauwavf(lb,ub,n,4);

% Plot complex Gaussian wavelet of order 4.
subplot(211)
plot(x,real(psi)),
title('Complex Gaussian wavelet of order 4')
xlabel('Real part'), grid
 subplot(212)
 plot(x,imag(psi))
 xlabel('Imaginary part'), grid
```

Complex Gaussian wavelet of order 4

Real part

Imaginary part
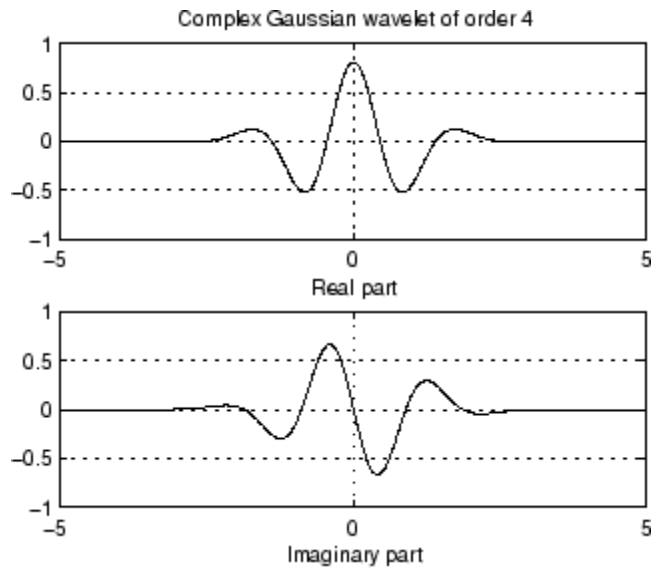
**See Also**        waveinfo

# chgwdeccfs

| **Purpose** | Change multisignal 1-D decomposition coefficients |
|---|---|

**Syntax**

```
DEC = chgwdeccfs(DEC,'ca',COEFS)
DEC = chgwdeccfs(DEC,'cd',COEFS,LEV)
DEC = chgwdeccfs(DEC,'all',CA,CD)
DEC = chgwdeccfs(DEC,'all',V)
DEC = chgwdeccfs(...,IDXSIG)
```

**Description**

DEC = chgwdeccfs(DEC,'ca',COEFS) replaces the approximation coefficients at level DEC.level with those contained in the matrix COEFS. If COEFS is a single value *V*, all coefficients are replaced by *V*.

DEC = chgwdeccfs(DEC,'cd',COEFS,LEV) replaces the detail coefficients at level LEV with those contained in the matrix COEFS. If COEFS is a single value *V*, then LEV can be a vector of levels and all the coefficients that belong to these levels are replaced by *V*. LEV must be such that $1 \leq \text{LEV} \leq \text{DEC.level}$

DEC = chgwdeccfs(DEC,'all',CA,CD) replaces all the approximation and detail coefficients. *CA* must be a matrix and *CD* must be a cell array of length DEC.level.

If COEFS (or *CA* or *CD*) is a single number, then it replaces all the related coefficients. Otherwise, COEFS (or *CA*, or *CD*) must be a matrix of appropriate size.

For a real value *V*, DEC = chgwdeccfs(DEC,'all',V) replaces all the coefficients by *V*.

DEC = chgwdeccfs(...,IDXSIG) replaces the coefficients for the signals whose indices are given by the vector IDXSIG. If the initial data are stored row-wise or column-wise in a matrix X, then IDXSIG contains the row or column indices, respectively, of the data.

**Examples**

```
% Load original 1D-multisignal
load thinker

% Perform a decomposition at level 2 using wavelet db2
dec = mdwtdec('r',X,2,'db2');
```

```
% Change the coefficients of details at level 1.
% Replace all values by 0.
decBIS = chgwdeccfs(dec,'cd',0,1);

% Change the coefficients of details at level 1 and
% level 2 for signals 31 to 35. Replace all values by 0.
decTER = chgwdeccfs(dec,'cd',0,1:2,31:35);

% Compare original and new coefficients for details
% at level 1 for signals 31 to 35.
plot(dec.cd{1}(31:35,:)','b'); hold on;
plot(decTER.cd{1}(31:35,:)','r')
```
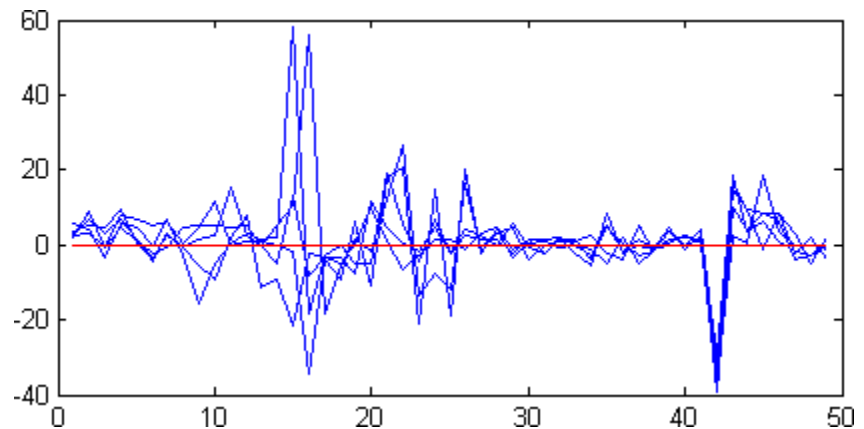


**See Also**    mdwtdec | mdwtrec

# cmddenoise

**Purpose**          Interval-dependent denoising

**Syntax**           sigden = cmddenoise(sig,wname,level)
sigden = cmddenoise(sig,wname,level,sorh)
sigden = cmddenoise(sig,wname,level,sorh,nb_inter)
sigden =
cmddenoise(sig,wname,level,sorh,nb_inter,thrParamsIn)

[sigden,coefs] = cmddenoise( ___ )
[sigden,coefs,thrParamsOut] = cmddenoise( ___ )

[sigden,coefs,thrParamsOut,int_DepThr_Cell] =
cmddenoise(sig,wname,
    level,sorh,nb_inter)
[sigden,coefs,thrParamsOut,int_DepThr_Cell,
    BestNbofInt] = cmddenoise(sig,wname,level,sorh,nb_inter)

**Description**      sigden = cmddenoise(sig,wname,level) returns the denoised signal,
sigden, obtained from an interval-dependent denoising of the signal,
sig, using the orthogonal or biorthogonal wavelet and scaling filters,
wname. cmddenoise thresholds the wavelet (detail) coefficients down
to level, level, and reconstructs a signal approximation using the
modified detail coefficients. cmddenoise partitions the signal into
intervals based on variance change points in the first level detail
coefficients and thresholds each interval separately. The location
and number of variance change points are automatically selected
using a penalized contrast function [2]. The minimum delay between
change points is 10 samples. Thresholds are obtained using a minimax
threshold rule and soft thresholding is used to modify the wavelet
coefficients [1] .

sigden = cmddenoise(sig,wname,level,sorh) returns the denoised
signal, sigden, using the thresholding method, sorh, to modify the
wavelet coefficients. Valid choices for sorh are 's' for soft thresholding
or 'h' for hard thresholding.

sigden = cmddenoise(sig,wname,level,sorh,nb_inter) returns the denoised signal, sigden, with the number of denoising intervals as a positive integer between 1 and 6: $1 \leq$ nb_inter $\leq 6$. For nb_inter $\geq 2$, cmddenoise estimates the location of the change points with a contrast function [2].

sigden = cmddenoise(sig,wname,level,sorh,nb_inter,thrParamsIn) returns the denoised signal, sigden, with the denoising intervals and corresponding thresholds specified as a cell array of matrices with length equal to level. Each element of the cell array contains the interval and threshold information for the corresponding level of the wavelet transform. The elements of thrParamsIn are N-by-3 matrices with N equal to the number of intervals. The 1st and 2nd columns contain the beginning and ending indices of the intervals and the 3rd column contains the corresponding threshold value. If you specify thrParamsIn, cmddenoise ignores the value of nb_inter.

[sigden,coefs] = cmddenoise( ___ ) returns the approximation (scaling) and detail (wavelet) coefficients, coefs. The organization of coefs is identical to the structure returned by wavedec. This syntax can include any of the input arguments used in previous syntaxes.

[sigden,coefs,thrParamsOut] = cmddenoise( ___ ) returns a cell array, thrParamsOut, with length equal to level. Each element of thrParamsOut is an N-by-3 matrix. The row dimension of the matrix elements is the number of intervals and is determined by the value of the input arguments. Each row of the matrix contains the beginning and end points (indices) of the thresholded interval and the corresponding threshold value.

[sigden,coefs,thrParamsOut,int_DepThr_Cell] = cmddenoise(sig,wname, level,sorh,nb_inter) returns a cell array, int_DepThr_Cell, with length equal to 6. int_DepThr_Cell contains interval and threshold information assuming the number of change

points ranges from 0 to 5. The N-th element of `int_DepThr_Cell` is a N-by-3 matrix containing the interval information assuming N-1 change points. Each row of the matrix contains the beginning and end points (indices) of the thresholded interval and the corresponding threshold value. Attempting to output `int_DepThr_Cell` if you use the input argument, `thrParamsIn`, results in an error.

`[sigden,coefs,thrParamsOut,int_DepThr_Cell, BestNbofInt] = cmddenoise(sig,wname,level,sorh,nb_inter)` returns the optimal number of signal intervals based on the estimated variance change points in the level-1 detail coefficients. To estimate the number of change points, `cmddenoise` assumes the total number is less than or equal to 6 and uses a penalized contrast [2]. Attempting to output `BestNbofInt` if you use the input argument, `thrParamsIn`, results in an error.

**Input Arguments**

### sig - Signal for interval-dependent denoising

1-D row or column vector

Input signal, specified as a 1-D row or column vector. `sig` is the real-valued input signal for interval-dependent denoising. The elements of `sig` are assumed to be equally spaced in time or space. If `sig` contains unequally-sampled data, `cmddenoise` is not appropriate. Use a lifting transform instead. See `lwt` for details.

**Data Types**
`double`

### wname - Wavelet name

string

Wavelet name, specified as a character array. `wname` is any valid orthogonal or biorthogonal wavelet. You can use the command: `wtype = wavemngr('fields',wname,'type','file');` to determine if the wavelet name is valid to use with `cmddenoise`. Valid wavelet names return a 1 or 2 for `wtype`.

**Example:** `'bior2.2'`, `'db4'`, `'sym4'`

**Data Types**
char

### level - Level of the decimated wavelet transform (multiresolution analysis)
positive integer

Wavelet transform (multiresolution analysis) level, specified as a positive integer. `level` gives the level of the multiresolution decomposition of the input signal using the decimated 1-D discrete wavelet transform, `wavedec`.

**Data Types**
double

### sorh - Threshold rule
'`s`' (default) | '`h`'

Thresholding rule, specified as a character array. `sorh` is the threshold rule used in the modification of the detail coefficients. Valid choices for `sorh` are '`s`' (default) and '`h`' for soft and hard thresholding.

### nb_inter - Number of intervals
positive integer in the set {1,2,3,4,5,6} | NaN

Number of intervals, specified as a positive integer less than 7. `cmddenoise` divides the input signal into `nb_inter` intervals. `cmddenoise` determines the location of the `nb_inter` change points using a contrast function [2]. If you enter `NaN` for `nb_inter`, `cmddenoise` ignores the input. If you use the input argument `thrParamsIn`, `cmddenoise` disregards any value you enter for `nb_inter`.

**Data Types**
double

### thrParamsIn - Intervals and thresholds by level
cell array of matrices

Intervals and thresholds by level, specified as a cell array of matrices equal in length to `level`. Each element of `thrParamsIn` contains the

# cmddenoise

interval and threshold information for the corresponding level of the multiresolution analysis. The elements of thrParamsIn are N-by-3 matrices with N equal to the number of intervals. The 1st and 2nd columns contain the beginning and ending indices of the intervals and the 3rd column contains the corresponding threshold value. If you specify thrParamsIn, you cannot specify the output arguments int_DepThr_Cell or BestNbofInt.

**Data Types**
cell

**Output Arguments**

### sigden - Denoised signal
1-D row or column vector

sigden is the denoised version of the input sig. sigden is a 1-D row vector equal in length to sig.

### coefs - Approximation coefficients and thresholded wavelet coefficients
1-D row vector of approximation coefficients and thresholded wavelet coefficients

coefs is a row vector of approximation (scaling) and thresholded detail (wavelet) coefficients. The ordering of the approximation and detail coefficients by level in coefs is the same as the output of wavedec. cmddenoise does not apply thresholding to the approximation coefficients.

**Data Types**
double

### thrParamsOut - Intervals and thresholds by level
cell array of matrices

thrParamsOut is a cell array of matrices equal in length to level. Each element of the cell array contains the interval and threshold information for the corresponding level of the multiresolution analysis. The elements of thrParamsOut are N-by-3 matrices with N equal to the number of intervals. N is determined by the value of the input

arguments. The 1st and 2nd columns contain the beginning and ending indices of the intervals and the 3rd column contains the corresponding threshold value.

**Data Types**
`cell`

### int_DepThr_Cell - Intervals and thresholds assuming 0 to 5 change points
cell array of matrices

`int_DepThr_Cell` contains interval and threshold information assuming the number of change points ranges from 0 to 5. The N-th element of `int_DepThr_Cell` is a N-by-3 matrix containing the interval information assuming N-1 change points. Each row of the matrix contains the beginning and ending indices of the thresholded interval and the corresponding threshold value. Attempting to output `int_DepThr_Cell` if you input the number of intervals and thresholds, `thrParamsIn`, results in an error. `int_DepThr_Cell{BestNbofInt}` or `int_DepThr_Cell{nb_inter}` is equal to the matrix elements of `thrParamsOut`.

**Data Types**
`cell`

### BestNbofInt - Optimal number of intervals
positive integer ≤6

`BestNbofInt` is the optimal number of intervals based on estimated change points in the variance of the level-1 detail coefficients. The number and location of the change points are estimated using a penalized contrast method [2]. Attempting to output `BestNbofInt` if you input the number of intervals and thresholds, `thrParamsIn`, results in an error.

# cmddenoise

**Examples**

### Denoising Blocks Signal with Haar Wavelet

Load the noisy blocks signal, `nblocr1.mat`. The signal consists of a piecewise constant signal in addtive white Gaussian noise. The variance of the additive noise differs in three disjoint intervals.

```
load nblocr1;
```

Apply interval-dependent denoising down to level 4 using the Haar wavelet. `cmddenoise` automatically determines the optimal number and locations of the variance change points. Plot the denoised and original signal for comparison.

```
sigden = cmddenoise(nblocr1,'db1',4);
plot(nblocr1);
hold on;
plot(sigden,'r','linewidth',2);
axis tight;
legend('Original Signal','Denoised Signal','Location','NorthWest');
```
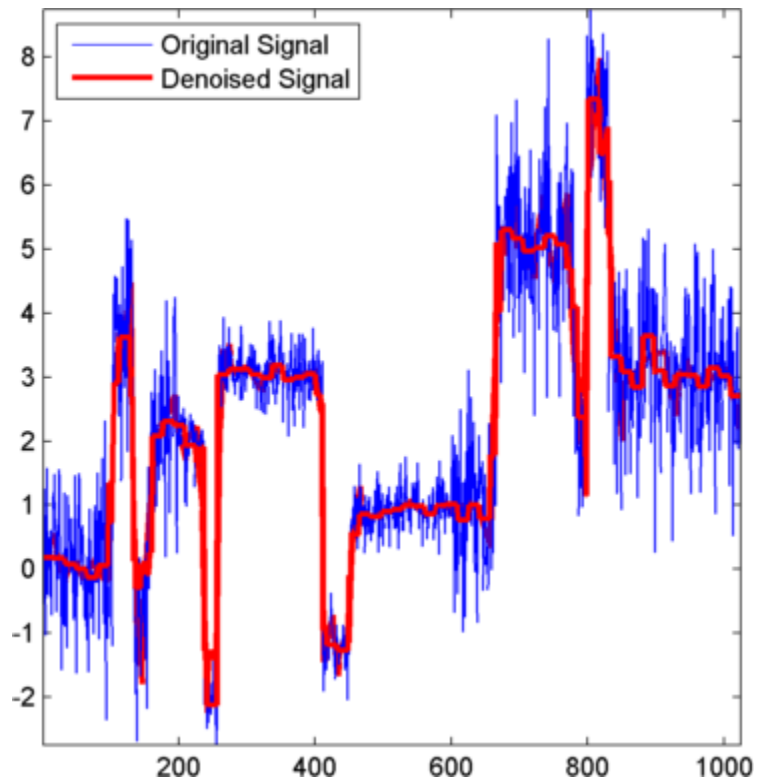
**Denoising Blocks Signal with Hard Thresholding**

Load the noisy blocks signal, nblocr1.mat. The signal consists of a piecewise constant signal in additive white Gaussian noise. The variance of the additive noise differs in three disjoint intervals.

```
load nblocr1;
```

Apply interval-dependent denoising down to level 4 using the Haar wavelet and a hard thresholding rule. cmddenoise automatically determines the optimal number and locations of the intervals. Plot the original and denoised signals.

```
sorh = 'h';
sigden = cmddenoise(nblocr1,'db1',4,sorh);
plot(nblocr1);
hold on;
plot(sigden,'r','linewidth',2);
axis tight;
legend('Original Signal','Denoised Signal','Location','NorthWest');
```

### Specify the Number of Intervals

Create a signal sampled at 1 kHz. The signal consists of a series of bumps of various widths.

```
t = [0.1  0.13  0.15  0.23  0.25  0.40  0.44  0.65  0.76  0.78  0.81];
h = [4   -5 3 -4 5  -4.2   2.1   4.3  -3.1   5.1  -4.2];
h  = abs(h);
len = 1000;
w  = 0.01*[0.5 0.5 0.6 1 1 3 1 1 0.5 0.8 0.5];
tt = linspace(0,1,len);  x = zeros(1,len);
for j=1:11
  x = x + ( h(j) ./ (1+ ((tt-t(j))/w(j)).^4));
end
plot(tt,x); title('Original Signal');
```

Add white Gaussian noise with different variances to two disjoint segments of the signal. Add zero-mean white Gaussian noise with variance equal to 2 to the signal segment from 0 to 0.3 seconds. Add zero-mean white Gaussian noise with unit variance to the signal segment from 0.3 seconds to 1 second. Set the random number generator to the default settings for reproducible results.

```
rng default;
nv1 = sqrt(2).*randn(size(tt)).*(tt<=0.3);
nv2 = randn(size(tt)).*(tt>0.3);
xx = x+nv1+nv2;
```

Apply interval-dependent denoising using the Daubechies' least-asymmetric wavelet with 5 vanishing moments down to level 3.

Set the number of intervals to 2. Plot the noisy signal, original signal, and denoised signal for comparison.

```
sigden = cmddenoise(xx,'sym5',3,'s',2);
subplot(211)
plot(tt,xx); title('Noisy Signal');
subplot(212)
plot(tt,x,'k-.','linewidth',2);
hold on;
plot(tt,sigden,'r','linewidth',2);
legend('Original Signal','Denoised Signal','Location','SouthEast');
```
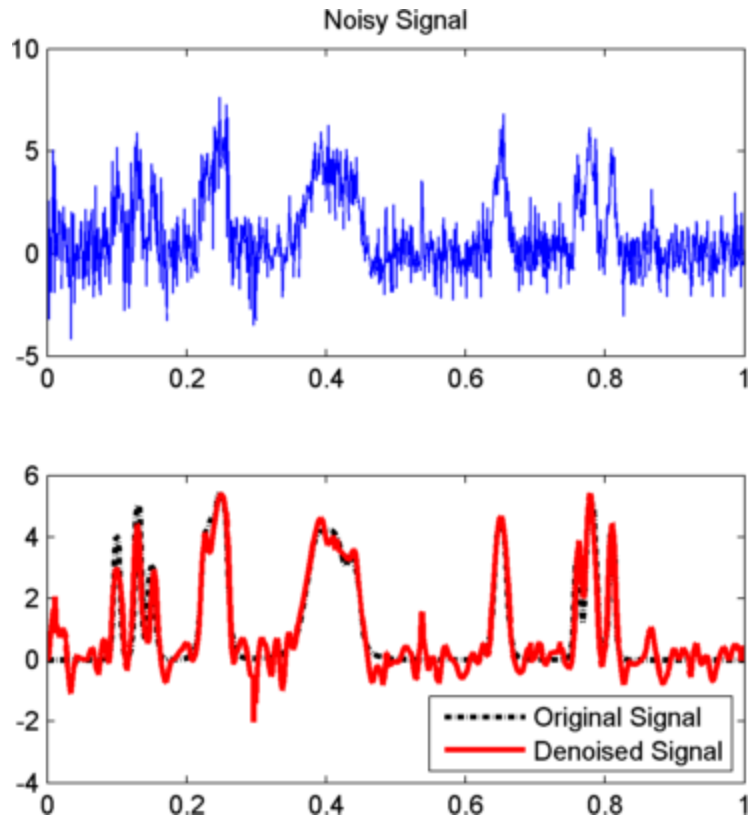
### Specify Intervals and Thresholds

Load the example signal nbumpr1.mat. The variance of the additive noise differs in three disjoint intervals.

```
load nbumpr1.mat;
```

Use a level-5 multiresolution analysis. Create a cell array of length 5 consisting of 3-by-3 matrices. The first two elements of each row contain the beginning and ending indices of the interval and the last element of each row is the corresponding threshold.

```
thrParamsIn =  {...
    [...
    1     207       1.0482; ...
    207   613       2.5110; ...
    613   1024      1.0031; ...
    ]; ...
    [...
    1     207       1.04824; ...
    207   613       3.8718; ...
    613   1024      1.04824; ...
    ]; ...
    [...
    1     207       1.04824; ...
    207   613       1.99710; ...
    613   1024      1.65613; ...
    ]; ...
    [...
    1     207       1.04824; ...
    207   613       2.09117; ...
    613   1024      1.04824; ...
    ]; ...
    [...
    1     207       1.04824; ...
    207   613       1.78620; ...
    613   102       1.04824; ...
    ]; ...
```

```
    };
```

Denoise the signal using the threshold settings and the Daubechies'
least-asymmetric wavelet with 4 vanishing moments. Use a soft
thresholding rule. Plot the noisy and denoised signals for comparison.

```
wname = 'sym4';
level = 5;
sorh = 's';
sigden = cmddenoise(nbumpr1,wname,level,sorh,NaN,thrParamsIn);
plot(nbumpr1); hold on;
plot(sigden,'r','linewidth',2); axis tight;
legend('Noisy Signal','Denoised Signal','Location','NorthEast');
```
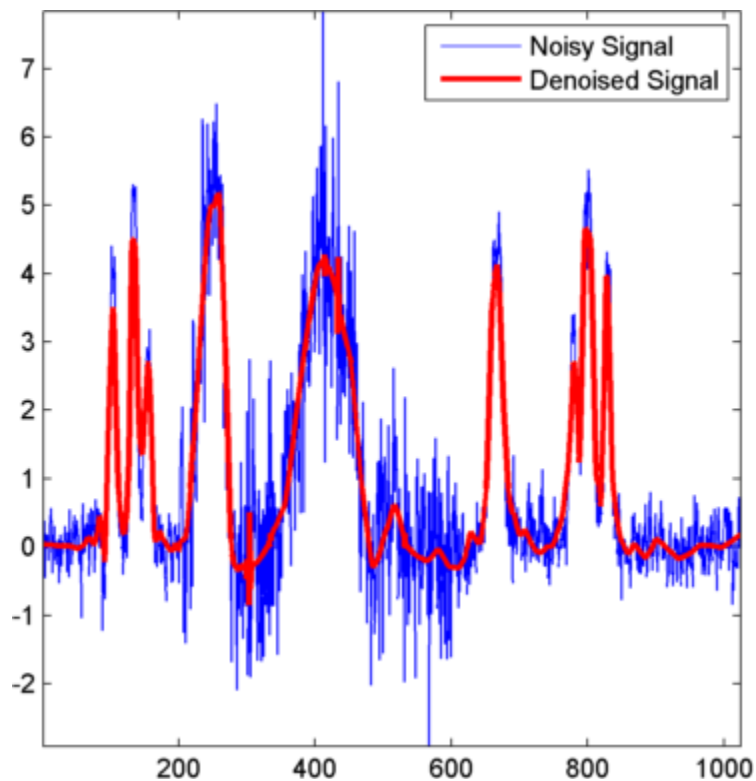
**Return Denoised Wavelet Coefficients**

Load the example signal nblocr1.mat. Use the Haar wavelet and decompose the signal down to level 2. Obtain the discrete wavelet transform and denoise the signal. Return the wavelet coefficients of the noisy and denoised signals.

```
load nblocr1.mat;
[sigden,coefs] = cmddenoise(nblocr1,'db1',2);
[C,L] = wavedec(nblocr1,2,'db1');
```

Plot reconstructions based on the level-2 approximation and level-2 and level-1 detail coefficients for the noisy signal.

```
app = wrcoef('a',C,L,'db1',2);
subplot(3,1,1);
plot(app); title('Approximation Coefficients');
for nn = 1:2
    det = wrcoef('d',C,L,'db1',nn);
    subplot(3,1,nn+1)
    plot(det); title(['Noisy Wavelet Coefficients - Level ' num2str(nn
end
```
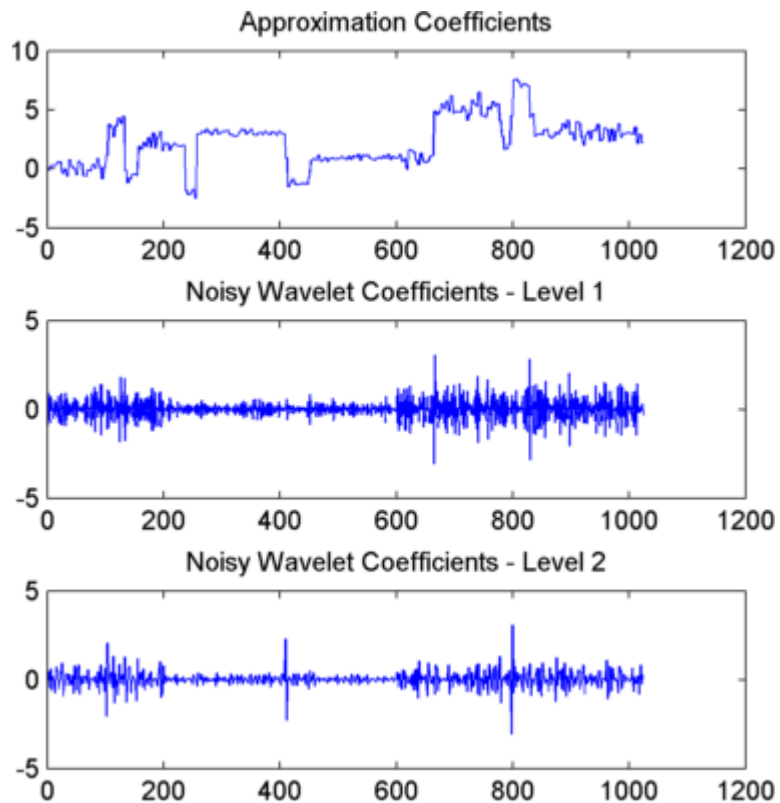


Plot reconstructions based on the approximation and detail coefficients for the denoised signal at the same levels.

```
figure;
app = wrcoef('a',coefs,L,'db1',2);
subplot(3,1,1);
plot(app); title('Approximation Coefficients');
for nn = 1:2
    det = wrcoef('d',coefs,L,'db1',nn);
    subplot(3,1,nn+1)
    plot(det);
    title(['Thresholded Wavelet Coefficients - Level '  num2str(nn)]);
end
```
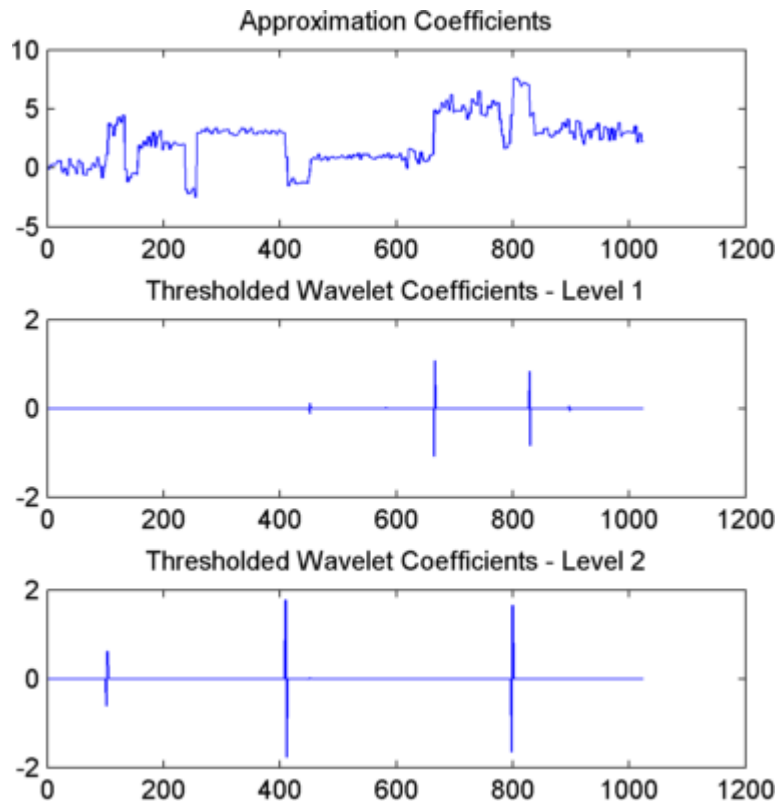
The approximation coefficients are identical in the noisy and denoised signal, but most of the detail coefficients in the denoised signal are close to zero.

### Output Intervals and Thresholds

Create a signal sampled at 1 kHz. The signal consists of a series of bumps of various widths.

```
t = [0.1  0.13  0.15  0.23  0.25  0.40  0.44  0.65  0.76  0.78  0.81];
h = [4  -5 3 -4 5  -4.2  2.1  4.3 -3.1  5.1  -4.2];
h  = abs(h);
len = 1000;
w  = 0.01*[0.5 0.5 0.6 1 1 3 1 1 0.5 0.8 0.5];
tt = linspace(0,1,len);  x = zeros(1,len);
for j=1:11
  x = x + ( h(j) ./ (1+ ((tt-t(j))/w(j)).^4));
end
plot(tt,x); title('Original Signal');
```

Add white Gaussian noise with different variances to two disjoint segments of the signal. Add zero-mean white Gaussian noise with variance equal to 2 to the signal segment from 0 to 0.3 seconds. Add zero-mean white Gaussian noise with unit variance to the signal segment from 0.3 seconds to 1 second. Set the random number generator to the default settings for reproducible results.

```
rng default;
nv1 = sqrt(2).*randn(size(tt)).*(tt<=0.3);
nv2 = randn(size(tt)).*(tt>0.3);
xx = x+nv1+nv2;
```

Apply interval-dependent denoising using the Daubechies' least-asymmetric wavelet with 4 vanishing moments down to level 5. Automatically choose the number of intervals and output the result.

```
[sigden,coefs,thrParamsOut] = cmddenoise(xx,'sym4',5);
thrParamsOut{1}
```

cmddenoise identifies one variance change point in the 1st level detail coefficients defining two intervals. The first interval contains samples 1 to 293. The second interval contains samples 293 to 1000. This is close to the true variance change point, which occurs at sample 299.

### Partition Signal into Increasing Numbers of Intervals with Thresholds

Load the example signal, nbumpr1.mat. Partition the signal into 1 to 6 intervals assuming 0 to 5 change points. Compute the thresholds for each interval. Using the Daubechies' least-asymmetric wavelet with 4 vanishing moments return the intervals and corresponding thresholds. Display the results in the command window.

```
load nbumpr1.mat;
[sigden,~,~,int_DepThr_Cell] = cmddenoise(nbumpr1,'sym4',1);
format bank;
disp('          Begin        End          Threshold ');
cellfun(@disp,int_DepThr_Cell,'UniformOutput',false);
format;
```

### Detect Number of Change Points

Load the example signal, nbumpr1.mat. The signal has two variance change points, which results in three intervals.

Use cmddenoise to detect the number of change points. Print the result.

```
load nbumpr1.mat;
[sigden,~,thrParamsOut,~,bestNbofInt] = cmddenoise(nbumpr1,'sym4',1);
fprintf('Found %d change points.\n',bestNbofInt-1);
```

## References

[1] Donoho, D. and Johnstone, I. "Ideal spatial adaptation by wavelet shrinkage", *Biometrika*, 1994, 81,3, 425–455.

[2] Lavielle, M. "Detection of multiple changes in a sequence of dependent variables", *Stochastic Processes and their Applications*, 1999, 83, 79–102.

**See Also**    thselect **|** wavedec **|** wthresh **|** wvarchg

# cmorwavf

| | |
|---|---|
| **Purpose** | Complex Morlet wavelet |
| **Syntax** | `[PSI,X] = cmorwavf(LB,UB,N,FB,FC)` |

**Description**   `[PSI,X] = cmorwavf(LB,UB,N,FB,FC)` returns values of the complex Morlet wavelet defined by a positive bandwidth parameter *FB*, a wavelet center frequency *FC*, and the expression

```
PSI(X) = ((pi*FB)^(-0.5))*exp(2*i*pi*FC*X)*exp(-X^2/FB)
```

on an N point regular grid for the interval [*LB*,*UB*].

Output arguments are the wavelet function PSI computed on the grid X.

**Complex Morlet Wavelet**   Construct a complex-valued Morlet wavelet with a bandwidth parameter of 1.5 and a center frequency of 1. Set the effective support to $[-8, 8]$ and the length of the wavelet to 1000.

```
N = 1000;
Lb = -8;
Ub = 8;
fb = 1.5;
fc = 1;
[psi,x] = cmorwavf(Lb,Ub,N,fb,fc);
```

Plot the real and imaginary parts of the wavelet.

```
subplot(2,1,1)
plot(x,real(psi)); title('Real Part');
subplot(2,1,2)
plot(x,imag(psi)); title('Imaginary Part');
```
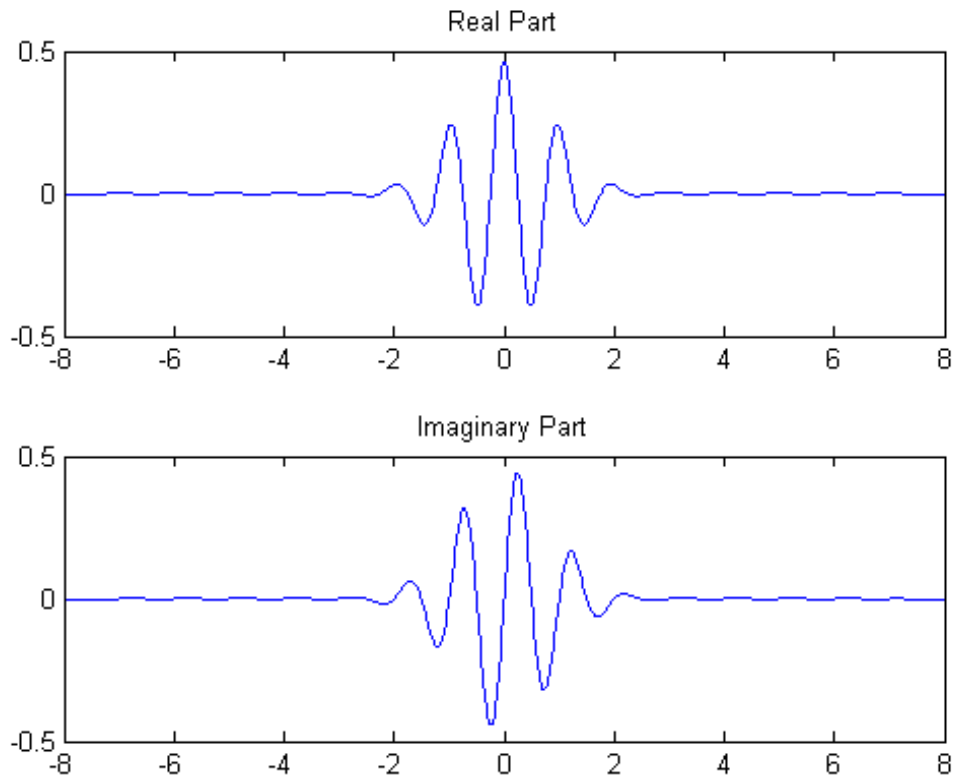
**References**  Teolis, A. (1998), *Computational signal processing with wavelets*, Birkhauser, p. 65.

**See Also**  `waveinfo`

# coifwavf

**Purpose**      Coiflet wavelet filter

**Syntax**       F = coifwavf(*W*)

**Description**  F = coifwavf(*W*) returns the scaling filter associated with the Coiflet
wavelet specified by the string *W* where W = 'coifN'. Possible values
for N are 1, 2, 3, 4, or 5.

**Examples**     % Set coiflet wavelet name.
wname = 'coif2';

% Compute the corresponding scaling filter.
f = coifwavf(wname)

f =
Columns 1 through 7
0.0116  -0.0293  -0.0476 0.2730 0.5747  0.2949  -0.0541

Columns 8 through 12
-0.0420  0.0167  0.0040 -0.0013 -0.0005

**See Also**     waveinfo

| | |
|---|---|
| **Purpose** | Cone of influence |
| **Syntax** | cone = conofinf(wname,scales,LenSig,SigVal)<br>[cone,PL,PR] = conofinf(wname,scales,LenSig,SigVal)<br>[cone,PL,PR,PLmin,PRmax] =<br>conofinf(wname,scales,LenSig,SigVal)<br>[PLmin,PRmax] = conofinf(wname,scales,LenSig)<br>[...] = conofinf(...,'plot') |

**Description**   cone = conofinf(wname,scales,LenSig,SigVal) returns the cone of influence (COI) for the wavelet wname at the scales in scales and positions in SigVal. LenSig is the length of the input signal. If SigVal is a scalar, cone is a matrix with row dimension length(scales) and column dimension LenSig. If isa vector, cone is cell array of matrices.

[cone,PL,PR] = conofinf(wname,scales,LenSig,SigVal) returns the left and right boundaries of the cone of influence atscale1for the points in . PL and PR are length(SigVal)-by-2 matrices. The left boundaries are(1-PL(:,2))./PL(:,1) and therightboundariesare(1-PR(:,2))./PR(:,1).

[cone,PL,PR,PLmin,PRmax] =
conofinf(wname,scales,LenSig,SigVal) returns the equations of the lines that define the minimal left and maximal right boundaries of the cone of influence. PLmin and PRmax are 1-by-2 row vectors where PLmin(1) and PRmax(1) are the slopes of the lines. PLmin(2) and PRmax(2) are the points where the lines intercept the scale axis.

[PLmin,PRmax] = conofinf(wname,scales,LenSig) returns the slope and intercept terms for the first-degree polynomials defining the minimal left and maximal right vertices of the cone of influence.

[...] = conofinf(...,'plot') plots the cone of influence.

**Input Arguments**   **wname**

wname is a string corresponding to a valid wavelet. To verify that wname is a valid wavelet, wavemngr('fields',wname) must return a struct array with a type field of 1 or 2, or a nonempty bound field.

### scales

scales is a vector of scales over which to compute the cone of influence. Larger scales correspond to stretched versions of the wavelet and larger boundary values for the cone of influence.

### LenSig

LenSig is the signal length and must exceed the maximum of SigVal.

### SigVal

SigVal is a vector of signal values at which to compute the cone of influence. The largest value of SigVal must be less than the signal length, LenSig.If SigVal is empty, conofinf returns the slope and intercept terms for the minimal left and maximal right vertices of the cone of influence.

**Output Arguments**

### cone

cone isthe cone of influence. If SigVal is a scalar, cone is a matrix. The row dimension is equal to the number of scales and column dimension equal to the signal length, LenSig. If SigVal is a vector, cone is a cell array of matrices. The elements of each row of the matrix are equal to 1 in the interval around SigVal corresponding to the cone of influence.

### PL

PL is the minimum value of the cone of influence on the position (time) axis.

### PR

PR is the maximum value of the cone of influence on the position (time) axis.

### PLmin

PLmin is a 1-by-2 row vector containing the slope and scale axis intercept of the line defining the minimal left vertex of the cone of influence. PLmin(1) is the slope and PLmin(2) is the point where the line intercepts the scale axis.

**PRmax**

PRmax is a 1-by-2 row vector containing the slope and scale axis intercept of the line defining the maximal right vertex of the cone of influence. PRmax(1) is the slope and PRmax(2) is the point where the line intercepts the scale axis.

**Definitions**     **Cone of Influence**

Let ψ(t) be an admissible wavelet. Assume that the effective support of ψ(t) is [-B,B]. Letting $u$ denote the translation parameter and $s$ denote the scale parameter, the dilated and translated wavelet is:

$$\psi_{u,s}(t) = \frac{1}{\sqrt{s}}\psi(\tfrac{t-u}{s})$$

andhas effective support [u-sB,u+sB]. The cone of influence (COI) is the set of all $t$ included in the effective support of the wavelet at a given position and scale. This set is equivalent to:

$$|t-u| \leq sB$$

At each scale, the COI determines the set of wavelet coefficients influenced by the value of the signal at a specified position.

**Examples**     Cone of influence for Mexican hat wavelet:

```
load cuspamax
signal = cuspamax;
wname  = 'mexh';
scales = 1:64;
lenSIG = length(signal);
x = 500;
```

# conofinf

```
figure;
cwt(signal,scales,wname,'plot');
hold on
[cone,PL,PR,Pmin,Pmax] = conofinf(wname,scales,lenSIG,x,'plot');
set(gca,'Xlim',[1 lenSIG])
```

Left minimal and right maximal vertices for the cone of influence (Morlet wavelet):

```
[PLmin,PRmax] = conofinf('morl',1:32,1024,[],'plot');
% PLmin = -0.1245*u+ 32.0000
% PRmax =  0.1250*u-96.0000
```

**References**    Mallat, S. *A Wavelet Tour of Signal Processing*, London:Academic Press, 1999, p. 174.

**See Also**    cwt | wavsupport

**How To**    • "Continuous Wavelet Transform"

**Purpose**      Continuous 1-D wavelet transform

**Syntax**
```
coefs = cwt(x,scales,'wname')
coefs = cwt(x,scales,'wname','plot')
coefs = cwt(x,scales,'wname','coloration')
[coefs,sgram] = cwt(x,scales,'wname','scal')
[coefs,sgram] = cwt(x,scales,'wname','scalCNT')
coefs = cwt(x,scales,'wname','coloration',xlim)
```

**Description**      coefs = cwt(x,scales,'*wname*') computes the continuous wavelet transform (CWT) coefficients of the real-valued signal x at real, positive scales, using wavelet '*wname*' (see waveinfo for more information). The analyzing wavelet can be real or complex. coefs is an *la*-by-*lx* matrix, where *la* is the length of scales and *lx* is the length of the input x. coefs is a real or complex matrix, depending on the wavelet type.

coefs = cwt(x,scales,'*wname*','plot') plots the continuous wavelet transform coefficients, using default coloration 'absglb'.

coefs = cwt(x,scales,'*wname*','*coloration*') uses the specified coloration.

[coefs,sgram] = cwt(x,scales,'*wname*','scal') displays a scaled image of the scalogram.

[coefs,sgram] = cwt(x,scales,'*wname*','scalCNT') displays a contour representation of the scalogram.

coefs = cwt(x,scales,'*wname*','*coloration*',xlim) colors the coefficients using coloration and xlim, where xlim is a vector, [x1 x2], with $1 \leq$ x1 < x2 $\leq$ length(x).

**Definitions**      **Scale values**

*Scale values* determine the degree to which the wavelet is compressed or stretched. Low scale values compress the wavelet and correlate better with high frequencies. The low scale CWT coefficients represent the fine-scale features in the input signal vector. High scale values stretch the wavelet and correlate better with the low frequency content

of the signal. The high scale CWT coefficients represent the coarse-scale features in the input signal.

### Coloration

*Coloration* is the method used to scale the coefficient values for plotting. Each coefficient is divided by the resulting coloration value.

- `'lvl'` — uses maximum value in each scale

- `'glb'` — uses maximum value in all scales

- `'abslvl'` or `'lvlabs'` — uses maximum absolute value in each scale

- `'absglb'` or `'glbabs'` — uses maximum absolute value in all scales

- `'scal'` — produces a scaled image of the scalogram

- `'scalCNT'` — produces a contour representation of the scalogram

For 3-D plots (surfaces), use the coloration parameter preceded by `'3D'`, such as coefs = cwt(...,'3Dplot') or coefs = cwt(...,'3Dlvl') ...

### Scalogram

*Scalograms* are plots that represent the percentage energy for each coefficient.
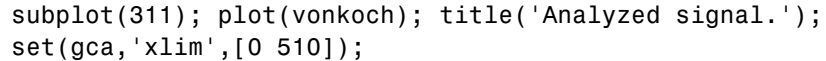
**Examples**  Plot the continuous wavelet transform and scalogram using sym2 wavelet at all integer scales from 1 to 32, using a fractal signal as input:

```
load vonkoch
vonkoch=vonkoch(1:510);
len = length(vonkoch);
cw1 = cwt(vonkoch,1:32,'sym2','plot');
title('Continuous Transform, absolute coefficients.')
ylabel('Scale')
[cw1,sc] = cwt(vonkoch,1:32,'sym2','scal');
title('Scalogram')
ylabel('Scale')
```

Compare discrete and continuous wavelet transforms, using a fractal signal as input:
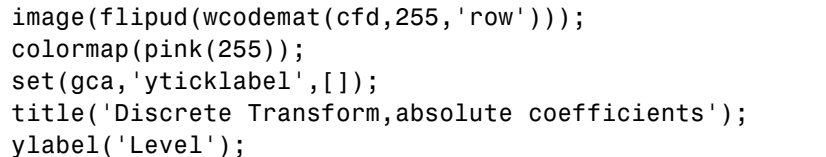
```
load vonkoch
vonkoch=vonkoch(1:510);
len=length(vonkoch);
[c,l]=wavedec(vonkoch,5,'sym2');
% Compute and reshape DWT to compare with CWT.
cfd=zeros(5,len);
for k=1:5
    d=detcoef(c,l,k);
    d=d(ones(1,2^k),:);
    cfd(k,:)=wkeep(d(:)',len);
end
cfd=cfd(:);
I=find(abs(cfd) <sqrt(eps));
cfd(I)=zeros(size(I));
cfd=reshape(cfd,5,len);
% Plot DWT.
subplot(311); plot(vonkoch); title('Analyzed signal.');
set(gca,'xlim',[O 510]);
subplot(312);
image(flipud(wcodemat(cfd,255,'row')));
colormap(pink(255));
set(gca,'yticklabel',[]);
title('Discrete Transform,absolute coefficients');
ylabel('Level');
% Compute CWT and compare with DWT
subplot(313);
ccfs=cwt(vonkoch,1:32,'sym2','plot');
title('Continuous Transform, absolute coefficients');
set(gca,'yticklabel',[]);
ylabel('Scale');
```
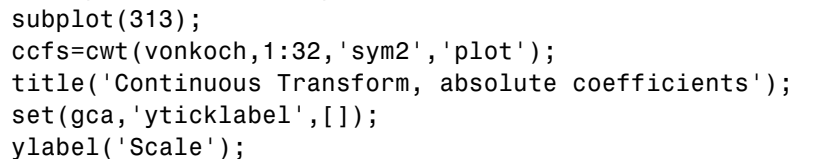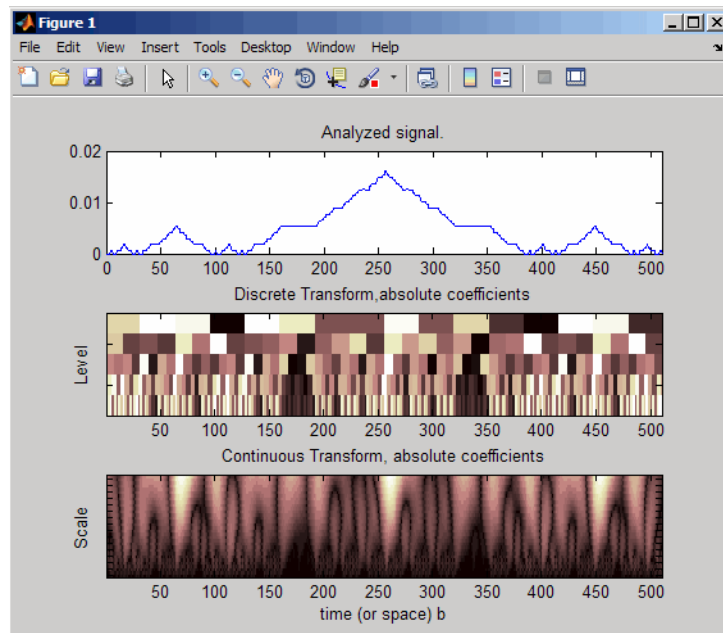
**References**    Daubechies, I. *Ten Lectures on Wavelets*, Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 1992.

Mallat, S. *A Wavelet Tour of Signal Processing*, San Diego, CA: Academic Press, 1998.

**How To**    "Continuous Wavelet Transform"

"1-D Continuous Wavelet Analysis"

"New Wavelet for CWT"

**See Also**    cwtext | dwt | wavedec | wavefun | waveinfo | wcodemat

**Purpose**     Continuous wavelet transform using FFT algorithm

**Syntax**      ```
cwtstruct = cwtft(sig)
cwtstruct = cwtft(sig,Name,Value)
cwtstruct = cwtft(...,'plot')
```

**Description**  `cwtstruct = cwtft(sig)` returns the continuous wavelet transform (CWT) of the 1–D input signal `sig`. `cwtft` uses an FFT algorithm to compute the CWT. `sig` can be a vector, a structure array, or a cell array. If the sampling interval of your signal is not equal to 1, you must input the sampling period with `sig` in a cell array or a structure array to obtain correct results. If `sig` is a cell array, `sig{1}` is equal to your signal and `sig{2}` is equal to the sampling interval. If `sig` is a structure array, the field `sig.val` contains your signal and `sig.period` contains the sampling interval.

By default, `cwtft` uses the analytic Morlet wavelet. See "Wavelet Definitions" on page 1-80 for a description of valid analyzing wavelets.

For additional default values, see `scales` in "Name-Value Pair Arguments" on page 1-78.

`cwtstruct = cwtft(sig,Name,Value)` returns the continuous wavelet transform (CWT) of the 1–D input signal `sig` with additional options specified by one or more `Name,Value` pair arguments. See "Name-Value Pair Arguments" on page 1-78 for a comprehensive list.

`cwtstruct = cwtft(...,'plot')` plots the continuous wavelet transform. If the analyzing wavelet is real-valued, the original signal along with the CWT coefficient magnitudes and signed CWT coefficients are plotted. If the analyzing wavelet is complex-valued, the original signal is plotted along with the moduli, real parts, imaginary parts, and angles of the CWT coefficients. You can select the radio button in the bottom left of the plot to superimpose the signal's reconstruction using `icwtft`.

**Input Arguments**

**sig**

The 1–D input signal. `sig` can be a vector, a structure array, or a cell array. If `sig` is a structure array, `sig` contains two fields: `val` and `period`. `sig.val` is the signal vector and `sig.period` is the sampling period. If `sig` is a cell array, `sig{1}` is the signal vector and `sig{2}` is the sampling period.

If `sig` is a vector, the sampling period defaults to `1`.

---

**Note** If the sampling interval of your input signal is not 1, you must input the sampling interval with `sig` in a cell array or structure array to obtain correct results. If `sig` is a cell array, `sig{1}` is the 1–D input signal and `sig{2}` is the sampling period. If `sig` is a structure array, the field `sig.val` is the 1–D input signal and `sig.period` is the sampling interval.

---

**Name-Value Pair Arguments**

**'scales'**

Scales over which to compute the CWT. The value of `scales` can be a vector, a structure array, or a cell array. If `scales` is a structure array, it contains at most five fields. The first three fields are mandatory. The last two fields are optional.

**1** `s0` — The smallest scale. The default `s0` depends on the wavelet. See "Wavelet Definitions" on page 1-80 for the wavelet-dependent default.

**2** `ds` — Spacing between scales. The default `ds` depends on the wavelet. See "Wavelet Definitions" on page 1-80 for the wavelet-dependent default. You can construct a linear or logarithmic scale vector using `ds`. See type for a description of the type of spacing.

**3** `nb` — Number of scales. The default `nb` depends on the wavelet. See "Wavelet Definitions" on page 1-80 for the wavelet-dependent default.

**4** type — Type of spacing between scales. `type` can be one of `'pow'` or `'lin'`. The default is `'pow'`. If `type` is equal to `'pow'`, the CWT scales are `s0*pow.^((0:nb-1)*ds)`. This results in a constant spacing of `ds` if you take the logarithm to the base `power` of the scales vector. If `type` is equal to `'lin'`, the CWT scales are linearly spaced by `s0 + (0:nb-1)*ds`.

Use the default power of two spacing to ensure an accurate approximation to the original signal based only on select scales. See the second example in "Examples" on page 1-82 for a signal approximation based on select scales.

**5** pow — The base for `'pow'` spacing. The default is 2. This input is valid only if the `type` argument is `'pow'`.

If `scales` is a cell array, the first three elements of the cell array are identical to the first three elements of the structure array described in the preceding list. The last two elements of the cell array are optional and match the two optional inputs in the structure array described in the preceding list.

**'wavelet'**

Analyzing wavelet. The supported analyzing wavelets are:

- `'dog'` — $m$-th order derivative of a Gaussian wavelet where $m$ is a positive even integer. The default value of $m$ is 2.

- `'morl'` — Morlet wavelet. Results in an analytic Morlet wavelet. The Fourier transform of an analytic wavelet is zero for negative frequencies.

- `'morlex'` — non-analytic Morlet wavelet

- `'morl0'` — non-analytic Morlet wavelet with zero mean

- `'mexh'` — Mexican hat wavelet. The Mexican hat wavelet is a special case of the $m$-th order derivative of a Gaussian wavelet with $m$=2.

- `'paul'` — Paul wavelet

See "Wavelet Definitions" on page 1-80 for formal definitions of the supported analyzing wavelets and associated defaults.

> **Default:** `'morl'`

**Output Arguments**

**cwtstruct**

A structure array with six fields. The fields of the structure array are:

- `dt` — The sampling interval of the 1–D input signal

- `cfs` — The CWT coefficient matrix. `cwtstruct.cfs` is an `nb`-by-N matrix where `nb` is the number of scales and N is the length of the input signal.

- `meanSIG` — Mean of the analyzed signal

- `omega` — Vector of angular frequencies

- `scales` — Vector of scales at which the CWT is computed. The length of `cwtstruct.scales` is equal to the row dimension of `cwtstruct.cfs`.

- `wav` — Analyzing wavelet

**Definitions**

**Wavelet Definitions**

**Morlet Wavelet**

Both non-analytic and analytic Morlet wavelets are supported. The analytic Morlet wavelet, `'morl'`, is defined in the Fourier domain by:

$$\hat{\Psi}(s\omega) = \pi^{-1/4} e^{(s\omega-\omega_0)^2/2} U(s\omega)$$

where $U(\omega)$ is the Heaviside step function [5].

The non-analytic Morlet wavelet, `'morlex'`, is defined in the Fourier domain by:

$$\hat{\Psi}(s\omega) = \pi^{-1/4} e^{(s\omega-\omega_0)^2/2}$$

`'morl0'` defines a non-analytic Morlet wavelet in the Fourier domain with exact zero mean:

$$\hat{\Psi}(s\omega) = \pi^{-1/4}\{e^{(s\omega - \omega_0)^2/2} - e^{\omega_0^2/2}\}$$

The default value of $\omega_0$ is 6.

The scale-to-frequency Fourier factor for the Morlet wavelet is:

$$\frac{4\pi s}{\omega_0 + \sqrt{2 + \omega_0^2}}$$

The default smallest scale for the Morlet wavelets is `2*dt` where `dt` is the sampling period.

The default spacing between scales for the Morlet wavelets is `ds=0.4875`.

The default number of scales for the Morlet wavelets is `fix(log2(length(sig))/ds)+1`.

### *m*-th Order Derivative of Gaussian Wavelets

In the Fourier domain, the *m*-th order derivative of Gaussian wavelets, `'dog'`, are defined by:

$$\hat{\Psi}(s\omega) = -\frac{1}{\sqrt{\Gamma(m + 1/2)}}(js\omega)^m e^{-(s\omega)^2/2}$$

where $\Gamma(\ )$ denotes the gamma function [5].

The derivative must be an even order. The default order of the derivative is 2, which is also known as the Mexican hat wavelet .

The scale-to-frequency Fourier factor for the DOG wavelet is:

$$\frac{2\pi s}{\sqrt{m + \frac{1}{2}}}$$

The default smallest scale for the DOG wavelet is `2*dt` where `dt` is the sampling period.

The default spacing between scales for the DOG wavelet is `ds=0.4875`.

The default number of scales for the DOG wavelet is `max([fix(log2(length(sig))/ds),1])`.

**Paul Wavelet**

The Fourier transform of the analytic Paul wavelet, `'paul'`, of order *m* is:

$$\hat{\Psi}(s\omega) = 2^m \sqrt{m(2m-1)!}\,(s\omega)^m\, e^{-s\omega} U(sw)$$

where *U(ω)* is the Heaviside step function [5].

The default order of the Paul wavelet is 4.

The scale-to-frequency Fourier factor for the Paul wavelet is:

$$\frac{4\pi s}{2m+1}$$

The default smallest scale for the Paul wavelet is `2*dt` where `dt` is the sampling period.

The default spacing between scales for the Paul wavelet is `ds=0.4875`.

The default number of scales for the Paul wavelet is `fix(log2(length(sig))/ds)+1`.

**Examples**

Compute and display the CWT of sine waves with disjoint support. The sampling interval is 1/1023.

```
N = 1024;
% Sampling interval is 1/1023
t = linspace(0,1,N);
y = sin(2*pi*4*t).*(t<=0.5)+sin(2*pi*8*t).*(t>0.5);
% Because the sampling interval differs from the default
% you must input it along with the signal
% Using cell array input
```

```
sig = {y,1/1023};
cwtS1 = cwtft(sig,'plot');
```



You can display or hide the reconstructed signal using the radio button at the bottom left of the figure. When you select the radio button, the maximum and quadratic relative errors are computed and displayed along with the reconstructed signal.

Reconstruct an approximation to a sum of disjoint sine waves in noise using cwtft to decompose the signal and icwtft to reconstruct the approximation. Use the CWT coefficients to identify the scales isolating the sinusoidal components. Reconstruct an approximation to the signal based on those scales using the inverse CWT. To ensure an accurate approximation to the based on select scales, use the default power of two spacing in the CWT.

```
rng default % Reset random number generator for reproducible results
N = 1024;
```

```
% Sampling interval is 1/1023
t = linspace(0,1,N);
y = sin(2*pi*4*t).*(t<=0.5)+sin(2*pi*8*t).*(t>0.5);
ynoise = y+randn(size(t));
% Because the sampling interval differs from the default
% you must input it along with the signal
% Using structure array input
sig = struct('val',ynoise,'period',1/1023);
cwtS1 = cwtft(sig);
scales = cwtS1.scales;
MorletFourierFactor = 4*pi/(6+sqrt(2+6^2));
freq = 1./(scales.*MorletFourierFactor);
contour(t,freq,real(cwtS1.cfs));
xlabel('Seconds'); ylabel('Pseudo-frequency');
axis([0 t(end) 0 15]);
```

Extract the scales dominated by energy from the two sine waves and reconstruct a signal approximation using the inverse CWT.

```
cwtS2 = cwtS1;
cwtS2.cfs = zeros(size(cwtS1.cfs));
cwtS2.cfs(13:15,:) = cwtS1.cfs(13:15,:);
xrec = icwtft(cwtS2);
subplot(2,1,1);
plot(t,ynoise);
title('Sum of Disjoint Sinusoids in Noise');
subplot(2,1,2);
plot(t,xrec,'b'); hold on; axis([0 1 -4 4]);
plot(t,y,'r');
legend('Reconstructed Signal','Original Signal',...
```

```
'Location','NorthWest');
xlabel('Seconds'); ylabel('Amplitude');
```



**Algorithms**   cwtft implements the following algorithm:

- Obtain the discrete Fourier transform (DFT) of the signal using fft.

- Obtain the DFT of the analyzing wavelet at the appropriate angular frequencies. Scale the DFT of the analyzing wavelet at different scales to ensure different scales are directly comparable.

- Take the product of the signal DFT and the wavelet DFT over all the scales. Invert the DFT to obtain the CWT coefficients.

For a mathematical motivation for the FFT-based algorithm see "DFT-Based Continuous Wavelet Transform".

**References**

[1] Daubechies, I. *Ten Lectures on Wavelets*, Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 1992.

[2] Farge, M. "Wavelet Transforms and Their Application to Turbulence", *Ann. Rev. Fluid. Mech.*, 1992, 24, 395–457.

[3] Mallat, S. *A Wavelet Tour of Signal Processing*, San Diego, CA: Academic Press, 1998.

[4] Sun,W. "Convergence of Morlet's Reconstruction Formula", *preprint*, 2010.

[5] Torrence, C. and G.P. Compo. "A Practical Guide to Wavelet Analysis", *Bull. Am. Meteorol. Soc.*, 79, 61–78, 1998.

**Alternatives**

- cwt — Computes the CWT using convolutions. cwt supports a wider choice of analyzing wavelets than cwtft, but may be more computationally expensive. The output of cwt is not compatible with the inverse CWT implemented with icwtft. To use icwtft, obtain the CWT with cwtft.

**See Also**

cwt | icwtft

**How To**

- "Continuous Wavelet Transform"
- "DFT-Based Continuous Wavelet Transform"
- "Inverse Continuous Wavelet Transform"

# cwtftinfo

| | |
|---|---|
| **Purpose** | Valid analyzing wavelets for FFT-based CWT |
| **Syntax** | `cwtftinfo` |
| **Description** | `cwtftinfo` displays expressions for the Fourier transforms of valid analyzing wavelets for use with `cwtft`. |

**Definitions**

### Wavelet Definitions

#### Morlet Wavelet

Both non-analytic and analytic Morlet wavelets are supported. The analytic Morlet wavelet, `'morl'`, is defined in the Fourier domain by:

$$\hat{\Psi}(s\omega) = \pi^{-1/4} e^{(s\omega-\omega_0)^2/2} U(s\omega)$$

where $U(\omega)$ is the Heaviside step function.

The non-analytic Morlet wavelet, `'morlex'`, is defined in the Fourier domain by:

$$\hat{\Psi}(s\omega) = \pi^{-1/4} e^{(s\omega-\omega_0)^2/2}$$

`'morl0'` defines a non-analytic Morlet wavelet in the Fourier domain with exact zero mean:

$$\hat{\Psi}(s\omega) = \pi^{-1/4} \{ e^{(s\omega-\omega_0)^2/2} - e^{\omega_0^2/2} \}$$

The default value of $\omega_0$ is 6.

The scale-to-frequency Fourier factor for the Morlet wavelet is:

$$\frac{4\pi s}{\omega_0 + \sqrt{2 + \omega_0^2}}$$

#### *m*-th Order Derivative of Gaussian Wavelets

In the Fourier domain, the *m*-th order derivative of Gaussian wavelets, `'dog'`, is defined by:

$$\hat{\Psi}(s\omega) = -\frac{1}{\sqrt{\Gamma(m+1/2)}}(js\omega)^m e^{-(s\omega)^2/2}$$

The derivative must be an even order. The default order of the derivative is 2, which is also known as the *Mexican hat* wavelet.

Because the unit imaginary, *j*, is always raised to an even power, the Fourier transform is real-valued.

The scale-to-frequency Fourier factor for the DOG wavelet is:

$$\frac{2\pi s}{\sqrt{m+\frac{1}{2}}}$$

**Paul Wavelet**

The Fourier transform of the Paul wavelet, `'paul'`, of order *m* is:

$$\hat{\Psi}(s\omega) = 2^m \sqrt{m(2m-1)!}(s\omega)^m e^{-s\omega}U(sw)$$

where *U(ω)* is the Heaviside step function. The Paul wavelet is analytic.

The scale-to-frequency Fourier factor for the Paul wavelet is:

$$\frac{4\pi s}{2m+1}$$

The default order of the Paul wavelet is 4.

**Examples**    Display a list of Fourier transforms for all valid analyzing wavelets.

```
cwtftinfo
```

**References**    [1] Daubechies, I. *Ten Lectures on Wavelets*, Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 1992.

[2] Farge, M. *Wavelet Transforms and Their Application to Turbulence*, Ann. Rev. Fluid. Mech., 1992, 24, 395–457.

[3] Mallat, S. *A Wavelet Tour of Signal Processing*, San Diego, CA: Academic Press, 1998.

[4] Torrence, C. and G.P. Compo *A Practical Guide to Wavelet Analysis*, Bull. Am. Meteorol. Soc., 79, 61–78, 1998.

**See Also**      `cwtft | icwtft`

**How To**
- "Continuous Wavelet Transform"
- "DFT-Based Continuous Wavelet Transform"
- "Inverse Continuous Wavelet Transform"

**Purpose**     Supported 2-D CWT wavelets and Fourier transforms

**Syntax**      cwtftinfo2
                cwtftinfo2(wname)

**Description**  cwtftinfo2 lists the supported 2-D continuous wavelet transform
                (CWT) wavelets and corresponding parameters for use with cwtft2.

                cwtftinfo2(wname) displays the equation for the 2-D Fourier transform
                of the wavelet, wname. The figure with the 2-D Fourier transform of
                the analyzing wavelet has a drop-down list from which you can select
                other wavelets.

**Input Arguments**

**wname - Wavelet name**
string

Wavelet name, specified as a string. The following table lists the
supported wavelets for the 2-D CWT and associated parameters:

| Wavelet name | Parameters |
| --- | --- |
| 'morl' | {'Omega0',6;'Sigma',1;'Epsilon',1} |
| 'mexh' | {'p',2;'sigmax',1;'sigmay',1} |
| 'paul' | {'p',4} |
| 'dog' | {'alpha',1.25} |
| 'cauchy' | {'alpha','pi/6';'L',4;'M',4;'sigma',1} |
| 'escauchy' | {'alpha','pi/6';'L',4;'M',4;'sigma',1} |
| 'gaus' | {'p',1;'sigmax',1;'sigmay',1} |
| 'wheel' | {'sigma',2} |
| 'fan' | {'Omega0X',5.336;'Sigma',1;'Epsilon',1; |
| 'pethat' | None |
| 'dogpow' | {'alpha',1.25;'p',2} |

| Wavelet name | Parameters |
|---|---|
| `'esmorl'` | `{'Omega0',6;'Sigma',1;'Epsilon',1}` |
| `'esmexh'` | `{'Sigma',1;'Epsilon',0.5}` |
| `'gaus2'` | `{'p',1;'sigmax',1;'sigmay',1}` |
| `'gaus3'` | `{'A',1;'B',1;'p',1;'sigmax',1;'sigmay',1}` |
| `'isodog'` | `{'alpha',1.25,1.25}` |
| `'dog2'` | `{'alpha',1.25,1.25}` |
| `'isomorl'` | `{'Omega0',6;'Sigma',1}` |
| `'rmorl'` | `{'Omega0',6;'Sigma',1;'Epsilon',1}` |
| `'endstop1'` | `{'Omega0',6}` |
| `'endstop2'` | `{'Omega0',6;'Sigma',1}` |
| `'gabmexh'` | `{'Omega0',5.336;'Epsilon',1}` |
| `'sinc'` | `{'Ax',1;'Ay',1;'p',1;'Omega0X',0;'Omega0Y'` |

**Example:** cwtftinfo2('paul')

**Data Types**
char

**Examples**  **Available Wavelets with Parameters**

cwtftinfo2

**Display the Expression for the 2-D Fourier Transform**

Display the expression for the 2-D Fourier transform of the Cauchy wavelet.

cwtftinfo2('cauchy')

## cauchy

$$\widehat{\psi}(\omega_x, \omega_y) = \left[\sin(\alpha)\omega_x + \cos(\alpha)\omega_y\right]^L \ldots$$

$$\left[-\sin(\alpha)\omega_x + \cos(\alpha)\omega_y\right]^M \left[\tan(\alpha)\omega_x > |\omega_y|\right] e^{-\sigma \frac{|(\omega_y)}{}}$$

$$\sigma \in \left]0, +\infty\right[, \; \alpha \in \left]0, \frac{\pi}{2}\right[, \; L, M \left]0, +\infty\right[$$

cauchy ▼

After displaying the Fourier transform for any wavelet, you can use the drop-down list in the bottom left to view the Fourier transform for any supported wavelet.

**See Also**    cwtft2

# cwtft2

| | |
|---|---|
| **Purpose** | 2-D continuous wavelet transform |

**Syntax**
```
cwtstruct = cwtft2(x)
cwtstruct = cwtft2(x,'plot')
cwtstruct = cwtft2(x,Name,Value)
```

**Description**   cwtstruct = cwtft2(x) returns the 2-D continuous wavelet transform (CWT) of the 2-D matrix, x. cwtft2 uses a Fourier transform-based algorithm in which the 2-D Fourier transforms of the input data and analyzing wavelet are multiplied together and inverted.

cwtstruct = cwtft2(x,'plot') plots the data and the 2-D CWT.

cwtstruct = cwtft2(x,Name,Value) uses additional options specified by one or more Name,Value pair arguments.

**Input Arguments**

**x - Input data**
array

Input data, specified as a 2-D matrix or 3-D array. If the input data is a 3-D array, the input matrix is a truecolor image.

**Example:** X = imread('stars.jpg');

**Data Types**
double | uint8

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

**Example:** 'wavelet','paul','scales',2^(0:5) specifies to use the Paul wavelet and a vector of scales.

**'angles' - Angles**
0 (default) | scalar | vector

Angles in radians, specified as a comma-separated pair consisting of 'angles' and either a scalar or a vector.

**Example:** 'angles',[0 pi/2 pi]

**'norm' - Normalization**
'L2' (default) | 'L1' | 'L0'

Normalization used in the 2-D CWT, specified as a comma-separated pair consisting of 'norm' and one of these strings:

- 'L2' — The Fourier transform of the analyzing wavelet at a given scale is multiplied by the corresponding scale. 'L2' is the default normalization.

- 'L1' — The Fourier transform of the analyzing wavelet is multiplied by 1 at all scales.

- 'L0' — The Fourier transform of the analyzing wavelet at a given scale is multiplied by the square of the corresponding scale.

**Example:** 'norm','L1'

**'scales' - Scales**
2^(0:5) (default) | scalar | vector

Scales, specified as a comma-separated pair consisting of 'scales' and either a positive real-valued scalar or a vector of positive real numbers.

**Example:** 'scales',2^(1:6)

**'wavelet' - Analyzing wavelet**
'morl' (default) | string | structure | cell array

Analyzing wavelet, specified as a comma-separated pair consisting of 'wavelet' and a string, a structure, or a cell array. cwtftinfo2 provides a comprehensive list of supported wavelets and associated parameters.

If you specify `'wavelet'` as a structure, the structure must contain two fields:

- `name` — the string corresponding to a supported wavelet.

- `param` — a cell array with the parameters of the wavelet.

If you specify `'wavelet'` as a cell array, `wav`, the cell array must contain two elements:

- `wav{1}` — the string corresponding to a supported wavelet.

- `wav{2}` — a cell array with the parameters of the wavelet.

**Example:** `'wavelet',{'morl',{6,1,1}}`

**Example:** `'wavelet',struct('name','paul','param',{'p',2})`

**Output Arguments**

**cwtstruct - 2-D CWT**
structure

The 2-D CWT, returned as a structure with the following fields:

**wav - Analyzing wavelet and parameters**
structure

Analyzing wavelet and parameters, returned as a structure with the following fields:

- `wname` — name

- `param` — parameters

**wav_norm - Normalization constants**
matrix

Normalization constants, returned as a *M*-by-*N* matrix where *M* is the number of scales and *N* is the number of angles.

**cfs - CWT coefficients**
array

CWT coefficients, returned as an N-D array. The row and column dimensions of the array equal the row and column dimensions of the input data. The third page of the array is equal to 1 or 3 depending on whether the input data is a grayscale or truecolor image. The fourth page of the array is equal to the number of scales and the fifth page of the array is equal to the number of angles.

### scales - Scales
vector

Scales for the 2-D CWT, returned as a row vector.

### angles - Angles
vector

Angles for the 2-D CWT, returned as a row vector.

### meanSIG - Mean
scalar

Mean of the input data, returned as a scalar

**Examples**     **Compare Isotropic and Anisotropic Wavelets**

This example shows how an isotropic wavelet does not discern the orientation of features while an anisotropic wavelet does. The example uses the Mexican hat isotropic wavelet and the directional (anisotropic) Cauchy wavelet.

Load and view the hexagon image.

```
Im = imread('hexagon.jpg');
imagesc(Im); colormap(jet);
```

Obtain the scale-one 2-D CWT with both the Mexican hat and Cauchy wavelets. Specify a vector of angles going from 0 to 15π/8 in π/8 increments.

```
cwtcauchy = cwtft2(Im,'wavelet','cauchy','scales',1,...
    'angles',0:pi/8:2*pi-pi/8);
```

```
cwtmexh = cwtft2(Im,'wavelet','mexh','scales',1,...
    'angles',0:pi/8:2*pi-pi/8);
```

Visualize the scale-one 2-D CWT coefficient magnitudes at each angle.

```
angz = {'0', 'pi/8', 'pi/4', '3pi/8', 'pi/2', '5pi/8', '3pi/4', ...
    '7pi/8','pi', '9pi/8', '5pi/4', '11pi/8', '3pi/2', ...
    '13pi/8' '7pi/4', '15pi/8'};
for angn = 1:length(angz)
    subplot(211)
    imagesc(abs(cwtmexh.cfs(:,:,1,1,angn)));
    title(['Mexican hat at ' angz(angn) 'radians']);
    subplot(212)
    imagesc(abs(cwtcauchy.cfs(:,:,1,1,angn)));
    title(['Cauchy wavelet at ' angz(angn) 'radians']);
    pause(1);
end
```

### Plot 2-D CWT

Load an image of a woman, obtain the 2-D CWT using the Morlet wavelet, and plot the CWT coefficients.

```
load woman;
cwtmorl = cwtft2(X,'scales',1:4,'angles',0:pi/2:3*pi/2,'plot');
```

### 2-D CWT with Morlet Wavelet

Obtain the 2-D CWT of the star image using the default Morlet wavelet, scales `2^(0:5)`, and an angle of 0.

```
Im = imread('star.jpg');
cwtout = cwtft2(Im);
```

**See Also**     cwtftinfo2

**Related Examples**
- "Two-Dimensional CWT of Noisy Pattern"
- "2-D Continuous Wavelet Transform App"

**Purpose**         Real or complex continuous 1-D wavelet coefficients using extension
                    parameters

**Syntax**          COEFS = cwtext(S,SCALES,'*wname*')
                    COEFS = cwtext(S,SCALES,'*wname*',PropName1,ProVal1, ...)
                    EXTMODE = struct('Mode',ModeVAL,'Side',SideVAL,'Len',LenVAL);
                    EXTMODE = {ModeVAL,SideVAL,LenVAL};
                    COEFS = cwtext(...,'PlotMode',PLOTMODE)

**Description**     COEFS = cwtext(S,SCALES,'*wname*') computes the continuous wavelet
                    coefficients of the vector S at real, positive SCALES, using a wavelet
                    named '*wname*'. The signal S is real; the wavelet can be real or complex.

                    COEFS = cwtext(S,SCALES,'*wname*',PropName1,ProVal1, ...)
                    computes and plots the continuous wavelet transform coefficients using
                    extra parameters. Valid values for PropName are:

- 'ExtMode'

- 'ExtSide'

- 'ExtLen'

- 'PlotMode'

- 'xlim'

                    The continuous wavelet transform coefficients are computed using the
                    extension parameters:

- 'ExtMode'

- 'ExtSide'

- 'ExtLen'

                    Valid values for *ExtMode* are:

- 'zpd' (zero padding)

- 'sp0' (smooth extension of order 0)

- 'sp1' (smooth extension of order 1)

etc.

Valid values for *ExtSide* are:

- ExtSide = 'l' (or 'u') for left (or up) extension

- ExtSide = 'r' (or 'd') for right (or down) extension

- ExtSide = 'b' for extension on both sides

- ExtSide = 'n' null extension

For the complete list of valid values for ExtMode and ExtSide, see wextend.

ExtLen is the length of extension.

Default values for extension parameters are 'zpd' and 'b'. ExtLen is computed using the maximum of SCALES.

Instead of three parameters, use the following syntaxes:

```
EXTMODE =
struct('Mode',ModeVAL,'Side',SideVAL,'Len',LenVAL);
```

```
EXTMODE = {ModeVAL,SideVAL,LenVAL};
```

COEFS = cwtext(...,'PlotMode',PLOTMODE) computes and plots the continuous wavelet transform coefficients. Coefficients are colored using PLOTMODE:

- PLOTMODE = 'lvl' (By scale)

- PLOTMODE = 'glb' (All scales)

- PLOTMODE = 'abslvl' or 'lvlabs' (Absolute value and By scale)

- PLOTMODE = 'absglb' or 'glbabs' (Absolute value and All scales)

You get 3-D plots (surfaces) using the same keywords listed above for the PLOTMODE parameter, preceded by '3D', for example, PLOTMODE = '3Dlvl'.

When `PLOTMODE` = `'scal'` or `'scalCNT'` the continuous wavelet transform coefficients and the corresponding scalogram (percentage of energy for each coefficient) are computed.

When `PLOTMODE` is `'scal'`, a scaled image of scalogram is displayed. When `PLOTMODE` is `'scalCNT'`, a contour representation of scalogram is displayed.

If the `XLIM` parameter is given, the continuous wavelet transform coefficients are colored using `PLOTMODE` and `XLIM`.

`XLIM = [x1 x2]` with `1 <= x1 < x2 <= length(S)`.

For each given scale `a` within the vector `SCALES`, the wavelet coefficients `C(a,b)` are computed for `b = 1` to `ls = length(S)`, and are stored in `COEFS(i,:)` if `a = SCALES(i)`.

Output argument `COEFS` is a `la`-by-`ls` matrix where `la` is the length of `SCALES`. `COEFS` is a real or complex matrix depending on the wavelet type.

Examples of valid use are as follows:

```
t = linspace(-1,1,512);
s = 1-abs(t);
c = cwtext(s,1:32,'cgau4');
c = cwtext(s,[64 32 16:-2:2],'morl');
c = cwtext(s,[3 18 12.9 7 1.5],'db2');
c = cwtext(s,1:32,'sym2','plotMode','lvl');
c = cwtext(s,1:64,'sym4','plotMode','abslvl','xlim',[100 400]);

[c,Sc] = cwtext(s,1:64,'sym4','plotMode','scal');
[c,Sc] = cwtext(s,1:64,'sym4','plotMode','scalCNT');
[c,Sc] = cwtext(s,1:64,'sym4','plotMode','scalCNT', ...
                          'extMode','sp1');

c = cwtext(s,1:64,'sym4','plotMode','lvl','extMode','sp0');
c = cwtext(s,1:64,'sym4','plotMode','lvl','extMode','sp1');
c = cwtext(s,1:64,'sym4','plotMode','lvl', ...
                        'extMode',{'sp1','b',300});
```

```
ext = struct('Mode','sp1','Side','b','Len',300);
c = cwtext(s,1:64,'sym4','plotMode','lvl','extMode',ext);
```

**Examples**  This example demonstrates the difference between a continuous wavelet transform which deals with signal extension and one which does not.

```
% Load and plot the signal
load wcantor
plot(wcantor)
```



```
% Compute and plot the coefficients
cwt(wcantor,(1:256),'mexh','absglb');
colormap(pink(4))
```

Absolute Values of Ca,b Coefficients for a =  1 2 3 4 5 ...



In this figure above, which is produced by the `cwt` function, the values of coefficients are tremendously affected by the boundary effect due to the discontinuity of the signal on the right. The default (zero-padding) extension mode on the right explains this important discontinuity because the last value is 1. On the left there is no effect because the first value is 0.

```
% Compute and plot the coefficients with adapted extension mode
figure;
cwtext(wcantor,(1:256),'mexh','extmode','sp0','extLen',2000, ...
                                'plotMode','absglb');
colormap(pink(4))
```

Absolute Values of Ca,b Coefficients for a = 1 2 3 4 5 ...

In this figure, produced by the cwtext function, the suitable extension
mode of the signal is very efficient, giving as it can be seen, a good result.

**See Also**    cwt | wavedec | wavefun | waveinfo | wcodemat

**Purpose**          Daubechies wavelet filter computation

**Syntax**           W = dbaux(N,SUMW)
                     W = dbaux(N)
                     W = dbaux(N,0)

**Description**      W = dbaux(N,SUMW) is the order *N* Daubechies scaling filter such that
                    sum(W) = SUMW. Possible values for *N* are 1, 2, 3, ...

---

**Note** Instability may occur when *N* is too large.

---

W = dbaux(N) is equivalent to W = dbaux(N,1)

W = dbaux(N,0) is equivalent to W = dbaux(N,1)

**Daubechies'**     This example shows to determine the Daubechies' extremal phase
**Extremal**        scaling filter with a specified sum. The two most common values for the
**Phase**           sum are $\sqrt{2}$ and 1.
**Scaling**
**Filter with**     Construct two versions of the 'db4' scaling filter. One scaling filter
**Specified**       sums to $\sqrt{2}$ and the other version sums to 1.
**Sum**
                    NumVanishingMoments = 4;
                    h = dbaux(NumVanishingMoments,sqrt(2));
                    mO = dbaux(NumVanishingMoments,1);

                    The filter with sum equal to $\sqrt{2}$ is the synthesis (reconstruction) filter
                    returned by wfilters and used in the discrete wavelet transform.

                    [LoD,HiD,LoR,HiR] = wfilters('db4');
                    max(abs(LoR-h))


                    ans =

```
   4.2613e-13
```

For orthogonal wavelets, the analysis (decomposition) filter is the time-reverse of the synthesis filter.

```
max(abs(LoD-fliplr(h)))
```

```
ans =

   4.2613e-13
```

**Algorithms**     The algorithm used is based on a result obtained by Shensa (see "References"), showing a correspondence between the "Lagrange à trous" filters and the convolutional squares of the Daubechies wavelet filters.

The computation of the order $N$ Daubechies scaling filter $w$ proceeds in two steps: compute a "Lagrange à trous" filter $P$, and extract a square root. More precisely:

- P the associated "Lagrange à trous" filter is a symmetric filter of length 4N-1. P is defined by

  $P = [a(N)\ 0\ a(N\text{-}1)\ 0\ ...\ 0\ a(1)\ 1\ a(1)\ 0\ a(2)\ 0\ ...\ 0\ a(N)]$

- where

$$a(k) = \frac{\displaystyle\prod_{\substack{i=-N+1 \\ i \neq k}}^{N}\left(\frac{1}{2}-i\right)}{\displaystyle\prod_{\substack{i=-N+1 \\ i \neq k}}^{N}(k-i)} \quad \text{for } k = 1,...,N$$

- Then, if $w$ denotes db$N$ Daubechies scaling filter of sum $\sqrt{2}$, $w$ is a square root of $P$:

  $P = $ conv(wrev($w$),$w$) where $w$ is a filter of length 2$N$.

  The corresponding polynomial has $N$ zeros located at $-1$ and $N-1$ zeros less than 1 in modulus.

Note that other methods can be used; see various solutions of the spectral factorization problem in Strang-Nguyen (p. 157).

**Limitations** The computation of the db$N$ Daubechies scaling filter requires the extraction of the roots of a polynomial of order 4$N$. Instability may occur when $N$ is too large.

**References** Daubechies, I. (1992), *Ten lectures on wavelets*, CBMS-NSF conference series in applied mathematics, SIAM Ed.

Shensa, M.J. (1992), "The discrete wavelet transform: wedding the a trous and Mallat Algorithms," *IEEE Trans. on Signal Processing*, vol. 40, 10, pp. 2464-2482.

Strang, G.; T. Nguyen (1996), *Wavelets and Filter Banks*, Wellesley-Cambridge Press.

**See Also** dbwavf | wfilters

# dbwavf

**Purpose**    Daubechies wavelet filter

**Syntax**    F = dbwavf(W)

**Description**    `F = dbwavf(W)` returns the scaling filter associated with Daubechies wavelet specified by the string `W` where `W = 'dbN'`. Possible values for N are 1, 2, 3, ..., 45.

**Examples**
```
% Set Daubechies wavelet name.
wname = 'db4';

% Compute the corresponding scaling filter.
f = dbwavf(wname)

f =
Columns 1 through 7
0.1629  0.5055  0.4461 -0.0198 -0.1323  0.0218  0.0233
Column 8
-0.0075
```

**See Also**    dbaux | waveinfo | wfilters

| | |
|---|---|
| **Purpose** | Default values for denoising or compression |
| **Syntax** | [THR,SORH,KEEPAPP,CRIT] = ddencmp(IN1,IN2,X)<br>[THR,SORH,KEEPAPP] = ddencmp(IN1,'wv',X)<br>[THR,SORH,KEEPAPP,CRIT] = ddencmp(IN1,'wp',X) |
| **Description** | ddencmp returns default values for denoising or compression for the critically-sampled discrete wavelet or wavelet packet transform.<br><br>You can use ddencmp for 1-D signals or 2-D images.<br><br>[THR,SORH,KEEPAPP,CRIT] = ddencmp(IN1,IN2,X) returns default values for denoising or compression, using wavelets or wavelet packets, of an input vector or matrix X, which can be a one- or two-dimensional signal. THR is the threshold, SORH is for soft or hard thresholding, KEEPAPP allows you to keep approximation coefficients, and CRIT (used only for wavelet packets) is the entropy name (see wentropy for more information).<br><br>IN1 is 'den' for denoising or 'cmp' for compression.<br><br>IN2 is 'wv' for wavelet or 'wp' for wavelet packet.<br><br>For wavelets (three output arguments):<br><br>[THR,SORH,KEEPAPP] = ddencmp(IN1,'wv',X) returns default values for denoising (if IN1 = 'den') or compression (if IN1 = 'cmp') of X. These values can be used for wdencmp.<br><br>For wavelet packets (four output arguments):<br><br>[THR,SORH,KEEPAPP,CRIT] = ddencmp(IN1,'wp',X) returns default values for denoising (if IN1 = 'den') or compression (if IN1 = 'cmp') of X. These values can be used for wdencmp. |
| **Examples** | **Default Global Threshold for Wavelet Denoising**<br><br>Determine the default global denoising threshold for an N(0,1) white noise input. |

# ddencmp

Create an N(0,1) white noise input. Set the random number generator to the default initial settings for reproducible results.

```
dwtmode('per');
rng default;
x = randn(512,1);
```

Use ddencmp to obtain the default global threshold for wavelet denoising. Demonstrate that the threshold is equal to the universal threshold of Donoho and Johnstone scaled by a robust estimate of the variance.

```
[thr,sorh,keepapp] = ddencmp('den','wv',x);
[A,D] = dwt(x,'db1');
noiselev = median(abs(D))/0.6745;
thresh = sqrt(2*log(length(x)))*noiselev;
```

Compare the value of the variable thr to the value of thresh.

### Default Global Threshold for Wavelet Packet Compression

Determine the default global compression threshold for an N(0,1) white noise input.

Create an N(0,1) white noise input. Set the random number generator to the default initial settings for reproducible results.

```
dwtmode('per');
rng default;
x = randn(512,1);
```

Use ddencmp with the 'cmp' and 'wp' input arguments to return the default global compression threshold for a wavelet packet transform.

```
[thr,sorh,keepapp,crit] = ddencmp('den','wp',x) ;
```

**References** Donoho, D.L. (1995), "De-noising by soft-thresholding," *IEEE*, *Trans. on Inf. Theory*, 41, 3, pp. 613–627.

Donoho, D.L.; I.M. Johnstone (1994), "Ideal spatial adaptation by wavelet shrinkage," *Biometrika*, vol 81, pp. 425–455.

Donoho, D.L.; I.M. Johnstone (1994), "Ideal de-noising in an orthonormal basis chosen from a library of bases," *C.R.A.S. Paris*, *Ser. I*, t. 319, pp. 1317–1322.

**See Also**    wdencmp | wenergy | wpdencmp

# dddtree

| | |
|---|---|
| **Purpose** | Dual-tree and double-density 1-D wavelet transform |
| **Syntax** | `wt = dddtree(typetree,x,level,fdf,df)` |
| | `wt = dddtree(typetree,x,level,fname)` |
| | `wt = dddtree(typetree,x,level,fname1,fname2)` |

**Description**

`wt = dddtree(typetree,x,level,fdf,df)` returns the `typetree` discrete wavelet transform (DWT) of the 1-D input signal, `x`, down to level, `level`. The wavelet transform uses the decomposition (analysis) filters, `fdf`, for the first level and the analysis filters, `df`, for subsequent levels. Supported wavelet transforms are the critically sampled DWT, double-density, dual-tree complex, and dual-tree double-density complex wavelet transform. The critically sampled DWT is a filter bank decomposition in an orthogonal or biorthogonal basis (nonredundant). The other wavelet transforms are oversampled filter banks.

`wt = dddtree(typetree,x,level,fname)` uses the filters specified by `fname` to obtain the wavelet transform. Valid filter specifications depend on the type of wavelet transform. See `dtfilters` for details.

`wt = dddtree(typetree,x,level,fname1,fname2)` uses the filters specified in `fname1` for the first stage of the dual-tree wavelet transform and the filters specified in `fname2` for subsequent stages of the dual-tree wavelet transform. Specifying different filters for stage 1 is valid and necessary only when `typetree` is `'cplxdt'` or `'cplxdddt'`.

**Input Arguments**

**typetree - Type of wavelet decomposition**
`'dwt'` | `'ddt'` | `'cplxdt'` | `'cplxdddt'`

Type of wavelet decomposition, specified as one of `'dwt'`, `'ddt'`, `'cplxdt'`, or `'cplxdddt'`. The type, `'dwt'`, gives a critically sampled (nonredundant) discrete wavelet transform. The other decomposition types produce oversampled wavelet transforms. `'ddt'` produces a double-density wavelet transform. `'cplxdt'` produces a dual-tree complex wavelet transform. `'cplxdddt'` produces a double-density dual-tree complex wavelet transform.

### x - Input signal
vector

Input signal, specified as an even-length row or column vector. If *L* is the value of the `level` of the wavelet decomposition, $2^L$ must divide the length of x. Additionally, the length of the signal must be greater than or equal to the product of the maximum length of the decomposition (analysis) filters and $2^{(L-1)}$.

**Data Types**
double

### level - Level of wavelet decomposition
positive integer

Level of the wavelet decomposition, specified as an integer. If *L* is the value of `level`, $2^L$ must divide the length of x . Additionally, the length of the signal must be greater than or equal to the product of the maximum length of the decomposition (analysis) filters and $2^{(L-1)}$.

**Data Types**
double

### fdf - Level-one analysis filters
matrix | cell array

The level-one analysis filters, specified as a matrix or cell array of matrices. Specify `fdf` as a matrix when `typetree` is `'dwt'` or `'ddt'`. The size and structure of the matrix depend on the `typetree` input as follows:

- `'dwt'` — This is the critically sampled discrete wavelet transform. In this case, `fdf` is a two-column matrix with the lowpass (scaling) filter in the first column and the highpass (wavelet) filter in the second column.

- `'ddt'` — This is the double-density wavelet transform. The double-density DWT is a three-channel perfect reconstruction filter bank. `fdf` is a three-column matrix with the lowpass (scaling) filter in the first column and the two highpass (wavelet) filters in the

second and third columns. In the double-density wavelet transform, the single lowpass and two highpass filters constitute a three-channel perfect reconstruction filter bank. This is equivalent to the three filters forming a tight frame. You cannot arbitrarily choose the two wavelet filters in the double-density DWT. The three filters together must form a tight frame.

Specify `fdf` as a 1-by-2 cell array of matrices when `typetree` is a dual-tree transform, `'cplxdt'` or `'cplxdddt'`. The size and structure of the matrix elements depend on the `typetree` input as follows:

- For the dual-tree complex wavelet transform, `'cplxdt'`, `fdf{1}` is a two-column matrix containing the lowpass (scaling) filter and highpass (wavelet) filters for the first tree. The scaling filter is the first column and the wavelet filter is the second column. `fdf{2}` is a two-column matrix containing the lowpass (scaling) and highpass (wavelet) filters for the second tree. The scaling filter is the first column and the wavelet filter is the second column.

- For the double-density dual-tree complex wavelet transform, `'cplxdddt'`, `fdf{1}` is a three-column matrix containing the lowpass (scaling) and two highpass (wavelet) filters for the first tree and `fdf{2}` is a three-column matrix containing the lowpass (scaling) and two highpass (wavelet) filters for the second tree.

**Data Types**
double

### df - Analysis filters for levels > 1
matrix | cell array

Analysis filters for levels > 1, specified as a matrix or cell array of matrices. Specify `df` as a matrix when `typetree` is `'dwt'` or `'ddt'`. The size and structure of the matrix depend on the `typetree` input as follows:

- `'dwt'` — This is the critically sampled discrete wavelet transform. In this case, `df` is a two-column matrix with the lowpass (scaling) filter in the first column and the highpass (wavelet) filter in the second

column. For the critically sampled orthogonal or biorthogonal DWT, the filters in `df` and `fdf` must be identical.

- `'ddt'` — This is the double-density wavelet transform. The double-density DWT is a three-channel perfect reconstruction filter bank. `df` is a three-column matrix with the lowpass (scaling) filter in the first column and the two highpass (wavelet) filters in the second and third columns. In the double-density wavelet transform, the single lowpass and two highpass filters must constitute a three-channel perfect reconstruction filter bank. This is equivalent to the three filters forming a tight frame. For the double-density DWT, the filters in `df` and `fdf` must be identical.

Specify `df` as a 1-by-2 cell array of matrices when `typetree` is a dual-tree transform, `'cplxdt'` or `'cplxdddt'`. For dual-tree transforms, the filters in `fdf` and `df` must be different. The size and structure of the matrix elements in the cell array depend on the `typetree` input as follows:

- For the dual-tree complex wavelet transform, `'cplxdt'`, `df{1}` is a two-column matrix containing the lowpass (scaling) and highpass (wavelet) filters for the first tree. The scaling filter is the first column and the wavelet filter is the second column. `df{2}` is a two-column matrix containing the lowpass (scaling) and highpass (wavelet) filters for the second tree. The scaling filter is the first column and the wavelet filter is the second column.

- For the double-density dual-tree complex wavelet transform, `'cplxdddt'`, `df{1}` is a three-column matrix containing the lowpass (scaling) and two highpass (wavelet) filters for the first tree and `df{2}` is a three-column matrix containing the lowpass (scaling) and two highpass (wavelet) filters for the second tree.

**Data Types**
double

**fname - Filter name**
string

Filter name, specified as a string. For the critically sampled DWT, specify any valid orthogonal or biorthogonal wavelet filter. See `wfilters` for details. For the double-density wavelet transform, `'ddt'`, valid choices are `'filters1'` and `'filters2'`. For the complex dual-tree wavelet transform, valid choices are `'dtfP'` with P = 1, 2, 3, 4. For the double-density dual-tree wavelet transform, the only valid choice is `'dddtf1'`. See `dtfilters` for more details on valid filter strings for the oversampled wavelet filter banks.

**Data Types**
char

**fname1 - First-stage filter name**
string

First-stage filter name, specified as a string. Specifying a different filter for the first stage is valid and necessary only in the dual-tree transforms, `'cplxdt'` and `'cplxddt'`. In the complex dual-tree wavelet transform, you can use any valid wavelet filter for the first stage. In the double-density dual-tree wavelet transform, the first-stage filters must form a three-channel perfect reconstruction filter bank.

**Data Types**
char

**fname2 - Filter name for stages > 1**
string

Filter name for stages > 1, specified as a string. You must specify a first-level filter that is different from the wavelet and scaling filters in subsequent levels when using the dual-tree wavelet transforms, `'cplxdt'` or `'cplxdddt'`. See `dtfilters` for valid choices.

**Data Types**
char

**Output Arguments**

**wt - Wavelet transform**
structure

Wavelet transform, returned as a structure with these fields:

### type - Type of wavelet decomposition (filter bank)
`'dwt'` | `'ddt'` | `'cplxdt'` | `'cplxdddt'`

Type of wavelet decomposition (filter bank) used in the analysis, returned as one of `'dwt'`, `'ddt'`, `'cplxdt'`, or `'cplxdddt'`. The type, `'dwt'`, gives a critically sampled discrete wavelet transform. The other types correspond to oversampled wavelet transforms. `'ddt'` is a double-density wavelet transform, `'cplxdt'` is a dual-tree complex wavelet transform, and `'cplxdddt'` is a double-density dual-tree complex wavelet transform.

### level - Level of the wavelet decomposition
positive integer

Level of wavelet decomposition, returned as a positive integer.

### filters - Decomposition (analysis) and reconstruction (synthesis) filters
structure

Decomposition (analysis) and reconstruction (synthesis) filters, returned as a structure with these fields:

### Fdf - First-stage analysis filters
matrix | cell array

First-stage analysis filters, returned as an $N$-by-2 or $N$-by-3 matrix for single-tree wavelet transforms, or a cell array of two $N$-by-2 or $N$-by-3 matrices for dual-tree wavelet transforms. The matrices are $N$-by-3 for the double-density wavelet transforms. For an $N$-by-2 matrix, the first column of the matrix is the scaling (lowpass) filter and the second column is the wavelet (highpass) filter. For an $N$-by-3 matrix, the first column of the matrix is the scaling (lowpass) filter and the second and third columns are the wavelet (highpass) filters. For the dual-tree transforms, each element of the cell array contains the first-stage analysis filters for the corresponding tree.

### Df - Analysis filters for levels > 1
matrix | cell array

Analysis filters for levels > 1, returned as an $N$-by-2 or $N$-by-3 matrix for single-tree wavelet transforms, or a cell array of two $N$-by-2 or $N$-by-3 matrices for dual-tree wavelet transforms. The matrices are $N$-by-3 for the double-density wavelet transforms. For an $N$-by-2 matrix, the first column of the matrix is the scaling (lowpass) filter and the second column is the wavelet (highpass) filter. For an $N$-by-3 matrix, the first column of the matrix is the scaling (lowpass) filter and the second and third columns are the wavelet (highpass) filters. For the dual-tree transforms, each element of the cell array contains the analysis filters for the corresponding tree.

### Frf - First-level reconstruction filters
matrix | cell array

First-level reconstruction filters, returned as an $N$-by-2 or $N$-by-3 matrix for single-tree wavelet transforms, or a cell array of two $N$-by-2 or $N$-by-3 matrices for dual-tree wavelet transforms. The matrices are $N$-by-3 for the double-density wavelet transforms. For an $N$-by-2 matrix, the first column of the matrix is the scaling (lowpass) filter and the second column is the wavelet (highpass) filter. For an $N$-by-3 matrix, the first column of the matrix is the scaling (lowpass) filter and the second and third columns are the wavelet (highpass) filters. For the dual-tree transforms, each element of the cell array contains the first-stage synthesis filters for the corresponding tree.

### Rf - Reconstruction filters for levels > 1
matrix | cell array

Reconstruction filters for levels > 1, returned as an $N$-by-2 or $N$-by-3 matrix for single-tree wavelet transforms, or a cell array of two $N$-by-2 or $N$-by-3 matrices for dual-tree wavelet transforms. The matrices are $N$-by-3 for the double-density wavelet transforms. For an $N$-by-2 matrix, the first column of the matrix is the scaling (lowpass) filter and the second column is the wavelet (highpass) filter. For an $N$-by-3 matrix, the first column of the matrix is the scaling (lowpass) filter and the second and third columns are the wavelet (highpass) filters. For the dual-tree transforms, each element of the cell array contains the synthesis filters for the corresponding tree.

### cfs - Wavelet transform coefficients
cell array of matrices

Wavelet transform coefficients, returned as a 1-by-(level+1) cell array of matrices. The size and structure of the matrix elements of the cell array depend on the type of wavelet transform, typetree, as follows:

- 'dwt' — cfs{j}
  - j = 1,2,..., level is the level.
  - cfs{level+1} are the lowpass, or scaling, coefficients.
- 'ddt' — cfs{j}(:,:,k)
  - j = 1,2,..., level is the level.
  - k = 1,2 is the wavelet filter.
  - cfs{level+1}(:,:) are the lowpass, or scaling, coefficients.
- 'cplxdt' — cfs{j}(:,:,m)
  - j = 1,2,..., level is the level.
  - m = 1,2 are the real and imaginary parts.
  - cfs{level+1}(:,:) are the lowpass, or scaling, coefficients.
- 'cplxdddt' — cfs{j}(:,:,k,m)
  - j = 1,2,..., level is the level.
  - k = 1,2 is the wavelet filter.
  - m = 1,2 are the real and imaginary parts.
  - cfs{level+1}(:,:) are the lowpass, or scaling, coefficients.

**Examples**  **Complex Dual-Tree Wavelet Transform**

Obtain the complex dual-tree wavelet transform of the noisy Doppler signal. The FIR filters in the first and subsequent stages result in an approximately analytic wavelet as required.

Create the first-stage analysis filters for the two trees.

```
   Faf{1} = [0            0
    -0.0884    -0.0112
     0.0884     0.0112
     0.6959     0.0884
     0.6959     0.0884
     0.0884    -0.6959
    -0.0884     0.6959
     0.0112    -0.0884
     0.0112    -0.0884
          0          0];
Faf{2} = [ 0.0112  0
     0.0112          0
    -0.0884    -0.0884
     0.0884    -0.0884
     0.6959     0.6959
     0.6959    -0.6959
     0.0884     0.0884
    -0.0884     0.0884
          0     0.0112
          0    -0.0112];
```

Create the analysis filters for subsequent stages of the multiresolution analysis.

```
af{1} = [ 0.0352          0
          0          0
    -0.0883    -0.1143
     0.2339          0
     0.7603     0.5875
     0.5875    -0.7603
          0     0.2339
    -0.1143     0.0883
          0          0
          0    -0.0352];

af{2} = [0    -0.0352
          0          0
```

```
        -0.1143      0.0883
              0      0.2339
         0.5875     -0.7603
         0.7603      0.5875
         0.2339           0
        -0.0883     -0.1143
              0           0
         0.0352           0];
```

Load the noisy Doppler signal and obtain the complex dual-tree wavelet transform down to level 4.

```
load noisdopp;
wt = dddtree('cplxdt',noisdopp,4,Faf,af);
```

Plot an approximation based on the level-four approximation coefficients.

```
xapp = dddtreecfs('r',wt,'scale',{5});
plot(noisdopp);  hold on;
plot(cell2mat(xapp),'r','linewidth',3);
axis tight;
```

**Double-Density Wavelet Transform**

Obtain the double-density wavelet transform of a signal with two discontinuities. Use the level-one detail coefficients to localize the discontinuities.

Create a signal consisting of a 2-Hz sine wave with a duration of 1 second. The sine wave has discontinuities at 0.3 and 0.72 seconds.

```
N = 1024;
t = linspace(0,1,1024);
x = 4*sin(4*pi*t);
x = x - sign(t - .3) - sign(.72 - t);
```

```
plot(t,x); xlabel('t'); ylabel('x');
title('Original Signal');
```

Obtain the double-density wavelet transform of the signal, reconstruct an approximation based on the level-one detail coefficients, and plot the result.

```
wt = dddtree('ddt',x,1,'filters1');
wt.cfs{2} = zeros(1,512);
xrec = idddtree(wt);
plot(t,xrec,'linewidth',2)
set(gca,'xtick',[0 0.3 0.72 1]); set(gca,'xgrid','on');
```

# dddtree

### First-Level Detail Coefficients Approximation — Complex Dual-Tree

Obtain the complex dual-tree wavelet transform of a signal with two discontinuities. Use the first-level detail coefficients to localize the discontinuities.

Create a signal consisting of a 2-Hz sine wave with a duration of 1 second. The sine wave has discontinuities at 0.3 and 0.72 seconds.

```
N = 1024;
t = linspace(0,1,1024);
x = 4*sin(4*pi*t);
x = x - sign(t - .3) - sign(.72 - t);
plot(t,x); xlabel('t'); ylabel('x');
title('Original Signal');
```

Obtain the dual-tree wavelet transform of the signal, reconstruct an approximation based on the level-one detail coefficients, and plot the result.

```
wt = dddtree('cplxdt',x,1,'FSfarras','qshift06');
wt.cfs{2} = zeros(1,512,2);
xrec = idddtree(wt);
plot(t,xrec,'linewidth',2)
set(gca,'xtick',[0 0.3 0.72 1]); set(gca,'xgrid','on');
```

**See Also**    dddtree2 | dddtreecfs | dtfilters | idddtree

**Related Examples**
- "Analytic Wavelets Using the Dual-Tree Wavelet Transform"

**Concepts**
- "Critically Sampled and Oversampled Wavelet Filter Banks"

**Purpose**        Extract dual-tree/double-density wavelet coefficients or projections

**Syntax**         out = dddtreecfs(outputtype,wt,outputspec,outputindices)
                   out =
                   dddtreecfs(outputtype,wt,outputspec,outputindices,'plot')

**Description**    out = dddtreecfs(outputtype,wt,outputspec,outputindices)
                   extracts the coefficients or subspace projections from the 1-D or 2-D
                   wavelet decomposition, wt. If outputtype equals 'e', out contains
                   wavelet or scaling coefficients. If outputtype equals 'r', out contains
                   wavelet or scaling subspace projections (reconstructions).

                   out =
                   dddtreecfs(outputtype,wt,outputspec,outputindices,'plot')
                   plots the signal or image reconstruction or specified analysis
                   coefficients.  You can include the 'plot' option anywhere
                   after the wt input.

**Input
Arguments**        **outputtype - Output type**
                   'e' | 'r'

                   Output type, specified as 'e' or 'r'. Use 'e' to obtain the scaling or
                   wavelet coefficients. Use 'r' to obtain a projection, or reconstruction,
                   onto the appropriate scaling or wavelet subspace.

                   **wt - Wavelet transform**
                   structure

                   Wavelet transform, specified as a structure. The structure array is the
                   output of dddtree or dddtree2.

                   **outputspec - Output specification**
                   'lowpass' | 'scale' | 'ind' | 'cumind'

                   Output specification, specified as one of 'lowpass', 'scale', 'ind', or
                   'cumind'. The output specifications are defined as follows:

- `'lowpass'` — Outputs the lowpass, or scaling, coefficients or a signal/image approximation based on the scaling coefficients. If you set the output specification to `'lowpass'`, do not specify `outputindices`. If the `outputtype` is `'e'`, `out` is a structure array with fields identical to the input structure array `wt` except that all wavelet (detail) coefficients are equal to zero. If the `outputtype` is `'r'`, `out` is a signal or image approximation based on the scaling coefficients. The signal or image approximation is equal in size to the original input to `dddtree` or `dddtree2`.

- `'scale'` — Outputs the coefficients or a signal/image approximation based on the scales specified in `outputindices`. If the `outputtype` is `'e'`, `out` is a cell array of structure arrays. The fields of the structure arrays in `out` are identical to the fields of the input structure array `wt`. The coefficients in the `cfs` field are all equal to zero except the coefficients corresponding to the scales in `outputindices`. If the `outputtype` is `'r'`, `out` is a signal or image approximation based on the scales in `outputindices`. The signal or image approximation is equal in size to the original input to `dddtree` or `dddtree2`.

- `'ind'` — Outputs the coefficients or a signal/image approximation based on the tree-position indices specified in `outputindices`. If the `outputtype` is `'e'`, `out` is a cell array of vectors or matrices containing the coefficients specified by the tree-position indices in `outputindices`. If the `outputtype` is `'r'`, `out` is a cell array of vectors or matrices containing signal or image approximations based on the corresponding tree-position indices in `outputindices`.

- `'cumind'` — Outputs the coefficients or a signal/image approximation based on the tree-position indices specified in `outputindices`. If the `outputtype` is `'e'`, `out` is a structure array. The fields of the structure array are identical to the fields of the input structure array `wt`. The coefficients in the `cfs` field are all equal to zero except the coefficients corresponding to the tree positions in `outputindices`. If the `outputtype` is `'r'`, `out` is a signal or image approximation based on the coefficients corresponding to the tree-position indices in `outputindices`.

**Example:** `'ind',{[1 1]; [1 2]}`

**outputindices - Output indices**
cell array

Output indices, specified as a cell array with scalar or vector elements. If `outputspec` equals `'scale'`, a scalar element selects the corresponding element in the `cfs` field of `wt`. If `outputspec` equals `'ind'` or `'cumind'`, the elements of `outputspec` are row vectors. The first element of the row vector corresponds to the element in the `cfs` field of `wt`. Subsequent elements in the row vector correspond to the indices of the array contained in the cell array element.

**Example:** `'scale',{1;2;3}`

**Output Arguments**

**out - Signal or image reconstruction or coefficients**
cell array | structure | vector | matrix

Signal or image reconstruction or coefficients, returned as a vector, matrix, structure array, cell array of vectors or matrices, or cell array of structure arrays. The form of `out` depends on the value of `outputspec` and `outputindices`.

**Examples**

**Reconstruction from 1-D Complex Dual-Tree Wavelet Transform**

Obtain the complex dual-tree wavelet transform of the 1-D noisy Doppler signal. Reconstruct an approximation based on the level-three detail coefficients

Load the noisy Doppler signal. Obtain the complex dual-tree transform down to level 3.

```
load noisdopp;
wt = dddtree('cplxdt',noisdopp,3,'dtf1');
```

Plot a reconstruction of the original signal based on the level-three detail coefficients

```
xr = dddtreecfs('r',wt,'scale',{3},'plot');
```

### Coefficients from 1-D Complex Dual-Tree Wavelet Transform

Load the noisy Doppler signal. Obtain the complex dual-tree transform down to level 3.

```
load noisdopp;
wt = dddtree('cplxdt',noisdopp,3,'dtf1');
```

Create a cell array of vectors to obtain the second- and third-level detail coefficients from each of the wavelet filter bank trees.

```
outputindices = {[2 1]; [2 2]; [3 1]; [3 2]};
```

The first element of each vector in the cell array denotes the level, or stage. The second element denotes the tree.

Extract the detail coefficients.

```
out = dddtreecfs('e',wt,'ind',outputindices);
```

out is a 1-by-4 cell array. The cell array elements contain the wavelet coefficients corresponding to the elements in outputindices. For example, out{1} contains the level-two detail coefficients from the first tree.

### 1-D Complex Dual-Tree Wavelet Transform Structure

Load the noisy Doppler signal. Obtain the complex dual-tree transform down to level 3.

```
load noisdopp;
wt = dddtree('cplxdt',noisdopp,3,'dtf1');
```

Create a cell array of vectors to obtain the second- and third-level detail coefficients from each of the wavelet filter bank trees.

```
outputindices = {[2 1]; [2 2]; [3 1];[3 2]};
```

The first element of each vector in the cell array denotes the level, or stage. The second element denotes the tree.

Create a structure array identical to the wt output of dddtree with all the coefficients equal to zero except the first- and second-level detail coefficients.

```
out = dddtreecfs('e',wt,'cumind',outputindices);
```

### Extract Diagonal Features from Image

Use the complex dual-tree wavelet transform to isolate diagonal features in an image at +45 and −45 degrees.

Load and display the xbox image.

```
load xbox;
imagesc(xbox)
```

Obtain the complex dual-tree wavelet transform down to level 3.

```
fdf = dtfilters('FSfarras');
df = dtfilters('qshift10');
wt = dddtree2('cplxdt',xbox,3,fdf,df);
```

Isolate the +45 and -45 diagonal image features in the level-one wavelet coefficients. Plot the result.

```
out = dddtreecfs('e',wt,'ind',{[1 3 1 2]; [1 3 2 2]},'plot');
```

**See Also**    dddtree **|** dddtree2 **|** plotdt

# dddtree2

**Purpose**      Dual-tree and double-density 2-D wavelet transform

**Syntax**       wt = dddtree2(typetree,x,level,fdf,df)
                 wt = dddtree2(typetree,x,level,fname)
                 wt = dddtree2(typetree,x,level,fname1,fname2)

**Description**  wt = dddtree2(typetree,x,level,fdf,df) returns the typetree
                 discrete wavelet transform of the 2-D input image, x, down to level,
                 level. The wavelet transform uses the decomposition (analysis) filters,
                 fdf, for the first level and the analysis filters, df, for subsequent
                 levels. Supported wavelet transforms are the critically sampled DWT,
                 double-density, real oriented dual-tree, complex oriented dual-tree,
                 real oriented dual-tree double-density, and complex oriented dual-tree
                 double-density wavelet transform. The critically sampled DWT is
                 a filter bank decomposition in an orthogonal or biorthogonal basis
                 (nonredundant). The other wavelet transforms are oversampled filter
                 banks with differing degrees of directional selectivity.

                 wt = dddtree2(typetree,x,level,fname) uses the filters specified
                 by fname to obtain the wavelet transform. Valid filter specifications
                 depend on the type of wavelet transform. See dtfilters for details.

                 wt = dddtree2(typetree,x,level,fname1,fname2) uses the filters
                 specified in fname1 for the first stage of the dual-tree wavelet transform
                 and the filters specified in fname2 for subsequent stages of the dual-tree
                 wavelet transform. Specifying different filters for stage 1 is valid and
                 necessary only when typetree is 'realdt', 'cplxdt', 'realdddt',
                 or 'cplxdddt'.

**Input
Arguments**      **typetree - Type of wavelet decomposition**
                 'dwt' | 'ddt' | 'realdt' | 'cplxdt' | 'realdddt' | 'cplxdddt'

                 Type of wavelet decomposition, specified as one of 'dwt', 'ddt',
                 'realdt', 'cplxdt', 'realdddt', or 'cplxdddt'. The type, 'dwt',
                 produces a critically sampled (nonredundant) discrete wavelet
                 transform. The other decomposition types produce oversampled wavelet

transforms. `'ddt'` produces a double-density wavelet transform with one scaling and two wavelet filters for both row and column filtering. The double-density wavelet transform uses the same filters at all stages. `'realdt'` and `'cplxdt'` produce oriented dual-tree wavelet transforms consisting of two and four separable wavelet transforms. `'realdddt'` and `'cplxdddt'` produce double-density dual-tree wavelet transforms. The dual-tree wavelet transforms use different filters for the first stage (level).

### x - Input image
matrix

Input image, specified as a matrix with even-length row and column dimensions. Both the row and column dimensions must be divisible by $2^L$, where $L$ is the level of the wavelet transform. Additionally, the minimum of the row and column dimensions of the image must be greater than or equal to the product of the maximum length of the decomposition (analysis) filters and $2^{(L-1)}$.

**Data Types**
double

### level - Level of wavelet decomposition
integer

Level of the wavelet decomposition, specified as a positive integer. If $L$ is the value of `level`, $2^L$ must divide both the row and column dimensions of `x`. Additionally, the minimum of the row and column dimensions of the image must be greater than or equal to the product of the maximum length of the decomposition (analysis) filters and $2^{(L-1)}$.

### fdf - Level-one analysis filters
matrix | cell array

The level-one analysis filters, specified as a matrix or cell array of matrices. Specify `fdf` as a matrix when `typetree` is `'dwt'` or `'ddt'`. The size and structure of the matrix depend on the `typetree` input as follows:

- `'dwt'` — This is the critically sampled discrete wavelet transform. In this case, `fdf` is a two-column matrix with the lowpass (scaling) filter in the first column and the highpass (wavelet) filter in the second column.

- `'ddt'` — This is the double-density wavelet transform. The double-density DWT is a three-channel perfect reconstruction filter bank. `fdf` is a three-column matrix with the lowpass (scaling) filter in the first column and the two highpass (wavelet) filters in the second and third columns. In the double-density wavelet transform, the single lowpass and two highpass filters constitute a three-channel perfect reconstruction filter bank. This is equivalent to the three filters forming a tight frame. You cannot arbitrarily choose the two wavelet filters in the double-density DWT. The three filters together must form a tight frame.

Specify `fdf` as a 1-by-2 cell array of matrices when `typetree` is a dual-tree transform, `'realdt'`, `'cplxdt'`, `'realdddt'`, or `'cplxdddt'`. The size and structure of the matrix elements in the cell array depend on the `typetree` input as follows:

- For the dual-tree complex wavelet transforms, `'realdt'` and `'cplxdt'`, `fdf{1}` is an *N*-by-2 matrix containing the lowpass (scaling) and highpass (wavelet) filters for the first tree and `fdf{2}` is an *N*-by-2 matrix containing the lowpass (scaling) and highpass (wavelet) filters for the second tree.

- For the double-density dual-tree complex wavelet transforms, `'realdddt'` and `'cplxdddt'`, `fdf{1}` is an *N*-by-3 matrix containing the lowpass (scaling) and two highpass (wavelet) filters for the first tree and `fdf{2}` is an *N*-by-3 matrix containing the lowpass (scaling) and two highpass (wavelet) filters for the second tree.

### df - Analysis filters for levels > 1
matrix | cell array

Analysis filters for levels > 1, specified as a matrix or cell array of matrices. Specify `df` as a matrix when `typetree` is `'dwt'` or `'ddt'`.

The size and structure of the matrix depend on the `typetree` input as follows:

- `'dwt'` — This is the critically sampled discrete wavelet transform. In this case, `df` is a two-column matrix with the lowpass (scaling) filter in the first column and the highpass (wavelet) filter in the second column. For the critically sampled orthogonal or biorthogonal DWT, the filters in `df` and `fdf` must be identical.

- `'ddt'` — This is the double-density wavelet transform. The double-density DWT is a three-channel perfect reconstruction filter bank. `df` is a three-column matrix with the lowpass (scaling) filter in the first column and the two highpass (wavelet) filters in the second and third columns. In the double-density wavelet transform, the single lowpass and two highpass filters constitute a three-channel perfect reconstruction filter bank. This is equivalent to the three filters forming a tight frame. For the double-density DWT, the filters in `df` and `fdf` must be identical.

Specify `df` as a 1-by-2 cell array of matrices when `typetree` is a dual-tree transform, `'realdt'`, `'cplxdt'`, `'realdddt'`, or `'cplxdddt'`. For dual-tree transforms, the filters in `fdf` and `df` must be different. The size and structure of the matrix elements in the cell array depend on the `typetree` input as follows:

- For the dual-tree wavelet transforms, `'realdt'` and `'cplxdt'`, `df{1}` is an *N*-by-2 matrix containing the lowpass (scaling) and highpass (wavelet) filters for the first tree and `df{2}` is an *N*-by-2 matrix containing the lowpass (scaling) and highpass (wavelet) filters for the second tree.

- For the double-density dual-tree complex wavelet transforms, `'realdddt'` and `'cplxdddt'`, `df{1}` is an *N*-by-3 matrix containing the lowpass (scaling) and two highpass (wavelet) filters for the first tree and `df{2}` is an *N*-by-3 matrix containing the lowpass (scaling) and two highpass (wavelet) filters for the second tree.

**fname - Filter name**
string

Filter name, specified as a string. For the critically sampled DWT, specify any valid orthogonal or biorthogonal wavelet filter. See `wfilters` for details. For the redundant wavelet transforms, see `dtfilters` for valid filter names.

**fname1 - First-stage filter name**
string

First-stage filter name, specified as a string. Specifying a first-level filter that is different from the wavelet and scaling filters in subsequent levels is valid and necessary only with the dual-tree wavelet transforms, `'realdt'`, `'cplxdt'`, `'realdddt'`, and `'cplxdddt'`.

**fname2 - Filter name for stages > 1**
string

Filter name for stages > 1, specified as a string. Specifying a different filter for stages > 1 is valid and necessary only with the dual-tree wavelet transforms, `'realdt'`, `'cplxdt'`, `'realdddt'`, and `'cplxdddt'`.

**Output Arguments**

**wt - Wavelet transform**
structure

Wavelet transform, returned as a structure with these fields:

**type - Type of wavelet decomposition (filter bank)**
`'dwt'` | `'ddt'` | `'realdt'` | `'cplxdt'` | `'realdddt'` | `'cplxdddt'`

Type of wavelet decomposition used in the analysis returned as one of `'dwt'`, `'ddt'`, `'realdt'`, `'cplxdt'`, `'realdddt'`, or `'cplxdddt'`. `'dwt'` is the critically sampled DWT. `'ddt'` produces a double-density wavelet transform with one scaling and two wavelet filters for both row and column filtering. `'realdt'` and `'cplxdt'` produce oriented dual-tree wavelet transforms consisting of 2 and 4 separable wavelet transforms. `'realdddt'` and `'cplxdddt'` produce double-density dual-tree wavelet transforms consisting of two and four separable wavelet transforms.

**level - Level of wavelet decomposition**
positive integer

Level of wavelet decomposition, returned as a positive integer.

### filters - Decomposition (analysis) and reconstruction (synthesis) filters
structure

Decomposition (analysis) and reconstruction (synthesis) filters, returned as a structure with these fields:

### Fdf - First-stage analysis filters
matrix | cell array

First-stage analysis filters, returned as an $N$-by-2 or $N$-by-3 matrix for single-tree wavelet transforms, or a 1-by-2 cell array of two $N$-by-2 or $N$-by-3 matrices for dual-tree wavelet transforms. The matrices are $N$-by-3 for the double-density wavelet transforms. For an $N$-by-2 matrix, the first column of the matrix is the scaling (lowpass) filter and the second column is the wavelet (highpass) filter. For an $N$-by-3 matrix, the first column of the matrix is the scaling (lowpass) filter and the second and third columns are the wavelet (highpass) filters. For the dual-tree transforms, each element of the cell array contains the first-stage analysis filters for the corresponding tree.

### Df - Analysis filters for levels > 1
matrix | cell array

Analysis filters for levels > 1, returned as an $N$-by-2 or $N$-by-3 matrix for single-tree wavelet transforms, or a 1-by-2 cell array of two $N$-by-2 or $N$-by-3 matrices for dual-tree wavelet transforms. The matrices are $N$-by-3 for the double-density wavelet transforms. For an $N$-by-2 matrix, the first column of the matrix is the scaling (lowpass) filter and the second column is the wavelet (highpass) filter. For an $N$-by-3 matrix, the first column of the matrix is the scaling (lowpass) filter and the second and third columns are the wavelet (highpass) filters. For the dual-tree transforms, each element of the cell array contains the analysis filters for the corresponding tree.

### Frf - First-level reconstruction filters

matrix | cell array

First-level reconstruction filters, returned as an *N*-by-2 or *N*-by-3 matrix for single-tree wavelet transforms, or a 1-by-2 cell array of two *N*-by-2 or *N*-by-3 matrices for dual-tree wavelet transforms. The matrices are *N*-by-3 for the double-density wavelet transforms. For an *N*-by-2 matrix, the first column of the matrix is the scaling (lowpass) filter and the second column is the wavelet (highpass) filter. For an *N*-by-3 matrix, the first column of the matrix is the scaling (lowpass) filter and the second and third columns are the wavelet (highpass) filters. For the dual-tree transforms, each element of the cell array contains the first-stage synthesis filters for the corresponding tree.

### Rf - Reconstruction filters for levels > 1

matrix | cell array

Reconstruction filters for levels > 1, returned as an N-by-2 or N-by-3 matrix for single-tree wavelet transforms, or a 1-by-2 cell array of two N-by-2 or N-by-3 matrices for dual-tree wavelet transforms. The matrices are N-by-3 for the double-density wavelet transforms. For an N-by-2 matrix, the first column of the matrix is the scaling (lowpass) filter and the second column is the wavelet (highpass) filter. For an N-by-3 matrix, the first column of the matrix is the scaling (lowpass) filter and the second and third columns are the wavelet (highpass) filters. For the dual-tree transforms, each element of the cell array contains the first-stage analysis filters for the corresponding tree.

### cfs - Wavelet transform coefficients

cell array of matrices

Wavelet transform coefficients, specified as a 1-by-(`level`+1) cell array of matrices. The size and structure of the matrix elements of the cell array depend on the type of wavelet transform, `typetree` as follows:

- `'dwt'` — `cfs{j}(:,:,d)`
  - `j = 1,2,...,` `level` is the level.
  - `d = 1,2,3` is the orientation.

- cfs{level+1}(:,:) are the lowpass, or scaling, coefficients.

- 'ddt' — cfs{j}(:,:,d)

  - j = 1,2,..., level is the level.

  - d = 1,2,3,4,5,6,7,8 is the orientation.

  - cfs{level+1}(:,:) are the lowpass, or scaling, coefficients.

- 'realddt' — cfs{j}(:,:,d,k)

  - j = 1,2,..., level is the level.

  - d = 1,2,3 is the orientation.

  - k = 1,2 is the wavelet transform tree.

  - cfs{level+1}(:,:) are the lowpass, or scaling, coefficients.

- 'cplxdt' — cfs{j}(:,:,d,k,m)

  - j = 1,2,..., level is the level.

  - d = 1,2,3 is the orientation.

  - k = 1,2 is the wavelet transform tree.

  - m = 1,2 are the real and imaginary parts.

  - cfs{level+1}(:,:) are the lowpass, or scaling, coefficients.

- 'realdddt' — cfs{j}(:,:,d,k)

  - j = 1,2,..., level is the level.

  - d = 1,2,3 is the orientation.

  - k = 1,2 is the wavelet transform tree.

  - cfs{level+1}(:,:) are the lowpass, or scaling, coefficients.

- 'cplxdddt' — cfs{j}(:,:,d,k,m)

  - j = 1,2,..., level is the level.

  - d = 1,2,3 is the orientation.

  - k = 1,2 is the wavelet transform tree.

- `m = 1,2` are the real and imaginary parts.

- `cfs{level+1}(:,:)` are the lowpass, or scaling, coefficients.

**Examples**     **Real Oriented Dual-Tree Wavelets**

Visualize the six directional wavelets of the real oriented dual-tree wavelet transform.

Create the first-stage analysis filters for the two trees.

```
Faf{1} = [0          0
  -0.0884   -0.0112
   0.0884    0.0112
   0.6959    0.0884
   0.6959    0.0884
   0.0884   -0.6959
  -0.0884    0.6959
   0.0112   -0.0884
   0.0112   -0.0884
        0         0];
Faf{2} = [ 0.0112  0
   0.0112        0
  -0.0884   -0.0884
   0.0884   -0.0884
   0.6959    0.6959
   0.6959   -0.6959
   0.0884    0.0884
  -0.0884    0.0884
        0    0.0112
        0   -0.0112];
```

Create the analysis filters for subsequent stages of the multiresolution analysis.

```
af{1} = [ 0.0352        0
        0         0
  -0.0883   -0.1143
   0.2339        0
```

```
      0.7603     0.5875
      0.5875    -0.7603
           0     0.2339
     -0.1143     0.0883
           0          0
           0    -0.0352];

af{2} = [0     -0.0352
           0          0
     -0.1143     0.0883
           0     0.2339
      0.5875    -0.7603
      0.7603     0.5875
      0.2339          0
     -0.0883    -0.1143
           0          0
      0.0352         0];
```

Obtain the real dual-tree wavelet transform of an image of zeros down to level 4.

```
J = 4;
L = 3*2^(J+1);
N = L/2^J;
x = zeros(2*L,3*L);
wt = dddtree2('realdt',x,J,Faf,af);
```

Insert a 1 in one position of the six subbands and invert the wavelet transform.

```
wt.cfs{4}(N/2,N/2+0*N,1,1) = 1;
wt.cfs{4}(N/2,N/2+1*N,2,1) = 1;
wt.cfs{4}(N/2,N/2+2*N,3,1) = 1;
wt.cfs{4}(N/2+N,N/2+0*N,1,2) = 1;
wt.cfs{4}(N/2+N,N/2+1*N,2,2) = 1;
wt.cfs{4}(N/2+N,N/2+2*N,3,2) = 1;
xrec = idddtree2(wt);
```

Visualize the six directional wavelets.

```
imagesc(xrec);
colormap gray; axis off;
title('Real Oriented Dual-Tree Wavelets')
```

**Real Oriented Dual-Tree Wavelets**



### Double-Density Wavelet Transform

Obtain the double-density wavelet transform of an image.

Load the image and obtain the double-density wavelet transform.

```
load xbox;
```

```
imagesc(xbox); colormap gray;
wt = dddtree2('ddt',xbox,1,'filters1');
```



Visualize the diagonal details in the two wavelet HH subbands.

```
HH1 = wt.cfs{1}(:,:,5);
HH2 = wt.cfs{1}(:,:,8);
subplot(211)
imagesc(HH1);
colormap gray;
subplot(212);
imagesc(HH2);
```

### Complex Dual-Tree Wavelet Transform

Obtain the complex dual-tree wavelet transform of an image. Show that
the complex dual-tree wavelet transform can detect the two different
diagonal directions.

Load the image and obtain the complex dual-tree wavelet transform.

```
load xbox;
imagesc(xbox); colormap gray;
wt = dddtree2('cplxdt',xbox,1,'FSfarras','qshift10');
```

Obtain and display the imaginary parts of the 2 trees.

```
waveletcfs = wt.cfs{1};
subplot(211)
imagesc(waveletcfs(:,:,3,1,2));
colormap gray;
subplot(212)
imagesc(waveletcfs(:,:,3,2,2));
```

**See Also**      `dddtree` **|** `dddtreecfs` **|** `dtfilters` **|** `idddtree2`

**Related
Examples**      • "Analytic Wavelets Using the Dual-Tree Wavelet Transform"

**Concepts**      • "Critically Sampled and Oversampled Wavelet Filter Banks"

**Purpose**        Node depth-position to node index

**Syntax**

**Description**    depo2ind is a tree-management utility.

For a tree of order ORD, N = depo2ind(ORD,[D P]) computes the indices N of the nodes whose depths and positions are encoded within [D,P].

The nodes are numbered from left to right and from top to bottom. The root index is 0.

D and P are column vectors. The values of depths D and positions P must be such that $D \geq 0$ and $0 \leq P \leq ORD^{D-1}$.

Output indices N are such that $0 \leq N < (ORD^{\max(D)}-1)$ / (ORD-1).

Note that for a column vector X, we have depo2ind(0,X) = X.

**Examples**
```
% Create initial tree.
ord = 2;
t = ntree(ord,3);      % binary tree of depth 3.
t = nodejoin(t,5);
t = nodejoin(t,4);
plot(t)
```



```
% List t nodes (Depth_Position).
```

```
aln_depo = allnodes(t,'deppos')
aln_depo =
    0    0
    1    0
    1    1
    2    0
    2    1
    2    2
    2    3
    3    0
    3    1
    3    6
    3    7
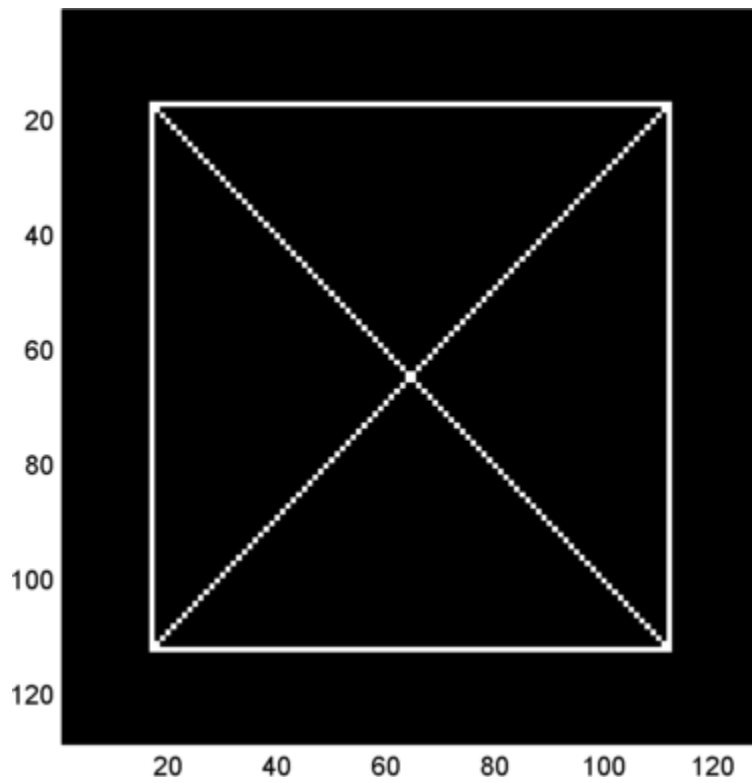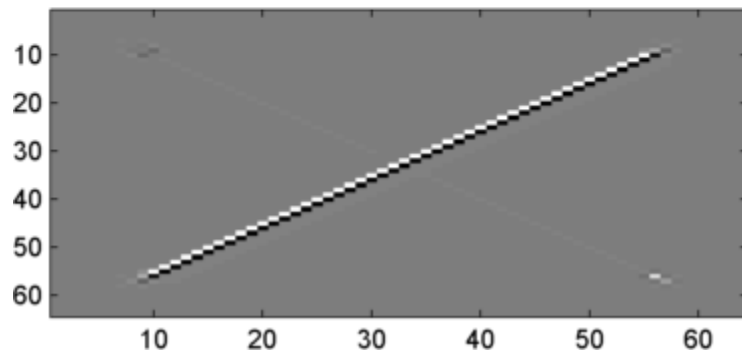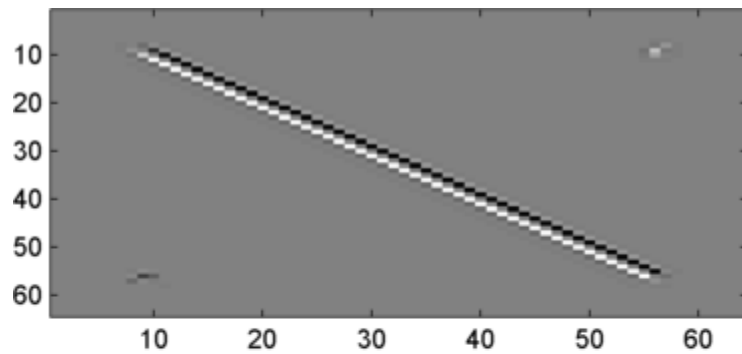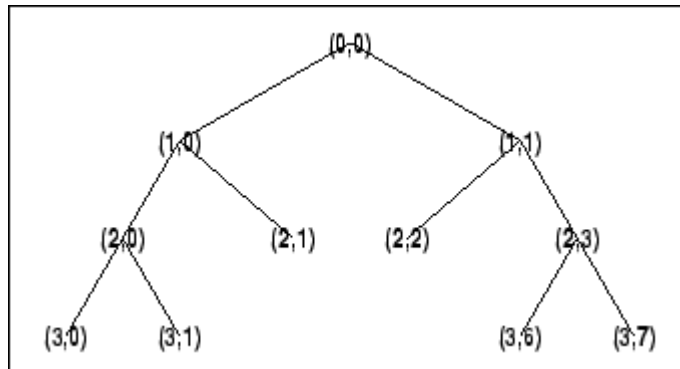
% Switch from Depth_Position to index.
aln_ind = depo2ind(ord,aln_depo)
aln_ind =
    0
    1
    2
    3
    4
    5
    6
    7
    8
   13
   14
```

**See Also**    ind2depo

**Purpose**     1-D detail coefficients

**Syntax**      D = detcoef(C,L,N)
                D = detcoef(C,L)

**Description**   detcoef is a one-dimensional wavelet analysis function.

D = detcoef(C,L,N) extracts the detail coefficients at level *N* from the wavelet decomposition structure [C,L]. See wavedec for more information on *C* and *L*.

Level N must be an integer such that $1 \leq N \leq NMAX$ where NMAX = length(L)-2.

D = detcoef(C,L) extracts the detail coefficients at last level NMAX.

If *N* is a vector of integers such that $1 \leq N(j) \leq NMAX$:

- DCELL = detcoef(C,L,N,'cells') returns a cell array where DCELL{j} contains the coefficients of detail N(j).

- If length(N) > 1, DCELL = detcoef(C,L,N) is equivalent to DCELL = detcoef(C,L,N,'*cells*').

- DCELL = detcoef(C,L,'cells') is equivalent to DCELL = detcoef(C,L,[1:NMAX]).

- [D1, ... ,Dp] = detcoef(C,L,[N(1), ... ,N(p)]) extracts the details coefficients at levels [N(1), ... ,N(p)].

**Examples**     % The current extension mode is zero-padding (see dwtmode).

% Load original one-dimensional signal.
load leleccum;
s = leleccum(1:3920);

% Perform decomposition at level 3 of s using db1.
[c,l] = wavedec(s,3,'db1');

% Extract detail coefficients at levels
% 1, 2 and 3, from wavelet decomposition

```
% structure [c,l].
[cd1,cd2,cd3] = detcoef(c,l,[1 2 3]);
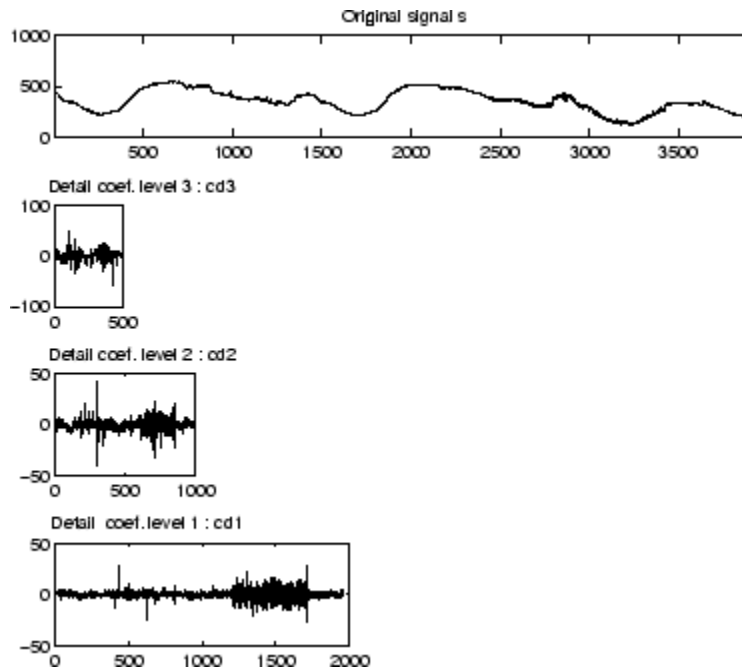
% Using some plotting commands,
% the following figure is generated.
```



**See Also**          appcoef | wavedec

**Purpose**    2-D detail coefficients

**Syntax**     D = detcoef2(O,C,S,N)

**Description**    detcoef2 is a two-dimensional wavelet analysis function.

D = detcoef2(O,C,S,N) extracts from the wavelet decomposition structure [C,S] the horizontal, vertical, or diagonal detail coefficients for O = 'h'(or 'v' or 'd', respectively), at level *N*, where *N* must be an integer such that $1 \leq N \leq$ size(S,1)-2. See wavedec2 for more information on C and S.

[H,V,D] = detcoef2('all',C,S,N) returns the horizontal H, vertical V, and diagonal D detail coefficients at level N.

D = detcoef2('compact',C,S,N) returns the detail coefficients at level N, stored row-wise.

detcoef2('a',C,S,N) is equivalent to detcoef2('all',C,S,N).

detcoef2('c',C,S,N) is equivalent to detcoef2('compact',C,S,N).

**Tips**    If C and S are obtained from an indexed image analysis or a truecolor image analysis, D is an m-by-n matrix or an m-by-n-by-3 array, respectively.

For more information on image formats, see the image and imfinfo reference pages.

**Examples**    % The current extension mode is zero-padding (see dwtmode).

% Load original image.
load woman;

% X contains the loaded image.

% Perform decomposition at level 2
% of X using db1.
[c,s] = wavedec2(X,2,'db1');

```
sizex = size(X)
sizex =
    256    256

sizec = size(c)
sizec =
    1    65536

val_s = s
val_s =
      64      64
      64      64
     128     128
     256     256

% Extract details coefficients at level 2
% in each orientation, from wavelet decomposition
% structure [c,s].
[chd2,cvd2,cdd2] = detcoef2('all',c,s,2);
sizecd2 = size(chd2)
sizecd2 =
      64      64

% Extract details coefficients at level 1
% in each orientation, from wavelet decomposition
% structure [c,s].
[chd1,cvd1,cdd1] = detcoef2('all',c,s,1);
sizecd1 = size(chd1)
sizecd1 =
     128     128
```

**See Also**    appcoef2 | wavedec2

**Purpose**     WPTREE information

**Syntax**      disp(*T*)

**Description**  disp(*T*) displays the content of the WPTREE object *T*.

**Examples**    ```
% Compute a wavelet packets tree
x = rand(1,1000);
t = wpdec(x,2,'db2');
disp(t)

 Wavelet Packet Object Structure
 =================================
 Size of initial data      : [1 1000]
 Order                     : 2
 Depth                     : 2
 Terminal nodes            : [3  4  5  6]
 ----------------------------------------------------
 Wavelet Name              : db2
 Low Decomposition filter  : [-0.1294  0.2241  0.8365  0.483]
 High Decomposition filter : [ -0.483  0.8365 -0.2241 -0.1294]
 Low Reconstruction filter : [  0.483  0.8365  0.2241 -0.1294]
 High Reconstruction filter : [-0.1294 -0.2241  0.8365 -0.483]
 ----------------------------------------------------
 Entropy Name              : shannon
 Entropy Parameter         : 0
 ----------------------------------------------------
```

**See Also**    get | read | set | write

# displs

| | |
|---|---|
| **Purpose** | Display lifting scheme |
| **Syntax** | S = displs(*LS*,*FRM*) |
| **Description** | S = displs(*LS*,*FRM*) returns a string describing the lifting scheme *LS*. The format string *FRM* (see sprintf) builds S.

displs(LS) is equivalent to DISPLS(LS,'%12.8f')

For more information about lifting schemes, see lsinfo. |

**Examples**

```
% Start from the Haar wavelet and get the
% corresponding lifting scheme.
lshaar = liftwave('haar');

% Visualize the obtained lifting scheme.
displs(lshaar);

lshaar = {...
'd'              [ -1.00000000]  [0]
'p'              [  0.50000000]  [0]
[  1.41421356]  [  0.70710678]  []
};

% Add a primal ELS to the lifting scheme.
els = {'p',[-0.125 0.125],0};
lsnew = addlift(lshaar,els);
displs(lsnew);

lsnew = {...
'd'              [ -1.00000000]                 [0]
'p'              [  0.50000000]                 [0]
'p'              [ -0.12500000   0.12500000]  [0]
[  1.41421356]  [  0.70710678]                 []
};
```

**See Also**    lsinfo

1-152

**Purpose**     Draw wavelet packet decomposition tree (GUI)

**Syntax**      drawtree(*T*)
                F = drawtree(*T*)
                drawtree(*T*,*F*)

**Description**  drawtree(*T*) draws the wavelet packet tree *T*, and F = drawtree(*T*)
                also returns the figure's handle.

                For an existing figure F produced by a previous call to the drawtree
                function, drawtree(*T*,*F*) draws the wavelet packet tree T in the figure
                whose handle is F.

**Examples**    x   = sin(8*pi*[0:0.005:1]);
                t   = wpdec(x,3,'db2');
                fig = drawtree(t);
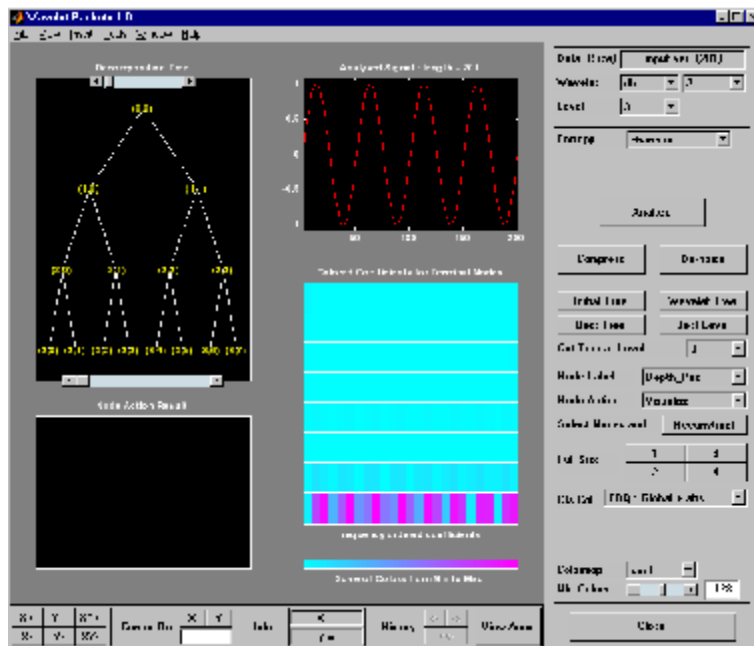
# drawtree

```
%--------------------------------------
% Use command line function to modify t.
%--------------------------------------
t   = wpjoin(t,2);
drawtree(t,fig);
```



**See Also**      readtree

1-154
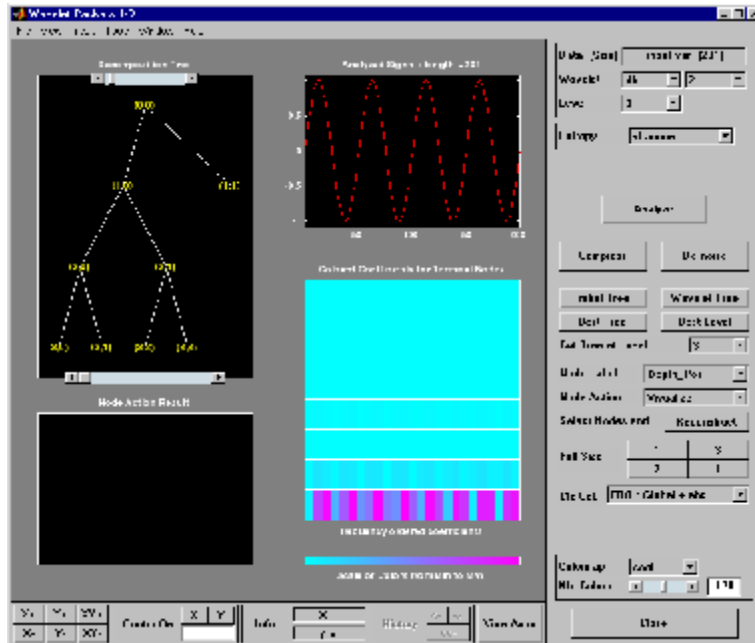
| | |
|---|---|
| **Purpose** | Analysis and synthesis filters for oversampled wavelet filter banks |
| **Syntax** | df = dtfilters(name)<br>[df,rf] = dtfilters(name) |
| **Description** | df = dtfilters(name) returns the decomposition (analysis) filters corresponding to the string, name.<br><br>[df,rf] = dtfilters(name) returns the reconstruction (synthesis) filters corresponding to the string, name. |

**Input Arguments**

**name - Filter name**

`'dtf1'` | `'dddtf1'` | `'self1'` | `'self2'` | ...

Filter name, specified as a string. Valid entries for name are:

- Any valid orthogonal or biorthogonal wavelet name. See `wfilters` for details. An orthogonal or biorthogonal wavelet is only valid when the filter bank type is `'dwt'`, or when you use the filter as the first stage in a complex dual-tree transform, `'realdt'` or `'cplxdt'`. An orthogonal or biorthogonal wavelet filter is not a valid filter if you have a double-density, `'ddt'` or dual-tree double-density, `'realdddt'` or `'cplxdddt'`, filter bank. An orthogonal or biorthogonal wavelet filter is not a valid filter for complex dual-tree filter banks for stages greater than 1.

- `'dtfP'` — With P equal to 1, 2, 3, or 4 returns the first-stage Farras filters (`'FSfarras'`) and Kingsbury Q-shift filters (`'qshiftN'` for subsequent stages. This input is only valid for a dual-tree transform, `'realdt'` or `'cplxdt'`. Setting P= 1, 2, 3, or 4 specifies the Kingsbury Q-shift filters with N = 6, 10, 14, or 18 taps respectively.

- `'dddtf1'` — Returns the filters for the first and subsequent stages of the double-density dual-tree transform. This input is only valid for the double-density dual-tree transforms, `'realdddt'` and `'cplxdddt'`.

- `'self1'` — Returns 10-tap filters for the double-density wavelet transform. This option is only valid for double-density wavelet transforms, `'ddt'`, `'realdddt'`, and `'cplxdddt'`.

- `'self2'` — Returns 16-tap filters for the double-density wavelet transform. This option is only valid for double-density wavelet transforms, `'ddt'`, `'realdddt'`, and `'cplxdddt'`.

- `'filters1'` — Returns 6-tap filters for the double-density wavelet transform, `'ddt'`.

- `'filters2'` — Returns 12-tap filters for the double-density wavelet transform, `'ddt'`.

- `'farass'` — Farrass nearly symmetric filters for a two-channel perfect reconstruction filter bank. This option is only valid for an orthogonal critically sampled wavelet transform, `'dwt'`.

- `'FSfarass'` — Farrass nearly symmetric first-stage filters for a dual-tree wavelet transform.

- `'qshiftN'` — Kingsbury Q-shift N-tap filters with N = 6,10,14, or 18. The Kingsbury Q-shift filters are most commonly used in dual-tree wavelet transforms for stages greater than 1.

- `'doubledualfilt'` — Filters for one stage of the double-density dual-tree wavelet transforms, `'realdddt'` or `'cplxdddt'`.

**Output Arguments**

**df - Decomposition (analysis) filters**

matrix | cell array

Decomposition (analysis) filters, returned as a matrix or cell array of matrices.

**rf - Reconstruction (synthesis) filters**

matrix | cell array

Reconstruction (synthesis) filters, returned as a matrix or cell array of matrices.

**Examples**     **Filters for Complex Dual-Tree Wavelet Transform**

Obtain valid filters for the complex dual-tree wavelet transform. The transform uses Farrass nearly symmetric filters for the first stage and Kingsbury Q-shift filters with 10 taps for subsequent stages.

Load the noisy Doppler signal. Obtain the filters for the first and subsequent stages of the complex dual-tree wavelet transform. Demonstrate perfect reconstruction using the complex dual-tree wavelet transform.

```
load noisdopp;
df = dtfilters('dtf2');
dt = dddtree('cplxdt',noisdopp,5,df{1},df{2});
xrec = idddtree(dt);
max(abs(noisdopp-xrec))
```

**Filters for Double-Density Wavelet Transform**

Obtain valid filters for the double-density wavelet transform.

Load the noisy Doppler signal. Obtain the filters for the double-density wavelet transform. The double-density wavelet transform uses the same filters at all stages. Demonstrate perfect reconstruction using the double-density wavelet transform.

```
df = dtfilters('filters1');
load noisdopp;
dt = dddtree('ddt',noisdopp,5,df,df);
xrec = idddtree(dt);
max(abs(noisdopp-xrec))
```

**See Also**     dddtree **|** dddtree2

# dtree

| | |
|---|---|
| **Purpose** | DTREE constructor |
| **Syntax** | `T = dtree(ORD,D,X)` |
| | `T = dtree(ORD,D,X,U)` |
| | `[T,NB] = dtree(...)` |
| | `[T,NB] =` |
| | `dtree('PropName1',PropValue1,'PropName2',PropValue2,...)` |

**Description**     `T = dtree(ORD,D,X)` returns a complete data tree (DTREE) object of order *ORD* and depth *D*. The data associated with the tree *T* is *X*.

With `T = dtree(ORD,D,X,U)` you can set a user data field.

`[T,NB] = dtree(...)` returns also the number of terminal nodes (leaves) of *T*.

`[T,NB] =`
`dtree('PropName1',PropValue1,'PropName2',PropValue2,...)` is the most general syntax to construct a DTREE object.

The valid choices for '*PropName*' are

| | |
|---|---|
| `'order'` | Order of the tree |
| `'depth'` | Depth of the tree |
| `'data'` | Data associated to the tree |
| `'spsch'` | Split scheme for nodes |
| `'ud'` | User data field |

The split scheme field is an order `ORD` by 1 logical array. The root of the tree can be split and it has `ORD` children. If `spsch(j) = 1`, you can split the `j`-th child. Each node that you can split has the same property as the root node.

For more information on object fields, type `help dtree/get`.

Class DTREE (Parent class: NTREE)

| **Fields** | dtree | Parent object |
|---|---|---|
| | allNI | All nodes information |
| | terNI | Terminal nodes information |

**Examples**
```
% Create a data tree.
x = [1:10];
t = dtree(3,2,x);
t = nodejoin(t,2);
```

**See Also**      ntree | wtbo

# dwt

**Purpose**        Single-level discrete 1-D wavelet transform

**Syntax**
```
[cA,cD] = dwt(X,'wname')
[cA,cD] = dwt(X,Lo_D,Hi_D)
[cA,cD] = dwt(...,'mode',MODE)
```

**Description**    The dwt command performs a single-level one-dimensional wavelet decomposition with respect to either a particular wavelet ('*wname*', see wfilters for more information) or particular wavelet decomposition filters (Lo_D and Hi_D) that you specify.

[cA,cD] = dwt(X,'*wname*') computes the approximation coefficients vector cA and detail coefficients vector cD, obtained by a wavelet decomposition of the vector X. The string '*wname*' contains the wavelet name.

[cA,cD] = dwt(X,Lo_D,Hi_D) computes the wavelet decomposition as above, given these filters as input:

- Lo_D is the decomposition low-pass filter.

- Hi_D is the decomposition high-pass filter.

Lo_D and Hi_D must be the same length.

Let $lx$ = the length of X and $lf$ = the length of the filters Lo_D and Hi_D; then length(cA) = length(cD) = la where la = ceil($lx$/2), if the DWT extension mode is set to periodization. For the other extension modes, la = floor($lx+lf-1$)/2).

For more information about the different Discrete Wavelet Transform extension modes, see dwtmode.

[cA,cD] = dwt(...,'mode',MODE) computes the wavelet decomposition with the extension mode MODE that you specify. MODE is a string containing the desired extension mode.

Example:

```
[cA,cD] = dwt(x,'db1','mode','sym');
```

**Examples**     **DWT Using Wavelet Name**

Obtain the level-1 DWT of the noisy Doppler signal using a wavelet name.

```
load noisdopp;
[A,D] = dwt(noisdopp,'sym4');
```

**DWT Using Wavelet and Scaling Filters**

Obtain the level-1 DWT of the noisy Doppler signal using wavelet and scaling filters.

```
load noisdopp;
[Lo_D,Hi_D] = wfilters('bior3.5','d');
[A,D] = dwt(noisdopp,Lo_D,Hi_D);
```

**Algorithms**     Starting from a signal *s*, two sets of coefficients are computed: approximation coefficients $CA_1$, and detail coefficients $CD_1$. These vectors are obtained by convolving *s* with the low-pass filter Lo_D for approximation and with the high-pass filter Hi_D for detail, followed by dyadic decimation.

More precisely, the first step is
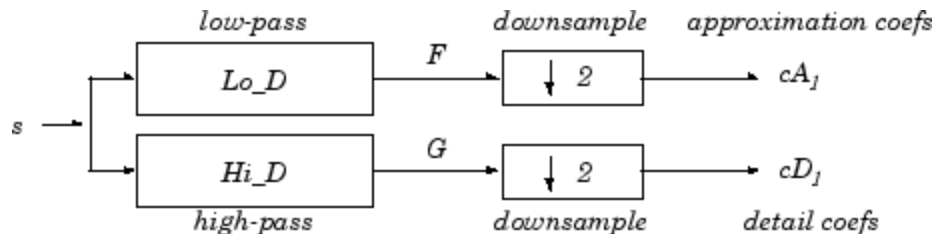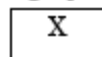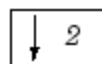


The length of each filter is equal to 2*N*. If $n = \texttt{length}(s)$, the signals *F* and *G* are of length $n + 2N - 1$, and then the coefficients $CA_1$ and $CD_1$ are of length

$$\text{floor}\left(\frac{n-1}{2}\right) + N$$

To deal with signal-end effects involved by a convolution-based algorithm, a global variable managed by dwtmode is used. This variable defines the kind of signal extension mode used. The possible options include zero-padding (used in the previous example) and symmetric extension, which is the default mode.

---

**Note** For the same input, this dwt function and the DWT block in the Signal Processing Toolbox™ do not produce the same results. The blockset is designed for real-time implementation while Wavelet Toolbox™ software is designed for analysis, so they produce handle boundary conditions and filter states differently.

To make the dwt function output match the DWT block output, set the function boundary condition to zero-padding by typing dwtmode('zpd') at the MATLAB® command prompt. To match the latency of the DWT block, which is implemented using FIR filters, add zeros to the input of the dwt function. The number of zeros you add must be equal to half the filter length.

---

**References**   Daubechies, I. (1992), *Ten lectures on wavelets*, CBMS-NSF conference series in applied mathematics. SIAM Ed.

Mallat, S. (1989), "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Pattern Anal. and Machine Intell.*, vol. 11, no. 7, pp. 674–693.

Meyer, Y. (1990), *Ondelettes et opérateurs*, Tome 1, Hermann Ed. (English translation: *Wavelets and operators*, Cambridge Univ. Press. 1993.)

**See Also**   dwtmode | idwt | wavedec | waveinfo

**Purpose**          Single-level discrete 2-D wavelet transform

**Syntax**           [cA,cH,cV,cD] = dwt2(X,'*wname*')
                     [cA,cH,cV,cD] = dwt2(X,Lo_D,Hi_D)
                     [cA,cH,cV,cD] = dwt2(...,'mode',MODE)

**Description**      The dwt2 command performs a single-level two-dimensional wavelet
                    decomposition with respect to either a particular wavelet ('*wname*', see
                    wfilters for more information) or particular wavelet decomposition
                    filters (Lo_D and Hi_D) you specify.

                    [cA,cH,cV,cD] = dwt2(X,'*wname*') computes the approximation
                    coefficients matrix cA and details coefficients matrices cH, cV, and cD
                    (horizontal, vertical, and diagonal, respectively), obtained by wavelet
                    decomposition of the input matrix X. The '*wname*' string contains the
                    wavelet name.

                    [cA,cH,cV,cD] = dwt2(X,Lo_D,Hi_D) computes the two-dimensional
                    wavelet decomposition as above, based on wavelet decomposition filters
                    that you specify.

                    • Lo_D is the decomposition low-pass filter.

                    • Hi_D is the decomposition high-pass filter.

                    Lo_D and Hi_D must be the same length.

                    Let sx = size(X) and lf = the length of filters; then
                    size(cA) = size(cH) = size(cV) = size(cD) = sa where sa =
                    ceil(sx/2), if the DWT extension mode is set to periodization. For the
                    other extension modes, sa = floor((sx+lf-1)/2).

                    For information about the different Discrete Wavelet Transform
                    extension modes, see dwtmode.

                    [cA,cH,cV,cD] = dwt2(...,'mode',MODE) computes the wavelet
                    decomposition with the extension mode MODE that you specify.

                    MODE is a string containing the desired extension mode.

                    An example of valid use is

```
[cA,cH,cV,cD] = dwt2(x,'db1','mode','sym');
```

**Tips**    When X represents an indexed image, then X, as well as the output
arrays cA,cH,cV,cD are m-by-n matrices. When X represents a truecolor
image, it is an m-by-n-by-3 array, where each m-by-n matrix represents a
red, green, or blue color plane concatenated along the third dimension.

For more information on image formats, see the image and imfinfo
reference pages.

**2-D DWT of Image**    This example shows how to obtain the 2-D DWT of an image.

Load the "woman" image and obtain the 2-D DWT using the 'sym4'
wavelet. Use the periodic extension mode.

```
load woman;
wname = 'sym4';
[CA,CH,CV,CD] = dwt2(X,wname,'mode','per');
```

Display the vertical detail image and the lowpass approximation.

```
subplot(211)
imagesc(CV); title('Vertical Detail Image');
colormap gray;
subplot(212)
imagesc(CA); title('Lowpass Approximation');
```

Vertical Detail Image



Lowpass Approximation
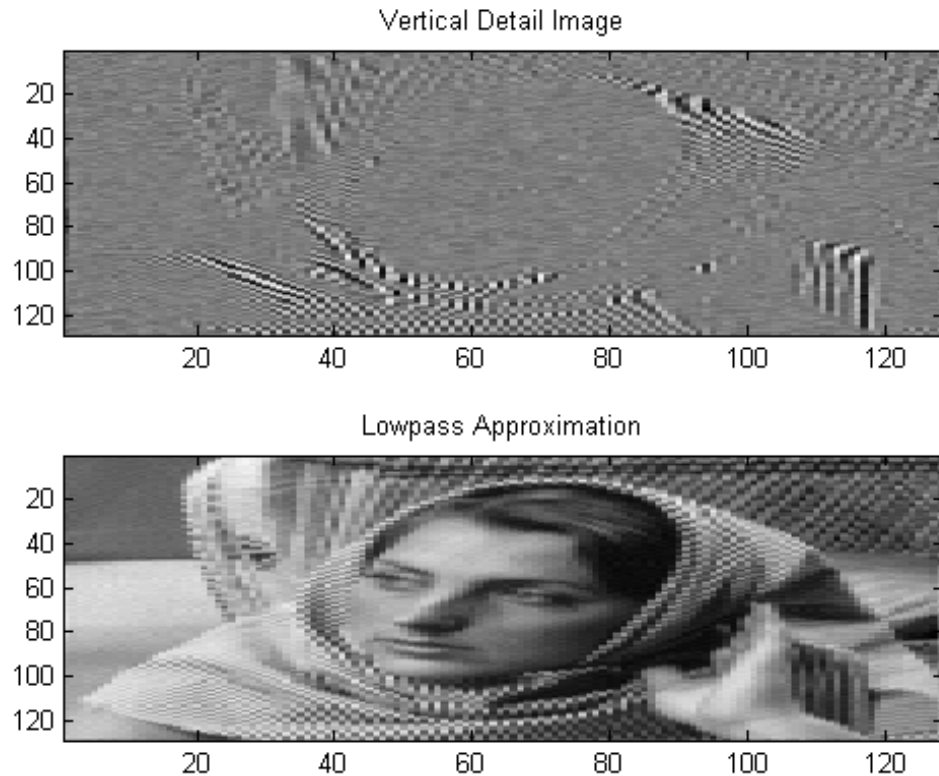


**Algorithms**     For images, there exist an algorithm similar to the one-dimensional case for two-dimensional wavelets and scaling functions obtained from one- dimensional ones by tensorial product.

This kind of two-dimensional DWT leads to a decomposition of approximation coefficients at level $j$ in four components: the approximation at level $j + 1$, and the details in three orientations (horizontal, vertical, and diagonal).

The following chart describes the basic decomposition steps for images:

## Two-Dimensional DWT

**Decomposition step**

$$CA_j \longrightarrow$$

rows: Lo_D → 2↓1 → columns Lo_D → 1↓2 → $CA_{j+1}$

columns Hi_D → 1↓2 → $CD_{j+1}^{(h)}$ horizontal

rows: Hi_D → 2↓1 → columns Lo_D → 1↓2 → $CD_{j+1}^{(v)}$ vertical

columns Hi_D → 1↓2 → $CD_{j+1}^{(d)}$ diagonal
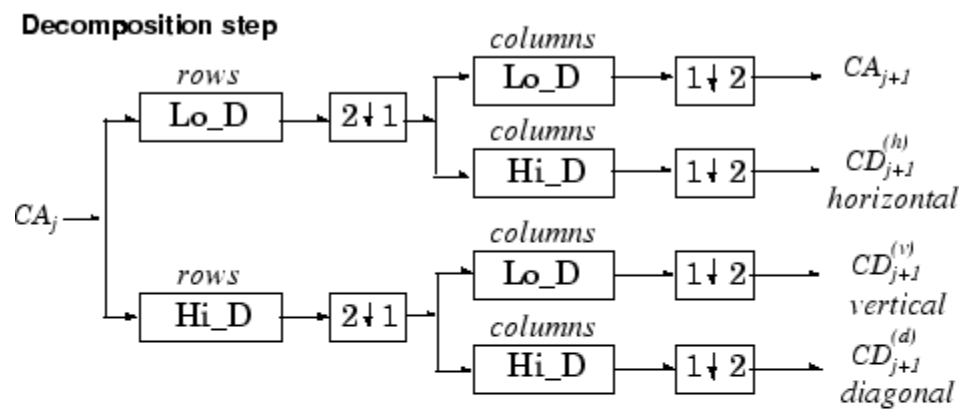
Where

2↓1  Downsample columns: keep the even indexed columns

1↓2  Downsample rows: keep the even indexed rows

rows
X  Convolve with filter X the rows of the entry

columns
X  Convolve with filter X the columns of the entry

**Initialization**   $CA_0 = s$ for the decomposition initialization

**Note** To deal with signal-end effects involved by a convolution-based algorithm, a global variable managed by dwtmode is used. This variable defines the kind of signal extension mode used. The possible options include zero-padding (used in the previous example) and symmetric extension, which is the default mode.

**References**   Daubechies, I. (1992), *Ten lectures on wavelets*, CBMS-NSF conference series in applied mathematics. SIAM Ed.

Mallat, S. (1989), "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Pattern Anal. and Machine Intell.*, vol. 11, no. 7, pp. 674–693.

Meyer, Y. (1990), *Ondelettes et opérateurs*, Tome 1, Hermann Ed. (English translation: *Wavelets and operators*, Cambridge Univ. Press. 1993.)

**See Also**     dwtmode | idwt2 | wavedec2 | waveinfo

# dwt3

| **Purpose** | Single-level discrete 3-D wavelet transform |
| --- | --- |

**Syntax**

```
WT = dwt3(X,'wname')
WT = dwt3(X,'wname','mode','ExtM')
WT = dwt3(X,W,...)
WT = dwt3(X,WF,...)
```

**Description**    dwt3 performs a single-level three-dimensional wavelet decomposition using either a particular wavelet ('*wname*') or the wavelet decomposition and reconstruction filters you specify. The decomposition also uses the specified DWT extension mode (see dwtmode).

WT = dwt3(X,'*wname*') returns the 3-D wavelet transform of the 3-D array X. '*wname*' is a string containing the wavelet name. The default extension mode is 'sym'. For more information on *wname*, see wfilters.

WT = dwt3(X,'*wname*','mode','*ExtM*') uses the extension mode '*ExtM*'.

WT is a structure with the following fields shown in the table.

| sizeINI | Size of the three-dimensional array X. |
| --- | --- |
| mode | Name of the wavelet transform extension mode. |
| filters | Structure with four fields: LoD, HiD, LoR, HiR, which are the filters used for DWT. |
| dec | 2 x 2 x 2 cell array containing the coefficients of the decomposition. <br><br> dec{i,j,k}, i,j,k = 1 or 2 contains the coefficients obtained by low-pass filtering (for i or j or k = 1) or high-pass filtering (for i or j or k = 2) |

WT = dwt3(X,W,...) specify three wavelets, one for each direction. W = {'wname1','wname2','wname3'} or W is a structure with 3 fields 'w1', 'w2', 'w3' containing strings that are the names of wavelets.

WT = dwt3(X,WF,...) specify four filters, two for decomposition, and two for reconstructionm or 3 x 4 filters (one quadruplet by direction). WF is either a cell array (1 x 4) or (3 x 4) : {LoD,HiD,LoR,HiR} or a structure with the four fields 'LoD','HiD','LoR','HiR'.

**Examples**

```
% Define the original 3-D data.
X = reshape(1:64,4,4,4)

X(:,:,1) =

     1     5     9    13
     2     6    10    14
     3     7    11    15
     4     8    12    16

X(:,:,2) =

    17    21    25    29
    18    22    26    30
    19    23    27    31
    20    24    28    32

X(:,:,3) =

    33    37    41    45
    34    38    42    46
    35    39    43    47
    36    40    44    48

X(:,:,4) =

    49    53    57    61
    50    54    58    62
    51    55    59    63
    52    56    60    64

% Perform single level decomposition of X using db1.
```

```
wt = dwt3(X,'db1')

wt =

    sizeINI: [4 4 4]
    filters: [1x1 struct]
       mode: 'sym'
        dec: {2x2x2 cell}

% Decompose X using db2.
[LoD,HiD,LoR,HiR] = wfilters('db2');
wt = dwt3(X,{LoD,HiD,LoR,HiR})

wt =

    sizeINI: [4 4 4]
    filters: [1x1 struct]
       mode: 'sym'
        dec: {2x2x2 cell}


% Decompose X using different wavelets, one for
% each orientation (db1, db2 and again db1).
WS = struct('w1','db1','w2','db2','w3','db1');
wt = dwt3(X,WS,'mode','per')

wt =

    sizeINI: [4 4 4]
    filters: [1x1 struct]
       mode: 'per'
        dec: {2x2x2 cell}

WF = wt.filters;

% Decompose X using the filters given by WF and
% set the extension mode to symmetric.
```

```
wtBIS = dwt3(X,WF,'mode','sym')

wtBIS =

    sizeINI: [4 4 4]
    filters: [1x1 struct]
       mode: 'sym'
        dec: {2x2x2 cell}
```

**See Also**      dwtmode | idwt3 | wavedec3 | waverec3 | waveinfo | wfilters

# dwtmode

**Purpose**     Discrete wavelet transform extension mode

**Syntax**      ST = dwtmode
                ST = dwtmode('status')
                dwtmode('*mode*')

**Description**  dwtmode sets the signal or image extension mode for discrete wavelet
                and wavelet packet transforms. The extension modes represent
                different ways of handling the problem of border distortion in signal
                and image analysis. For more information, see "Border Effects" , in
                the User's Guide.

                dwtmode or dwtmode('status') display the current mode.

                ST = dwtmode or ST = dwtmode('status') display and returns in ST
                the current mode.

                ST = dwtmode('status','nodisp') returns in ST the current mode
                and no text (status or warning) is displayed in the MATLAB Command
                Window.

                dwtmode('*mode*') sets the DWT extension mode according to the value
                of '*mode*':

| '*mode*' | DWT Extension Mode |
|---|---|
| 'sym' or 'symh' | Symmetric-padding (half-point): boundary value symmetric replication — default mode |
| 'symw' | Symmetric-padding (whole-point): boundary value symmetric replication |
| 'asym' or 'asymh' | Antisymmetric-padding (half-point): boundary value antisymmetric replication |
| 'asymw' | Antisymmetric-padding (whole-point): boundary value antisymmetric replication |

| '*mode*'          | DWT Extension Mode                                                          |
| ----------------- | -------------------------------------------------------------------------- |
| 'zpd'             | Zero-padding                                                               |
| 'spd' or 'sp1'    | Smooth-padding of order 1 (first derivative interpolation at the edges)    |
| 'sp0'             | Smooth-padding of order 0 (constant extension at the edges)                |
| 'ppd'             | Periodic-padding (periodic extension at the edges)                         |

For more information on symmetric extension modes see "References".

The DWT associated with these five modes is slightly redundant. But, the IDWT ensures a perfect reconstruction for any of the five previous modes whatever is the extension mode used for DWT.

dwtmode('per') sets the DWT mode to periodization.

This mode produces the smallest length wavelet decomposition. But, the extension mode used for IDWT must be the same to ensure a perfect reconstruction.

Using this mode, dwt and dwt2 produce the same results as the obsolete functions dwtper and dwtper2, respectively.

All functions and GUI tools involving the DWT (1-D & 2-D) or Wavelet Packet transform (1-D & 2-D) use the specified DWT extension mode.

dwtmode updates a global variable allowing the use of these six signal extensions. The extension mode should only be changed using this function. Avoid changing the global variable directly.

The default mode is loaded from the file DWTMODE.DEF (in the current path) if it exists. If not, the file DWTMODE.CFG (in the toolbox/wavelet/wavelet folder) is used.

dwtmode('save',MODE) saves MODE as the new default mode in the file DWTMODE.DEF (in the current folder). If a file with the same name already exists in the current folder, it is deleted before saving.

dwtmode('save') is equivalent to dwtmode('save',CURRENTMODE).

In these last two cases, the new default mode saved in the file
DWTMODE.DEF will be active as default mode in the next MATLAB
session.

**Examples**

```
% If the DWT extension mode global variable does not
% exist, default is Symmetrization.
clear global
dwtmode

****************************************
**  DWT Extension Mode: Symmetrization  **
****************************************


% Display current DWT signal extension mode.
dwtmode

****************************************
**  DWT Extension Mode: Symmetrization  **
****************************************
% Change to Periodization extension mode.
dwtmode('per')

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!  WARNING: Change DWT Extension Mode  !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

****************************************
**  DWT Extension Mode: Periodization  **
****************************************


% Display current DWT signal extension mode.
dwtmode

****************************************
**  DWT Extension Mode: Periodization  **
```

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

---

**Note** You should change the extension mode only by using dwtmode. Avoid changing the global variable directly.

---

**References**    Strang, G.; T. Nguyen (1996), *Wavelets and filter banks*, Wellesley-Cambridge Press.

**See Also**    idwt | idwt2 | dwt | dwt2 | wextend

# dyaddown

**Purpose**      Dyadic downsampling

**Syntax**       Y = dyaddown(X,EVENODD)
                 Y = dyaddown(X)
                 Y = dyaddown(X,EVENODD,'*type*')
                 Y = dyaddown(X,'*type*',EVENODD)
                 Y = dyaddown(X)
                 Y = dyaddown(X,'*type*')
                 Y = dyaddown(X,0,'*type*')
                 Y = dyaddown(X,EVENODD)
                 Y = dyaddown(X,EVENODD,'c')

**Description**  Y = dyaddown(X,EVENODD) where *X* is a *vector*, returns a version of
                *X* that has been downsampled by 2. Whether Y contains the even- or
                odd-indexed samples of *X* depends on the value of positive integer
                EVENODD:

- If EVENODD is even, then Y(k) = X(2k).

- If EVENODD is odd,   then Y(k) = X(2k+1).

Y = dyaddown(X) is equivalent to Y = dyaddown(X,0) (even-indexed
samples).

Y = dyaddown(X,EVENODD,'*type*') or Y =
dyaddown(X,'*type*',EVENODD), where *X* is a *matrix*, returns a version
of *X* obtained by suppressing one out of two:

| Columns of *X* | If '*type*'= 'c' |
| Rows of *X* | If '*type*'= 'r' |
| Rows and columns of *X* | If '*type*'= 'm' |

according to the parameter *EVENODD*, which is as above.

If you omit the *EVENODD* or '*type*' arguments, dyaddown defaults to
EVENODD = 0 (even-indexed samples) and '*type*'= 'c' (columns).

Y = dyaddown(X) is equivalent to Y = dyaddown(X,0,'c').
Y = dyaddown(X,'*type*') is equivalent to Y = dyaddown(X,0,'*type*').

Y = dyaddown(X,EVENODD) is equivalent to Y =
dyaddown(X,EVENODD,'c').

**Examples**

```
% For a vector.
s = 1:10
s =
   1   2   3   4   5   6   7   8   9  10

dse = dyaddown(s)   % Downsample elements with even indices.
dse =
   2   4   6   8  10
% or equivalently
dse = dyaddown(s,0)
dse =
   2   4   6   8  10

dso = dyaddown(s,1) % Downsample elements with odd indices.
dso =
   1   3   5   7   9

% For a matrix.
s = (1:3)'*(1:4)
s =
   1   2   3   4
   2   4   6   8
   3   6   9  12

dec = dyaddown(s,0,'c') % Downsample columns with even indices.
dec =
   2   4
   4   8
   6  12

der = dyaddown(s,1,'r') % Downsample rows with odd indices.
der =
  1   2   3   4
  3   6   9  12
```

```
dem = dyaddown(s,1,'m')  % Downsample rows and columns
                          % with odd indices.
dem =
     1     3
     3     9
```

**References**    Strang, G.; T. Nguyen (1996), *Wavelets and Filter Banks*,
                  Wellesley-Cambridge Press.

**See Also**      dyadup

**Purpose**        Dyadic upsampling

**Syntax**
```
Y = dyadup(X,EVENODD)
Y = dyadup(X)
Y = dyadup(X,EVENODD,'type')
Y = dyadup(X,'type',EVENODD)
Y = dyadup(X)
Y = dyaddown(X,1,'c')
Y = dyadup(X,'type')
Y = dyadup(X,1,'type')
Y = dyadup(X,EVENODD)
Y = dyadup(X,EVENODD,'c')
```

**Description**    dyadup implements a simple zero-padding scheme very useful in the wavelet reconstruction algorithm.

Y = dyadup(X,EVENODD), where *X* is a *vector*, returns an extended copy of vector *X* obtained by inserting zeros. Whether the zeros are inserted as even- or odd-indexed elements of *Y* depends on the value of positive integer EVENODD:

- If *EVENODD* is even, then Y(2k 1) = X(k), Y(2k) = 0.

- If *EVENODD* is odd, then Y(2k 1) = 0, Y(2k) = X(k).

Y = dyadup(*X*) is equivalent to Y = dyadup(*X*,1) (odd-indexed samples).

Y = dyadup(*X*,*EVENODD*,'*type*') or Y = dyadup(*X*,'*type*',*EVENODD*), where *X* is a *matrix*, returns extended copies of *X* obtained by inserting

| | |
|---|---|
| Columns in *X* | If '*type*'= 'c' |
| Rows in *X* | If '*type*'= 'r' |
| Rows and columns in *X* | If '*type*'= 'm' |

according to the parameter *EVENODD*, which is as above.

# dyadup

If you omit the *EVENODD* or '*type*' arguments, dyadup defaults to
EVENODD = 1 (zeros in odd-indexed positions) and '*type*'= 'c' (insert
columns).

Y = dyadup(*X*) is equivalent to Y = dyaddown(X,1,'c').

Y = dyadup(*X*,'*type*') is equivalent to Y = dyadup(*X*,1,'*type*').
Y = dyadup(*X*,*EVENODD*) is equivalent to Y = dyadup(*X*,*EVENODD*,'c').

**Examples**
```
% For a vector.
s = 1:5
s =
    1 2 3 4 5

dse = dyadup(s) % Upsample elements at odd indices.
dse =
    0 1 0 2 0 3 0 4 0 5 0

% or equivalently
dse = dyadup(s,1)
dse =
    0 1 0 2 0 3 0 4 0 5 0

dso = dyadup(s,0) % Upsample elements at even indices.
dso =
    1 0 2 0 3 0 4 0 5

% For a matrix.
s = (1:2)'*(1:3)
s =
    1 2 3
    2 4 6

der = dyadup(s,1,'r') % Upsample rows at even indices.
der =
    0 0 0
    1 2 3
    0 0 0
```

```
                       2 4 6
                       0 0 0

      doc = dyadup(s,0,'c') % Upsample columns at odd indices.
      doc =
         1 0 2 0 3
         2 0 4 0 6
      dem = dyadup(s,1,'m') % Upsample rows and columns
                            % at even indices.
      dem =
          0    0    0    0    0    0    0
          0    1    0    2    0    3    0
          0    0    0    0    0    0    0
          0    2    0    4    0    6    0
          0    0    0    0    0    0    0

      % Using default values for dyadup and dyaddown, we have:
      % dyaddown(dyadup(s)) = s.
      s = 1:5
      s =
         1 2 3 4 5

      uds = dyaddown(dyadup(s))
      uds =
         1 2 3 4 5

      % In general reversed identity is false.
```

**References**   Strang, G.; T. Nguyen (1996), *Wavelets and Filter Banks*,
Wellesley-Cambridge Press.

**See Also**   dyaddown

# entrupd

**Purpose**      Entropy update (wavelet packet)

**Syntax**       T = entrupd(*T*,*ENT*)
                 T = entrupd(*T*,*ENT*,*PAR*)

**Description**  entrupd is a one- or two-dimensional wavelet packet utility.

T = entrupd(*T*,*ENT*) or T = entrupd(*T*,*ENT*,*PAR*) returns for a given
wavelet packet tree *T*, the updated tree using the entropy function *ENT*
with the optional parameter *PAR* (see wenergy for more information).

**Examples**
```
% The current extension mode is zero-padding (see dwtmode).

% Load signal.
load noisdopp; x = noisdopp;

% Decompose x at depth 2 with db1 wavelet packets
% using shannon entropy.
t = wpdec(x,2,'db1','shannon');

% Read entropy of all the nodes.
nodes = allnodes(t);
ent = read(t,'ent',nodes);
ent'
ent =
    1.0e+04 *
    -5.8615 -6.8204 -0.0350 -7.7901 -0.0497 -0.0205 -0.0138

% Update nodes entropy.
t = entrupd(t,'threshold',0.5);
nent = read(t,'ent');
nent'
nent =
    937 488 320 241 175 170 163
```

**See Also**     wenergy | wpdec | wpdec2

**Purpose**     Complex frequency B-spline wavelet

**Syntax**     `[PSI,X] = fbspwavf(LB,UB,N,M,FB,FC)`

**Description**     `[PSI,X] = fbspwavf(LB,UB,N,M,FB,FC)` returns values of the complex frequency B-Spline wavelet defined by the order parameter *M* (*M* is an integer such that $1 \le M$), a bandwidth parameter FB, and a wavelet center frequency *FC*.

The function `PSI` is computed using the explicit expression

`PSI(X) = (FB^0.5)*((sinc(FB*X/M).^M).*exp(2*i*pi*FC*X))`

on an *N* point regular grid in the interval `[LB,UB]`.

*FB* and *FC* must be such that `FC > 0` and `> FB > 0`.

Output arguments are the wavelet function `PSI` computed on the grid *X*.

**Examples**
```
% Set order, bandwidth and center frequency parameters.
m = 2; fb = 0.5; fc = 1;

% Set effective support and grid parameters.
lb = -20; ub = 20; n = 1000;

% Compute complex Frequency B-Spline wavelet fbsp2-0.5-1.
[psi,x] = fbspwavf(lb,ub,n,m,fb,fc);

% Plot complex Frequency B-Spline wavelet.
subplot(211)
plot(x,real(psi))
title('Complex Frequency B-Spline wavelet fbsp2-0.5-1')
xlabel('Real part'), grid
subplot(212)
plot(x,imag(psi))
xlabel('Imaginary part'), grid
```

# fbspwavf



Complex Frequency B–Spline wavelet fbsp2–0.5–1

Real part

Imaginary part
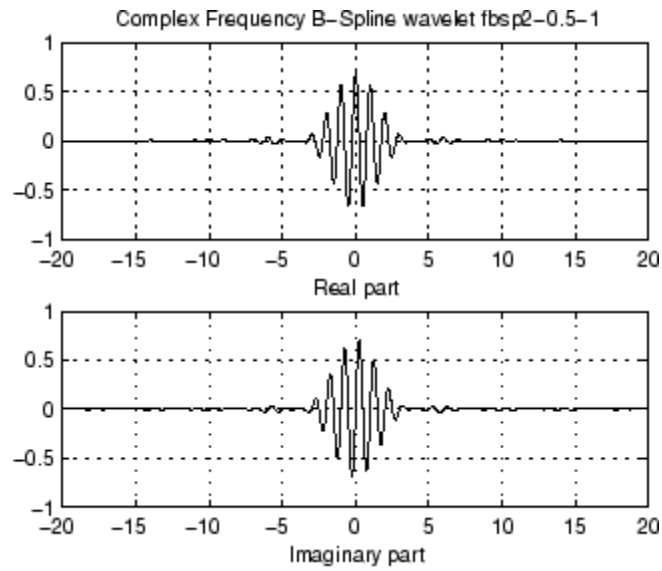
**References**     Teolis, A. (1998), *Computational signal processing with wavelets*, Birkhauser, p. 63.

**See Also**      waveinfo

**Purpose**     Transform quadruplet of filters to lifting scheme

**Syntax**      LS = filt2ls(LoD,HiD,LoR,HiR)

**Description**  LS = filt2ls(LoD,HiD,LoR,HiR) returns the lifting scheme LS
associated with the four input filters LoD, HiD, LoR, and HiR that verify
the perfect reconstruction condition.

**Examples**
```
[LoD,HiD,LoR,HiR] = wfilters('db2')

LoD =

   -0.1294    0.2241    0.8365    0.4830

HiD =

   -0.4830    0.8365   -0.2241   -0.1294

LoR =

    0.4830    0.8365    0.2241   -0.1294

HiR =

   -0.1294   -0.2241    0.8365   -0.4830

LS = filt2ls(LoD,HiD,LoR,HiR);
displs(LS);

LS = {...
'd'            [ -1.73205081]                [0]
'p'            [ -0.06698730  0.43301270]    [1]
'd'            [  1.00000000]                [-1]
[  1.93185165] [  0.51763809]                []
};
```

```
LSref = liftwave('db2');
displs(LSref);

LSref = {...
'd'              [ -1.73205081]                    [0]
'p'              [ -0.06698730   0.43301270]  [1]
'd'              [  1.00000000]                    [-1]
[  1.93185165]  [  0.51763809]                  []
};
```

**See Also**    ls2filt | lsinfo

# gauswavf

**Purpose**        Gaussian wavelet

**Syntax**
```
[PSI,X] = gauswavf(LB,UB,N,P)
[PSI,X] = gauswavf(LB,UB,N)
[PSI,X] = gauswavf(LB,UB,N,1)
```

**Description**    [PSI,X] = gauswavf(LB,UB,N,P) returns values of the *P*-th derivative
                   of the Gaussian function on an *N* point regular grid for the interval
                   [LB,UB]. *Cp* is such that the 2-norm of the *P*-th derivative of *F* is equal
                   to 1.

                   For P > 8, Symbolic Math Toolbox software is required.

                   Output arguments are the wavelet function PSI computed on the grid X.

                   [PSI,X] = gauswavf(LB,UB,N) is equivalent to
                   [PSI,X] = gauswavf(LB,UB,N,1).

                   These wavelets have an effective support of [-5 5].

**Examples**       % Set effective support and grid parameters.
                   lb = -5; ub = 5; n = 1000;

                   % Compute Gaussian wavelet of order 8.
                   [psi,x] = gauswavf(lb,ub,n,8);

                   % Plot Gaussian wavelet of order 8.
                   plot(x,psi),
                   title('Gaussian wavelet of order 8'), grid

# gauswavf



Gaussian wavelet of order 8
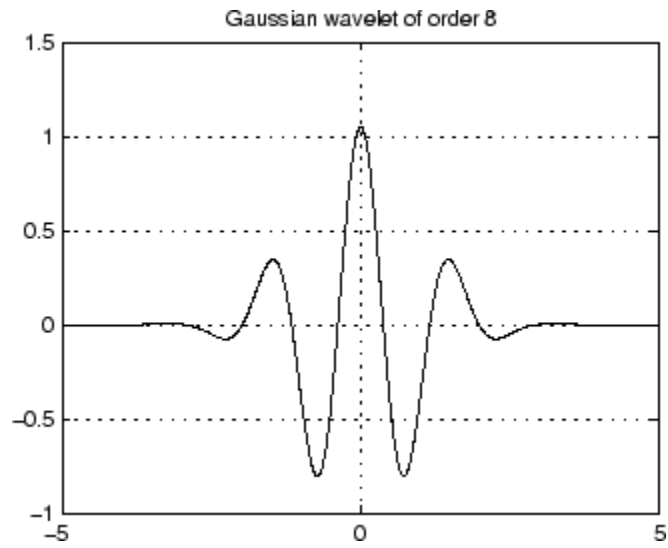
**See Also**     `waveinfo`

| | |
|---|---|
| **Purpose** | WPTREE contents |

**Syntax**  `[FieldValue1,FieldValue2, ...] =`
`get(T,'`*`FieldName1`*`','`*`FieldName2`*`',`
`    ...)`
`[FieldValue1,FieldValue2, ...] = get(`*`T`*`)`

**Description**  `[FieldValue1,FieldValue2, ...]   =`
`get(T,'`*`FieldName1`*`','`*`FieldName2`*`', ...)` returns the content of
the specified fields for the WPTREE object T.

For the fields that are objects or structures, you can get the subfield
contents, giving the name of these subfields as '*`FieldName`*' values.
(See "Examples" below.)

`[FieldValue1,FieldValue2, ...]   = get(`*`T`*`)` returns all the field
contents of the tree *`T`*.

The valid choices for '*`FieldName`*' are

| | |
|---|---|
| `'dtree'` | DTREE parent object |
| `'wavInfo'` | Structure (wavelet information) |

The fields of the wavelet information structure, `'wavInfo'`, are also
valid for '*`FieldName`*':

| | |
|---|---|
| `'wavName'` | Wavelet name |
| `'Lo_D'` | Low Decomposition filter |
| `'Hi_D'` | High Decomposition filter |
| `'Lo_R'` | Low Reconstruction filter |
| `'Hi_R'` | High Reconstruction filter |

| | |
|---|---|
| `'entInfo'` | Structure (entropy information) |

The fields of the entropy information structure, 'entInfo', are also valid for '*FieldName*':

| | |
|---|---|
| 'entName' | Entropy name |
| 'entPar' | Entropy parameter |

Or fields of DTREE parent object:

| | |
|---|---|
| 'ntree' | NTREE parent object |
| 'allNI' | All nodes information |
| 'terNI' | Terminal nodes information |

Or fields of NTREE parent object:

| | |
|---|---|
| 'wtbo' | WTBO parent object |
| 'order' | Order of the tree |
| 'depth' | Depth of the tree |
| 'spsch' | Split scheme for nodes |
| 'tn' | Array of terminal nodes of the tree |

Or fields of WTBO parent object:

| | |
|---|---|
| 'wtboInfo' | Object information |
| 'ud' | Userdata field |

**Examples**
```
% Compute a wavelet packets tree
x = rand(1,1000);
t = wpdec(x,2,'db2');
```

```
o = get(t,'order');
[o,tn] = get(t,'order','tn');
[o,allNI,tn] = get(t,'order','allNI','tn');
[o,wavInfo,allNI,tn] = get(t,'order','wavInfo','allNI','tn');
[o,tn,Lo_D,EntName] = get(t,'order','tn','Lo_D','EntName');
[wo,nt,dt] = get(t,'wtbo','ntree','dtree');
```

**See Also**      disp | read | set | write

# icwtft

**Purpose**     Inverse CWT

**Syntax**
```
xrec = icwtft(cwtstruct)
xrec = icwtft(cwtstruct,'plot')
xrec = icwtft(cwtstruct,'signal',SIG,'plot')
```

**Description**     `xrec = icwtft(cwtstruct)` returns the inverse continuous wavelet transform of the CWT coefficients contained in the `cfs` field of the structure array `cwtstruct`. Obtain the structure array `cwtstruct` as the output of `cwtft`.

`xrec = icwtft(cwtstruct,'plot')` plots the reconstructed signal.

`xrec = icwtft(cwtstruct,'signal',SIG,'plot')` places a radio button in the bottom left corner of the plot. Enabling the radio button superimposes the plot of the input signal `SIG` on the plot of the reconstructed signal. By default the radio button is not enabled and only the reconstructed signal is plotted.

**Input Arguments**

**cwtstruct**

Structure array containing six fields.

- `dt` — The sampling period
- `cfs` — CWT coefficient matrix
- `scales` — Vector of scales
- `wav` — Analyzing wavelet used in the CWT
- `omega` — Angular frequencies used in the Fourier transform
- `meanSig` — Mean of the analyzed signal

`cwtstruct` is the output of `cwtft`.

**Output Arguments**

**xrec**

Reconstructed signal

**Definitions**    **Inverse CWT**

icwtft computes the inverse CWT based on a discretized version of the single integral formula due to Morlet. The Wavelet Toolbox Getting Started Guide contains a brief description of the theoretical foundation for the single integral formula in "Inverse Continuous Wavelet Transform". The discretized version of this integral is presented in [5]

**Examples**    Compute the CWT and inverse CWT of two sinusoids with disjoint support.

```
N = 1024;
t = linspace(0,1,N);
y = sin(2*pi*8*t).*(t<=0.5)+sin(2*pi*16*t).*(t>0.5);
dt = 0.05;
s0 = 2*dt;
ds = 0.4875;
NbSc = 20;
wname = 'morl';
sig = {y,dt};
sca = {s0,ds,NbSc};
wave = {wname,[]};
cwtsig = cwtft(sig,'scales',sca,'wavelet',wave);

% Compute inverse CWT and plot reconstructed signal with original
sigrec = icwtft(cwtsig,'signal',sig,'plot');
```

Select the radio button in the bottom left corner of the plot.
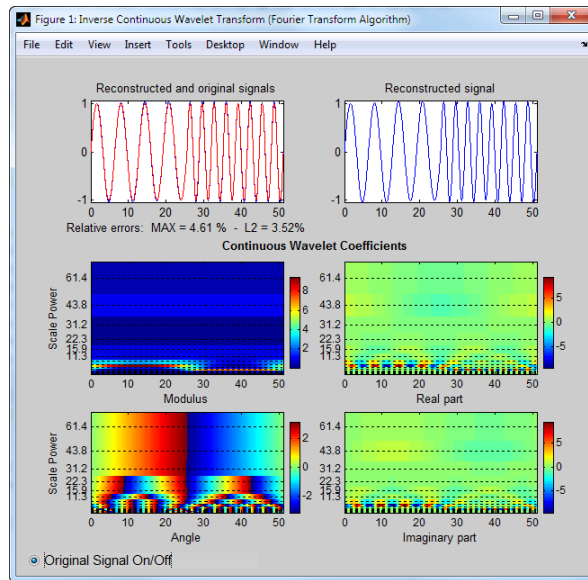
Use the inverse CWT to approximate a trend in a time series. Construct a time series consisting of a polynomial trend, a sinewave (oscillatory component), and additive white Gaussian noise. Obtain the CWT of the input signal and use the inverse CWT based on only the coarsest scales to reconstruct an approximation to the trend. To obtain an accurate approximation based on select scales use the default power of two spacing for the scales in the continuous wavelet transform. See cwtft for details.

```
t = linspace(0,1,1e3);
% Polynomial trend
x = t.^3-t.^2;
% Periodic term
x1 = 0.25*cos(2*pi*250*t);
% Reset random number generator for reproducible results
rng default
y = x+x1+0.1*randn(size(t));
```

```
% Obtain CWT of input time series
cwty = cwtft({y,0.001},'wavelet','morl');
% Zero out all but the coarsest scale CWT coefficients
cwty.cfs(1:16,:) = 0;
% Reconstruct a signal approximation based on the coarsest scales
xrec = icwtft(cwty);
plot(t,y,'k'); hold on;
xlabel('Seconds'); ylabel('Amplitude');
plot(t,x,'b','linewidth',2);
plot(t,xrec,'r','linewidth',2);
legend('Original Signal','Polynomial Trend','Inverse CWT Approximation
figure
plot(t,x,'b'); hold on;
xlabel('Seconds'); ylabel('Amplitude');
plot(t,xrec,'r','linewidth',2);
legend('Polynomial Trend','Inverse CWT Approximation');
```
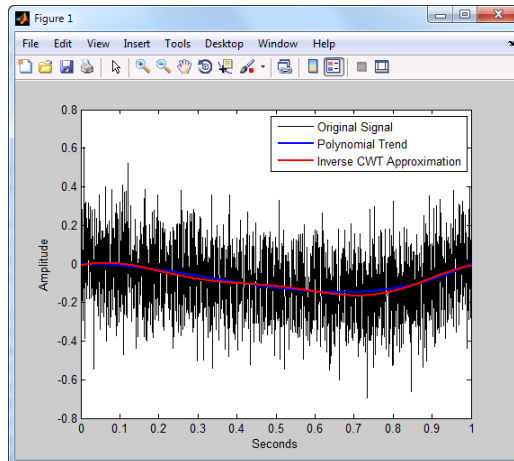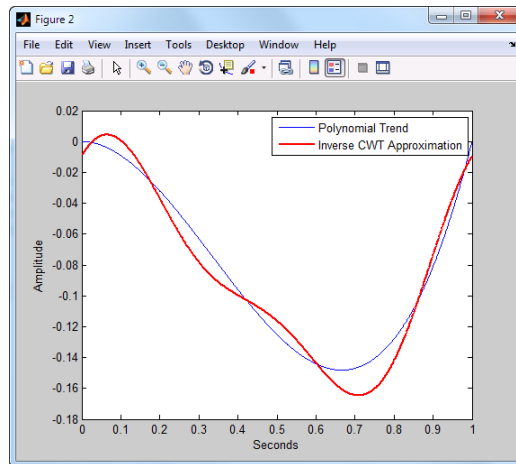
You can also use the following syntax to plot the approximation. Select the radio button to view the original polynomial trend superimposed on the wavelet approximation.

```
% Input the polynomial trend as the value of 'signal'
xrec = icwtft(cwty,'signal',x,'plot');
```

**References**    [1] Daubechies, I. *Ten Lectures on Wavelets*, Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 1992.

[2] Farge, M. "Wavelet Transforms and Their Application to Turbulence", *Ann. Rev. Fluid. Mech.*, 1992, 24, 395–457.

[3] Mallat, S. *A Wavelet Tour of Signal Processing*, San Diego, CA: Academic Press, 1998.

[4] Sun,W. "Convergence of Morlet's Reconstruction Formula", *preprint*, 2010.

[5] Torrence, C. and G.P. Compo "A Practical Guide to Wavelet Analysis", *Bull. Am. Meteorol. Soc.*, 79, 61–78, 1998.

**See Also**    cwt | cwtft

# icwtft

**How To**
- "Continuous Wavelet Transform"
- "DFT-Based Continuous Wavelet Transform"
- "Inverse Continuous Wavelet Transform"

**Purpose**     Inverse continuous wavelet transform (CWT) for linearly spaced scales

**Syntax**
```
xrec = icwtlin(cwtstruct)
xrec = icwtlin(wav,meanSIG,cfs,scales,dt)
xrec = icwtin(...,'plot')
xrec = icwtlin (...,'signal',SIG,'plot')
xrec = icwtlin(...,Name,Value)
```

**Description**     `xrec = icwtlin(cwtstruct)` returns the inverse continuous wavelet transform (CWT) of the CWT coefficients obtained at linearly-spaced scales.

---

**Note** To use `icwtlin` you must:

• Use linearly-spaced scales in the CWT. `icwtlin` does not verify that the scales are linearly-spaced.

• Use one of the supported wavelets. See "Input Arguments" on page 1-200 for a list of supported wavelets.

---

`xrec = icwtlin(wav,meanSIG,cfs,scales,dt)` returns the inverse CWT of the coefficients in `cfs`. The inverse CWT is obtained using the wavelet `wav`, the linearly spaced scales `scales`, the sampling period `dt`, and the mean signal value `meanSig`.

`xrec = icwtin(...,'plot')` plots the reconstructed signal `xrec` along with the CWT coefficients and CWT moduli. If the analyzing wavelet is complex-valued, the plot includes the real and imaginary parts of the CWT coefficients.

`xrec = icwtlin (...,'signal',SIG,'plot')` places a radio button in the bottom-left corner of the plot. Enabling the radio button superimposes the plot of the input signal `SIG` on the plot of the reconstructed signal. `SIG` can be a structure array, a cell array, or a vector. If `SIG` is a structure array, there must be two fields: `val` and `period`. The `val` field contains the signal and the `period` field contains

the sampling period. If `SIG` is a cell array, `SIG{1}` contains the signal and `SIG{2}` is the sampling period.

`xrec = icwtlin(...,Name,Value)` returns the inverse CWT transform with additional options specified by one or more `Name,Value` pair arguments.

**Input Arguments**

**cwtstruct**

A structure array that is the output of `cwtft` or constructed from the output of `cwt`. If you obtain `cwtstruct` from `cwtft`, the structure array contains six fields:

- `cfs` — CWT coefficient matrix

- `scales` — Vector of linearly spaced scales. The scale vector must be linearly-spaced to ensure accurate reconstruction. `icwtlin` does not check that the spacing of your scale vector is linear.

- `wav` — Analyzing wavelet. `icwtlin` uses this wavelet as the reconstruction wavelet. The supported wavelets are:

  - `'dog'` — An *m*-th order derivative of Gaussian wavelet where *m* is a positive even integer

  - `'morl'` — Analytic Morlet wavelet

  - `'morlex'` — Nonanalytic Morlet wavelet

  - `'morl0'` — Nonanalytic Morlet wavelet with exact zero mean

  - `'mexh'` — Mexican-hat wavelet. This argument represents a special case of the derivative of Gaussian wavelet with *m*=2.

  - `'paul'` — Paul wavelet

- `omega` — Angular frequencies used in the Fourier transform in radians/sample

- `MeanSIG` — Signal mean

- `dt` — Sampling period in seconds

If you create `cwtstruct` from the output of `cwt`, `cwtstruct` contains all of the preceding fields except `omega`.

Using `cwt` to obtain the CWT coefficients, the valid analyzing wavelets are:

- Coiflets — `'coif1'`,`'coif2'` ,`'coif3'` ,`'coif4'`, `'coif5'`

- Biorthogonal wavelets — `'bior2.2'`, `'bior2.4'`, `'bior2.6'`, `'bior2.8'`, `'bior4.4'`, `'bior5.5'`, `bior6.8`

- Reverse biorthogonal wavelets — `'rbio2.2'`, `'rbio2.4'`, `'rbio2.6'`, `'rbio2.8'`, `'rbio4.4'`, `'rbio5.5'`, `'rbio6.8'`

- Complex Gaussian wavelets — `'cgau2'`, `'cgau4'`, `'cgau6'`, `'cgau8'`

### Name-Value Pair Arguments

**'IdxSc'**

Vector of scales to use in the signal reconstruction. Specifying a subset of scales results in a scale-localized approximation of the analyzed signal.

**Output Arguments**

**xrec**

Reconstructed signal. Signal approximation based on the input CWT coefficient matrix, analyzing wavelet, selected scales, and sampling period.

The purpose of the CWT inversion algorithm is not to produce a perfect reconstruction of the input signal. The inversion preserves time and scale-localized features in the reconstructed signal. The amplitude scaling in the reconstructed signal, however, can be significantly different. This difference in scaling can occur whether or not you use all the CWT coefficients in the inversion.

**Examples**

Compute the inverse CWT of a sum of sine waves with disjoint support.

```
% Define the signal
N = 100;
```

```
t = linspace(0,1,N);
Y = sin(8*pi*t).*(t<=0.5) + sin(16*pi*t).*(t>0.5) ;

% Define parameters before analysis
dt = 0.001;
maxsca = 1; s0 = 2*dt; ds = 2*dt;
scales = s0:ds:maxsca;
wname = 'morl';
SIG = {Y,dt};
WAV = {wname,[]};

% Compute the CWT using cwtft with linear scales
cwtS = cwtft(SIG,'scales',scales,'wavelet',WAV);
% Compute inverse CWT using linear scales
Yrec = icwtlin(cwtS,'Signal',Y,'plot');
```
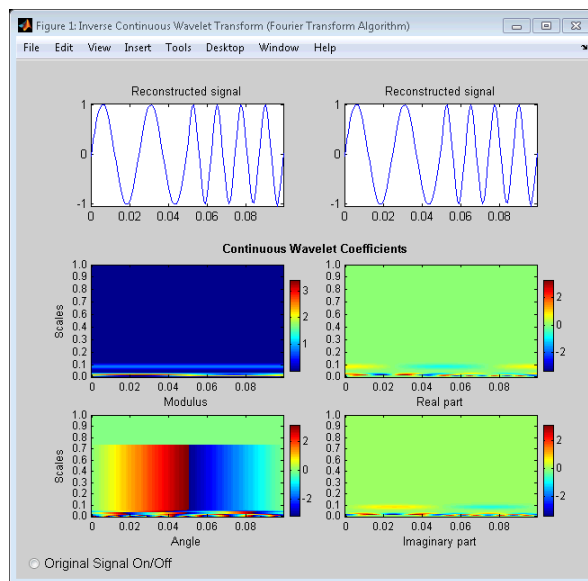
Reconstruct an approximation to a noisy Doppler signal based on thresholded coefficients. Use the universal threshold. Assume the sampling period is 0.05 seconds.

```
load noisdopp;
Y = noisdopp;
N = length(Y);

% Define parameters before analysis
% Assume sampling period is 0.05
dt = 0.05;
maxsca = 100; s0 = 2*dt; ds = 4*dt;
scales = s0:ds:maxsca;
wname = 'morl';
SIG = {Y,dt};
WAV = {wname,[]};

% Compute CWT
cwtS = cwtft(SIG,'scales',scales,'wavelet',WAV,'plot');

% Select subset of coefficients
cwtS1 = cwtS;
Hfreq = cwtS.cfs(1:10,:);
% Set threshold
thr = sqrt(2*log(N))*median(abs(Hfreq(:)))/0.6745;
newCFS = cwtS.cfs;
% Set coefficients smaller than threshold in absolute value to 0
newCFS(abs(newCFS)<thr) = 0;
cwtS1.cfs = newCFS;


% Reconstruction from the modified structure
YRDen = icwtlin(cwtS1,'signal',Y,'plot');
```

Enable the **Reconstructed Signal On/Off** radio button in the bottom-left corner.
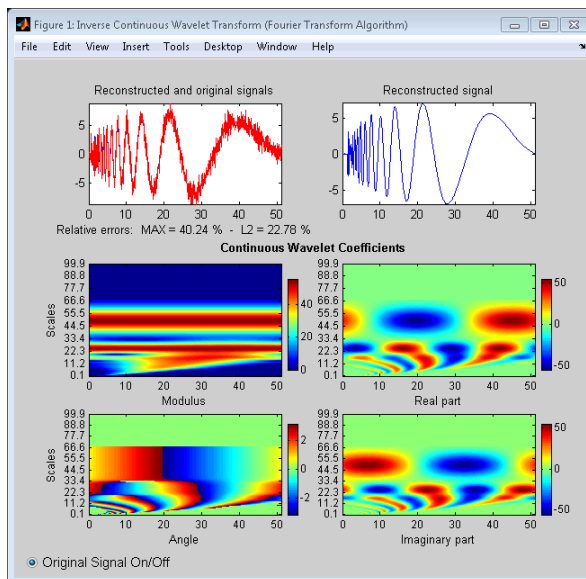
# icwtlin



**Algorithms**   See [4] for a description of the inverse CWT algorithm for linearly
spaced scales. The icwtlin function uses heuristic scaling factors for
the analyzing wavelets. These scaling factors can result in significant
differences in the amplitude scaling of the reconstructed signal.

**References**   [1] Daubechies, I. *Ten Lectures on Wavelets*, Philadelphia, PA: Society
for Industrial and Applied Mathematics (SIAM), 1992.

[2] Farge, M. "Wavelet Transforms and Their Application to
Turbulence", *Ann. Rev. Fluid. Mech.*, 1992, 24, 395–457.

[3] Mallat, S. *A Wavelet Tour of Signal Processing*, San Diego, CA:
Academic Press, 1998.

[4] Sun,W. "Convergence of Morlet's Reconstruction Formula", *preprint*,
2010.

[5] Torrence, C. and G.P. Compo. "A Practical Guide to Wavelet Analysis", *Bull. Am. Meteorol. Soc.*, 79, 61–78, 1998.

**Alternatives**
- `icwtft` — Computes the inverse for the CWT obtained using `cwtft` with logarithmically spaced scales. If you use linearly spaced scales in `cwtft`, or you obtain the CWT with `cwt`, use `icwtlin` to compute the inverse.

**See Also**    icwtft | cwtft | cwt

**How To**
- "Continuous Wavelet Transform"
- "DFT-Based Continuous Wavelet Transform"
- "Inverse Continuous Wavelet Transform"

# idddtree

| | |
|---|---|
| **Purpose** | Inverse dual-tree and double-density 1-D wavelet transform |
| **Syntax** | `xrec = idddtree(wt)` |

**Description**   `xrec = idddtree(wt)` returns the inverse wavelet transform of the wavelet decomposition (analysis filter bank), `wt`. `wt` is the output of `dddtree`.

**Input Arguments**

**wt - Wavelet transform**
structure

Wavelet transform, returned as a structure from `dddtree` with these fields:

**type - Type of wavelet decomposition (filter bank)**
`'dwt'` | `'ddt'` | `'cplxdt'` | `'cplxdddt'`

Type of wavelet decomposition (filter bank), specified as one of `'dwt'`, `'ddt'`, `'cplxdt'`, or `'cplxdddt'`. The type,`'dwt'`, gives a critically sampled discrete wavelet transform. The other types are oversampled wavelet transforms. `'ddt'` is a double-density wavelet transform, `'cplxdt'` is a dual-tree complex wavelet transform, and `'cplxdddt'` is a double-density dual-tree complex wavelet transform.

**level - Level of wavelet decomposition**
positive integer

Level of wavelet decomposition, specified as a positive integer.

**filters - Decomposition (analysis) and reconstruction (synthesis) filters**
structure

Decomposition (analysis) and reconstruction (synthesis) filters, specified as a structure with these fields:

**Fdf - First-stage analysis filters**
matrix | cell array

First-stage analysis filters, specified as an *N*-by-2 or *N*-by-3 matrix for single-tree wavelet transforms, or a cell array of two *N*-by-2 or *N*-by-3 matrices for dual-tree wavelet transforms. The matrices are *N*-by-3 for the double-density wavelet transforms. For an *N*-by-2 matrix, the first column of the matrix is the scaling (lowpass) filter and the second column is the wavelet (highpass) filter. For an *N*-by-3 matrix, the first column of the matrix is the scaling (lowpass) filter and the second and third columns are the wavelet (highpass) filters. For the dual-tree transforms, each element of the cell array contains the first-stage analysis filters for the corresponding tree.

### Df - Analysis filters for levels > 1
matrix | cell array

Analysis filters for levels > 1, specified as an *N*-by-2 or *N*-by-3 matrix for single-tree wavelet transforms, or a cell array of two *N*-by-2 or *N*-by-3 matrices for dual-tree wavelet transforms. The matrices are *N*-by-3 for the double-density wavelet transforms. For an *N*-by-2 matrix, the first column of the matrix is the scaling (lowpass) filter and the second column is the wavelet (highpass) filter. For an *N*-by-3 matrix, the first column of the matrix is the scaling (lowpass) filter and the second and third columns are the wavelet (highpass) filters. For the dual-tree transforms, each element of the cell array contains the analysis filters for the corresponding tree.

### Frf - First-level reconstruction filters
matrix | cell array

First-level reconstruction filters, specified as an *N*-by-2 or *N*-by-3 matrix for single-tree wavelet transforms, or a cell array of two *N*-by-2 or *N*-by-3 matrices for dual-tree wavelet transforms. The matrices are *N*-by-3 for the double-density wavelet transforms. For an *N*-by-2 matrix, the first column of the matrix is the scaling (lowpass) filter and the second column is the wavelet (highpass) filter. For an *N*-by-3 matrix, the first column of the matrix is the scaling (lowpass) filter and the second and third columns are the wavelet (highpass) filters. For the dual-tree transforms, each element of the cell array contains the first-stage synthesis filters for the corresponding tree.

### Rf - Reconstruction filters for levels > 1

matrix | cell array

Reconstruction filters for levels > 1, specified as an *N*-by-2 or *N*-by-3 matrix for single-tree wavelet transforms, or a cell array of two *N*-by-2 or *N*-by-3 matrices for dual-tree wavelet transforms. The matrices are N-by-3 for the double-density wavelet transforms. For an *N*-by-2 matrix, the first column of the matrix is the scaling (lowpass) filter and the second column is the wavelet (highpass) filter. For an *N*-by-3 matrix, the first column of the matrix is the scaling (lowpass) filter and the second and third columns are the wavelet (highpass) filters. For the dual-tree transforms, each element of the cell array contains the synthesis filters for the corresponding tree.

### cfs - Wavelet transform coefficients

cell array of matrices

Wavelet transform coefficients, specified as a 1-by-(`level`+1) cell array of matrices. The size and structure of the matrix elements of the cell array depend on the type of wavelet transform as follows:

- `'dwt'` — `cfs{j}`

  - `j = 1,2,...,` `level` is the level.

  - `cfs{level+1}` are the lowpass, or scaling, coefficients.

- `'ddt'` — `cfs{j}(:,:,k)`

  - `j = 1,2,...,` `level` is the level.

  - `k = 1,2` is the wavelet filter.

  - `cfs{level+1}(:,:)` are the lowpass, or scaling, coefficients.

- `'cplxdt'` — `cfs{j}(:,:,m)`

  - `j = 1,2,...,` `level` is the level.

  - `m = 1,2` are the real and imaginary parts.

  - `cfs{level+1}(:,:)` are the lowpass, or scaling, coefficients.

- `'cplxdddt'` — `cfs{j}(:,:,k,m)`

- $j = 1,2,...,$ `level` is the level.

- $k = 1,2$ is the wavelet filter.

- $m = 1,2$ are the real and imaginary parts.

- `cfs{level+1}(:,:)` are the lowpass, or scaling, coefficients.

**Output Arguments**

**xrec - Synthesized 1-D signal**
vector

Synthesized 1-D signal, returned as a vector. The row or column orientation of `xrec` depends on the row or column orientation of the 1-D signal input to `dddtree`.

**Data Types**
double

**Examples**

**Perfect Reconstruction Using the Dual-Tree Double-Density Wavelet Filter Bank**

Demonstrate perfect reconstruction of a signal using a dual-tree double-density wavelet transform.

Load the noisy Doppler signal. Obtain the dual-tree double-density wavelet transform down to level 5. Invert the transform and demonstrate perfect reconstruction.

```
load noisdopp;
wt = dddtree('cplxdddt',noisdopp,5,'FSdoubledualfilt','doubledualfilt
xrec = idddtree(wt);
max(abs(noisdopp-xrec))
```

**See Also**

`dddtree` | `dddtreecfs` | `plotdt`

**Related Examples**

- "Analytic Wavelets Using the Dual-Tree Wavelet Transform"

**Concepts**

- "Critically Sampled and Oversampled Wavelet Filter Banks"

# idddtree2

**Purpose**        Inverse dual-tree and double-density 2-D wavelet transform

**Syntax**         xrec = idddtree2(wt)

**Description**    xrec = idddtree2(wt) returns the inverse wavelet transform of
                   the 2-D decomposition (analysis filter bank), wt. wt is the output of
                   dddtree2.

**Input
Arguments**

**wt - Wavelet transform**
structure

Wavelet transform, returned as a structure from dddtree2 with these
fields:

**type - Type of wavelet decomposition (filter bank)**
'dwt' | 'ddt' | 'realdt' | 'cplxdt' | 'realdddt' | 'cplxdddt'

Type of wavelet decomposition (filter bank), specified as one of 'dwt',
'ddt', 'realdt', 'cplxdt', 'realdddt', or 'cplxdddt'. 'dwt' is
the critically sampled DWT. 'ddt' produces a double-density wavelet
transform with one scaling and two wavelet filters for both row and
column filtering. 'realdt' and 'cplxdt' produce oriented dual-tree
wavelet transforms consisting of two and four separable wavelet
transforms. 'realdddt' and 'cplxdddt' produce double-density
dual-tree wavelet transforms consisting of two and four separable
wavelet transforms.

**level - Level of the wavelet decomposition**
positive integer

Level of the wavelet decomposition, specified as a positive integer.

**filters - Decomposition (analysis) and reconstruction (synthesis)
filters**
structure

Decomposition (analysis) and reconstruction (synthesis) filters, specified
as a structure with these fields:

### Fdf - First-stage analysis filters
matrix | cell array

First-stage analysis filters, specified as an $N$-by-2 or $N$-by-3 matrix for single-tree wavelet transforms, or a 1-by-2 cell array of two $N$-by-2 or $N$-by-3 matrices for dual-tree wavelet transforms. The matrices are $N$-by-3 for the double-density wavelet transforms. For an $N$-by-2 matrix, the first column of the matrix is the scaling (lowpass) filter and the second column is the wavelet (highpass) filter. For an $N$-by-3 matrix, the first column of the matrix is the scaling (lowpass) filter and the second and third columns are the wavelet (highpass) filters. For the dual-tree transforms, each element of the cell array contains the first-stage analysis filters for the corresponding tree.

### Df - Analysis filters for levels > 1
matrix | cell array

Analysis filters for levels > 1, specified as an $N$-by-2 or $N$-by-3 matrix for single-tree wavelet transforms, or a 1-by-2 cell array of two $N$-by-2 or $N$-by-3 matrices for dual-tree wavelet transforms. The matrices are $N$-by-3 for the double-density wavelet transforms. For an $N$-by-2 matrix, the first column of the matrix is the scaling (lowpass) filter and the second column is the wavelet (highpass) filter. For an $N$-by-3 matrix, the first column of the matrix is the scaling (lowpass) filter and the second and third columns are the wavelet (highpass) filters. For the dual-tree transforms, each element of the cell array contains the analysis filters for the corresponding tree.

### Frf - First-level reconstruction filters
matrix | cell array

First-level reconstruction filters, specified as an $N$-by-2 or $N$-by-3 matrix for single-tree wavelet transforms, or a 1-by-2 cell array of two $N$-by-2 or $N$-by-3 matrices for dual-tree wavelet transforms. The matrices are $N$-by-3 for the double-density wavelet transforms. For an $N$-by-2 matrix, the first column of the matrix is the scaling (lowpass) filter and the second column is the wavelet (highpass) filter. For an $N$-by-3 matrix, the first column of the matrix is the scaling (lowpass)

filter and the second and third columns are the wavelet (highpass) filters. For the dual-tree transforms, each element of the cell array contains the first-stage synthesis filters for the corresponding tree.

### Rf - Reconstruction filters for levels > 1

matrix | cell array

Reconstruction filters for levels > 1, specified as an *N*-by-2 or *N*-by-3 matrix for single-tree wavelet transforms, or a 1-by-2 cell array of two *N*-by-2 or *N*-by-3 matrices for dual-tree wavelet transforms. The matrices are *N*-by-3 for the double-density wavelet transforms. For an *N*-by-2 matrix, the first column of the matrix is the scaling (lowpass) filter and the second column is the wavelet (highpass) filter. For an *N*-by-3 matrix, the first column of the matrix is the scaling (lowpass) filter and the second and third columns are the wavelet (highpass) filters. For the dual-tree transforms, each element of the cell array contains the first-stage analysis filters for the corresponding tree.

### cfs - Wavelet transform coefficients

cell array of matrices

Wavelet transform coefficients, specified as a 1-by-(`level`+1) cell array of matrices. The size and structure of the matrix elements of the cell array depend on the type of wavelet transform as follows:

- `'dwt'` — `cfs{j}(:,:,d)`
  - `j = 1,2,...,` `level` is the level.
  - `d = 1,2,3` is the orientation.
  - `cfs{level+1}(:,:)` are the lowpass, or scaling, coefficients.
- `'ddt'` — `cfs{j}(:,:,d)`
  - `j = 1,2,...,` `level` is the level.
  - `d = 1,2,3,4,5,6,7,8` is the orientation.
  - `cfs{level+1}(:,:)` are the lowpass, or scaling, coefficients.
- `'realddt'` — `cfs{j}(:,:,d,k)`

- $j = 1,2,...,$ `level` is the level.
- $d = 1,2,3$ is the orientation.
- $k = 1,2$ is the wavelet transform tree.
- `cfs{level+1}(:,:)` are the lowpass, or scaling, coefficients.

- `'cplxdt'` — `cfs{j}(:,:,d,k,m)`
  - $j = 1,2,...,$ `level` is the level.
  - $d = 1,2,3$ is the orientation.
  - $k = 1,2$ is the wavelet transform tree.
  - $m = 1,2$ are the real and imaginary parts.
  - `cfs{level+1}(:,:)` are the lowpass, or scaling, coefficients..

- `'realdddt'` — `cfs{j}(:,:,d,k)`
  - $j = 1,2,...,$ `level` is the level.
  - $d = 1,2,3$ is the orientation.
  - $k = 1,2$ is the wavelet transform tree.
  - `cfs{level+1}(:,:)` are the lowpass, or scaling, coefficients.

- `'cplxdddt'` — `cfs{j}(:,:,d,k,m)`
  - $j = 1,2,...,$ `level` is the level.
  - $d = 1,2,3$ is the orientation.
  - $k = 1,2$ is the wavelet transform tree.
  - $m = 1,2$ are the real and imaginary parts.
  - `cfs{level+1}(:,:)` are the lowpass, or scaling, coefficients.

**Output Arguments**

**xrec - Synthesized 2-D image**

matrix

Synthesized image, returned as a matrix.

**Data Types**
`double`

**Examples**

**Perfect Reconstruction Using the Complex Oriented Dual-Tree Wavelet Filter Bank**

Demonstrate perfect reconstruction of an image using a complex oriented dual-tree wavelet transform.

Load the image and obtain the complex oriented dual-tree wavelet transform down to level 5 using `dddtree2`. Reconstruct the image using `idddtree2` and demonstrate perfect reconstruction.

```
load woman;
wt = dddtree2('cplxdt',X,5,'dtf2');
xrec = idddtree2(wt);
max(max(abs(X-xrec)))
```

**See Also**   `dddtree2` | `dddtreecfs`

**Related Examples**
- "Analytic Wavelets Using the Dual-Tree Wavelet Transform"

**Concepts**
- "Critically Sampled and Oversampled Wavelet Filter Banks"

| **Purpose** | Single-level inverse discrete 1-D wavelet transform |
|---|---|

**Syntax**

```
X = idwt(cA,cD,'wname')
X = idwt(cA,cD,Lo_R,Hi_R)
X = idwt(cA,cD,'wname',L)
X = idwt(cA,cD,Lo_R,Hi_R,L)
idwt(cA,cD,'wname')
X = idwt(...,'mode',MODE)
X = idwt(cA,[],...)
X = idwt([],cD,...)
```

**Description**

The idwt command performs a single-level one-dimensional wavelet reconstruction with respect to either a particular wavelet ('*wname*', see wfilters for more information) or particular wavelet reconstruction filters (Lo_R and Hi_R) that you specify.

X = idwt(cA,cD,'*wname*') returns the single-level reconstructed approximation coefficients vector X based on approximation and detail coefficients vectors cA and cD, and using the wavelet '*wname*'.

X = idwt(cA,cD,Lo_R,Hi_R) reconstructs as above using filters that you specify.

- Lo_R is the reconstruction low-pass filter.

- Hi_R is the reconstruction high-pass filter.

Lo_R and Hi_R must be the same length.

Let la be the length of cA (which also equals the length of cD) and lf the length of the filters Lo_R and Hi_R; then length(X) = LX where LX = 2*la if the DWT extension mode is set to periodization. For the other extension modes LX = 2*la-lf+2.

For more information about the different Discrete Wavelet Transform extension modes, see dwtmode.

X = idwt(cA,cD,'*wname*',L) or X = idwt(cA,cD,Lo_R,Hi_R,L) returns the length-L central portion of the result obtained using idwt(cA,cD,'*wname*'). L must be less than LX.

X = idwt(...,'*mode*',MODE) computes the wavelet reconstruction using the specified extension mode MODE.

X = idwt(cA,[],...) returns the single-level reconstructed approximation coefficients vector X based on approximation coefficients vector cA.

X = idwt([],cD,...) returns the single-level reconstructed detail coefficients vector X based on detail coefficients vector cD.

**Examples**

### Inverse DWT Using Orthogonal Wavelet

Demonstrate perfect reconstruction using dwt and idwt with an orthonormal wavelet.

```
load noisdopp;
[A,D] = dwt(noisdopp,'sym4');
x = idwt(A,D,'sym4');
max(abs(noisdopp-x))
```

### Inverse DWT Using Biorthgonal Wavelet

Demonstrate perfect reconstruction using dwt and idwt with a biorthogonal wavelet.

```
load noisdopp;
[Lo_D,Hi_D,Lo_R,Hi_R] = wfilters('bior3.5');
[A,D] = dwt(noisdopp,Lo_D,Hi_D);
x = idwt(A,D,Lo_R,Hi_R);
max(abs(noisdopp-x))
```

**Algorithms**

Starting from the approximation and detail coefficients at level *j*, *cAj* and *cD$_j$*, the inverse discrete wavelet transform reconstructs *cA$_{j-1}$*, inverting the decomposition step by inserting zeros and convolving the results with the reconstruction filters.

**One-Dimensional IDWT**
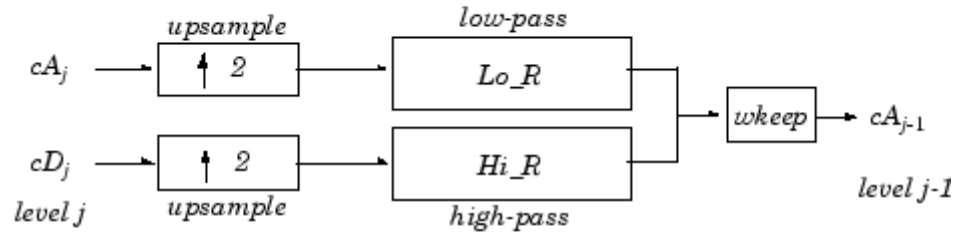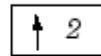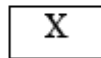
## Reconstruction step



References

Daubechies, I. (1992), *Ten lectures on wavelets*, CBMS-NSF conference series in applied mathematics. SIAM Ed.

Mallat, S. (1989), "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Pattern Anal. and Machine Intell.*, vol. 11, no. 7, pp. 674–693.

Meyer, Y. (1990), *Ondelettes et opérateurs*, Tome 1, Hermann Ed. (English translation: *Wavelets and operators*, Cambridge Univ. Press. 1993.)

See Also

dwt | dwtmode | upwlev

# idwt2

| | |
|---|---|
| **Purpose** | Single-level inverse discrete 2-D wavelet transform |
| **Syntax** | X = idwt2(cA,cH,cV,cD,'*wname*')<br>X = idwt2(cA,cH,cV,cD,Lo_R,Hi_R)<br>X = idwt2(cA,cH,cV,cD,'*wname*',S)<br>X = idwt2(cA,cH,cV,cD,Lo_R,Hi_R,S)<br>idwt2(cA,cH,cV,cD,'*wname*')<br>X = idwt2(...,'mode',MODE)<br>X = idwt2(cA,[],[],[],...)<br>X = idwt2([],cH,[],[],...) |

**Description**   The idwt2 command performs a single-level two-dimensional wavelet reconstruction with respect to either a particular wavelet ('*wname*', see wfilters for more information) or particular wavelet reconstruction filters (Lo_R and Hi_R) that you specify.

X = idwt2(cA,cH,cV,cD,'*wname*') uses the wavelet '*wname*' to compute the single-level reconstructed approximation coefficients matrix *X*, based on approximation matrix cA and details matrices cH,cV, and cD (horizontal, vertical, and diagonal, respectively).

X = idwt2(cA,cH,cV,cD,Lo_R,Hi_R) reconstructs as above, using filters that you specify.

- Lo_R is the reconstruction low-pass filter.

- Hi_R is the reconstruction high-pass filter.

Lo_R and Hi_R must be the same length.

Let sa = size(cA) = size(cH) = size(cV) = size(cD) and lf the length of the filters; then size(X) = SX, where SX = 2* SA, if the DWT extension mode is set to periodization. For the other extension modes, SX = 2*size(cA)-lf+2.

For more information about the different Discrete Wavelet Transform extension modes, see dwtmode.

X = idwt2(cA,cH,cV,cD,'*wname*',S) and X = idwt2(cA,cH,cV,cD,Lo_R,Hi_R,S) return the size-S central portion of

the result obtained using the syntax `idwt2(cA,cH,cV,cD,'wname')`.
S must be less than SX.

`X = idwt2(...,'mode',MODE)` computes the wavelet reconstruction
using the extension mode MODE that you specify.

`X = idwt2(cA,[],[],[],...)` returns the single-level reconstructed
approximation coefficients matrix X based on approximation coefficients
matrix cA.

`X = idwt2([],cH,[],[],...)` returns the single-level reconstructed
detail coefficients matrix X based on horizontal detail coefficients
matrix cH.

The same result holds for `X = idwt2([],[],cV,[],...)` and
`X = idwt2([],[],[],cD,...)`, based on vertical and diagonal details.

More generally, `X = idwt2(AA,HH,VV,DD,...)` returns the single-level
reconstructed matrix X, where AA can be cA or [], and so on.

`idwt2` is the inverse function of dwt2 in the sense that the abstract
statement
`idwt2(dwt2(X,'wname'),'wname')` would give back X.

**Tips**  If cA,cH,cV,cD are obtained from an indexed image analysis or a
truecolor image analysis, they are m-by-n matrices or m-by-n-by-3 arrays,
respectively.

For more information on image formats, see the `image` and `imfinfo`
reference pages.

**Examples**
```
% The current extension mode is zero-padding (see dwtmode).

% Load original image.
load woman;

% X contains the loaded image.
sX = size(X);

% Perform single-level decomposition
```

# idwt2

```
% of X using db4.
[cA1,cH1,cV1,cD1] = dwt2(X,'db4');

% Invert directly decomposition of X
% using coefficients at level 1.
A0 = idwt2(cA1,cH1,cV1,cD1,'db4',sX);

% Check for perfect reconstruction.
max(max(abs(X-A0)))
ans =
    3.4176e-10
```

**Algorithms**

## Two-Dimensional IDWT
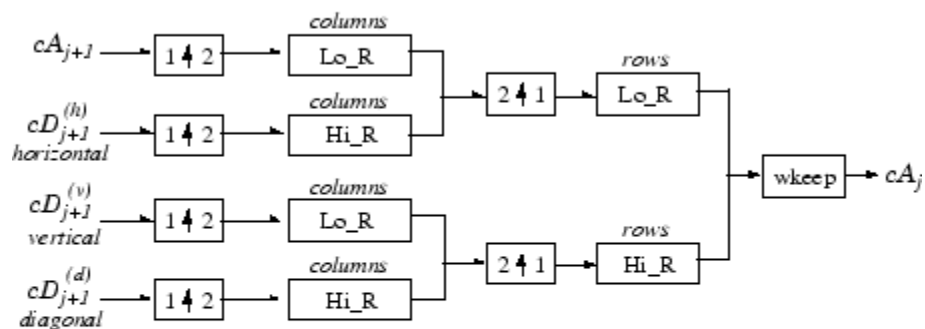
### Reconstruction step



1-220

**See Also**　　　dwt2 | dwtmode | upwlev2

# idwt3

**Purpose**      Single-level inverse discrete 3-D wavelet transform

**Syntax**       X = idwt3(WT)
                 C = idwt3(WT,TYPE)

**Description**  The idwt3 command performs a single-level three-dimensional wavelet
                 reconstruction starting from a single-level three-dimensional wavelet
                 decomposition.

                 X = idwt3(WT) computes the single-level reconstructed 3-D array X,
                 based on the three-dimensional wavelet decomposition stored in the WT
                 structure. This structure contains the following fields.

| sizeINI | Size of the three-dimensional array X. |
|---------|----------------------------------------|
| mode    | Name of the wavelet transform extension mode. |
| filters | Structure with 4 fields, LoD, HiD, LoR, HiR, which contain the filters used for DWT. |
| dec     | 2 x 2 x 2 cell array containing the coefficients of the decomposition. |
|         | dec{i,j,k}, i,j,k = 1 or 2 contains the coefficients obtained by low-pass filtering (for i or j or k = 1) or high-pass filtering (for i or j or k = 2). |

C = idwt3(WT,TYPE) computes the single-level reconstructed
component based on the three-dimensional wavelet decomposition.
Valid values for TYPE are:

- A group of three characters 'xyz', one per direction, with 'x','y'
  and 'z' selected in the set {'a','d','l','h'} or in the corresponding
  uppercase set {'A','D','L','H'}), where 'A' (or 'L') specifies
  low-pass filter and 'D' (or 'H') specifies high-pass filter.

- The char 'd' (or 'h' or 'D' or 'H') which specifies the sum of all the
  components different from the low-pass component.

**Examples**
```
% Define original 3D data.
X  = reshape(1:64,4,4,4);

% Decompose X using db1.
wt = dwt3(X,'db1');

% Reconstruct X from coefficients.
XR = idwt3(wt);

% Compute reconstructed approximation, i.e. the
% low-pass component.
A  = idwt3(wt,'aaa');

% Compute the sum of all the components different
% from the low-pass component.
D  = idwt3(wt,'d');

% Reconstruct the component associated with low-pass in the
% X and Z directions and high-pass in the Y direction.
ADA  = idwt3(wt,'ada');
```

**See Also**    dwt3 | wavedec3 | waverec3

# ilwt

| | |
|---|---|
| **Purpose** | Inverse 1-D lifting wavelet transform |
| **Syntax** | X = ilwt(AD_In_Place,W)<br>X = ilwt(CA,CD,W)<br>X = ilwt(AD_In_Place,W,LEVEL)<br>X = ILWT(CA,CD,W,LEVEL)<br>X = ilwt(AD_In_Place,W,LEVEL,'typeDEC',typeDEC)<br>X = ilwt(CA,CD,W,LEVEL,'typeDEC',typeDEC) |
| **Description** | ilwt performs a 1-D lifting wavelet reconstruction with respect to a particular lifted wavelet that you specify.<br><br>X = ilwt(AD_In_Place,W) computes the reconstructed vector X using the approximation and detail coefficients vector AD_In_Place obtained by a lifting wavelet reconstruction. W is a lifted wavelet name (see liftwave).<br><br>X = ilwt(CA,CD,W) computes the reconstructed vector X using the approximation coefficients vector CA and detail coefficients vector CD obtained by a lifting wavelet reconstruction.<br><br>X = ilwt(AD_In_Place,W,LEVEL) or X = ILWT(CA,CD,W,LEVEL) computes the lifting wavelet reconstruction, at level LEVEL.<br><br>X = ilwt(AD_In_Place,W,LEVEL,'typeDEC',typeDEC) or X = ilwt(CA,CD,W,LEVEL,'typeDEC',typeDEC) with typeDEC = 'w' or 'wp' computes the wavelet or the wavelet packet decomposition using lifting, at level LEVEL.<br><br>Instead of a lifted wavelet name, you may use the associated lifting scheme LS: X = ilwt(...,LS,...) instead of X = ILWT(...,W,...).<br><br>For more information about lifting schemes, see lsinfo. |
| **Examples** | % Start from the Haar wavelet and get the<br>% corresponding lifting scheme.<br>lshaar = liftwave('haar');<br><br>% Add a primal ELS to the lifting scheme. |

```
els = {'p',[-0.125 0.125],0};
lsnew = addlift(lshaar,els);

% Perform LWT at level 1 of a simple signal.
x = 1:8;
[cA,cD] = lwt(x,lsnew);

% Perform integer LWT of the same signal.
lshaarInt = liftwave('haar','int2int');
lsnewInt = addlift(lshaarInt,els);
[cAint,cDint] = lwt(x,lsnewInt);

% Invert the two transforms.
xRec = ilwt(cA,cD,lsnew);
err = max(max(abs(x-xRec)))


err =

  4.4409e-016

xRecInt = ilwt(cAint,cDint,lsnewInt);
errInt = max(max(abs(x-xRecInt)))

errInt =

     0
```

**See Also**        lwt

# ilwt2

| | |
|---|---|
| **Purpose** | Inverse 2-D lifting wavelet transform |
| **Syntax** | `X = ilwt2(AD_In_Place,W)` |
| | `X = ilwt2(CA,CH,CV,CD,W)` |
| | `X = ilwt2(AD_In_Place,W,LEVEL)` |
| | `X = ILWT2(CA,CH,CV,CD,W,LEVEL)` |
| | `X = ilwt2(AD_In_Place,W,LEVEL,'typeDEC',typeDEC)` |
| | `X = ilwt2(CA,CH,CV,CD,W,LEVEL,'typeDEC',typeDEC)` |

**Description**  `ilwt2` performs a 2-D lifting wavelet reconstruction with respect to a particular lifted wavelet that you specify.

`X = ilwt2(AD_In_Place,W)` computes the reconstructed matrix `X` using the approximation and detail coefficients matrix `AD_In_Place`, obtained by a lifting wavelet decomposition. `W` is a lifted wavelet name (see `liftwave`).

`X = ilwt2(CA,CH,CV,CD,W)` computes the reconstructed matrix `X` using the approximation coefficients vector `CA` and detail coefficients vectors `CH`, `CV`, and `CD` obtained by a lifting wavelet decomposition.

`X = ilwt2(AD_In_Place,W,LEVEL)` or `X = ILWT2(CA,CH,CV,CD,W,LEVEL)` computes the lifting wavelet reconstruction, at level `LEVEL`.

`X = ilwt2(AD_In_Place,W,LEVEL,'typeDEC',typeDEC)` or `X = ilwt2(CA,CH,CV,CD,W,LEVEL,'typeDEC',typeDEC)` with `typeDEC = 'w'` or `'wp'` computes the wavelet or the wavelet packet decomposition using lifting, at level `LEVEL`.

Instead of a lifted wavelet name, you may use the associated lifting scheme LS: `X = ilwt2(...,LS,...)` instead of `X = ilwt2(...,W,...)`.

For more information about lifting schemes, see `lsinfo`.

**Tips**  If `AD_In_Place` or cA,cH,cV,cD are obtained from an indexed image analysis or a truecolor image analysis, they are m-by-n matrices or m-by-n-by-3 arrays, respectively.

For more information on image formats, see the `image` and `imfinfo` reference pages.

**Examples**
```
% Start from the Haar wavelet and get the
% corresponding lifting scheme.
lshaar = liftwave('haar');

% Add a primal ELS to the lifting scheme.
els = {'p',[-0.125 0.125],0};
lsnew = addlift(lshaar,els);

% Perform LWT at level 1 of a simple image.
x = reshape(1:16,4,4);
[cA,cH,cV,cD] = lwt2(x,lsnew);

% Perform integer LWT of the same image.
lshaarInt = liftwave('haar','int2int');
lsnewInt = addlift(lshaarInt,els);
[cAint,cHint,cVint,cDint] = lwt2(x,lsnewInt);

% Invert the two transforms.
xRec = ilwt2(cA,cH,cV,cD,lsnew);
err = max(max(abs(x-xRec)))

err =

     0

xRecInt = ilwt2(cAint,cHint,cVint,cDint,lsnewInt);
errInt = max(max(abs(x-xRecInt)))

errInt =

     0
```

**See Also**    `lwt2`

# ind2depo

| | |
|---|---|
| **Purpose** | Node index to node depth-position |
| **Syntax** | `[D,P] = ind2depo(ORD,[D P])` |
| **Description** | `ind2depo` is a tree-management utility. |

For a tree of order `ORD`, `[D,P] = ind2depo(ORD,N)` computes the depths `D` and the positions `P` (at these depths `D`) for the nodes with indices `N`.

The nodes are numbered from left to right and from top to bottom. The root index is 0.

`N` must be a column vector of integers (`N    0`).

Note that `[D,P] = ind2depo(ORD,[D P])`.

**Depth and Position in Wavelet Packet Tree**

Create a binary wavelet packet tree with three levels.

```
Ord = 2;
Lev = 3;
T = ntree(Ord,Lev);
```

Plot the binary wavelet packet tree.

```
plot(T)
```
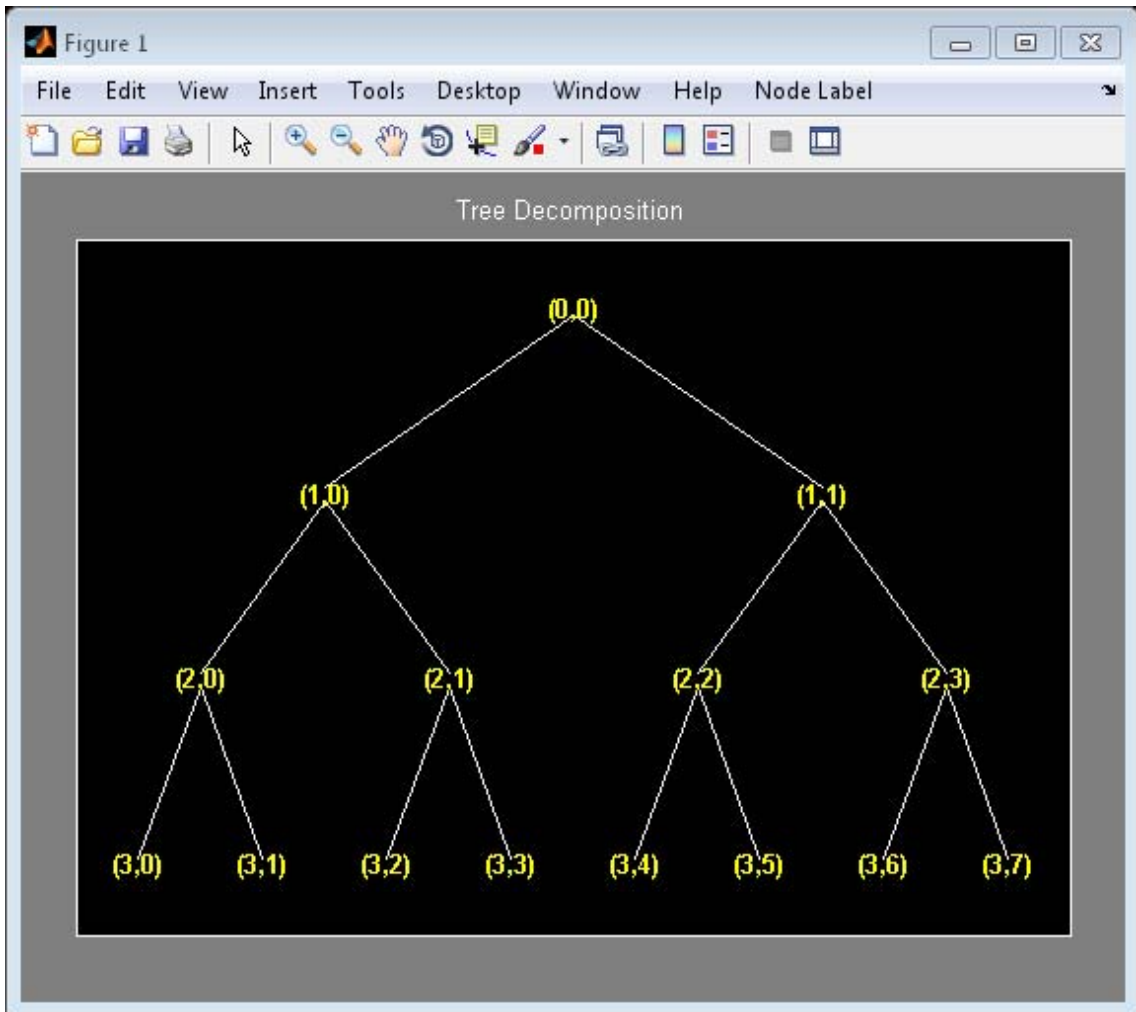
Obtain the indices of the nodes in linear order.

```
idx = allnodes(T);
```

Conver the indices to depth-position format.

```
[depth,pos] = ind2depo(Ord,idx);
table(depth,pos)


ans =

    depth    pos
    _____    ___

    0        0
    1        0
    1        1
    2        0
    2        1
    2        2
    2        3
    3        0
    3        1
    3        2
    3        3
    3        4
    3        5
    3        6
    3        7
```

**See Also**      depo2ind

**Purpose**     Inverse nondecimated 1-D wavelet transform

**Syntax**      C = indwt(*W*,*TYPE*,*N*)
                C = indwt(*W*,*TYPE*)
                C = indwt(*W*,*TYPE*,*N*)
                X = indwt(*W*)
                X = indwt(*W*,'a',0)
                X = indwt(*W*,'ca',0)

**Description**  indwt performs a multilevel nondecimated 1-D wavelet reconstruction
                starting from a multilevel nondecimated 1-D wavelet decomposition.
                You can also use indwt to extract coefficients from a multilevel
                nondecimated 1-D wavelet decomposition.

                C = indwt(*W*,*TYPE*,*N*) computes the reconstructed components at level
                *N* of a non-decimated 1-D wavelet decomposition. *N* must be a positive
                integer less or equal to the level of the decomposition. The valid value
                for *TYPE* is a char:

                • 'a' (or 'l' or 'A' or 'L'), which gives the low-pass component

                • 'd' (or 'h' or 'D' or 'H'), which gives the high-pass component

                where 'A' (or 'L') specifies low-pass filter and 'D' (or 'H') specifies
                high-pass filter.

                For extraction, the valid values for *TYPE* are the same but prefixed by
                'c' or 'C'.

                See ndwt for more information about the decomposition structure *W*.

                C = indwt(*W*,*TYPE*) is equivalent to C = indwt(*W*,*TYPE*,*N*) with *N*
                equal to the level of the decomposition.

                X = indwt(*W*), X = indwt(*W*,'a',0) or X = indwt(*W*,'ca',0)
                reconstructs the vector *X* based on the nondecimated 1-D wavelet
                decomposition structure *W*.

**Examples**    % Load the signal
                load noissin;

```
x = noissin;

% Decompose X at level 3 using db1.
W1 = ndwt(x,3,'db1');

% Reconstruct the original signal from the
% decomposition W1 structure.
a0 = indwt(W1,'a',0);

% Check for perfect reconstruction.
err = max(abs(x(:)-a0(:)))

err =

  8.8818e-016

% Decompose X at level 3 using db3 and periodic extension mode.
W2 = ndwt(x,3,'db3','mode','per');

% Reconstruct approximation at level 2.
a2 = indwt(W2,'a',2);

% Reconstruct detail at level 2.
d2 = indwt(W2,'d',2);

% Reconstruct detail at level 1.
d1 = indwt(W2,'d',1);
```

**See Also**     dwtmode | ndwt | waveinfo

**Purpose**     Inverse nondecimated 2-D wavelet transform

**Syntax**      C = indwt2(*W*,*TYPE*,*N*)
                C = indwt2(*W*,*TYPE*)
                C = indwt2(*W*,*TYPE*,*N*)
                X = indwt2(*W*)
                X = indwt2(*W*,'a',0)
                X = indwt2(*W*,'ca',0)

**Description**  indwt2 performs a multilevel nondecimated 2-D wavelet reconstruction
                starting from a multilevel nondecimated 2-D wavelet decomposition.
                You can also use indwt2 to extract coefficients from a multilevel
                nondecimated 2-D wavelet decomposition.

                C = indwt2(*W*,*TYPE*,*N*) computes the reconstructed or the extracted
                components at level *N* of a nondecimated 2-D wavelet decomposition. *N*
                must be a positive integer less or equal to the level of the decomposition.
                The valid values for *TYPE* are:

                - A group of 2 chars 'xy', one per direction, with 'x' and 'y' in
                  the set {'a','d','l','h'} or in the corresponding uppercase set
                  {'A','D','L','H'}), where 'A' (or 'L') stands for low-pass filter
                  and 'D' (or 'H') stands for high-pass.

                - The char 'd' (or 'h' or 'D' or 'H') specifies the sum of the
                  components different from the low-pass one.

                For extraction, the valid values for *TYPE* are the same as above prefixed
                by 'c' or 'C'.

                See ndwt2 for more information about the decomposition structure *W*.

                C = indwt2(*W*,*TYPE*) is equivalent to C = indwt2(*W*,*TYPE*,*N*) with *N*
                equal to the level of the decomposition.

                X = indwt2(*W*), X = indwt2(*W*,'a',0) or X = indwt2(*W*,'ca',0)
                reconstructs the matrix *X* based on the nondecimated 2-D wavelet
                decomposition structure *W*.

**Examples**

```
% Load original image.
load noiswom

% Decompose X at level 3 using db1.
W = ndwt2(X,3,'db1');

% Reconstruct approximations at levels 1 to 3.
A = cell(1,3);
for k=1:3, A{k} = indwt2(W,'aa',k); end

 % Plot original image at the top and approximations
% at the bottom.
figure; colormap(pink(255))
subplot(2,3,2);image(X);
for k=1:3
    subplot(2,3,k+3);image(A{k});
end
```
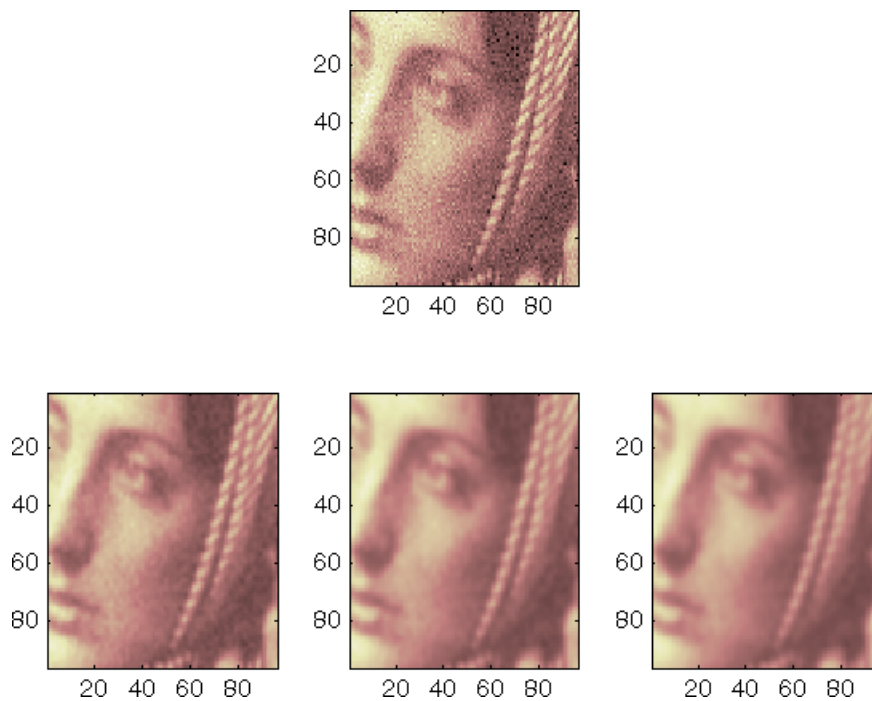
```
% Reconstruct detail at level 1.
D = indwt2(W,'d',1);

% Display reconstructed detail at level 1.
figure; colormap(pink(255));imagesc(abs(D))
```
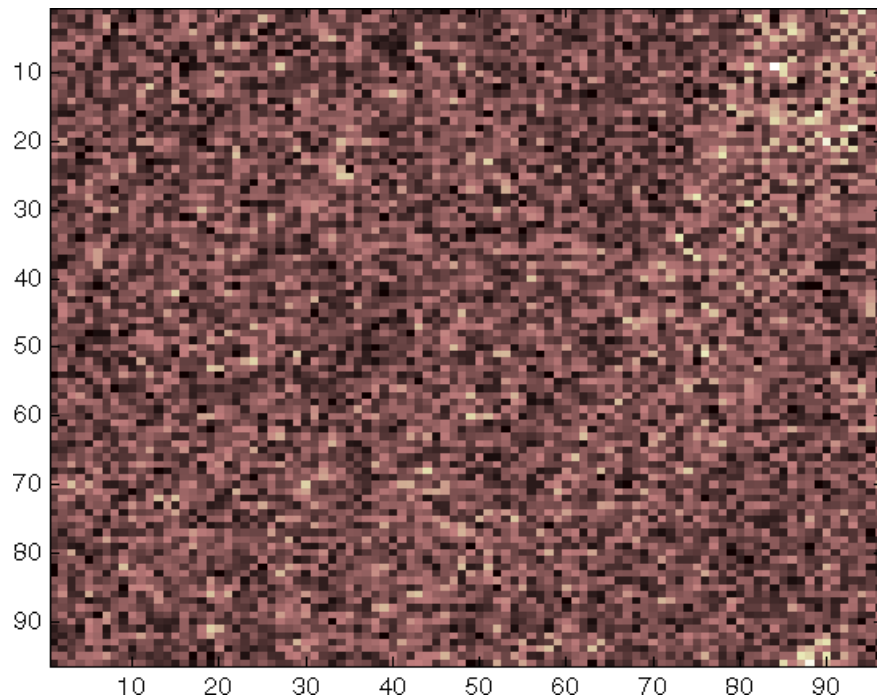
```
% Compute reconstructed approximation and detail at level 1.
A1 = indwt2(W,'aa',1);
D1 = indwt2(W,'d',1);

% Check that X = A1 + D1.
E1 = X-A1-D1;
err1 = max(abs(E1(:)))

err1 =

  2.6645e-013

% Compute reconstructed approximation and detail at level 2.
A2 = indwt2(W,'aa',2);
```

```
D2 = indwt2(W,'d',2);

% Check that X = A2 + D2.
E2 = X-A2-D2;
err2 = max(abs(E2(:)))

err2 =

  2.5668e-013
```

**See Also**     dwtmode | ndwt2 | waveinfo

| | |
|---|---|
| **Purpose** | Integrate wavelet function psi (ψ) |
| **Syntax** | `[INTEG,XVAL] = intwave('wname',PREC)` |
| | `[INTDEC,XVAL,INTREC] = intwave('wname',PREC)` |
| | `[INTEG,XVAL] = intwave('wname',PREC)` |
| | `[INTEG,XVAL] = intwave('wname',PREC,0)` |
| | `[INTEG,XVAL] = intwave('wname')` |
| | `[INTEG,XVAL] = intwave('wname',8)` |
| | `intwave('wname',IN2,IN3), PREC = max(IN2,IN3)` |
| | `intwave('wname',0)` |
| | `intwave('wname',8,IN3)` |
| | `intwave('wname')` |
| | `intwave('wname',8)` |

**Description**  `[INTEG,XVAL] = intwave('wname',PREC)` computes the integral,

INTEG, of the wavelet function ψ (from −∞ to XVAL values): $\int_{-\infty}^{x} \psi(y)dy$
for $x$ in XVAL.

The function ψ is approximated on the $2^{PREC}$ points grid XVAL where *PREC* is a positive integer. `'wname'` is a string containing the name of the wavelet ψ (see wfilters for more information).

Output argument *INTEG* is a real or complex vector depending on the wavelet type.

For biorthogonal wavelets,

`[INTDEC,XVAL,INTREC] = intwave('wname',PREC)` computes the integrals, INTDEC and INTREC, of the wavelet decomposition function $\psi_{dec}$ and the wavelet reconstruction function $\psi_{rec}$.

`[INTEG,XVAL] = intwave('wname',PREC)` is equivalent to `[INTEG,XVAL] = intwave('wname',PREC,0)`.

`[INTEG,XVAL] = intwave('wname')` is equivalent to `[INTEG,XVAL] = intwave('wname',8)`.

When used with three arguments `intwave('wname',IN2,IN3)`, PREC = `max(IN2,IN3)` and plots are given.

When IN2 is equal to the special value 0, intwave('*wname*',0) is equivalent to intwave('*wname*',8,IN3).

intwave('*wname*') is equivalent to intwave('*wname*',8).

intwave is used only for continuous analysis (see cwt for more information).

**Examples**

```
% Set wavelet name.
wname = 'db4';

% Plot wavelet function.
[phi,psi,xval] = wavefun(wname,7);
subplot(211); plot(xval,psi); title('Wavelet');

% Compute and plot wavelet integrals approximations
% on a dyadic grid.
[integ,xval] = intwave(wname,7);
subplot(212); plot(xval,integ);
title(['Wavelet integrals over [-Inf x] ' ...
        'for each value of xval']);
```

Wavelet

Wavelet integrals over [–Inf x] for each value of xval
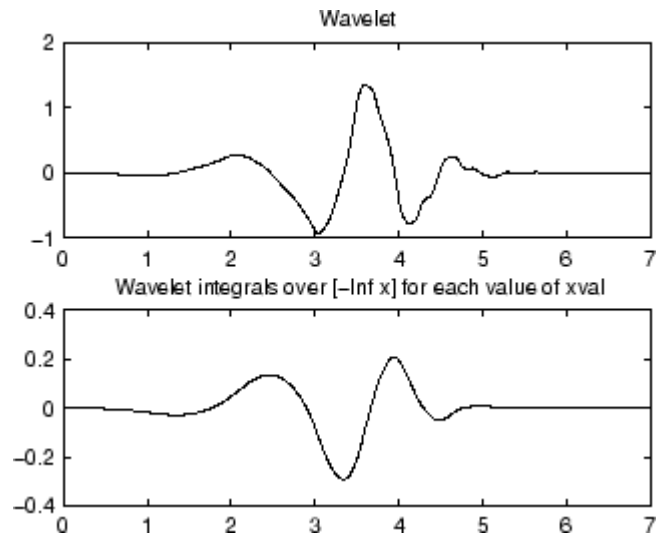
**Algorithms**    First, the wavelet function is approximated on a grid of $2^{\mathrm{PREC}}$ points using wavefun. A piecewise constant interpolation is used to compute the integrals using cumsum.

**See Also**    wavefun

**Purpose**       Existing node test

**Syntax**        R = isnode(*T*,*N*)

**Description**   isnode is a tree-management utility.

R = isnode(*T*,*N*) returns 1's for nodes *N*, which exist in the tree *T*, and 0's for others.

*N* can be a column vector containing the indices of nodes or a matrix, that contains the depths and positions of nodes.

In the last case, N(i,1) is the depth of the i-th node and N(i,2) is the position of the i-th node.

The nodes are numbered from left to right and from top to bottom. The root index is 0.

**Examples**
```
% Create initial tree.
ord = 2;
t = ntree(ord,3);    % binary tree of depth 3.
t = nodejoin(t,5);
t = nodejoin(t,4);
plot(t)
```



```
% Change Node Label from Depth_Position to Index
```
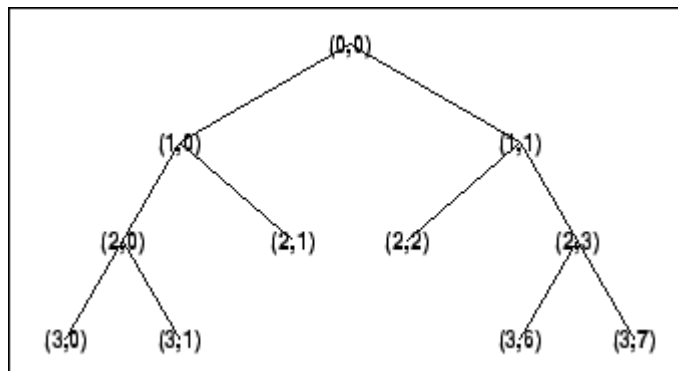
```
% (see the plot function).
```
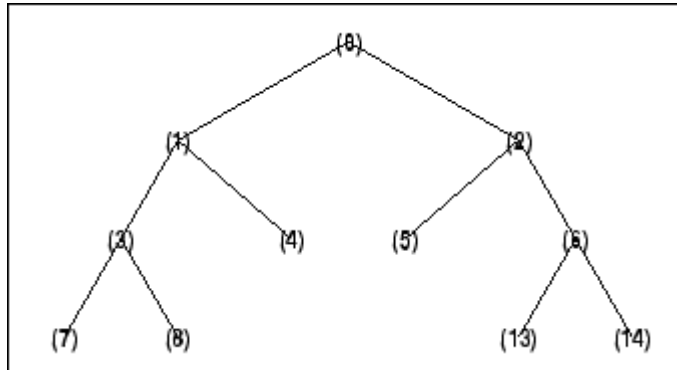


```
% Check node index.
isnode(t,[1;3;25])

ans =
     1
     1
     0

% Check node Depth_Position.
isnode(t,[1 0;3 1;4 5])

ans =
     1
     1
     0
```

**See Also**   istnode | wtreemgr

**Purpose**          Terminal nodes indices test

**Syntax**           R = istnode(T,N)

**Description**       istnode is a tree-management utility.

R = istnode(T,N) returns ranks (in left to right terminal nodes ordering) for terminal nodes *N* belonging to the tree *T*, and 0's for others.

*N* can be a column vector containing the indices of nodes or a matrix that contains the depths and positions of nodes.

In the last case, N(i,1) is the depth of the i-th node and N(i,2) is the position of the i-th node.

The nodes are numbered from left to right and from top to bottom. The root index is 0.

**Examples**
```
% Create initial tree.
ord = 2;
t = ntree(ord,3); % binary tree of depth 3.
t = nodejoin(t,5);
t = nodejoin(t,4);
plot(t)
```
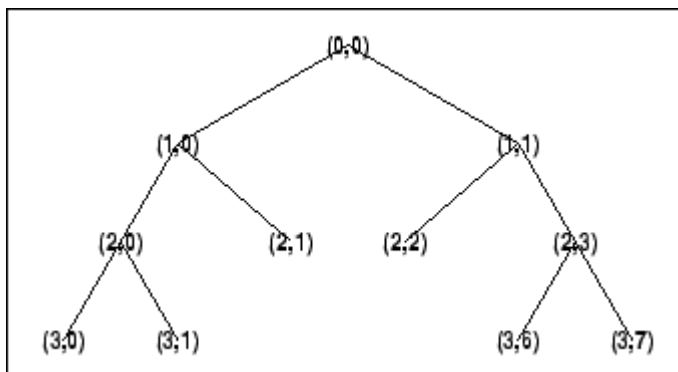


```
% Change Node Label from Depth_Position to Inde
```

```
% (see the plot function)x.
```
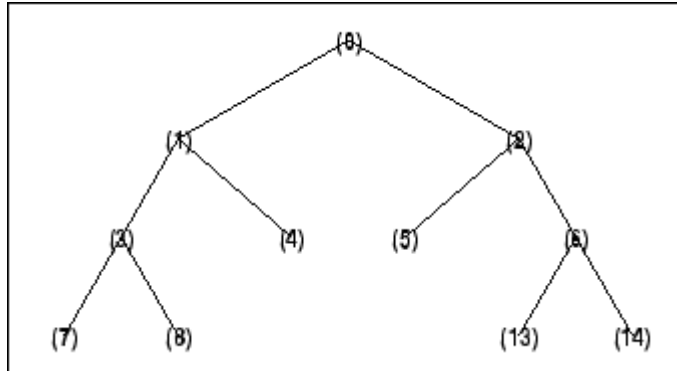


```
% Find terminal nodes and return indices for terminal
% nodes in the tree.
istnode(t,[14])
ans =
    6

istnode(t,[15])
ans =
    0

istnode(t,[1;7;14;25])
ans =
    0
    1
    6
    0

istnode(t,[1 0;3 1;4 5])
ans =
    0
    2
    0
```

**See Also**     isnode | wtreemgr

# iswt

| **Purpose** | Inverse discrete stationary wavelet transform 1-D |
|---|---|

**Syntax**

```
X = iswt(SWC,'wname')
X = iswt(SWA,SWD,'wname')
X = iswt(SWA(end,:),SWD,'wname')
X = iswt(SWC,Lo_R,Hi_R)
X = iswt(SWA,SWD,Lo_R,Hi_R)
X = iswt(SWA(end,:),SWD,Lo_R,Hi_R)
```

**Description**

iswt performs a multilevel 1-D stationary wavelet reconstruction using either a specific orthogonal wavelet ('wname', see wfilters for more information) or specific reconstruction filters (Lo_R and Hi_R).

X = iswt(SWC,'wname') or X = iswt(SWA,SWD,'wname') or X = iswt(SWA(end,:),SWD,'wname') reconstructs the signal X based on the multilevel stationary wavelet decomposition structure SWC or [SWA,SWD] (see swt for more information).

X = iswt(SWC,Lo_R,Hi_R) or X = iswt(SWA,SWD,Lo_R,Hi_R) or X = iswt(SWA(end,:),SWD,Lo_R,Hi_R) reconstruct as above, using filters that you specify.

- Lo_R is the reconstruction low-pass filter.

- Hi_R is the reconstruction high-pass filter.

Lo_R and Hi_R must be the same length.

**Examples**

```
% Load original 1D signal.
load noisbloc; s = noisbloc;

% Perform SWT decomposition at level 3 of s using db1.
swc = swt(s,3,'db1');
% Second usage.
[swa,swd] = swt(s,3,'db1');

% Reconstruct s from the stationary wavelet
% decomposition structure swc.
a0 = iswt(swc,'db1');
```

```
% Second usage.
aObis = iswt(swa,swd,'db1');


% Check for perfect reconstruction.
err = norm(s-aO)
err =
  9.6566e-014

errbis = norm(s-aObis)
errbis =
  9.6566e-014
```

**References**    Nason, G.P.; B.W. Silverman (1995), "The stationary wavelet transform and some statistical applications," *Lecture Notes in Statistics*, 103, pp. 281–299.

Coifman, R.R.; Donoho D.L. (1995), "Translation invariant de-noising," *Lecture Notes in Statistics*, 103, pp 125–150.

Pesquet, J.C.; H. Krim, H. Carfatan (1996), "Time-invariant orthonormal wavelet representations," *IEEE Trans. Sign. Proc.*, vol. 44, 8, pp. 1964–1970.

**See Also**    idwt | swt | waverec

# iswt2

**Purpose**　　　　Inverse discrete stationary wavelet transform 2-D

**Syntax**　　　　　X = iswt2(SWC,'*wname*')
X = iswt2(A,H,V,D,*wname*)
X = iswt2(A(:,:,end),H,V,D,'*wname*')
X = iswt2(SWC,Lo_R,Hi_R)
X = iswt2(A,H,V,D,Lo_R,Hi_R)
X = iswt2(A(:,:,end),H,V,D,Lo_R,Hi_R)

**Description**　　iswt2 performs a multilevel 2-D stationary wavelet reconstruction using either a specific orthogonal wavelet ('*wname*' see wfilters for more information) or specific reconstruction filters (Lo_R and Hi_R).

X = iswt2(SWC,'*wname*') or X = iswt2(A,H,V,D,*wname*) or X = iswt2(A(:,:,end),H,V,D,'*wname*') reconstructs the signal X, based on the multilevel stationary wavelet decomposition structure SWC or [A,H,V,D] (see swt2).

X = iswt2(SWC,Lo_R,Hi_R) or X = iswt2(A,H,V,D,Lo_R,Hi_R) or X = iswt2(A(:,:,end),H,V,D,Lo_R,Hi_R) reconstructs as in the previous syntax, using filters that you specify:

- Lo_R is the reconstruction low-pass filter.

- Hi_R is the reconstruction high-pass filter.

Lo_R and Hi_R must be the same length.

**Tips**　　　　　If SWC or (cA,cH,cV,cD) are obtained from an indexed image analysis or a truecolor image analysis, then X is an m-by-n matrix or an m-by-n-by-3 array, respectively.

For more information on image formats, see the image and imfinfo reference pages.

**Examples**　　　% Load original image.
load nbarb1;

% Perform SWT decomposition

```
% of X at level 3 using sym4.
swc = swt2(X,3,'sym4');
% Second usage.
[ca,chd,cvd,cdd] = swt2(X,3,'sym4');

% Reconstruct s from the stationary wavelet
% decomposition structure swc.
a0 = iswt2(swc,'sym4');
% Second usage.
a0 = iswt2(ca,chd,cvd,cdd,'sym4');
% Check for perfect reconstruction.
err = max(max(abs(X-a0)))
ans =
  2.3482e-010

errbis = max(max(abs(X-a0bis)))
ans =
  2.3482e-010
```

**References**    Nason, G.P.; B.W. Silverman (1995), "The stationary wavelet transform and some statistical applications," *Lecture Notes in Statistics*, 103, pp. 281–299.

Coifman, R.R.; Donoho D.L. (1995), "Translation invariant de-noising," *Lecture Notes in Statistics*, 103, pp. 125–150.

Pesquet, J.C.; H. Krim, H. Carfatan (1996), "Time-invariant orthonormal wavelet representations," *IEEE Trans. Sign. Proc.*, vol. 44, 8, pp. 1964–1970.

**See Also**    idwt2 | swt2 | waverec2

# laurmat

| | |
|---|---|
| **Purpose** | Laurent matrices constructor |
| **Syntax** | M = laurmat(*V*) |
| **Description** | M = laurmat(*V*) returns the Laurent matrix object M associated with *V* which can be a cell array (at most two dimensional) of Laurent polynomials (see laurpoly) or an ordinary matrix. |

**Examples**

```
% Define Laurent matrices.
M1 = laurmat(eye(2,2))

        | 1      0  |
        |           |
 M1 =   |           |
        |           |
        | 0      1  |

Z  = laurpoly(1,1);
M2 = laurmat({1 Z;0 1})

        | 1      z^(+1)  |
        |                |
 M2 =   |                |
        |                |
        | 0       1      |

% Calculus on Laurent polynomials.
P = M1 * M2

        | 1      z^(+1)  |
        |                |
 P  =   |                |
        |                |
        | 0       1      |

d = det(P)
```

```
d(z) = 1
```

**References**  Strang, G.; T. Nguyen (1996), *Wavelets and filter banks*, Wellesley-Cambridge Press.

Sweldens, W. (1998), "The Lifting Scheme: a Construction of Second Generation of Wavelets," *SIAM J. Math. Anal.*, 29 (2), pp. 511–546.

**See Also**  `laurpoly`

# laurpoly

| | |
|---|---|
| **Purpose** | Laurent polynomials constructor |
| **Syntax** | `P = laurpoly(C,d)`<br>`P = laurpoly(C,'dmin',d)`<br>`P = laurpoly(C,'dmax',d)`<br>`P = laurpoly(C,d)` |

**Description**

`P = laurpoly(C,d)` returns a Laurent polynomial object. *C* is a vector whose elements are the coefficients of the polynomial P and *d* is the highest degree of the monomials of P.

If m is the length of the vector *C*, *P* represents the following Laurent polynomial:

`P(z) = C(1)*z^d + C(2)*z^(d-1) + ... + C(m)*z^(d-m+1)`

`P = laurpoly(C,'dmin',d)` specifies the lowest degree instead of the highest degree of monomials of *P*. The corresponding output *P* represents the following Laurent polynomial:

`P(z) = C(1)*z^(d+m-1) + ... + C(m-1)*z^(d+1) + C(m)*z^d`

`P = laurpoly(C,'dmax',d)` is equivalent to `P = laurpoly(C,d)`.

**Examples**

```
% Define Laurent polynomials.
P = laurpoly([1:3],2);
P = laurpoly([1:3],'dmax',2)

P(z) = + z^(+2) + 2*z^(+1) + 3

P = laurpoly([1:3],'dmin',2)

P(z) = + z^(+4) + 2*z^(+3) + 3*z^(+2)

% Calculus on Laurent polynomials.
Z = laurpoly(1,1)
```

```
Z(z) = z^(+1)

Q = Z*P

Q(z) = + z^(+5) + 2*z^(+4) + 3*z^(+3)

R = Z^1 - Z^-1

R(z) = + z^(+1) - z^(-1)
```

**References**   Strang, G.; T. Nguyen (1996), *Wavelets and filter banks*, Wellesley-Cambridge Press.

Sweldens, W. (1998), "The Lifting Scheme: a Construction of Second Generation of Wavelets," *SIAM J. Math. Anal.*, 29 (2), pp. 511–546.

**See Also**   laurmat

# leaves

| | |
|---|---|
| **Purpose** | Determine terminal nodes |
| **Syntax** | `N = leaves(T)`<br>`[N,K] = leaves(T,'sort')`<br>`N = leaves(T,'dp')`<br>`[N,K] = leaves(T,'sortdp')`<br>`[N,K] = leaves(T,'sdp')` |
| **Description** | `N = leaves(T)` returns the indices of terminal nodes of the tree *T* where N is a column vector.<br><br>The nodes are ordered from left to right as in tree *T*.<br><br>`[N,K] = leaves(T,'s')` or `[N,K] = leaves(T,'sort')` returns sorted indices. `M = N(K)` are the indices reordered as in tree *T*, from left to right.<br><br>`N = leaves(T,'dp')` returns a matrix N, which contains the depths and positions of terminal nodes.<br><br>`N(i,1)` is the depth of the i-th terminal node, and `N(i,2)` is the position of the i-th terminal node.<br><br>`[N,K] = leaves(T,'sortdp')` or `[N,K] = leaves(T,'sdp')` returns sorted nodes. |
| **Examples** | ```
% Create initial tree.
ord = 2;
t = ntree(ord,3);        % binary tree of depth 3.
t=nodejoin(t,5);
t=nodejoin(t,4);
plot(t)
``` |

```
% List terminal nodes (index).
tnodes_ind = leaves(t)
tnodes_ind =
     7
     8
     4
     5
    13
    14
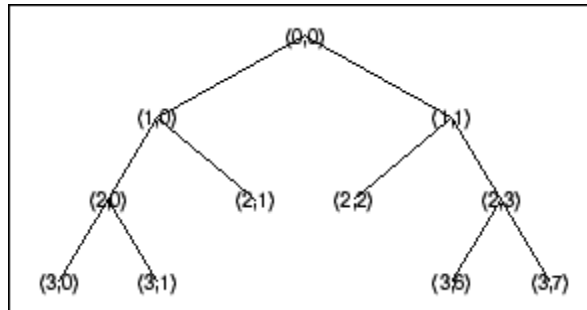
% List terminal nodes (sorted on index).
[tnodes_ind,Ind] = leaves(t,'sort')
tnodes_ind =
     4
     5
     7
     8
    13
    14

Ind =
     3
     4
     1
     2
     5
```

```
          6

% List terminal nodes (Depth_Position).
tnodes_depo = leaves(t,'dp')
tnodes_depo =
     3     0
     3     1
     2     1
     2     2
     3     6
     3     7

% List terminal nodes (sorted on Depth_Position).
[tnodes_depo,Ind] = leaves(t,'sortdp')
tnodes_depo =
     2     1
     2     2
     3     0
     3     1
     3     6
     3     7

Ind =
     3
     4
     1
     2
     5
     6
```

**See Also**     tnodes | noleaves

**Purpose** Apply elementary lifting steps on quadruplet of filters

**Syntax** 
```
[LoDN,HiDN,LoRN,HiRN] = liftfilt(LoD,HiD,LoR,HiR,ELS)
liftfilt(LoD,HiD,LoR,HiR,ELS,TYPE,VALUE)
```

**Description** `[LoDN,HiDN,LoRN,HiRN] = liftfilt(LoD,HiD,LoR,HiR,ELS)` returns the four filters `LoDN`, `HiDN`, `LoRN`, and `HiRN` obtained by an elementary lifting step (ELS) starting from the four filters `LoD`, `HiD`, `LoR`, and `HiR`. The four input filters verify the perfect reconstruction condition.

`ELS` is a structure such that

- `TYPE = ELS.type` contains the type of the elementary lifting step. The valid values for `TYPE` are `'p'` (primal) or `'d'` (dual).

- `VALUE = ELS.value` contains the Laurent polynomial `T` associated with the elementary lifting step (see `laurpoly`). If `VALUE` is a vector, the associated Laurent polynomial `T` is equal to `laurpoly(VALUE,0)`.

In addition, `ELS` may be a scaling step. In that case, `TYPE` is equal to `'s'` (scaling) and `VALUE` is a scalar different from zero.

`liftfilt(LoD,HiD,LoR,HiR,ELS,TYPE,VALUE)` gives the same outputs.

---

**Note** If `TYPE = 'p'`, `HiD` and `LoR` are unchanged.
If `TYPE = 'd'`, `LoD` and `HiR` are unchanged.
If `TYPE = 's'`, the four filters are changed.
If `ELS` is an array of elementary lifting steps, `liftfilt(...,ELS)` performs each step successively.

---

`liftfilt(...,FLAGPLOT)` plots the successive biorthogonal pairs—scaling function and wavelet.

**Examples** 
```
% Get Haar filters.
[LoD,HiD,LoR,HiR] = wfilters('haar');

% Lift the Haar filters.
```

```
twoels(1) = struct('type','p','value',...
laurpoly([0.125 -0.125],0));
twoels(2) = struct('type','p','value',...
laurpoly([0.125 -0.125],1));
[LoDN,HiDN,LoRN,HiRN] = liftfilt(LoD,HiD,LoR,HiR,twoels);

% The biorthogonal wavelet bior1.3 is obtained up to
% an unsignificant sign.
[LoDB,HiDB,LoRB,HiRB] = wfilters('bior1.3');
samewavelet = ...
isequal([LoDB,HiDB,LoRB,HiRB],[LoDN,-HiDN,LoRN,HiRN])

samewavelet =

     1
```

**See Also**      laurpoly

**Purpose**     Lifting schemes

**Syntax**      LS = liftwave(WNAME)
                LS = liftwave(WNAME,'Int2Int')

**Description**     LS = liftwave(WNAME) returns the lifting scheme associated with the
                wavelet specified by WNAME. LS is a structure, not an integer, and used
                by lwt, ilwt, lwt2, etc.

                LS = liftwave(WNAME,'Int2Int') performs an integer to integer
                wavelet transform. Using 'Int2Int' produces an LS such that when
                you use [CA,CD] = lwt(X,LS) or Y = lwt(X,LS) and X is a vector of
                integers, the resulting CA, CD, and Y are vectors of integers. If you omit
                'Int2Int' then lwt produces vectors of real numbers.

                The valid values for WNAME are

| WNAME Values | Comments |
| --- | --- |
| 'lazy' | A "lazy" wavelet is a second-generation wavelet and is not a true mathematical wavelet. |
| 'haar' | Same as 'db1', 'bior1.1', and 'cdf1.1' |
| 'db1', 'db2', 'db3', 'db4', 'db5', 'db6', 'db7', 'db8' | 'db2' same as 'sym2', 'db3', and 'sym4' |
| 'sym2', 'sym3', 'sym4', 'sym5', 'sym6', 'sym7', 'sym8' | |

| WNAME Values | Comments |
|---|---|
| Cohen-Daubechies-Feauveau wavelets<br><br>'cdf1.1','cdf1.3','cdf1.5'<br>'cdf3.1','cdf3.3','cdf3.5'<br>'cdf5.1','cdf5.3','cdf5.5'<br>'cdf2.2','cdf2.4','cdf2.6'<br>'cdf4.2','cdf4.4','cdf4.6'<br>'cdf6.2','cdf6.4','cdf6.6' | 'cdfX.Y' same as 'biorX.Y' except for bior4.4 and bior5.5. |
| 'biorX.Y' | See waveinfo |
| 'rbioX.Y' | Reverse of 'biorX.Y'.<br>See waveinfo |
| 'bs3' | Same as 'cdf4.2' |
| 'rbs3' | Reverse of 'bs3' |
| '9.7' | Same as 'bior4.4' |
| 'r9.7' | Reverse of '9.7' |

For more information about lifting schemes, see lsinfo.

**Examples**
```
% Start from the db2 wavelet and get the
% corresponding lifting scheme.
lsdb2 = liftwave('db2');

% Visualize the obtained lifting scheme.
displs(lsdb2);

lsdb2 = {...
'd'              [ -1.73205081]            [0]
'p'              [ -0.06698730  0.43301270]  [1]
'd'              [  1.00000000]            [-1]
[  1.93185165]  [  0.51763809]            []
```

```
};
```

**See Also**    laurpoly

# localmax

| | |
|---|---|
| **Purpose** | Identify and chain local maxima |

**Syntax**

```
[lmaxima,indices] = localmax(inputmatrix)
[lmaxima,indices] = localmax(inputmatrix,initrow)
[lmaxima,indices] = localmax(inputmatrix,initrow,regflag)
```

**Description**   [lmaxima,indices] = localmax(inputmatrix) identifies and chains the local maxima in the rows of inputmatrix.

[lmaxima,indices] = localmax(inputmatrix,initrow) initializes the chaining of local maxima begining with row initrow. If there are no local maxima in initrow, all rows in lmaxima with indices less than initrow consist of only zeros.

[lmaxima,indices] = localmax(inputmatrix,initrow,regflag) replaces initrow of inputmatrix with the level-5 approximation (scaling) coefficients obtained with the sym4 wavelet.

**Input Arguments**

**inputmatrix**

inputmatrix is a matrix of real or complex numbers. Most often, inputmatrix is a matrix of continuous wavelet transform (CWT) coefficients, and you use localmax to identify maxima lines. localmax operates on the absolute values of inputmatrix.

**initrow**

Initialization row for chaining local maxima. The chaining algorithm begins at initrow and decrements the row index by 1 until the first row of the matrix is reached. By specifying initrow, you can exclude rows from the chaining algorithm.

> **Default:** size(inputmatrix,1)

**regflag**

Regularization flag. If you set regflag to true, the row of inputmatrix corresponding to initrow is replaced by the level-5 approximation (scaling) coefficients obtained with the sym4 wavelet.

# localmax

- If the local maximum in row R lies between two local maxima in row R+1, the value of the local maximum in row R is the higher column index in row R+1.

  To illustrate this, if `inputmatrix` is:

  ```
  0    0    1    0    0    0
  0    1    0    1    0    0
  ```

  `lmaxima` with `initRow = 2` and `regflag = false` is:

  ```
  0    0    4    0    0    0
  0    2    0    4    0    0
  ```

  `lmaxima` with `initRow = 1` and `regflag = false` is:

  ```
  0    0    3    0    0    0
  0    1    0    1    0    0
  ```

**indices**

Linear indices of the nonzero values of `lmaxima`. Use `ind2sub` to convert the linear indices to matrix row and column indices.

**Examples**   Construct a 4-by-4 matrix with local maxima at the following row-column indices: (4,2), (3,3), (2,2), and (1,3). Set `initrow` to 4 and `regflag` to `false`.

```
inputmatrix = ...
[3    2    5    3
 4    6    3    2
 4    4    7    4
 4    6    2    2];
[lmaxima,indices] = localmax(inputmatrix,4,false);
lmaxima
```

Because `localmax` operates on the absolute values of `inputmatrix`, setting `inputmatrix(4,2) = -inputmatrix(4,2)` produces an identical `lmaxima`.

```
inputmatrix(4,2) = -inputmatrix(4,2);
[lmaxima1,indices1] = localmax(inputmatrix,4,false);
isequal(lmaxima,lmaxima1)
```

Determine the local maxima from the CWT of the `cuspamax` signal with the Haar wavelet. Plot the CWT coefficient moduli and the maxima lines.

```
load cuspamax;
x = 1:length(cuspamax);
scales = 1:32;
cfs = cwt(cuspamax,scales,'haar');
[lmaxima,indices] = localmax(cfs,[],false);
[iRow,iCol] = find(lmaxima);
subplot(211);
imagesc(abs(cfs)); axis xy;
axis([1 1024 1 32]);
ylabel('Scale'); title('CWT Coefficients (Moduli)');
subplot(212);
plot(x(iCol),scales(iRow),'marker','o','markerfacecolor',[0 0 1],...
    'linestyle','none');
xlabel('Position'); ylabel('Scale'); title('Maxima Lines');
axis([1 1024 1 32]);
```

CWT Coefficients (Moduli)

Maxima Lines

**Purpose**        Transform lifting scheme to quadruplet of filters

**Syntax**         [LoD,HiD,LoR,HiR] = ls2filt(LS)

**Description**    [LoD,HiD,LoR,HiR] = ls2filt(LS) returns the four filters LoD, HiD,
                   LoR, and HiR associated with the lifting scheme LS.

**Examples**
```
% Start from the db2 wavelet and get the
% corresponding lifting scheme.
LS = liftwave('db2')
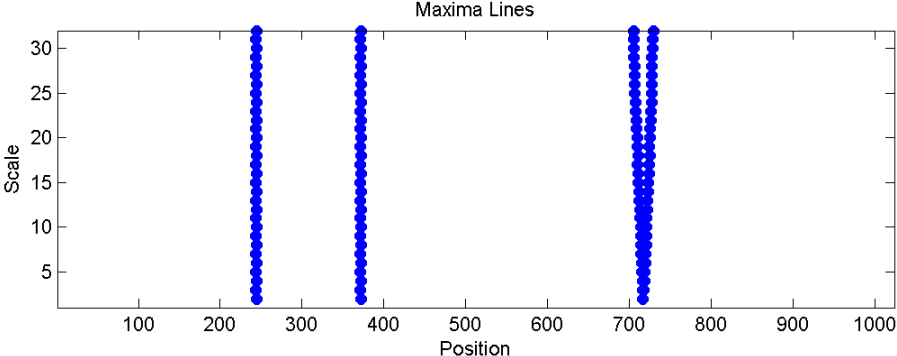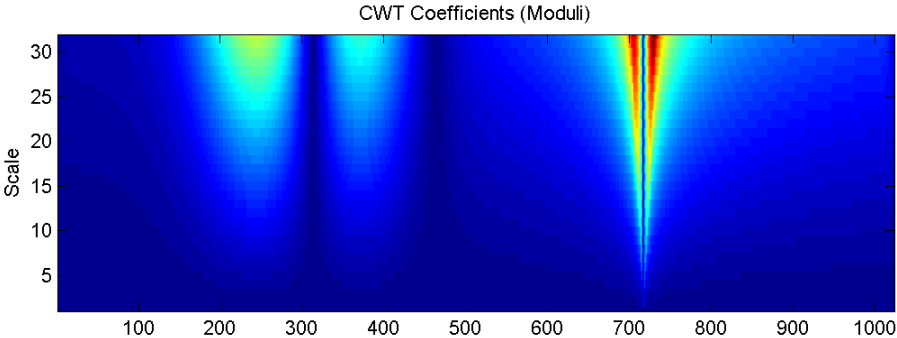
LS =

    'd'         [   -1.7321]    [ 0]
    'p'         [1x2 double]    [ 1]
    'd'         [         1]    [-1]
    [1.9319]    [    0.5176]      []

% Visualize the obtained lifting scheme.

displs(LS);

LS = {...
'd'              [ -1.73205081]                  [0]
'p'              [ -0.06698730   0.43301270]    [1]
'd'              [  1.00000000]                  [-1]
[  1.93185165]   [  0.51763809]                  []
};

% Get the filters from the lifting scheme.

[LoD,HiD,LoR,HiR] = ls2filt(LS)

LoD =

   -0.1294    0.2241    0.8365    0.4830
```

```
HiD =

   -0.4830    0.8365   -0.2241   -0.1294

LoR =

    0.4830    0.8365    0.2241   -0.1294

HiR =

   -0.1294   -0.2241    0.8365   -0.4830

% Get the db2 filters using wfilters.
% You can check the equality.

[LoDref,HiDref,LoRref,HiRref] = wfilters('db2')

LoDref =

   -0.1294    0.2241    0.8365    0.4830

HiDref =

   -0.4830    0.8365   -0.2241   -0.1294

LoRref =

    0.4830    0.8365    0.2241   -0.1294

HiRref =

   -0.1294   -0.2241    0.8365   -0.4830
```

**See Also**       filt2ls | lsinfo

**Purpose**    Lifting schemes information

**Syntax**     `lsinfo`

**Description**   `lsinfo` displays the following information about lifting schemes. A lifting scheme `LS` is a N x 3 cell array. The N-1 first rows of the array are elementary lifting steps (`ELS`). The last row gives the normalization of `LS`.

Each `ELS` has this format:

`{type, coefficients, max_degree}`

where `type` is `'p'` (primal) or `'d'` (dual), `coefficients` is a vector `C` of real numbers defining the coefficients of a Laurent polynomial `P` described below, and `max_degree` is the highest degree `d` of the monomials of `P`.

The Laurent polynomial `P` is of the form

$P(z) = C(1)*z^\wedge d + C(2)*z^\wedge(d-1) + ... + C(m)*z^\wedge(d-m+1)$

The lifting scheme `LS` is such that for

`k = 1:N-1`, `LS{k,:}` is an `ELS`, where

`LS{k,1}` is the lifting type `'p'` (primal) or `'d'` (dual).

`LS{k,2}` is the corresponding lifting filter.

`LS{k,3}` is the highest degree of the Laurent polynomial corresponding to the filter `LS{k,2}`.

`LS{N,1}` is the primal normalization (real number).

`LS{N,2}` is the dual normalization (real number).

`LS{N,3}` is not used.

Usually, the normalizations are such that `LS{N,1}*LS{N,2} = 1`.

For example, the lifting scheme associated with the wavelet db1 is

```
LS = {...
       'd'            [      -1]     [0]
       'p'            [0.5000]       [0]
       [1.4142]       [0.7071]       []
      }
```

**See Also**     displs | laurpoly

**Purpose**    1-D lifting wavelet transform

**Syntax**    [CA,CD] = lwt(*X*,*W*)
X_InPlace = lwt(X,W)
lwt(*X*,*W*,LEVEL)
X_InPlace = lwt(*X*,*W*,LEVEL,'typeDEC',typeDEC)
[CA,CD] = lwt(*X*,*W*,LEVEL,'typeDEC',typeDEC)

**Description**    lwt performs a 1-D lifting wavelet decomposition with respect to a particular lifted wavelet that you specify.

[CA,CD] = lwt(*X*,*W*) computes the approximation coefficients vector CA and detail coefficients vector CD, obtained by a lifting wavelet decomposition, of the vector *X*. *W* is a lifted wavelet name (see liftwave).

X_InPlace = lwt(X,W) computes the approximation and detail coefficients. These coefficients are stored in place:

CA = X_InPlace(1:2:end) and CD = X_InPlace(2:2:end)

lwt(*X*,*W*,LEVEL) computes the lifting wavelet decomposition at level LEVEL.

X_InPlace = lwt(*X*,*W*,LEVEL,'typeDEC',typeDEC) or [CA,CD] = lwt(*X*,*W*,LEVEL,'typeDEC',typeDEC) with typeDEC = 'w' or 'wp' computes the wavelet or the wavelet packet decomposition using lifting, at level LEVEL.

Instead of a lifted wavelet name, you may use the associated lifting scheme LS: lwt(X,LS,...) instead of lwt(X,W,...).

For more information about lifting schemes, see lsinfo.

**Examples**    
```
% Start from the Haar wavelet and get the
% corresponding lifting scheme.
lshaar = liftwave('haar');

% Add a primal ELS to the lifting scheme.
els = {'p',[-0.125 0.125],0};
lsnew = addlift(lshaar,els);
```

```
% Perform LWT at level 1 of a simple signal.
x = 1:8;
[cA,cD] = lwt(x,lsnew)

cA =

    1.9445    4.9497    7.7782   10.6066


cD =

    0.7071    0.7071    0.7071    0.7071

% Perform integer LWT of the same signal.
lshaarInt = liftwave('haar','int2int');
lsnewInt = addlift(lshaarInt,els);
[cAint,cDint] = lwt(x,lsnewInt)

cAint =

     1     3     5     7


cDint =

     1     1     1     1
```

**Algorithms**    This function uses the polyphase algorithm.

lwt reduces to dwt with zero-padding extension mode and without extra-coefficients.

**References**    Strang, G.; T. Nguyen (1996), *Wavelets and filter banks*, Wellesley-Cambridge Press.

Sweldens, W. (1998), "The Lifting Scheme: a Construction of Second Generation of Wavelets," *SIAM J. Math. Anal.*, 29 (2), pp. 511–546.

**See Also**       ilwt

# lwt2

**Purpose**　2-D lifting wavelet transform

**Syntax**
```
[CA,CH,CV,CD] = lwt2(X,W)
X_InPlace = lwt2(X,LS)
lwt2(X,W,LEVEL)
X_InPlace = lwt2(X,W,LEVEL,'typeDEC',typeDEC)
[CA,CH,CV,CD] = LWT2(X,W,LEVEL,'typeDEC',typeDEC)
```

**Description**　lwt2 performs a 2-D lifting wavelet decomposition with respect to a particular lifted wavelet that you specify.

`[CA,CH,CV,CD] = lwt2(X,W)` computes the approximation coefficients matrix CA and detail coefficients matrices CH, CV, and CD, obtained by a lifting wavelet decomposition, of the matrix *X*. *W* is a lifted wavelet name (see liftwave).

`X_InPlace = lwt2(X,LS)` computes the approximation and detail coefficients. These coefficients are stored in place:

- `CA = X_InPlace(1:2:end,1:2:end)`

- `CH = X_InPlace(2:2:end,1:2:end)`

- `CV = X_InPlace(1:2:end,2:2:end)`

- `CD = X_InPlace(2:2:end,2:2:end)`

`lwt2(X,W,LEVEL)` computes the lifting wavelet decomposition at level LEVEL.

`X_InPlace = lwt2(X,W,LEVEL,'typeDEC',typeDEC)` or `[CA,CH,CV,CD] = LWT2(X,W,LEVEL,'typeDEC',typeDEC)` with typeDEC = 'w' or 'wp' computes the wavelet or the wavelet packet decomposition using lifting, at level LEVEL.

Instead of a lifted wavelet name, you may use the associated lifting scheme LS: `lwt2(X,LS,...)` instead of `LWT2(X,W,...)`.

For more information about lifting schemes, see lsinfo.

**Tips**

When X represents an indexed image, X, as well as the output arrays cA,cH,cV,cD, or X_InPlace are m-by-n matrices. When X represents a truecolor image, it is an m-by-n-by-3 array, where each m-by-n matrix represents a red, green, or blue color plane concatenated along the third dimension.

For more information on image formats, see the image and imfinfo reference pages .

**Examples**

```
% Start from the Haar wavelet and get the
% corresponding lifting scheme.
lshaar = liftwave('haar');

% Add a primal ELS to the lifting scheme.
els = {'p',[-0.125 0.125],0};
lsnew = addlift(lshaar,els);

% Perform LWT at level 1 of a simple image.
x = reshape(1:16,4,4);
[cA,cH,cV,cD] = lwt2(x,lsnew)

cA =

    5.7500   22.7500
   10.0000   27.0000


cH =

    1.0000    1.0000
    1.0000    1.0000


cV =

    4.0000    4.0000
    4.0000    4.0000
```

```
cD =

     0     0
     0     0

% Perform integer LWT of the same image.
lshaarInt = liftwave('haar','int2int');
lsnewInt = addlift(lshaarInt,els);
[cAint,cHint,cVint,cDint] = lwt2(x,lsnewInt)

cAint =

     3    11
     5    13


cHint =

     1     1
     1     1


cVint =

     4     4
     4     4


cDint =

     0     0
     0     0
```

**Algorithms**  This function implements the polyphase algorithm.

lwt reduces to dwt with zero-padding extension mode and without extra-coefficients.

**References**  Strang, G.; T. Nguyen (1996), *Wavelets and filter banks*, Wellesley-Cambridge Press.

Sweldens, W. (1998), "The Lifting Scheme: a Construction of Second Generation of Wavelets," *SIAM J. Math. Anal.*, 29 (2), pp. 511–546.

**See Also**  ilwt2

# lwtcoef

**Purpose**    Extract or reconstruct 1-D LWT wavelet coefficients

**Syntax**     Y = lwtcoef(TYPE,XDEC,LS,LEVEL,LEVEXT)
               Y = lwtcoef(TYPE,XDEC,W,LEVEL,LEVEXT)

**Description**  Y = lwtcoef(TYPE,XDEC,LS,LEVEL,LEVEXT) returns the coefficients or
the reconstructed coefficients of level LEVEXT, extracted from XDEC, the
LWT decomposition at level LEVEL obtained with the lifting scheme LS.

The valid values for TYPE are

| TYPE Values | Description |
|---|---|
| 'a' | Approximations |
| 'd' | Details |
| 'ca' | Coefficients of approximations |
| 'cd' | Coefficients of details |

Y = lwtcoef(TYPE,XDEC,W,LEVEL,LEVEXT) returns the same output
using W, which is the name of a lifted wavelet.

**Examples**   ```
% Start from the Haar wavelet and get the
% corresponding lifting scheme.
lshaar = liftwave('haar');

% Add a primal ELS to the lifting scheme.
els = {'p',[-0.125 0.125],0};
lsnew = addlift(lshaar,els);

% Perform LWT at level 2 of a simple signal.
x = 1:8;
xDec = lwt(x,lsnew,2)

xDec =
```

```
      4.3438    0.7071    2.1250    0.7071   13.0313    0.7071
      2.0000    0.7071

% Extract approximation coefficients of level 1.
ca1 = lwtcoef('ca',xDec,lsnew,2,1)

ca1 =

    1.9445    4.9497    7.7782   10.6066

% Reconstruct approximations and details.
a1 = lwtcoef('a',xDec,lsnew,2,1)

a1 =

    1.3750    1.3750    3.5000    3.5000    5.5000    5.5000
    7.5000    7.5000

a2 = lwtcoef('a',xDec,lsnew,2,2)

a2 =

    2.1719    2.1719    2.1719    2.1719    6.5156    6.5156
    6.5156    6.5156

d1 = lwtcoef('d',xDec,lsnew,2,1)

d1 =

   -0.3750    0.6250   -0.5000    0.5000   -0.5000    0.5000
   -0.5000    0.5000

d2 = lwtcoef('d',xDec,lsnew,2,2)

d2 =

   -0.7969   -0.7969    1.3281    1.3281   -1.0156   -1.0156
```

```
      0.9844    0.9844

% Check perfect reconstruction.
err = max(abs(x-a2-d2-d1))

err =

  9.9920e-016
```

**See Also**    `ilwt | lwt`

**Purpose**      Extract or reconstruct 2-D LWT wavelet coefficients

**Syntax**       Y = lwtcoef2(TYPE,XDEC,LS,LEVEL,LEVEXT)
                 Y = lwtcoef2(TYPE,XDEC,W,LEVEL,LEVEXT)

**Description**  Y = lwtcoef2(TYPE,XDEC,LS,LEVEL,LEVEXT) returns the coefficients
                 or the reconstructed coefficients of level LEVEXT, extracted from XDEC,
                 the LWT decomposition at level LEVEL obtained with the lifting scheme
                 LS.

                 The valid values for TYPE are listed in this table.

| TYPE Values | Description |
|-------------|-------------|
| 'a'  | Approximations |
| 'h'  | Horizontal details |
| 'v'  | Vertical details |
| 'd'  | Diagonal details |
| 'ca' | Coefficients of approximations |
| 'ch' | Coefficients of horizontal details |
| 'cv' | Coefficients of vertical details |
| 'cd' | Coefficients of diagonal details |

                 Y = lwtcoef2(TYPE,XDEC,W,LEVEL,LEVEXT) returns the same output
                 using W, which is the name of a lifted wavelet.

**Tips**         If XDEC is obtained from an indexed image analysis or a truecolor image
                 analysis, it is an m-by-n matrix or an m-by-n-by-3 array, respectively.

                 For more information on image formats, see the image and imfinfo
                 reference pages.

**Examples**     % Start from the Haar wavelet and get the
                 % corresponding lifting scheme.

```
lshaar = liftwave('haar');

% Add a primal ELS to the lifting scheme.
els = {'p',[-0.125 0.125],0};
lsnew = addlift(lshaar,els);

% Perform LWT at level 2 of a simple image.
x = reshape(1:16,4,4);
xDec = lwt2(x,lsnew,2)

xDec =

    27.4375    4.0000   17.0000    4.0000
     1.0000         0    1.0000         0
     4.2500    4.0000    0.0000    4.0000
     1.0000         0    1.0000         0

% Extract approximation coefficients of level 1.
ca1 = lwtcoef2('ca',xDec,lsnew,2,1)

ca1 =

     5.7500   22.7500
    10.0000   27.0000

% Reconstruct approximations and details.
a1 = lwtcoef2('a',xDec,lsnew,2,1)

a1 =

     2.8750    2.8750   11.3750   11.3750
     2.8750    2.8750   11.3750   11.3750
     5.0000    5.0000   13.5000   13.5000
     5.0000    5.0000   13.5000   13.5000

a2 = lwtcoef2('a',xDec,lsnew,2,2)
```

```
a2 =

    6.8594    6.8594    6.8594    6.8594
    6.8594    6.8594    6.8594    6.8594
    6.8594    6.8594    6.8594    6.8594
    6.8594    6.8594    6.8594    6.8594

h1 = lwtcoef2('h',xDec,lsnew,2,1)

h1 =

   -0.3750   -0.3750   -0.3750   -0.3750
    0.6250    0.6250    0.6250    0.6250
   -0.5000   -0.5000   -0.5000   -0.5000
    0.5000    0.5000    0.5000    0.5000

v1 = lwtcoef2('v',xDec,lsnew,2,1)

v1 =

   -1.5000    2.5000   -2.0000    2.0000
   -1.5000    2.5000   -2.0000    2.0000
   -1.5000    2.5000   -2.0000    2.0000
   -1.5000    2.5000   -2.0000    2.0000

d1 = lwtcoef2('d',xDec,lsnew,2,1)

d1 =

    0    0    0    0
    0    0    0    0
    0    0    0    0
    0    0    0    0

h2 = lwtcoef2('h',xDec,lsnew,2,2)

h2 =
```

```
                  -0.7969    -0.7969    -0.7969    -0.7969
                  -0.7969    -0.7969    -0.7969    -0.7969
                   1.3281     1.3281     1.3281     1.3281
                   1.3281     1.3281     1.3281     1.3281

    v2 = lwtcoef2('v',xDec,lsnew,2,2)

    v2 =

                  -3.1875    -3.1875     5.3125     5.3125
                  -3.1875    -3.1875     5.3125     5.3125
                  -3.1875    -3.1875     5.3125     5.3125
                  -3.1875    -3.1875     5.3125     5.3125

    d2 = lwtcoef2('d',xDec,lsnew,2,2)

    d2 =

      1.0e-015 *

                   0.2498     0.2498    -0.4163    -0.4163
                   0.2498     0.2498    -0.4163    -0.4163
                  -0.4163    -0.4163     0.6939     0.6939
                  -0.4163    -0.4163     0.6939     0.6939

    % Check perfect reconstruction.
    err = max(max(abs(x-a2-h2-v2-d2-h1-v1-d1)))

    err =

      3.5527e-015
```

**See Also**     ilwt2 | lwt2

| **Purpose** | Multisignals 1-D clustering |
|---|---|

**Syntax**

```
S = mdwtcluster(X)
S = mdwtcluster(X,'PropName1',PropVal1,'PropName2',PropVal2,...)
```

**Description**    S = mdwtcluster(X) constructs clusters from a hierarchical cluster tree. The input matrix *X* is decomposed in row direction using the DWT function with the haar wavelet and the maximum allowed level.

```
S =
mdwtcluster(X,'PropName1',PropVal1,'PropName2',PropVal2,...)
```
allows you to modify some properties. The valid choices for *PropName* are:

---

**Note** mdwtcluster requires the Statistics Toolbox™

---

| 'dirDec' | 'r' (row) or 'c' (column). Default value is 'r'. |
|---|---|
| 'level' | Level of the DWT decomposition. Default value is:<br>level=fix(log2(size(X,d)))<br>where d=1 or d=2, depending on the dirDec value. |
| 'wname' | Wavelet name used for DWT. Default value is 'haar'. |
| 'dwtEXTM' | DWT extension mode (see dwtmode). |
| 'pdist' | See Statistics Toolbox pdist function. Default value is 'euclidean'. |
| 'linkage' | See Statistics Toolbox linkage function. Default value is 'ward'. |

| `'maxclust'` | Number of clusters. Default value is 6. The input variable can be a vector. |
|---|---|
| `'lst2clu'` | Cell array that contains the list of data to classify. |
| | If N is the level of decomposition, the allowed name values for the cells are: |
| | • `'s'` — Signal |
| | • `'aj'` — Approximation at level j |
| | • `'dj'` — Detail at level j |
| | • `'caj'` — Coefficients of approximation at level j |
| | • `'cdj'` — Coefficients of detail at level j |
| | Default value is `{'s';'ca1';...;'caN'}`. |

The output structure S is such that for each partition `j`:

| `S.Idx(:,j)` | Contains the cluster numbers obtained from the hierarchical cluster tree (see `cluster` in the Statistics Toolbox software). |
|---|---|
| `S.Incons(:,j)` | Contains the inconsistent values of each non-leaf node in the hierarchical cluster tree (see Statistics Toolbox software function `inconsistent`). |
| `S.Corr(j)` | Contains the cophenetic correlation coefficients of the partition (see Statistics Toolbox software function `cophenet`). |

> **Note** If maxclustVal is a vector, then IdxCLU is a multidimensional
> array such that IdxCLU(:,j,k) contains the cluster numbers obtained
> from the hierarchical cluster tree for k clusters.

**Examples**

```
load elecsig10
lst2clu = {'s','ca1','ca3','ca6'};

% Compute the structure resulting from multisignal clustering
S = mdwtcluster(signals,'maxclust',4,'lst2clu',lst2clu)

S =

    IdxCLU: [70x4 double]
    Incons: [69x4 double]
      Corr: [0.7920 0.7926 0.7947 0.7631]

% Retrieve indices of clusters
IdxCLU = S.IdxCLU;

% Plot the first cluster
plot(signals(IdxCLU(:,1)==1,:)','r');
hold on;

% Plot the third clustering
plot(signals(IdxCLU(:,1)==3,:)','b')
```
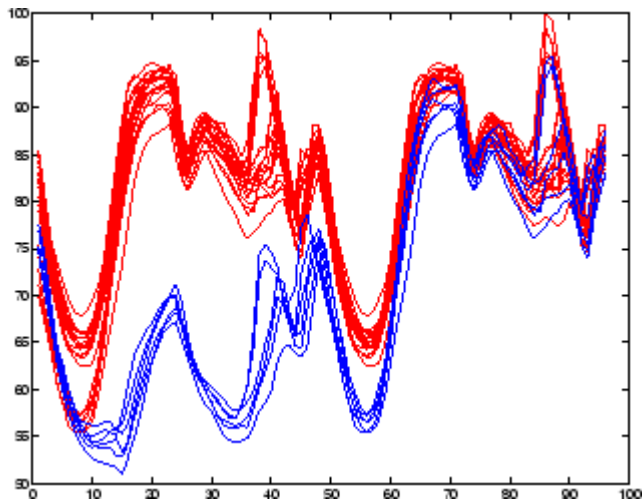
```
% Check the equality of partitions
equalPART = isequal(IdxCLU(:,1),IdxCLU(:,3))

equalPART =

      1

% So we can see that we obtain the same partitions using
% coefficents of approximation at level 3 instead of original
% signals. Much less information is then used.
```

**See Also**    mdwtdec | wavedec

| **Purpose** | Multisignal 1-D wavelet decomposition |

**Syntax**

```
DEC = mdwtdec(DIRDEC,X,LEV,WNAME)
DEC = mdwtdec(DIRDEC,X,LEV,LoD,HiD,LoR,HiR)
DEC = mdwtdec(...,'mode',EXTMODE)
```

**Description**

DEC = mdwtdec(DIRDEC,X,LEV,WNAME) returns the wavelet decomposition at level LEV of each row (if DIRDEC = 'r') or each column (if DIRDEC = 'c') of matrix *X*, using the wavelet WNAME.

The output DEC is a structure with the following fields:

| 'dirDec' | Direction indicator: 'r' (row) or 'c' (column) |
| --- | --- |
| 'level' | Level of the DWT decomposition |
| 'wname' | Wavelet name |
| 'dwtFilters' | Structure with four fields LoD, HiD, LoR, and HiR |
| 'dwtEXTM' | DWT extension mode (see dwtmode) |
| 'dwtShift' | DWT shift parameter (0 or 1) |
| 'dataSize' | Size of X |
| 'ca' | Approximation coefficients at level LEV |
| 'cd' | Cell array of detail coefficients, from level 1 to level LEV |

Coefficients cA and cD{k} (for k = 1 to LEV) are matrices and are stored in rows if DIRDEC = 'r' or in columns if DIRDEC = 'c'.

DEC = mdwtdec(DIRDEC,X,LEV,LoD,HiD,LoR,HiR) uses the four filters instead of the wavelet name.

DEC = mdwtdec(...,'mode',EXTMODE) computes the wavelet decomposition with the EXTMODE extension mode that you specify (see dwtmode for the valid extension modes).

# mdwtdec

**Examples**
```
% Load original 1D-multisignal.
load thinker

% Perform a decomposition at level 2 using wavelet db2.
dec = mdwtdec('r',X,2,'db2')
dec =
        dirDec: 'r'
         level: 2
         wname: 'db2'
     dwtFilters: [1x1 struct]
       dwtEXTM: 'sym'
       dwtShift: 0
       dataSize: [192 96]
             ca: [192x26 double]
             cd: {[192x49 double]  [192x26 double]}

% Compute the associated filters of db2 wavelet.
[LoD,HiD,LoR,HiR] = wfilters('db2');

% Perform a decomposition at level 2 using filters.
decBIS = mdwtdec('r',X,2,LoD,HiD,LoR,HiR)

decBIS =
        dirDec: 'r'
         level: 2
         wname: ''
     dwtFilters: [1x1 struct]
       dwtEXTM: 'sym'
       dwtShift: 0
       dataSize: [192 96]
             ca: [192x26 double]
             cd: {[192x49 double]  [192x26 double]}
```

**References** Daubechies, I. , *Ten lectures on wavelets*, CBMS-NSF conference series in applied mathematics. SIAM Ed., 1992.

Mallat, S., "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Pattern Anal. and Machine Intell.*, vol. 11, no. 7, 1989, pp. 674–693.

Meyer, Y. , *Ondelettes et opérateurs*, Tome 1, Hermann Ed. (English translation: *Wavelets and operators*, Cambridge Univ. Press. 1993.)

**See Also**     mdwtdec | wavedec

# mdwtrec

**Purpose**　　　Multisignal 1-D wavelet reconstruction

**Syntax**　　　```
X = mdwtrec(DEC)
X = mdwtrec(DEC,IDXSIG)
Y = mdwtrec(DEC,TYPE,LEV)
A = mdwtrec(DEC,'a')
A = mdwtrec(DEC,'a',LEVDEC)
D = mdwtrec(DEC,'d')
CA = mdwtrec(DEC,'ca')
CA = mdwtrec(DEC,'ca',LEVDEC)
CD = mdwtrec(DEC,'cd',MODE)
CFS = mdwtrec(DEC,'cfs',MODE)
Y = mdwtrec(...,IDXSIG)
```

**Description**　　X = mdwtrec(DEC) returns the original matrix of signals, starting from the wavelet decomposition structure DEC (see mdwtdec).

X = mdwtrec(DEC,IDXSIG) reconstructs the signals whose indices are given by the vector IDXSIG.

Y = mdwtrec(DEC,TYPE,LEV) extracts or reconstructs the detail or approximation coefficients at level LEV depending on the TYPE value. The maximum value for LEV is LEVDEC = DEC.level.

When TYPE is equal to:

- 'cd' or 'ca', coefficients of level LEV are extracted.

- 'd' or 'a', coefficients of level LEV are reconstructed.

- 'a' or 'ca', LEV must be such that $0 \leq LEV \leq LEVDEC$.

- 'd' or 'cd', LEV must be such that $1 \leq LEV \leq LEVDEC$.

A = mdwtrec(DEC,'a') is equivalent to A = mdwtrec(DEC,'a',LEVDEC).

D = mdwtrec(DEC,'d') returns a matrix containing the sum of all the details, so that X = A + D.

CA = mdwtrec(DEC,'ca') is equivalent to CA = mdwtrec(DEC,'ca',LEVDEC).

CD = mdwtrec(DEC,'cd',MODE) returns a matrix containing all the detail coefficients.

CFS = mdwtrec(DEC,'cfs',MODE) returns a matrix containing all the coefficients.

For MODE = 'descend' the coefficients are concatened from level LEVDEC to level 1 and MODE = 'descend' concatenates from level 1 to level LEVDEC). The default is MODE = 'descend'. The concatenation is made row-wise if DEC.dirDEC = 'r' or column-wise if DEC.dirDEC = 'c'.

Y = mdwtrec(...,IDXSIG) extracts or reconstructs the detail or the approximation coefficients for the signals whose indices are given by the vector IDXSIG.

**Examples**
```
% Load original 1D-multisignal.
load thinker

% Perform a decomposition at level 2 using wavelet db2.
dec = mdwtdec('r',X,2,'db2');

% Reconstruct the original matrix of signals, starting from
% the wavelet decomposition structure dec.
XR = mdwtrec(dec);

% Compute the reconstruction error.
errREC = max(max(abs(X-XR)))

errREC =
  2.1026e-010

% Reconstruct the original signal 31, the corresponding
% approximation at level 2, details at levels 1 and 2.
Y = mdwtrec(dec,31);
A2 = mdwtrec(dec,'a',2,31);
D2 = mdwtrec(dec,'d',2,31);
D1 = mdwtrec(dec,'d',1,31);
```

```
% Compute the reconstruction error for signal 31.
errREC = max(abs(Y-A2-D2-D1))

errREC =
  6.8390e-014
```

**References**    Daubechies, I., *Ten lectures on wavelets*, CBMS-NSF conference series in applied mathematics. SIAM Ed., 1992.

Mallat, S., "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Pattern Anal. and Machine Intell.*, vol. 11, no. 7, 1989, pp. 674–693.

Meyer, Y., *Ondelettes et opérateurs*, Tome 1, Hermann Ed. (English translation: *Wavelets and operators*, Cambridge Univ. Press. 1993.)

**See Also**    mdwtdec | waverec

**Purpose**     Approximation quality metrics

**Syntax**      [PSNR,MSE,MAXERR,L2RAT] = measerr(X,XAPP)
                [...] = measerr(...,BPS)

**Description**  [PSNR,MSE,MAXERR,L2RAT] = measerr(X,XAPP) returns the peak
                signal-to-noise ratio, PSNR, mean square error, MSE, maximum squared
                error, MAXERR, and ratio of squared norms, L2RAT, for an input signal
                or image, X, and its approximation, XAPP.

                [...] = measerr(...,BPS) uses the bits per sample, BPS, to
                determine the peak signal-to-noise ratio.

**Input**       **X**
**Arguments**
                X is a real-valued signal or image.

                **XAPP**

                XAPP is a real-valued signal or image approximation with a size equal
                to that of the input data, X.

                **BPS**

                BPS is the number of bits per sample in the data.

                    **Default:** 8

**Output**      **PSNR**
**Arguments**
                PSNR is the peak signal-to-noise ratio in decibels (dB). The PSNR is
                only meaningful for data encoded in terms of bits per sample, or bits
                per pixel. For example, an image with 8 bits per pixel contains integers
                from 0 to 255.

                **MSE**

The mean square error (MSE) is the squared norm of the difference between the data and the approximation divided by the number of elements.

### MAXERR

MAXERR is the maximum absolute squared deviation of the data, X, from the approximation, XAPP.

### L2RAT

L2RAT is the ratio of the squared norm of the signal or image approximation, XAPP, to the input signal or image, X.

**Definitions**

### Peak Signal to Noise Ratio (PSNR)

The following equation defines the PSNR:

$$20\log_{10}(\frac{2^B - 1}{\sqrt{MSE}})$$

where *MSE* represents the mean square error and *B* represents the bits per sample.

### Mean Square Error (MSE)

The mean square error between a signal or image, *X*, and an approximation, *Y*, is the squared norm of the difference divided by the number of elements in the signal or image:

$$\frac{||X - Y||^2}{N}$$

**Examples**

Approximate an image and calculate approximation quality metrics.

```
load woman;
Xapp = X;
Xapp(X<=50) = 1;
```

```
[psnr,mse,maxerr,L2rat] = measerr(X,Xapp);
figure; colormap(map);
subplot(1,2,1); image(X);
subplot(1,2,2); image(Xapp);
```

Measure approximation quality in an RGB image.

```
X = imread('africasculpt.jpg');
Xapp = X;
Xapp(X<=100) = 1;
[psnr,mse,maxerr,L2rat] = measerr (X,Xapp)
figure;
subplot(1,2,1); image(X);
subplot(1,2,2); image(Xapp);
```

**References**       Huynh-Thu, Q.*Scope of validity of PSNR in image/video quality assessment*, Electronics Letters, 44, 2008, pp. 800–801.

**See Also**        wden | wdencmp

**Tutorials**       • "Data Compression"

• "Denoising and Nonparametric Function Estimation"

# mexihat

**Purpose**      Mexican hat wavelet

**Syntax**       [PSI,X] = mexihat(LB,UB,N)

**Description**  [PSI,X] = mexihat(LB,UB,N) returns values of the Mexican hat
                 wavelet on an N point regular grid, X, in the interval [LB,UB].

                 Output arguments are the wavelet function PSI computed on the grid X.

                 This wavelet has [-5 5] as effective support. Although [-5 5] is the
                 correct theoretical effective support, a wider effective support, [-8 8], is
                 used in the computation to provide more accurate results.

                 This function is proportional to the second derivative function of the
                 Gaussian probability density function.

**Examples**     ```
                 % Set effective support and grid parameters.
                 lb = -5; ub = 5; n = 1000;

                 % Compute and plot Mexican hat wavelet.
                 [psi,x] = mexihat(lb,ub,n);
                 plot(x,psi), title('Mexican hat wavelet')
                 ```
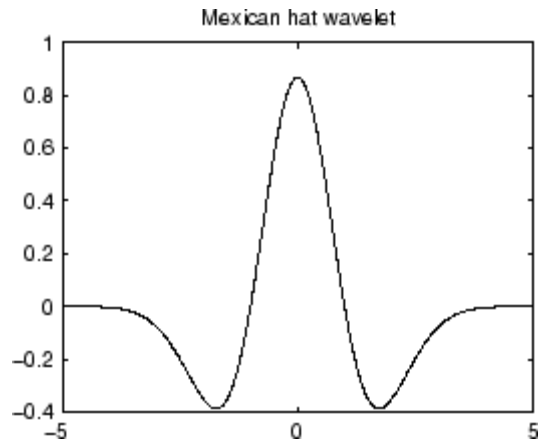
**See Also**     waveinfo

# meyer

**Purpose**      Meyer wavelet

**Syntax**       [PHI,PSI,T] = meyer(LB,UB,*N*)

**Description**  [PHI,PSI,T] = meyer(LB,UB,*N*) returns Meyer scaling and wavelet functions evaluated on an *N* point regular grid in the interval [LB,UB].

*N* must be a power of two.

Output arguments are the scaling function PHI and the wavelet function PSI computed on the grid *T*. These functions have [-8 8] as effective support.

If only one function is required, a fourth argument is allowed:

```
[PHI,T] = meyer(LB,UB,N,'phi')
[PSI,T] = meyer(LB,UB,N,'psi')
```

When the fourth argument is used, but not equal to 'phi' or 'psi', outputs are the same as in the main option.

The Meyer wavelet and scaling function are defined in the frequency domain.

By changing the auxiliary function (see meyeraux for more information), you get a family of different wavelets.

**Examples**
```
% Set effective support and grid parameters.
lb = -8; ub = 8; n = 1024;

% Compute and plot Meyer wavelet and scaling functions.
[phi,psi,x] = meyer(lb,ub,n);
subplot(211), plot(x,psi)
title('Meyer wavelet')
subplot(212), plot(x,phi)
title('Meyer scaling function')
```

Meyer wavelet

Meyer scaling function
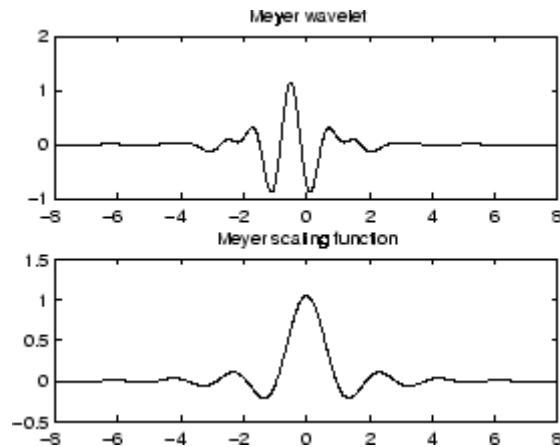
**Algorithms**    Starting from an explicit form of the Fourier transform $\hat{\phi}$ of φ, meyer

computes the values of $\hat{\phi}$ on a regular grid, and then the values of φ are computed using instdfft, the inverse nonstandard discrete FFT.

The procedure for ψ is along the same lines.

**References**    Daubechies, I. (1992), *Ten lectures on wavelets*, CBMS-NSF conference series in applied mathematics, SIAM Ed., pp. 117–119, 137, 152.

**See Also**    meyeraux | wavefun | waveinfo

# meyeraux

**Purpose**          Meyer wavelet auxiliary function

**Syntax**           Y = meyeraux(*X*)

**Description**      Y = meyeraux(*X*) returns values of the auxiliary function used for
                     Meyer wavelet generation evaluated at the elements of the vector or
                     matrix *X*.

                     The function is

                     $$35x^4 - 84x^5 + 70x^6 - 20x^7$$

**See Also**         meyer

**Purpose**      Morlet wavelet

**Syntax**       [PSI,X] = morlet(LB,UB,N)

**Description**  [PSI,X] = morlet(LB,UB,N) returns values of the Morlet wavelet on an N point regular grid in the interval [LB,UB].

Output arguments are the wavelet function PSI computed on the grid X, and the grid X.

This wavelet has [-4 4] as effective support. Although [-4 4] is the correct theoretical effective support, a wider effective support, [-8 8], is used in the computation to provide more accurate results.

$$\psi(x) = e^{-x^2/2}\cos(5x)$$

**Examples**
```
% Set effective support and grid parameters.
lb = -4; ub = 4; n = 1000;
% Compute and plot Morlet wavelet.
[psi,x] = morlet(lb,ub,n);
plot(x,psi), title('Morlet wavelet')
```
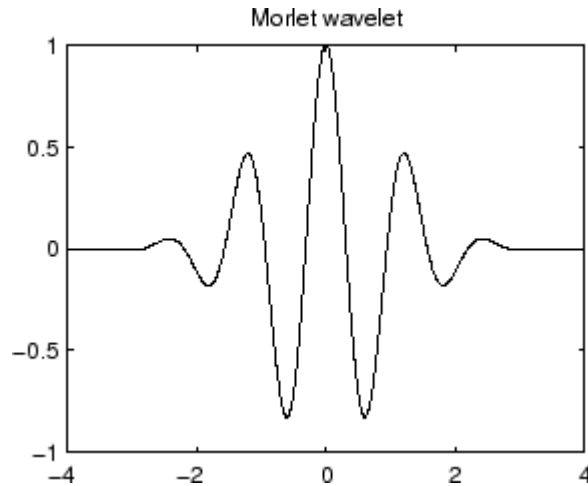


Morlet wavelet

# morlet

**See Also**     waveinfo

**Purpose**     Multisignal 1-D compression using wavelets

**Syntax**
```
[XC,DECCMP,THRESH] = mswcmp('cmp',DEC,METH)
[XC,DECCMP,THRESH] = mswcmp('cmp',DEC,METH,PARAM)
[XC,THRESH] = mswcmp('cmpsig',...)
[DECCMP,THRESH] = mswcmp('cmpdec',...)
THRESH = mswcmp('thr',...)
[...] = mswcmp(OPTION,DIRDEC,X,WNAME,LEV,METH)
 [...] = mswcmp(OPTION,DIRDEC,X,WNAME,LEV,METH,PARAM)
[...] = mswcmp(...,S_OR_H)
 [...] = mswcmp(...,S_OR_H,KEEPAPP)
 [...] = mswcmp(...,S_OR_H,KEEPAPP,IDXSIG)
```

**Description**    mswcmp computes thresholds and, depending on the selected option, performs compression of 1-D signals using wavelets.

[XC,DECCMP,THRESH] = mswcmp('cmp',DEC,METH) or
[XC,DECCMP,THRESH] = mswcmp('cmp',DEC,METH,PARAM) returns
a compressed (indicated by 'cmp' input) version XC of the original
multisignal matrix X, whose wavelet decomposition structure is DEC.
The output XC is obtained by thresholding the wavelet coefficients:
DECCMP, which is the wavelet decomposition associated with XC (see
mdwtdec), and THRESH is the matrix of threshold values. The input METH
is the name of the compression method and PARAM is the associated
parameter, if required.

Valid compression methods METH are shown in the following tables.
For methods that use an associated parameter, the range of allowable
PARAM values is also shown.

| | |
|---|---|
| 'rem_n0' | Remove near 0 |
| 'bal_sn' | Balance sparsity-norm |
| 'sqrtbal_sn' | Balance sparsity-norm (sqrt) |
| 'scarce' | Scarce, PARAM (any number) |
| 'scarcehi' | Scarce high, $2.5 \leq$ PARAM $\leq 10$ |

| `'scarceme'` | Scarce medium, $1.5 \leq$ PARAM $\leq 2.5$ |
|---|---|
| `'scarcelo'` | Scarce low, $1 \leq$ PARAM $\leq 2$ |

PARAM is a sparsity parameter, and it should be such that: $1 \leq$ PARAM $\leq 10$. For scarce method no control is done.

| `'L2_perf'` | Energy ratio |
|---|---|
| `'N0_perf'` | Zero coefficients ratio |

PARAM is a real number which represents the required performance:

$0 \leq$ PARAM $\leq 100$.

| `'glb_thr'` | Global threshold |
|---|---|

PARAM is a real positive number.

| `'man_thr'` | Manual method |
|---|---|

PARAM is an NbSIG-by-NbLEV matrix or NbSIG-by-(NbLEV+1) matrix such that:

- - PARAM(i,j) is the threshold for the detail coefficients of level j for the ith signal ($1 \leq$ j $\leq$ NbLEV).

- - PARAM(i,NbLEV+1) is the threshold for the approximation coefficients for the ith signal (if KEEPAPP is 0).

Where NbSIG is the number of signals and NbLEV the number of levels of decomposition.

```
[XC,THRESH] = mswcmp('cmpsig',...) or
[DECCMP,THRESH] = mswcmp('cmpdec',...) or
THRESH = mswcmp('thr',...)   Instead of the 'cmp' input OPTION,
you can use 'cmpsig', 'cmpdec' or 'thr' to select other output
```

arguments. `'thr'` returns the computed thresholds, but compression is not performed.

`[...]  = mswcmp(OPTION,DIRDEC,X,WNAME,LEV,METH)`
`[...]  = mswcmp(OPTION,DIRDEC,X,WNAME,LEV,METH,PARAM)` The decomposition structure input argument DEC can be replaced by four arguments: DIRDEC, X, WNAME, and LEV. Before performing a compression or computing thresholds, the multisignal matrix X is decomposed at level LEV using the wavelet WNAME, in the direction DIRDEC.

`[...]  = mswcmp(...,S_OR_H)`
`[...]  = mswcmp(...,S_OR_H,KEEPAPP)`
`[...]  = mswcmp(...,S_OR_H,KEEPAPP,IDXSIG)` Three more optional inputs may be used:

- S_OR_H (`'s'` or `'h'`) stands for soft or hard thresholding (see mswthresh for more details). Default is `'h'`.

- KEEPAPP (`true` or `false`) indicates whether to keep approximation coefficients (`true`) or not (`false`). Default is `false`.

- IDXSIG is a vector which contains the indices of the initial signals, or the string `'all'`. Default is `'all'`.

**Examples**

```
% Load original 1D-multisignal.
load thinker

% Perform a decomposition at level 2 using wavelet db2.
dec = mdwtdec('r',X,2,'db2');

% Compress the signals to obtain a percentage of zeros
% near 95% for the wavelet coefficients.
[XC,decCMP,THRESH] = mswcmp('cmp',dec,'NO_perf',95);
[Ecmp,PECcmp,PECFScmp] = wdecenergy(decCMP);

% Plot the original signals 1 and 31, and
% the corresponding compressed signals.
figure;
plot(X([1 31],:)','r--','linewidth',2);    hold on
```

```
plot(XC([1 31],:)','b','linewidth',2);
grid; set(gca,'Xlim',[1,96])
title('X dashed line and XC solid line')
```



X dashed line and XC solid line

**References**

Birgé L.; P. Massart (1997), "From Model Selection to Adaptive Estimation," in D. Pollard (ed), *Festchrift for L. Le Cam*, Springer, pp. 55–88.

DeVore, R.A.; B. Jawerth, B.J. Lucier (1992), "Image Compression Through Wavelet Transform Coding," *IEEE Trans. on Inf. Theory*, vol. 38, No 2, pp. 719–746.

Donoho, D.L. (1993), "Progress in Wavelet Analysis and WVD: a Ten Minute Tour," in Progress in Wavelet Analysis and Applications, Y. Meyer, S. Roques, pp. 109–128. Frontières Ed.

Donoho, D.L.; I.M. Johnstone(1994), "Ideal Spatial Adaptation by Wavelet Shrinkage," *Biometrika*, vol. 81, pp. 425–455.

Donoho, D.L.; I.M. Johnstone, G. Kerkyacharian, D. Picard (1995), "Wavelet Shrinkage: Asymptopia," *Jour. Roy. Stat. Soc., series B*, vol. 57 no. 2, pp. 301–369.

Donoho, D.L.; I.M. Johnstone, "Ideal De-noising in an Orthonormal Basis Chosen from a Library of Bases," C.R.A.S. Paris, t. 319, Ser. I, pp. 1317–1322.

Donoho, D.L. (1995), "De-noising by Soft-thresholding," *IEEE Trans. on Inf. Theory*, 41, 3, pp. 613–627.

**See Also**     mdwtdec | mdwtrec | mswthresh | wthresh

# mswcmpscr

**Purpose**      Multisignal 1-D wavelet compression scores

**Syntax**       `[THR,L2SCR,NOSCR,IDXSORT] = mswcmpscr(DEC)`

**Description**  `[THR,L2SCR,NOSCR,IDXSORT] = mswcmpscr(DEC)` computes four matrices: thresholds `THR`, compression scores `L2SCR` and `NOSCR`, and indices `IDXSORT`. The decomposition `DEC` corresponds to a matrix of wavelet coefficients `CFS` obtained by concatenation of detail and (optionally) approximation coefficients, where

`CFS = [cd{DEC.level}, ...  , cd{1}]` or `CFS = [ca, cd{DEC.level}, ...  , cd{1}]`

The concatenation is made rowwise if `DEC.dirDec` is equal to `'r'` or columnwise if `DEC.dirDec` is equal to `'c'`.

If `NbSIG` is the number of original signals and `NbCFS` the number of coefficients for each signal (all or only the detail coefficients), then `CFS` is an `NbSIG`-by-`NbCFS` matrix. Therefore,

- `THR`, `L2SCR`, `NOSCR` are `NbSIG`-by-`(NbCFS+1)` matrices

- `IDXSORT` is an `NbSIG`-by-`NbCFS` matrix

- `THR(:,2:end)` is equal to `CFS` sorted by row in ascending order with respect to the absolute value.

- For each row, `IDXSORT` contains the order of coefficients and `THR(:,1)=0`.

For the ith signal:

- `L2SCR(i,j)` is the percentage of preserved energy (L2-norm), corresponding to a threshold equal to `CFS(i,j-1)` ($2 \leq j \leq$ `NbCFS`), and `L2SCR(:,1)=100`.

- `NOSCR(i,j)` is the percentage of zeros corresponding to a threshold equal to `CFS(i,j-1)` ($2 \leq j \leq$ `NbCFS`), and `NOSCR(:,1)=0`.

Three more optional inputs may be used:

`[...]  = mswcmpscr(...,S_OR_H,KEEPAPP,IDXSIG)`

- S_OR_H (`'s'` or `'h'`) stands for soft or hard thresholding (see mswthresh for more details).

- KEEPAPP (`true` or `false`) indicates whether to keep approximation coefficients (`true`) or not (`false`).

- IDXSIG is a vector that contains the indices of the initial signals, or the string `'all'`.

The defaults are, respectively, `'h'`, false and `'all'`.

**Examples**
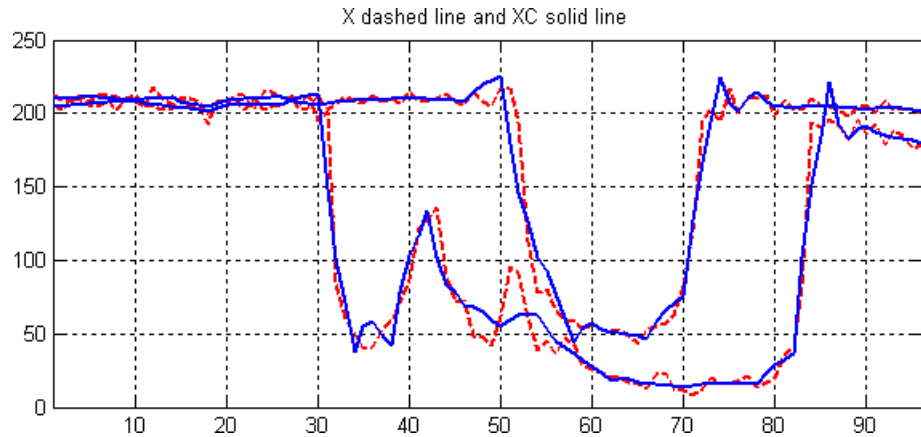```
% Load original 1D-multisignal.
load thinker

% Perform a decomposition at level 2 using wavelet db2.
dec = mdwtdec('r',X,2,'db2');

% Compute compression performances for soft an hard thresholding.
[THR_S,L2SCR_S,NOSCR_S] = mswcmpscr(dec,'s');
[THR_H,L2SCR_H,NOSCR_H] = mswcmpscr(dec,'h');
```

**References** Daubechies, I. (1992), *Ten lectures on wavelets*, CBMS-NSF conference series in applied mathematics. SIAM Ed.

Mallat, S. (1989), "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Pattern Anal. and Machine Intell.*, vol. 11, no. 7, pp. 674–693.

Meyer, Y. (1990), *Ondelettes et opérateurs*, Tome 1, Hermann Ed. (English translation: *Wavelets and operators*, Cambridge Univ. Press. 1993.)

**See Also** mdwtdec | mdwtrec | ddencmp | wdencmp

# mswcmptp

| | |
|---|---|
| **Purpose** | Multisignal 1-D compression thresholds and performances |
| **Syntax** | [THR_VAL,L2_Perf,NO_Perf] = mswcmptp(DEC,METH)<br>[THR_VAL,L2_Perf,NO_Perf] = mswcmptp(DEC,METH,PARAM) |
| **Description** | [THR_VAL,L2_Perf,NO_Perf] = mswcmptp(DEC,METH) or<br>[THR_VAL,L2_Perf,NO_Perf] = mswcmptp(DEC,METH,PARAM)<br>computes the vectors THR_VAL, L2_Perf and NO_Perf obtained after a compression using the METH method and, if required, the PARAM parameter (see mswcmp for more information on METH and PARAM). |

For the ith signal:

- THR_VAL(i) is the threshold applied to the wavelet coefficients. For a level dependent method, THR_VAL(i,j) is the threshold applied to the detail coefficients at level j.

- L2_Perf(i) is the percentage of energy (L2_norm) preserved after compression.

- NO_Perf(i) is the percentage of zeros obtained after compression.

You can use three more optional inputs:

[...] = mswcmptp(...,S_OR_H,KEEPAPP,IDXSIG)

- S_OR_H ('s' or 'h') stands for soft or hard thresholding (see mswthresh for more details).

- KEEPAPP (true or false) indicates whether to keep approximation coefficients (true) or not (false)

- IDXSIG is a vector which contains the indices of the initial signals, or the string 'all'.

The defaults are, respectively, 'h', false and 'all'.

| | |
|---|---|
| **Examples** | % Load original 1D-multisignal.<br>load thinker<br><br>% Perform a decomposition at level 2 using wavelet db2. |

```
dec = mdwtdec('r',X,2,'db2');

% Compute compression thresholds and exact performances
% obtained after a compression using the method 'NO_perf' and
% requiring a percentage of zeros near 95% for the wavelet
% coefficients.
[THR_VAL,L2_Perf,NO_Perf] = mswcmptp(dec,'NO_perf',95);
```

**References**     Daubechies, I. (1992), *Ten lectures on wavelets*, CBMS-NSF conference series in applied mathematics. SIAM Ed.

Mallat, S. (1989), "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Pattern Anal. and Machine Intell.*, vol. 11, no. 7, pp. 674–693.

Meyer, Y. (1990), *Ondelettes et opérateurs*, Tome 1, Hermann Ed. (English translation: *Wavelets and operators*, Cambridge Univ. Press. 1993.)

**See Also**      mdwtdec | mdwtrec | ddencmp | wdencmp

# mswden

| | |
|---|---|
| **Purpose** | Multisignal 1-D denoising using wavelets |
| **Syntax** | `[XD,DECDEN,THRESH] = mswden('den',...)`<br>`[XD,THRESH] = mswden('densig',...)`<br>`[DECDEN,THRESH] = mswden('dendec',...)`<br>`THRESH = mswden('thr',...)`<br>`[...] = mswden(OPTION,DIRDEC,X,WNAME,LEV,METH,PARAM)`<br>`[...] = mswden(...,S_OR_H)`<br>`[...] = mswden(...,S_OR_H,KEEPAPP)`<br>`[...] = mswden(...,S_OR_H,KEEPAPP,IDXSIG)` |

**Description**    mswden computes thresholds and, depending on the selected option, performs denoising of 1-D signals using wavelets.

`[XD,DECDEN,THRESH] = mswden('den',...)` returns a denoised version XD of the original multisignal matrix X, whose wavelet decomposition structure is DEC. The output XD is obtained by thresholding the wavelet coefficients, DECDEN is the wavelet decomposition associated to XD (see mdwtdec), and THRESH is the matrix of threshold values. The input METH is the name of the denoising method and PARAM is the associated parameter, if required.

Valid denoising methods METH and associated parameters PARAM are:

| | |
|---|---|
| `'rigrsure'` | Principle of Stein's Unbiased Risk |
| `'heursure'` | Heuristic variant of the first option |
| `'sqtwolog'` | Universal threshold `sqrt(2*log(.))` |
| `'minimaxi'` | Minimax thresholding (see thselect) |

For these methods PARAM defines the multiplicative threshold rescaling:

| 'one' | No rescaling |
|-------|--------------|
| 'sln' | Rescaling using a single estimation of level noise based on first level coefficients |
| 'mln' | Rescaling using a level dependent estimation of level noise |

### Penalization methods

| 'penal' | Penal |
|---------|-------|
| 'penalhi' | Penal high, $2.5 \Re\leq$ PARAM $\Re\leq 10$ |
| 'penalme' | Penal medium, $1.5 \Re\leq$ PARAM $\Re\leq 2.5$ |
| 'penallo' | Penal low, $1 \Re\leq$ PARAM $\Re\leq 2$ |

PARAM is a sparsity parameter, and it should be such that: $1 \leq$ PARAM $\leq 10$. For penal method, no control is done.

### Manual method

| 'man_thr' | Manual method |
|-----------|---------------|

PARAM is an NbSIG-by-NbLEV matrix or NbSIG-by-(NbLEV+1) matrix such that:

- PARAM(i,j) is the threshold for the detail coefficients of level j for the ith signal ($1 \leq j \leq$ NbLEV).

- PARAM(i,NbLEV+1) is the threshold for the approximation coefficients for the ith signal (if KEEPAPP is 0).

where NbSIG is the number of signals and NbLEV the number of levels of decomposition.

Instead of the 'den' input OPTION, you can use 'densig', 'dendec' or 'thr' OPTION to select output arguments:

```
[XD,THRESH] = mswden('densig',...) or [DECDEN,THRESH] =
mswden('dendec',...)
```

THRESH = mswden('thr',...) returns the computed thresholds, but denoising is not performed.

The decomposition structure input argument DEC can be replaced by four arguments: DIRDEC, X, WNAME and LEV.

[...] = mswden(OPTION,DIRDEC,X,WNAME,LEV,METH,PARAM) before performing a denoising or computing thresholds, the multisignal matrix X is decomposed at level LEV using the wavelet WNAME, in the direction DIRDEC.

You can use three more optional inputs:

[...]  = mswden(...,S_OR_H) or
[...]  = mswden(...,S_OR_H,KEEPAPP) or
[...]  = mswden(...,S_OR_H,KEEPAPP,IDXSIG)

- S_OR_H ('s' or 'h') stands for soft or hard thresholding (see mswthresh for more details).

- KEEPAPP (true or false) indicates whether to keep approximation coefficients (true) or not (false).

- IDXSIG is a vector that contains the indices of the initial signals, or the string 'all'.

The defaults are, respectively, 'h', false and 'all'.

**Examples**

```
% Load original 1D-multisignal.
load thinker

% Perform a decomposition at level 2 using the wavelet db2.
dec = mdwtdec('r',X,2,'db2');

% Denoise signals using the universal method
% of thresholding (sqtwolog) and the 'sln'
% threshold rescaling (with a single estimation
% of level noise, based on first level coefficients).
[XD,decDEN,THRESH] = mswden('den',dec,'sqtwolog','sln');

% Plot the original signals 1 and 31, and the
```

```
% corresponding denoised signals.
figure;
plot(X([1 31],:)','r--','linewidth',2);    hold on
plot(XD([1 31],:)','b','linewidth',2);
grid; set(gca,'Xlim',[1,96])
title('X dashed line and XD solid line')
```



X dashed line and XD solid line

**References**    Birgé, L.; P. Massart (1997), "From model selection to adaptive estimation," in D. Pollard (ed), *Festchrift for L. Le Cam*, Springer, pp. 55–88.

DeVore, R.A.; B. Jawerth, B.J. Lucier (1992), "Image compression through wavelet transform coding," *IEEE Trans. on Inf. Theory*, vol. 38, No 2, pp. 719–746.

Donoho, D.L. (1993), "Progress in wavelet analysis and WVD: a ten minute tour," in Progress in wavelet analysis and applications, Y. Meyer, S. Roques, pp. 109–128. Frontières Ed.

Donoho, D.L.; I.M. Johnstone(1994), "Ideal spatial adaptation by wavelet shrinkage," *Biometrika*, vol. 81, pp. 425–455.

Donoho, D.L.; I.M. Johnstone, G. Kerkyacharian, D. Picard (1995), "Wavelet shrinkage: asymptopia," *Jour. Roy. Stat. Soc.,series B*, vol. 57 no. 2, pp. 301–369.

# mswden

Donoho, D.L.; I.M. Johnstone, "Ideal de-noising in an orthonormal basis chosen from a library of bases," C.R.A.S. Paris, t. 319, Ser. I, pp. 1317–1322.

Donoho, D.L. (1995), "De-noising by soft-thresholding," *IEEE Trans. on Inf. Theory*, 41, 3, pp. 613–627.

**See Also**    mdwtdec | mdwtrec | mswthresh | wthresh

**Purpose**        Nondecimated 1-D wavelet transform

**Syntax**         WT = ndwt(X,N,'*wname*')
                   WT = ndwt(X,N,'*wname*','mode','*ExtM*')
                   WT = ndwt(X,N,WF,...)

**Description**    ndwt performs a multilevel 1-D nondecimated wavelet decomposition
                   using either a particular wavelet ('*wname*') or the wavelet filters you
                   specify. The decomposition also uses the specified DWT extension mode
                   (see dwtmode).

                   WT = ndwt(X,N,'*wname*') returns a structure which contains the
                   non-decimated wavelet transform of the vector X at the level N. N is a
                   positive integer, and '*wname*' is a string containing the wavelet name.
                   The default default extension mode is 'sym'. For more information on
                   *wname*, see wfilters.

                   WT = ndwt(X,N,'*wname*','mode','*ExtM*') uses the extension mode
                   specified in the string '*ExtM*'.

                   WT is a structure with the fields shown in the table.

                   Instead of a wavelet you can specify four filters (two for decomposition
                   and two for reconstruction).

                   WT = ndwt(X,N,WF,...) specifies four filters (two for decomposition
                   and two for reconstruction) instead of a wavelet name. WF is a 1-by-4
                   cell array {LoD,HiD,LoR,HiR} or a structure with the four fields 'LoD',
                   'HiD', 'LoR', 'HiR'.

| rowvect | Logical value which is true if X is a row vector |
|---------|--------------------------------------------------|
| level   | Level of the decomposition |
| mode    | Name of the wavelet transform extension mode |
| filters | Structure with 4 fields, LoD, HiD, LoR, HiR, which contain the filters used for DWT |

| dec | 1 by (level+1) cell array containing the coefficients of the decomposition. dec{1} contains the coefficients of the approximation and dec{j} (j = 2 to level+1), contains the coefficients of the detail of level (level+1-j) |
|---|---|
| longs | 1 by (level+2) vector containing the lengths of the components. longs is defined as (where N is the level)<br>longs(1) = length of app.  coef.(N)<br>longs(i) = length of det.  coef.(N-i+2)<br>for i = 2,...,N+1<br>longs(N+2) = length(X). |

**Examples**    **Localize Discontinuity with Nondecimated Wavelet Transform**

Use fine-scale nondecimated wavelet transform coefficients to localize a discontinuity.

Create signal consisting of a 1/2–hz sine wave sampled at 1 kHz with discontinuities at 0.3 and 0.72 seconds.

```
t = linspace(0,1,1000);
x = 4*sin(4*pi*t);
x = x - sign(t - .3) - sign(.72 - t);
plot(t,x); xlabel('t'); ylabel('x');
grid on;
```
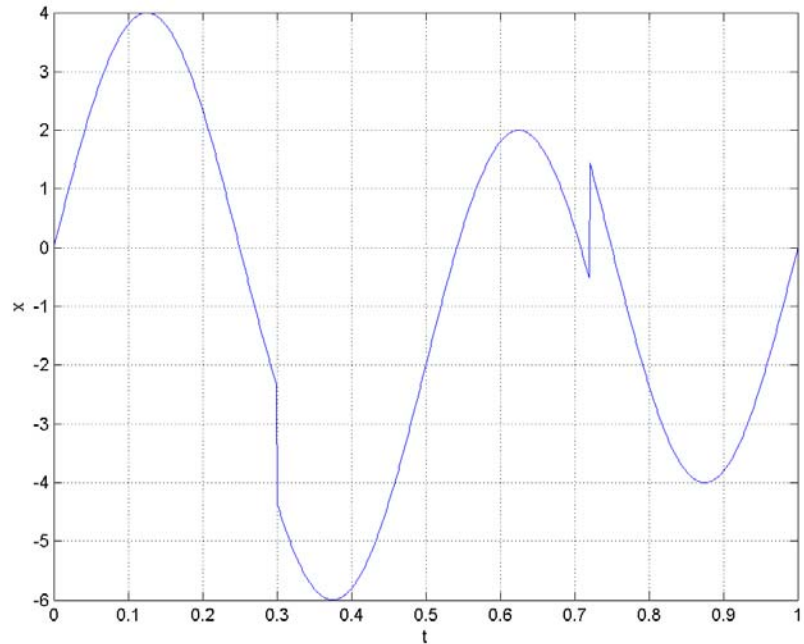
Obtain the nondecimated wavelet transform of the input signal down to level 4 using the Daubechies extremal phase wavelet with 2 vanishing moments and the default whole-point symmetric extension mode. Reconstruct a signal approximation based on the level-one wavelet coefficients.

```
W = ndwt(x,4,'db2','mode','per');
d1 = indwt(W,'d',1);
```

Plot the original signal and the signal approximation to visualize how the wavelet coefficients localize the discontinuities.

```
subplot(211);
plot(t,x); title('Original Signal');
```

```
grid on;
subplot(212);
plot(t,d1,'linewidth',2); title('Wavelet Approximation -- Level 1');
grid on;
```
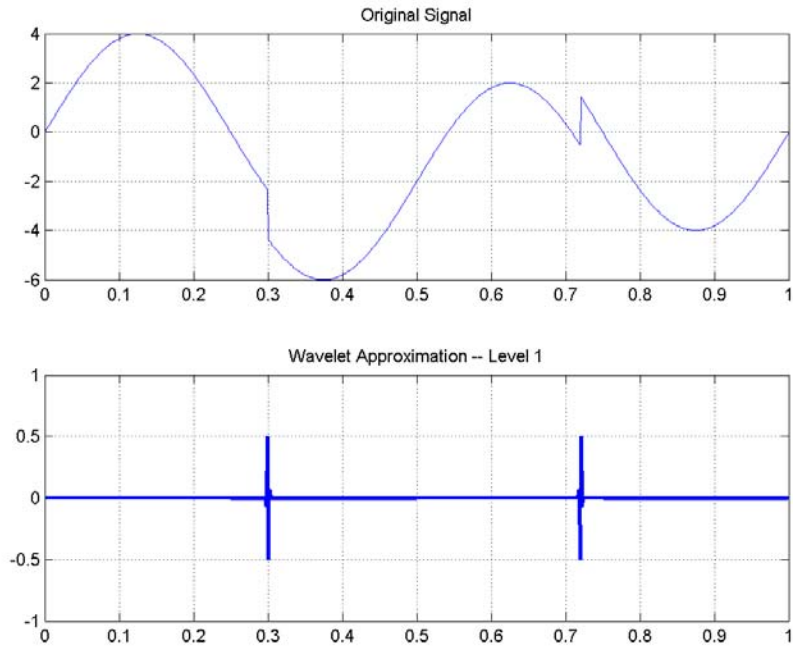


### Specify Extension Mode for Nondecimated Wavelet Transform

Specify an extension mode different from the default whole-point symmetric extension.

Load the freqbrk signal and obtain the nondecimated wavelet transform down to level 4 using the Daubechies extremal phase wavelet with 2 vanishing moments. Use the periodic extension mode.

```
load freqbrk;
W = ndwt(freqbrk,4,'db2','mode','per');
```

### Nondecimated Wavelet Transform Using Specified Decomposition and Reconstruction Filters

Specify the decomposition and reconstruction filters as a cell or structure array.

Obtain the decomposition and reconstruction filters for the biorthogonal spline wavelet with 3 vanishing moments in the reconstruction wavelet and 5 vanishing moments in the decomposition wavelet. Create a cell array with the scaling and wavelet filters and analyze the freqbrk signal.

```
[LoD,HiD,LoR,HiR] = wfilters('bior3.5');
WF = {LoD,HiD,LoR,HiR};
load freqbrk;
W = ndwt(freqbrk,4,WF);
```

Use a structure array to input the scaling and wavelet filters.

```
WF1 = struct('LoD',LoD,'HiD',HiD,'LoR',LoR,'HiR',HiR);
W1 = ndwt(freqbrk,4,WF1);
```

**See Also**    dwtmode | indwt | waveinfo | wfilters | wmaxlev

| | |
|---|---|
| **Purpose** | Nondecimated 2-D wavelet transform |

**Syntax**

```
WT = ndwt2(X,N,'wname')
WT = ndwt2(X,N,'wname','mode','ExtM')
WT = ndwt2(X,W,...)
WT = ndwt2(X,WF,...)
```

**Description**

ndwt2 performs a multilevel 2-D nondecimated wavelet decomposition using a particular wavelet ('wname') or the wavelet filters you specify. The decomposition also uses the specified DWT extension mode (see dwtmode).

WT = ndwt2(X,N,'wname') returns a structure which contains the nondecimated wavelet transform of the vector X at the level N. N is a positive integer and 'wname' is a string containing the wavelet name. The default default extension mode is'sym'. For more information on wname, see wfilters.

WT = ndwt2(X,N,'wname','mode','ExtM') uses the extension mode specified in the string 'ExtM'.

WT is a structure with the fields shown in the table.

| sizeINI | Size of the two-dimensional array X |
|---|---|
| level | Level of the decomposition |
| mode | Name of the wavelet transform extension mode |
| filters | Structure with 4 fields, LoD, HiD, LoR, HiR, which contain the filters used for DWT |

| dec | 1 by (3*level+1) cell array containing the coefficients of the decomposition. dec{1} contains the coefficients of the approximation and dec{j} (j = 2 to 3*level+1), contains the coefficients of the details from level level to level 1, three details by level (LH, HL and HH where L is low and H is high) |
|---|---|
| sizes | (level+1) by 2 array containing the size of the components |

IWT = ndwt2(X,W,...) specifies two wavelets (one for each direction) with W = {'wname1','wname2'} or W is a structure with two fields 'w1', 'w2' containing strings, which are the names of wavelets, one per direction.

Instead of one or two wavelets, you may specify four filters (two for decomposition and two for reconstruction) or 2 x 4 filters (one quadruplet per direction):

WT = ndwt2(X,WF,...) specifies four filters (two for decomposition and two for reconstruction) or 2 x 4 filters (one quadruplet per direction). WF is a cell array (1x4) or (2x4), {LoD,HiD,LoR,HiR}, or a structure with the four fields 'LoD', 'HiD', 'LoR', 'HiR'.

**Examples**
```
% Load original image.
load noiswom;

% Decompose X at level 2 using db1.
W1 = ndwt2(X,2,'db1')

W1 =

    sizeINI: [96 96]
      level: 2
    filters: [1x1 struct]
       mode: 'sym'
```

```
                 dec: {7x1 cell}
               sizes: [3x2 double]

% Decompose X at level 3 using db1 and periodic
% extension mode.
W2 = ndwt2(X,3,'db1','mode','per')

W2 =

    sizeINI: [96 96]
      level: 3
    filters: [1x1 struct]
       mode: 'per'
        dec: {10x1 cell}
      sizes: [4x2 double]

% Decompose X at level 3 using db1 for rows, and db2 for
% columns, using symmetric extension mode.
W3 = ndwt2(X,3,{'db1','db2'},'mode','sym')

W3 =

    sizeINI: [96 96]
      level: 3
    filters: [1x1 struct]
       mode: 'sym'
        dec: {10x1 cell}
      sizes: [4x2 double]

WF = W3.filters

WF =

    LoD: {[0.7071 0.7071]  [-0.1294 0.2241 0.8365 0.4830]}
    HiD: {[-0.7071 0.7071]  [-0.4830 0.8365 -0.2241 -0.1294]}
    LoR: {[0.7071 0.7071]  [0.4830 0.8365 0.2241 -0.1294]}
    HiR: {[0.7071 -0.7071]  [-0.1294 -0.2241 0.8365 -0.4830]}
```

```
% Decompose X using filters given by WF.
W4 = ndwt2(X,3,WF,'mode','sym')

W4 =

    sizeINI: [96 96]
      level: 3
    filters: [1x1 struct]
       mode: 'sym'
        dec: {10x1 cell}
      sizes: [4x2 double]
```

**See Also**      dwtmode | indwt2 | waveinfo | wfilters | wmaxlev

# mswthresh

| | |
|---|---|
| **Purpose** | Perform multisignal 1-D thresholding |

**Syntax**

```
Y = mswthresh(X,SORH,T)
Y = mswthresh(X,SORH,T,'c')
Y = mswthresh(X,'s',T)
Y = mswthresh(X,'h',T)
```

**Description**    Y = mswthresh(X,SORH,T) returns soft (if SORH='s') or hard (if SORH='h') T-thresholding of the input matrix X. T can be a single value, a matrix of the same size as X or a vector. In this last case, thresholding is performed rowwise and LT = length(T) must be such that size(X,1) $\leq$ LT.

Y = mswthresh(X,SORH,T,'c') performs a columnwise thresholding and size(X,2) $\leq$ LT.

Y = mswthresh(X,'s',T) returns Y = SIGN(X).(|X|-T)+, soft thresholding is shrinkage.

Y = mswthresh(X,'h',T) returns Y = X.1_(|X|>T), hard thresholding is cruder.

**See Also**    mswden | mswcmp | wthresh | wden | wdencmp | wpdencmp

**Purpose**          Node ascendants

**Syntax**           A = nodeasc(*T*,*N*)

**Description**      nodeasc is a tree-management utility.

A = nodeasc(*T*,*N*) returns the indices of all the ascendants of the node *N* in the tree *T* where N can be the index node or the depth and position of the node. A is a column vector with A(1) = index of node *N*.

A = nodeasc(*T*,*N*,'deppos') is a matrix, which contains the depths and positions of all ascendants. A(i,1) is the depth of the i-th ascendant and A(i,2) is the position of the i-th ascendant.

The nodes are numbered from left to right and from top to bottom. The root index is 0.

**Examples**         % Create binary tree of depth 3.
```
t = ntree(2,3);
t = nodejoin(t,5);
t = nodejoin(t,4);
plot(t)
```
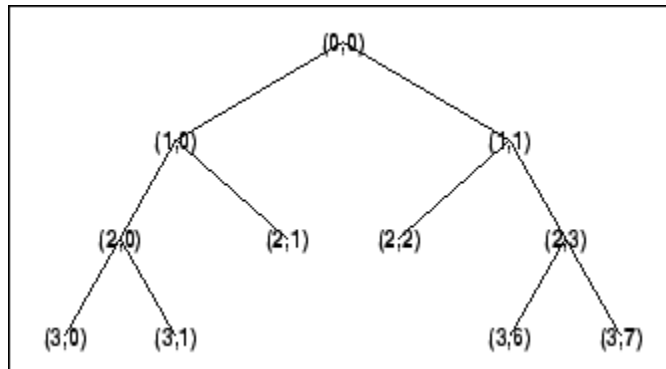


% Change Node Label from Depth_Position to Index
% (see the plot function).

# nodeasc
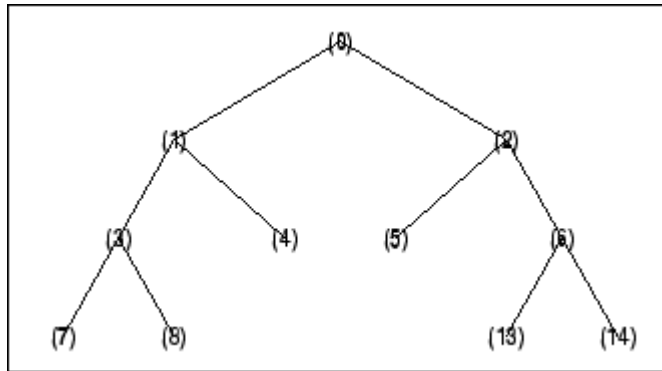


```
nodeasc(t,[2 2])
ans =
     5
     2
     0

nodeasc(t,[2 2],'deppos')
ans =
     2     2
     1     1
     0     0
```

**See Also**    nodedesc | nodepar | wtreemgr

**Purpose**    Node descendants

**Syntax**    D = nodedesc(*T*,*N*)
D = nodedesc(*T*,*N*,'deppos')

**Description**    nodedesc is a tree-management utility.

D = nodedesc(*T*,*N*) returns the indices of all the descendants of the node *N* in the tree *T* where *N* can be the index node or the depth and position of node. D is a column vector with D(1) = index of node *N*.

D = nodedesc(*T*,*N*,'deppos') is a matrix that contains the depths and positions of all descendants. D(i,1) is the depth of the i-th descendant and D(i,2) is the position of the i-th descendant.

The nodes are numbered from left to right and from top to bottom. The root index is 0.

**Examples**
```
% Create binary tree of depth 3.
t = ntree(2,3);
t = nodejoin(t,5);
t = nodejoin(t,4);
plot(t)
```
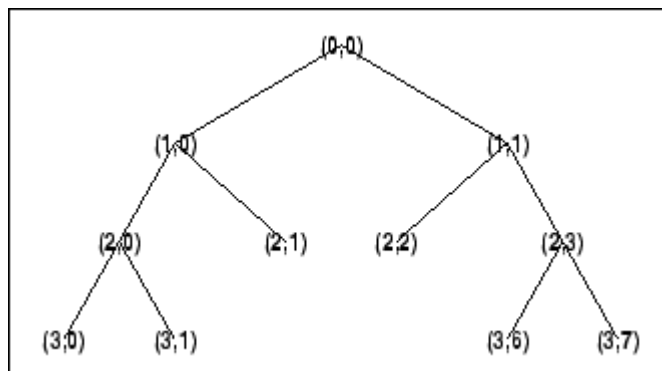


```
% Change Node Label from Depth_Position to Index
% (see the plot function).
```
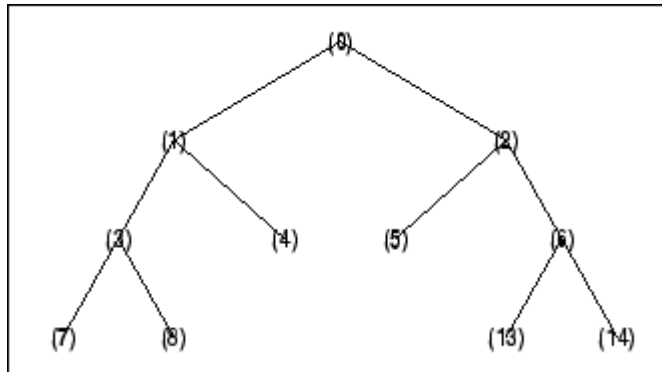
```
% Node descendants.
nodedesc(t,2)
ans =
     2
     5
     6
    13
    14

nodedesc(t,2,'deppos')
ans =
     1     1
     2     2
     2     3
     3     6
     3     7

nodedesc(t,[1 1],'deppos')
ans =
     1     1
     2     2
     2     3
     3     6
```

```
     3     7

nodedesc(t,[1 1])
ans =
     2
     5
     6
    13
    14
```

**See Also**       nodeasc | nodepar | wtreemgr

# nodejoin

| | |
|---|---|
| **Purpose** | Recompose node |
| **Syntax** | T = nodejoin(*T*,*N*)<br>T = nodejoin(*T*)<br>T = nodejoin(T,0) |
| **Description** | nodejoin is a tree-management utility. |
| | T = nodejoin(*T*,*N*) returns the modified tree *T* corresponding to a recomposition of the node *N*. |
| | The nodes are numbered from left to right and from top to bottom. The root index is 0. |
| | T = nodejoin(*T*) is equivalent to T = nodejoin(T,0). |
| **Examples** | ```
% Create binary tree of depth 3.
t = ntree(2,3);

% Plot tree t.
plot(t)
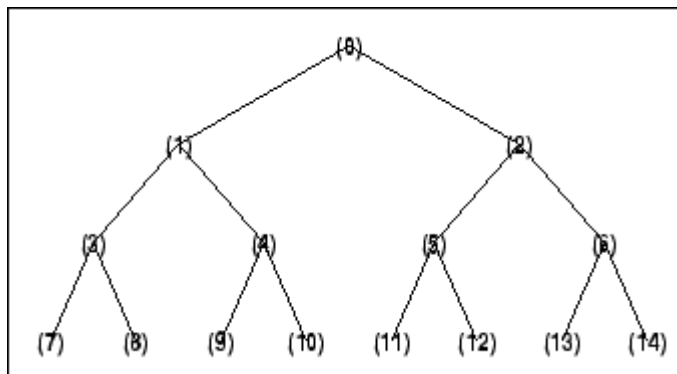
% Change Node Label from Depth_Position to Index
% (see the plot function).
``` |

```
% Merge nodes of indices 4 and 5.
t = nodejoin(t,5);
t = nodejoin(t,4);
% Plot new tree t.
plot(t)
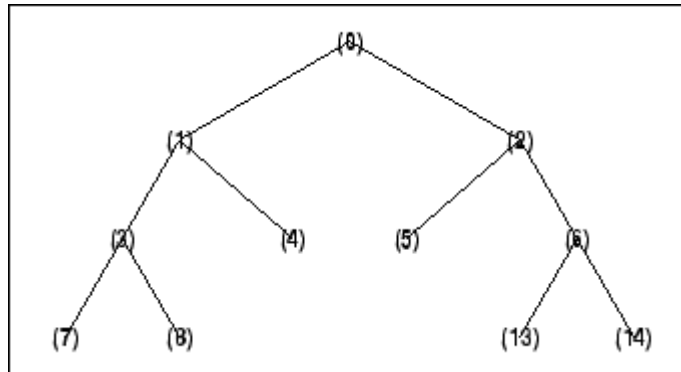
% Change Node Label from Depth_Position to Index
% (see the plot function).
```



**See Also**      nodesplt

# nodepar

**Purpose**        Node parent

**Syntax**         F = nodepar(T,N)
                   F = nodepar(T,N,'deppos')

**Description**    nodepar is a tree-management utility.

F = nodepar(T,N) returns the indices of the Äúparent(s)Äù of the nodes *N* in the tree *T* where *N* can be a column vector containing the indices of nodes or a matrix that contains the depths and positions of nodes. In the last case, N(i,1) is the depth of the i-th node and N(i,2) is the position of the i-th node.

F = nodepar(T,N,'deppos') is a matrix that contains the depths and positions of returned nodes. F(i,1) is the depth of the i-th node and F(i,2) is the position of the i-th node.

nodepar(T,0) or nodepar(T,[0,0]) returns -1.

nodepar(T,0,'deppos') or nodepar(T,[0,0],'deppos') returns [-1,0].

The nodes are numbered from left to right and from top to bottom. The root index is 0.

**Examples**
```
% Create binary tree of depth 3.
t = ntree(2,3);
t = nodejoin(t,5);
t = nodejoin(t,4);
plot(t)
```

```
% Change Node Label from Depth_Position to Index
% (see the plot function).
```



```
% Nodes parent.
nodepar(t,[2 2],'deppos')
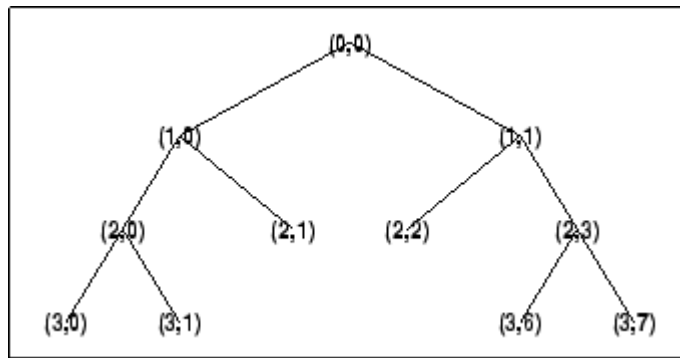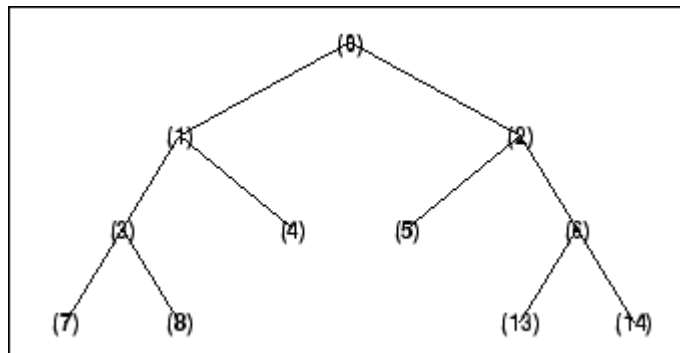
ans =
    1     1

nodepar(t,[1;7;14])

ans =
    0
```

# nodepar

```
        3
        6
```

**See Also**     nodeasc | nodedesc | wtreemgr

# nodesplt

**Purpose**     Split (decompose) node

**Syntax**     T = nodesplt(*T*,*N*)

**Description**     nodesplt is a tree-management utility.

T = nodesplt(*T*,*N*) returns the modified tree *T* corresponding to the decomposition of the node *N*.

The nodes are numbered from left to right and from top to bottom. The root index is 0.

**Examples**
```
% Create binary tree (tree of order 2) of depth 3.
t = ntree(2,3);

% Plot tree t.
plot(t)
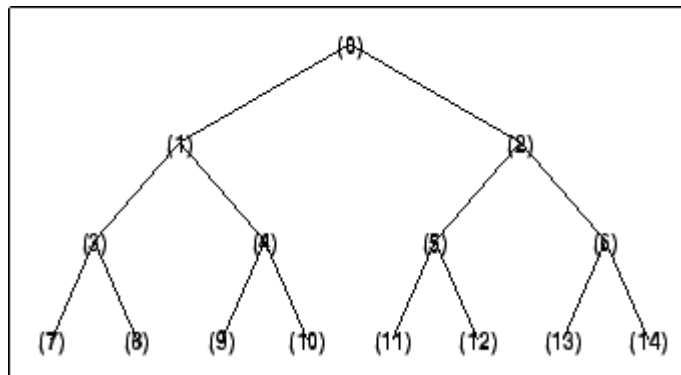
% Change Node Label from Depth_Position to Index
% (see the plot function).
```



```
% Split node of index 10.
t = nodesplt(t,10);
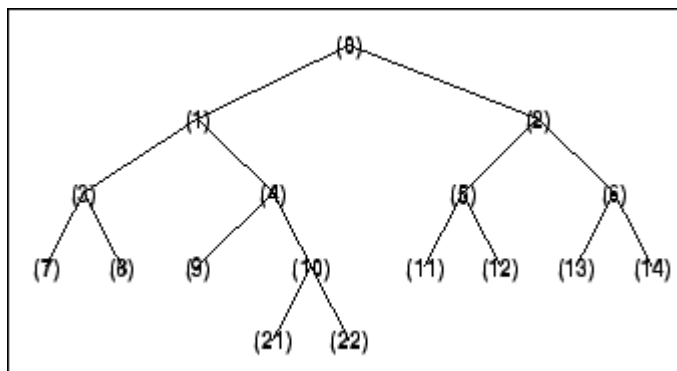
% Plot new tree t.
```

```
plot(t)
% Change Node Label from Depth_Position to Index
% (see the plot function).
```



**See Also**    nodejoin

| | |
|---|---|
| **Purpose** | Determine nonterminal nodes |
| **Syntax** | N = noleaves(*T*)<br>N = noleaves(*T*,'dp') |
| **Description** | N = noleaves(*T*) returns the indices of nonterminal nodes of the tree *T* (i.e., nodes that are not leaves). N is a column vector.<br><br>The nodes are ordered from left to right as in tree *T*.<br><br>N = noleaves(*T*,'dp') returns a matrix N, which contains the depths and positions of nonterminal nodes.<br><br>N(i,1) is the depth of the i-th nonterminal node and<br>N(i,2) is the position of the i-th nonterminal node. |
| **Examples** | ```matlab<br>% Create initial tree.<br>ord = 2;<br>t = ntree(ord,3);        % binary tree of depth 3.<br>t=nodejoin(t,5);<br>t=nodejoin(t,4);<br>plot(t)<br><br>% Change Node Label from Depth_Position to Index<br>% (see the plot function).<br>``` |

# noleaves

```
% List nonterminal nodes (index).
ntnodes_ind = noleaves(t)
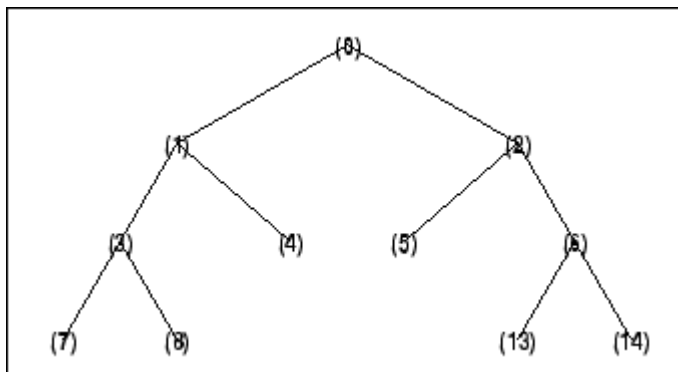
ntnodes_ind =
     0
     1
     2
     3
     6

% List nonterminal nodes (Depth_Position).
ntnodes_depo = noleaves(t,'dp')

ntnodes_depo =
     0     0
     1     0
     1     1
     2     0
     2     3
```

**See Also**      leaves

**Purpose**    Number of terminal nodes

**Syntax**    NB = ntnode(*T*)

**Description**    ntnode is a tree-management utility.

NB = ntnode(*T*) returns the number of terminal nodes in the tree *T*.

The nodes are numbered from left to right and from top to bottom. The root index is 0.

**Examples**
```
% Create binary tree (tree of order 2) of depth 3.
t = ntree(2,3);
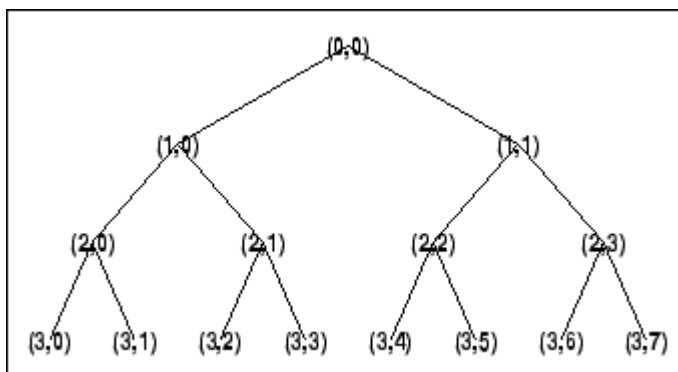
% Plot tree t.
plot(t)
```



```
% Number of terminal nodes.
ntnode(t)

ans =
    8
```

**See Also**    wtreemgr

# ntree

| | |
|---|---|
| **Purpose** | NTREE constructor |
| **Syntax** | `T = ntree(ORD,D)`<br>`T = ntree`<br>`T = ntree(2,0)`<br>`T = ntree(ORD)`<br>`T = ntree(ORD,0)`<br>`T = ntree(ORD,D,S)`<br>`T = ntree(ORD,D,S,U)` |
| **Description** | `T = ntree(ORD,D)` returns an NTREE object, which is a complete tree of order `ORD` and depth `D`.<br><br>`T = ntree` is equivalent to `T = ntree(2,0)`.<br><br>`T = ntree(ORD)` is equivalent to `T = ntree(ORD,0)`.<br><br>With `T = ntree(ORD,D,S)` you can set a "split scheme" for nodes. The split scheme field `S` is a logical array of size `ORD` by 1.<br><br>The root of the tree can be split and it has `ORD` children. You can split the j-th child if `S(j) = 1`.<br><br>Each node that you can split has the same property as the root node.<br><br>With `T = ntree(ORD,D,S,U)` you can, in addition, set a userdata field.<br><br>Inputs can be given in another way:<br><br>`T = ntree('order',ORD,'depth',D,'spsch',S,'ud',U)`. For "missing" inputs the defaults are `ORD = 2` , `D = 0` , `S = ones([1:ORD])` , `U = {}`.<br><br>`[T,NB] = ntree( ... )` returns also the number of terminal nodes (leaves) of `T`.<br><br>For more information on object fields, type `help ntree/get`.<br><br>Class NTREE (Parent class: WTBO) |

## Fields

| | |
|---|---|
| wtbo | Parent object |
| order | Tree order |
| depth | Tree depth |
| spsch | Split scheme for nodes |
| tn | Column vector with terminal node indices |

**Examples**

```
% Create binary tree (tree of order 2) of depth 3.
t2 = ntree(2,3);
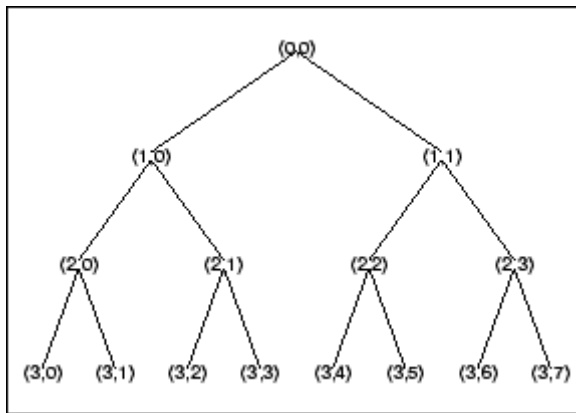
% Plot tree t2.
plot(t2)
```



```
% Create a quadtree (tree of order 4) of depth 2.
t4 = ntree(4,2,[1 1 0 1]);
```

```
% Plot tree t4.
plot(t4)
```



```
% Split and merge some nodes using the gui
% generated by plot (see the plot function).
% The figure becomes:
```

**See Also**       wtbo

# orthfilt

**Purpose**     Orthogonal wavelet filter set

**Syntax**      [Lo_D,Hi_D,Lo_R,Hi_R] = orthfilt(*W*)

**Description**  [Lo_D,Hi_D,Lo_R,Hi_R] = orthfilt(*W*) computes the four filters associated with the scaling filter *W* corresponding to a wavelet:

| | |
|---|---|
| Lo_D | Decomposition low-pass filter |
| Hi_D | Decomposition high-pass filter |
| Lo_R | Reconstruction low-pass filter |
| Hi_R | Reconstruction high-pass filter |

For an orthogonal wavelet, in the multiresolution framework, we start with the scaling function φ and the wavelet function ψ. One of the fundamental relations is the twin-scale relation:

$$\frac{1}{2}\phi\left(\frac{x}{2}\right) = \sum_{n \in Z} w_n \phi(x - n)$$

All the filters used in dwt and idwt are intimately related to the sequence $(w_n)_{n \in Z}$. Clearly if φ is compactly supported, the sequence $(w_n)$ is finite and can be viewed as a FIR filter. The scaling filter *W* is

- A low-pass FIR filter

- Of length 2*N*

- Of sum 1

- Of norm $\dfrac{1}{\sqrt{2}}$

For example, for the db3 scaling filter,

```
load db3
db3
```

```
db3 =
    0.2352 0.5706 0.3252 -0.0955 -0.0604 0.0249
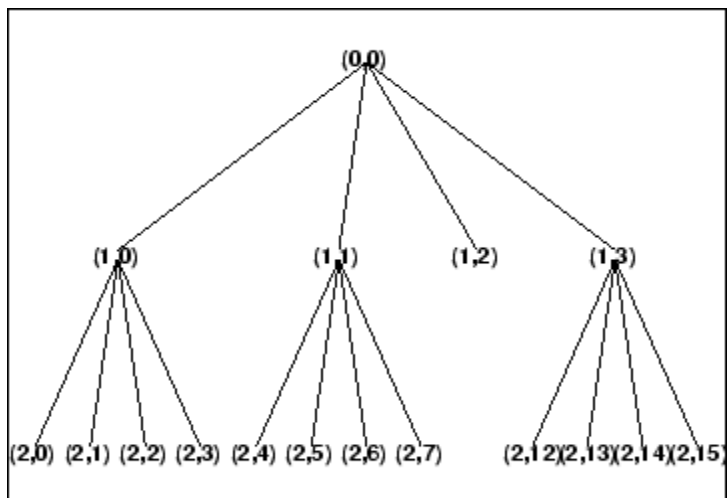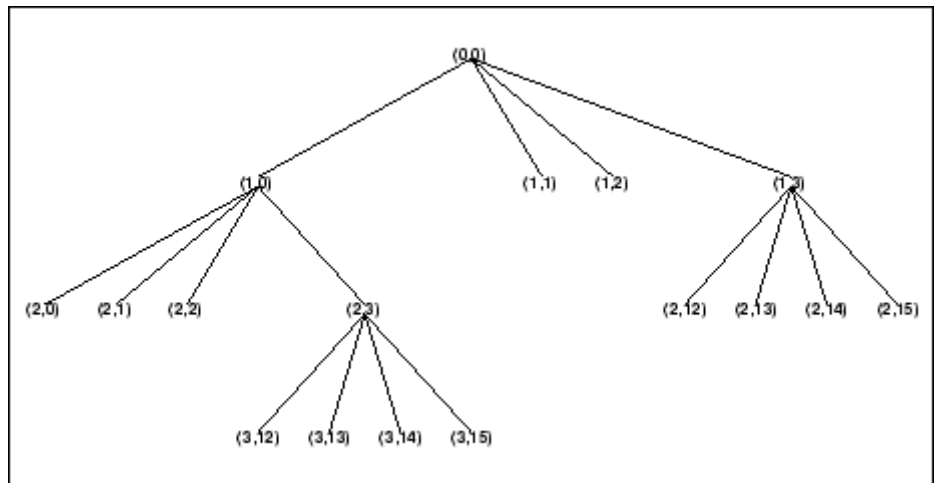
sum(db3)
ans =
    1.000
    norm(db3)
ans =
    0.7071
```

From filter $W$, we define four FIR filters, of length 2N and norm 1, organized as follows:

| Filters | Low-Pass | High-Pass |
|---|---|---|
| Decomposition | Lo_D | Hi_D |
| Reconstruction | Lo_R | Hi_R |

The four filters are computed using the following scheme:



where qmf is such that Hi_R and Lo_R are quadrature mirror filters (i.e., Hi_R(k) = (-1)$^k$Lo_R(2N + 1 - k), for k = 1, 2, ˜, 2N), and where wrev flips the filter coefficients. So Hi_D and Lo_D are also quadrature mirror filters. The computation of these filters is performed using orthfilt.

# orthfilt

**Examples**

```
% Load scaling filter.
load db8; w = db8;
subplot(421); stem(w);
title('Original scaling filter');
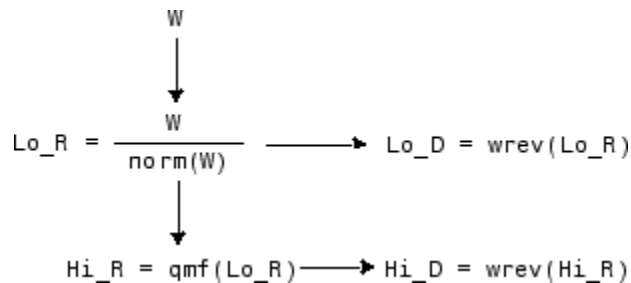
% Compute the four filters.
[Lo_D,Hi_D,Lo_R,Hi_R] = orthfilt(w);
subplot(423); stem(Lo_D);
title('Decomposition low-pass filter');
subplot(424); stem(Hi_D);
title('Decomposition high-pass filter');
subplot(425); stem(Lo_R);
title('Reconstruction low-pass filter');
subplot(426); stem(Hi_R);
title('Reconstruction high-pass filter');

% Check for orthonormality.
df = [Lo_D;Hi_D];
rf = [Lo_R;Hi_R];
id = df*df'

id =
    1.0000         0
         0    1.0000

id = rf*rf'

id =
    1.0000         0
         0    1.0000

% Check for orthogonality by dyadic translation, for example:
df = [Lo_D 0 0;Hi_D 0 0];
dft = [0 0 Lo_D; 0 0 Hi_D];
zer = df*dft'

zer =
```

```
    1.0e-12 *
    -0.1883 0.0000
    -0.0000 -0.1883


% High- and low-frequency illustration.
fftld = fft(Lo_D); ffthd = fft(Hi_D);
freq = [1:length(Lo_D)]/length(Lo_D);
subplot(427); plot(freq,abs(fftld));
title('Transfer modulus: low-pass');
subplot(428); plot(freq,abs(ffthd));
title('Transfer modulus: high-pass')
% Editing some graphical properties,
% the following figure is generated.
```

# orthfilt



Original scaling filter

Decomposition lowpass filter Lo_D

Decomposition highpass filter Hi_D

Reconstruction lowpass filter Lo_R

Reconstruction highpass filter Hi_R

Transfer modulus: lowpass (Lo_D or Lo_R)

Transfer modulus: highpass (Hi_D or Hi_R)

**References**     Daubechies, I. (1992), *Ten lectures on wavelets*, CBMS-NSF conference
series in applied mathematics, SIAM Ed. pp. 117–119, 137, 152.

**See Also**     biorfilt | qmf | wfilters

**Purpose**         Order terminal nodes of binary wavelet packet tree

**Syntax**          [Tn_Pal,Tn_Seq] = otnodes(WPT)
                    [Tn_Pal,Tn_Seq,I,J] = otnodes(WPT)
                    [DP_Pal,DP_Seq] = otnodes(WPT,'dp')

**Description**     [Tn_Pal,Tn_Seq] = otnodes(WPT) returns the terminal nodes of the
                    binary wavelet packet tree, WPT, in Paley (natural) ordering, Tn_Pal,
                    and sequency (frequency) ordering, Tn_Seq. Tn_Pal and Tn_Seq are
                    *N*-by-1 column vectors where *N* is the number of terminal nodes.

                    [Tn_Pal,Tn_Seq,I,J] = otnodes(WPT) returns the permutations of
                    the terminal node indices such that Tn_Seq = Tn_Pal(I) and Tn_Pal
                    = Tn_Seq(J).

                    [DP_Pal,DP_Seq] = otnodes(WPT,'dp') returns the Paley and
                    frequency-ordered terminal nodes in node depth-position format.
                    DP_Pal and DP_Seq are *N*-by-2 matrices. The first column contains the
                    depth index, and the second column contains the position index.

**Input**           **WPT**
**Arguments**
                    Binary wavelet packet tree. You can use treeord to determine the
                    order of your wavelet packet tree.

                    **dp**

                    String variable indicating that the Paley-ordered or sequency-ordered
                    nodes are returned in depth-position format.

**Output**          **Tn_Pal**
**Arguments**
                    Terminal nodes in Paley (natural) ordering

                    **Tn_Seq**

                    Terminal nodes in sequency ordering

                    **DP_Pal**

Paley-ordered terminal nodes in depth-position format. This output argument only applies when you use the `'dp'` input argument.

### DP_Seq

Sequency-ordered terminal nodes in depth-position format. This output argument only applies when you use the `'dp'` input argument.

**Definitions**     **Paley (Natural) and Sequency (Frequency) Ordering**

The discrete wavelet packet transform iterates on both approximation and detail coefficients at each level. In this transform, *A* denotes the lowpass (approximation) filter followed by downsampling. *D* denotes the highpass (detail) filter followed by downsampling. The following figure represents a wavelet packet transform in Paley ordering acting on a time series of length 8. The transform has a depth of two.

Because of aliasing introduced by downsampling, the frequency content extracted by the operator *AD* is higher than the frequency content extracted by the *DD* operator. Therefore, the terminal nodes in frequency (sequency) order are: *AA,DA,DD,AD*. The terminal nodes in Paley order have the following indices: 3,4,5,6. The frequency order has the indices: 3,4,6,5.

**Examples**   Order terminal nodes with Paley and frequency ordering:

```
x = randn(8,1);
wpt = wpdec(x,2,'haar');
[Tn_Pal,Tn_Seq] = otnodes(wpt);
% Tn_Pal is [3 4 5 6]
% Tn_Seq is [3 4 6 5]
```

Return permutations for Paley and frequency ordering:

```
load noisdopp;
wpt = wpdec(noisdopp,6,'sym4');
[Tn_Pal,Tn_Seq,I,J] = otnodes(wpt);
isequal(Tn_Seq(J),Tn_Pal)
isequal(Tn_Seq,Tn_Pal(I))
```

Order terminal nodes by depth and position:

```
x = randn(8,1);
wpt = wpdec(x,2,'haar');
[DP_Pal,DP_Seq] = otnodes(wpt,'dp');
```

Order terminal nodes from a modified wavelet packet tree:

```
 t = wptree(2,2,rand(1,512),'haar');
 t = wpsplt(t,4);
 t = wpsplt(t,5);
```

```
t = wpsplt(t,10);
plot(t);
[tn_Pal,tn_Seq,I,J] = otnodes(t);
```



**References**   Wickerhauser, M.V. *Lectures on Wavelet Packet Algorithms*, Technical Report, Washington University, Department of Mathematics, 1992.

**See Also**   leaves | treeord

**Purpose**      Build wavelet from pattern

**Syntax**       [PSI,XVAL,NC] = pat2cwav(YPAT,METHOD,POLDEGREE,REGULARITY)

**Description**  [PSI,XVAL,NC] = pat2cwav(YPAT,METHOD,POLDEGREE,REGULARITY)
computes an admissible wavelet for CWT (given by XVAL and PSI)
adapted to the pattern defined by the vector YPAT, and of norm equal
to 1.

The underlying x-values pattern is set to

xpat = linspace(0,1,length(YPAT))

The constant NC is such that NC*PSI approximates YPAT on the interval
[0,1] by least squares fitting using

- a polynomial of degree POLDEGREE when METHOD is equal to
  'polynomial'

- a projection on the space of functions orthogonal to constants when
  METHOD is equal to 'orthconst'

The REGULARITY parameter defines the boundary constraints at the
points 0 and 1. Allowable values are 'continuous', 'differentiable',
and 'none'.

When METHOD is equal to 'polynomial'

- if REGULARITY is equal to 'continuous', POLDEGREE must be greater
  than or equal to 3.

- if REGULARITY is equal to 'differentiable', POLDEGREE must be
  greater than or equal to 5.

**Examples**     The principle for designing a new wavelet for CWT is to approximate
a given pattern using least squares optimization under constraints
leading to an admissible wavelet well suited for the pattern detection
using the continuous wavelet transform (see Misiti et al.).

load ptpssin1;

```
plot(X,Y), title('Original Pattern')
```



Original Pattern

```
[psi,xval,nc] = pat2cwav(Y, 'polynomial',6, 'continuous') ;
plot(X,Y,'-',xval,nc*psi,'--'),
title('Original Pattern and Adapted Wavelet (dashed line)')
```



Original Pattern and Adapted Wavelet (dashed line)

You can check that psi satisfies the definition of a wavelet by noting that it integrates to zero and that its $L_2$ norm is equal to 1.

```
dx = xval(2)-xval(1);
Mu = sum(psi*dx)
L2norm = sum(abs(psi).^2*dx)
```

**References**    Misiti, M., Y. Misiti, G. Oppenheim, J.-M. Poggi (2003), "Les ondelettes et leurs applications," Hermes.

# plot

**Purpose**        Plot tree GUI

**Syntax**         plot(*T*)
                   plot(*T*,FIG)

**Description**    plot is a graphical tree-management utility.

                   plot(*T*) plots the tree *T*.

                   The figure that contains the tree is a GUI tool. It lets you change
                   the **Node Label** to **Depth_Position** or **Index**, and **Node Action** to
                   **Split-Merge** or **Visualize**.

                   The default values are **Depth_Position** and **Visualize**.

                   You can click the nodes to execute the current **Node Action**.

                   plot(*T*,FIG) plots the tree *T* in the figure whose handle is FIG. This
                   figure was already used to plot a tree, for example using the command

                   ```
                   FIG = plot(T)
                   ```

                   After some split or merge actions, you can get the new tree using
                   its parent figure handle. The following syntax lets you perform this
                   functionality:

                   ```
                   NEWT = plot(T,'read',FIG)
                   ```

                   In fact, the first argument is dummy. The most general syntax is

                   ```
                   NEWT = plot(DUMMY,'read',FIG)
                   ```

                   where DUMMY is any object parented by an NTREE object. More
                   generally, DUMMY can be any object constructor name returning an
                   NTREE parented object. For example:

                   ```
                   NEWT = plot(ntree,'read',FIG)
                   NEWT = plot(dtree,'read',FIG)
                   NEWT = plot(wptree,'read',FIG)
                   ```

**Examples**

```
% Create a wavelet packets tree (1-D)
load noisbloc
x = noisbloc;
t = wpdec(x,2,'db2');
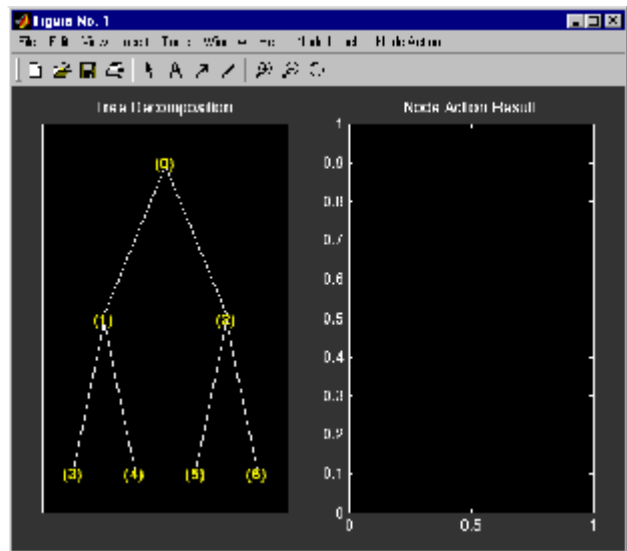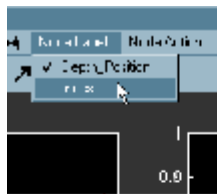
% Plot tree t.
plot(t)
```



% Change Node Label from Depth_Position to Index.

# plot



% Click the node (3). You get the following figure.

```
% Change Node Action from Visualize to Split_Merge.
```



```
% Merge the node (2) and split the node (3).
% Change Node Action from Split_Merge to Visualize.
% Click the node (7). You obtain the following figure,
% which represents the wavelet decomposition at level 3.
```

# plot

```
% Create a wavelet packets tree (2-D)
load woman2
t = wpdec2(X,1,'sym4');
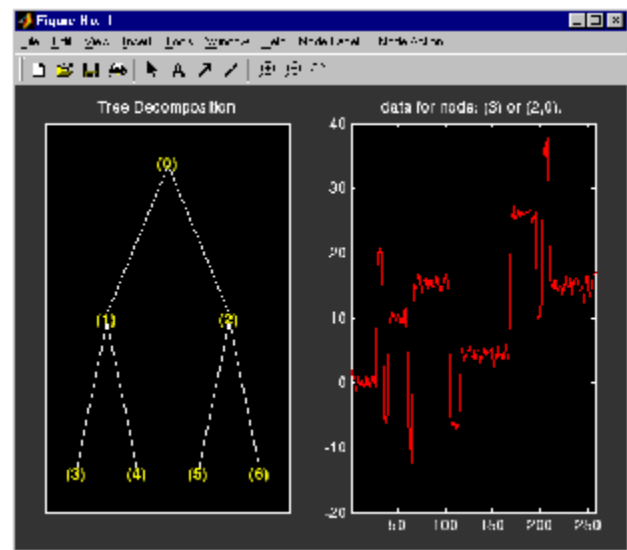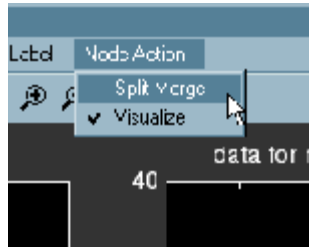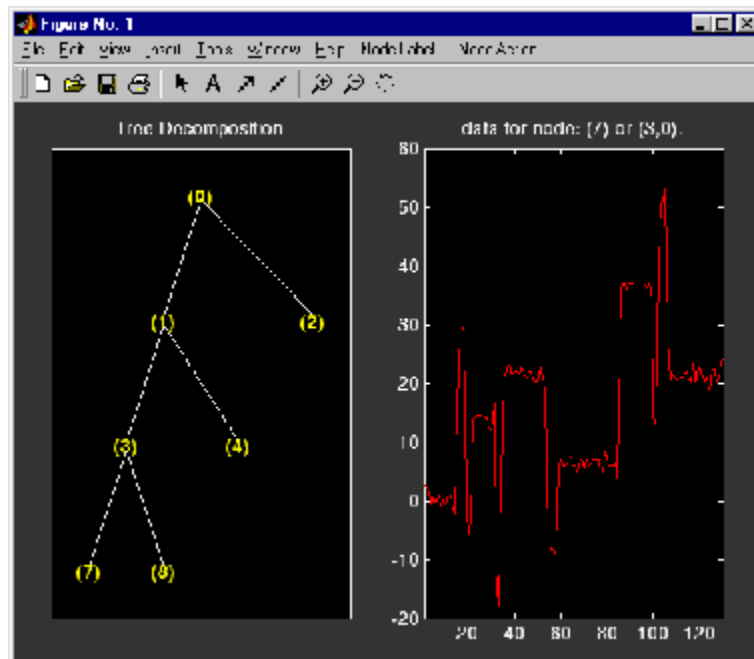
% Plot tree t.
plot(t)

% Change Node Label from Depth_Position to Index.
% Click the node (1). You get the following figure.
```

**Purpose**        Plot dual-tree or double-density wavelet transform

**Syntax**         plotdt(wt)

**Description**    plotdt(wt) plots the coefficients of the 1-D or 2-D wavelet filter bank
                   decomposition, wt.

**Input
Arguments**        **wt - Wavelet transform**
                   structure

                   Wavelet transform, returned as a structure from dddtree or dddtree2
                   with these fields:

                   **type - Type of wavelet decomposition (filter bank)**
                   'dwt' | 'ddt' | 'realdt' | 'cplxdt' | 'realdddt' | 'cplxdddt'

                   Type of wavelet decomposition (filter bank), specified as one of 'dwt',
                   'ddt', 'realdt', 'cplxdt',, 'realdddt', or 'cplxdddt'. 'realdt'
                   and 'realdddt' are only valid for the 2-D wavelet transform. The
                   type, 'dwt', is a critically sampled (nonredundant) discrete wavelet
                   transform for 1-D data or 2-D images. The other decomposition types
                   are oversampled wavelet transforms. For details about transform types
                   see dddtree for 1-D wavelet transforms and dddtree2 for 2-D wavelet
                   transforms.

                   **level - Level of the wavelet decomposition**
                   positive integer

                   Level of the wavelet decomposition, specified as a positive integer.

                   **filters - Decomposition (analysis) and reconstruction (synthesis)
                   filters**
                   structure

                   Decomposition (analysis) and reconstruction (synthesis) filters, specified
                   as a structure with these fields:

                   **Fdf - First-stage analysis filters**

matrix | cell array

First level decomposition filters specified as an *N*-by-2 or *N*-by-3 matrix
for single-tree wavelet transforms, or a 1-by-2 cell array of two *N*-by-2
or *N*-by-3 matrices for dual-tree wavelet transforms. The matrices
are *N*-by-3 for the double-density wavelet transforms. For an *N*-by-2
matrix, the first column of the matrix is the scaling (lowpass) filter
and the second column is the wavelet (highpass) filter. For an *N*-by-3
matrix, the first column of the matrix is the scaling (lowpass) filter and
the second and third columns are the wavelet (highpass) filters. For
the dual-tree transforms, each element of the cell array contains the
first-stage analysis filters for the corresponding tree.

### Df - Analysis filters for levels > 1

matrix | cell array

Analysis filters for levels > 1, specified as an *N*-by-2 or *N*-by-3 matrix
for single-tree wavelet transforms, or a 1-by-2 cell array of two *N*-by-2
or *N*-by-3 matrices for dual-tree wavelet transforms. The matrices
are *N*-by-3 for the double-density wavelet transforms. For an *N*-by-2
matrix, the first column of the matrix is the scaling (lowpass) filter
and the second column is the wavelet (highpass) filter. For an *N*-by-3
matrix, the first column of the matrix is the scaling (lowpass) filter
and the second and third columns are the wavelet (highpass) filters.
For the dual-tree transforms, each element of the cell array contains
the analysis filters for the corresponding tree.

### Frf - First-level reconstruction filters

matrix | cell array

First-level reconstruction filters, specified as an *N*-by-2 or *N*-by-3
matrix for single-tree wavelet transforms, or a 1-by-2 cell array of
two *N*-by-2 or *N*-by-3 matrices for dual-tree wavelet transforms. The
matrices are *N*-by-3 for the double-density wavelet transforms. For an
*N*-by-2 matrix, the first column of the matrix is the scaling (lowpass)
filter and the second column is the wavelet (highpass) filter. For an
*N*-by-3 matrix, the first column of the matrix is the scaling (lowpass)
filter and the second and third columns are the wavelet (highpass)

filters. For the dual-tree transforms, each element of the cell array contains the first-stage synthesis filters for the corresponding tree.

### Rf - Reconstruction filters for levels > 1
matrix | cell array

Reconstruction filters for levels > 1, specified as an *N*-by-2 or *N*-by-3 matrix for single-tree wavelet transforms, or a 1-by-2 cell array of two *N*-by-2 or *N*-by-3 matrices for dual-tree wavelet transforms. The matrices are *N*-by-3 for the double-density wavelet transforms. For an *N*-by-2 matrix, the first column of the matrix is the scaling (lowpass) filter and the second column is the wavelet (highpass) filter. For an *N*-by-3 matrix, the first column of the matrix is the scaling (lowpass) filter and the second and third columns are the wavelet (highpass) filters. For the dual-tree transforms, each element of the cell array contains the first-stage synthesis filters for the corresponding tree.

### cfs - Wavelet transform coefficients
cell array of matrices

Wavelet transform coefficients, specified as a 1-by-(`level`+1) cell array of matrices. The size and structure of the matrix elements of the cell array depend on the type of wavelet transform and whether the decomposition is 1-D or 2-D. For a 1-D wavelet transform, the coefficients are organized by transform type as follows:

- `'dwt'` — `cfs{j}`
  - `j = 1,2,...,` `level` is the level.
  - `cfs{level+1}` are the lowpass, or scaling, coefficients.
- `'ddt'` — `cfs{j}(:,:,k)`
  - `j = 1,2,...,` `level` is the level.
  - `k = 1,2` is the wavelet filter.
  - `cfs{level+1}(:,:)` are the lowpass, or scaling, coefficients.
- `'cplxdt'` — `cfs{j}(:,:,m)`

- ▪ `j = 1,2,...,` `level` is the level.
- ▪ `m = 1,2` are the real and imaginary parts.
- ▪ `cfs{level+1}(:,:)` are the lowpass, or scaling, coefficients.

- `'realdddt'` — `cfs{j}(:,:,d,k)`
  - ▪ `j = 1,2,...,` `level` is the level.
  - ▪ `d = 1,2,3` is the orientation.
  - ▪ `k = 1,2` is the wavelet transform tree.
  - ▪ `cfs{level+1}(:,:)` are the lowpass, or scaling, coefficients.

- `'cplxdddt'` — `cfs{j}(:,:,d,k,m)`
  - ▪ `j = 1,2,...,` `level` is the level.
  - ▪ `k = 1,2` is the wavelet transform tree.
  - ▪ `m = 1,2` are the real and imaginary parts.
  - ▪ `cfs{level+1}(:,:)` are the lowpass, or scaling, coefficients.

For a 2-D wavelet transform, the coefficients are organized by transform type as follows:

- `'dwt'` — `cfs{j}(:,:,d)`
  - ▪ `j = 1,2,...,` `level` is the level.
  - ▪ `d = 1,2,3` is the orientation.
  - ▪ `cfs{level+1}(:,:)` are the lowpass, or scaling, coefficients.

- `'ddt'` — `cfs{j}(:,:,d)`
  - ▪ `j = 1,2,...,` `level` is the level.
  - ▪ `d = 1,2,3,4,5,6,7,8` is the orientation.
  - ▪ `cfs{level+1}(:,:)` are the lowpass, or scaling, coefficients.

- `'realddt'` — `cfs{j}(:,:,d,k)`
  - ▪ `j = 1,2,...,` `level` is the level.

- $d = 1,2,3$ is the orientation.

  - $k = 1,2$ is the wavelet transform tree.

  - `cfs{level+1}(:,:)` are the lowpass, or scaling, coefficients.

- `'cplxdt'` — `cfs{j}(:,:,d,k,m)`

  - $j = 1,2,...,$ `level` is the level.

  - $d = 1,2,3$ is the orientation.

  - $k = 1,2$ is the wavelet transform tree.

  - $m = 1,2$ are the real and imaginary parts.

  - `cfs{level+1}(:,:)` are the lowpass, or scaling, coefficients.

- `'realdddt'` — `cfs{j}(:,:,d,k)`

  - $j = 1,2,...,$ `level` is the level.

  - $d = 1,2,3$ is the orientation.

  - $k = 1,2$ is the wavelet transform tree.

  - `cfs{level+1}(:,:)` are the lowpass, or scaling, coefficients.

- `'cplxdddt'` — `cfs{j}(:,:,d,k,m)`

  - $j = 1,2,...,$ `level` is the level.

  - $d = 1,2,3$ is the orientation.

  - $k = 1,2$ is the wavelet transform tree.

  - $m = 1,2$ are the real and imaginary parts.

  - `cfs{level+1}(:,:)` are the lowpass, or scaling, coefficients.

**Examples**      **Plot Complex Dual-Tree Wavelet Transform of 1-D Signal**

Plot the complex dual-tree wavelet transform of the noisy Doppler signal.

Load the noisy Doppler signal. Obtain the complex dual-tree wavelet transform down to level 4.

```
load noisdopp;
wt = dddtree('cplxdt',noisdopp,4,'dtf1');
```

Plot the coefficients.

```
plotdt(wt)
```

### Plot Complex Oriented Dual-Tree Wavelet Transform of 2-D Image

Plot the complex oriented dual-tree wavelet transform of an image.

Load the "xbox" image. Obtain the complex oriented dual-tree wavelet transform down to level 3.

```
load xbox;
wt = dddtree2('cplxdt',xbox,3,'dtf1');
```

Plot the coefficients.

```
plotdt(wt)
```

Select the level-one detail coefficients from the drop-down list in the lower left corner.

Coefficients of level 1

$C_{111}$  $C_{121}$  $C_{112}$  $C_{122}$

$C_{211}$  $C_{221}$  $C_{212}$  $C_{222}$

$C_{311}$  $C_{321}$  $C_{312}$  $C_{322}$

Level 1

# plotdt

**See Also**  dddtree **|** dddtree2 **|** dddtreecfs

**Related Examples**  • "Analytic Wavelets Using the Dual-Tree Wavelet Transform"

**Concepts**  • "Critically Sampled and Oversampled Wavelet Filter Banks"

**Purpose**          Scaling and Wavelet Filter

**Syntax**           Y = qmf(*X*,*P*)
                     Y = qmf(*X*)
                     Y = qmf(X,0)

**Description**      Y = qmf(*X*,*P*) changes the signs of the even index entries of the
                     reversed vector filter coefficients *X* if *P* is even. If *P* is odd the same holds
                     for odd index entries. Y = qmf(*X*) is equivalent to Y = qmf(X,0).

                     Let x be a finite energy signal. Two filters $F_0$ and $F_1$ are quadrature
                     mirror filters (QMF) if, for any *x*,

                     $$\|y_0\|^2 + \|y_1\|^2 = \|x\|^2$$

                     where $y_0$ is a decimated version of the signal *x* filtered with $F_0$ so $y_0$
                     defined by $x_0 = F_0(x)$ and $y_0(n) = x_0(2n)$, and similarly, $y_1$ is defined by $x_1$
                     $= F_1(x)$ and $y_1(n) = x_1(2n)$. This property ensures a perfect reconstruction
                     of the associated two-channel filter banks scheme (see Strang-Nguyen
                     p. 103).

                     For example, if $F_0$ is a Daubechies scaling filter and $F_1 = $ qmf($F_0$), then
                     the transfer functions $F_0(z)$ and $F_1(z)$ of the filters $F_0$ and $F_1$ satisfy the
                     condition (see the example for *db*10):

                     $$|F_0(z)|^2 + |F_1(z)|^2 = 1$$

**Examples**        % Load scaling filter associated with an orthogonal wavelet.
                     load db10;
                     subplot(321); stem(db10); title('db10 low-pass filter');

                     % Compute the quadrature mirror filter.
                     qmfdb10 = qmf(db10);
                     subplot(322); stem(qmfdb10); title('QMF db10 filter');

                     % Check for frequency condition (necessary for orthogonality):

```
% abs(fft(filter))^2 + abs(fft(qmf(filter))^2 = 1 at each
% frequency.
m = fft(db10);
mt = fft(qmfdb10);
freq = [1:length(db10)]/length(db10);
subplot(323); plot(freq,abs(m));
title('Transfer modulus of db10')
subplot(324); plot(freq,abs(mt));
title('Transfer modulus of QMF db10')
subplot(325); plot(freq,abs(m).^2 + abs(mt).^2);
title('Check QMF condition for db10 and QMF db10')
xlabel(' abs(fft(db10))^2 + abs(fft(qmf(db10))^2 = 1')
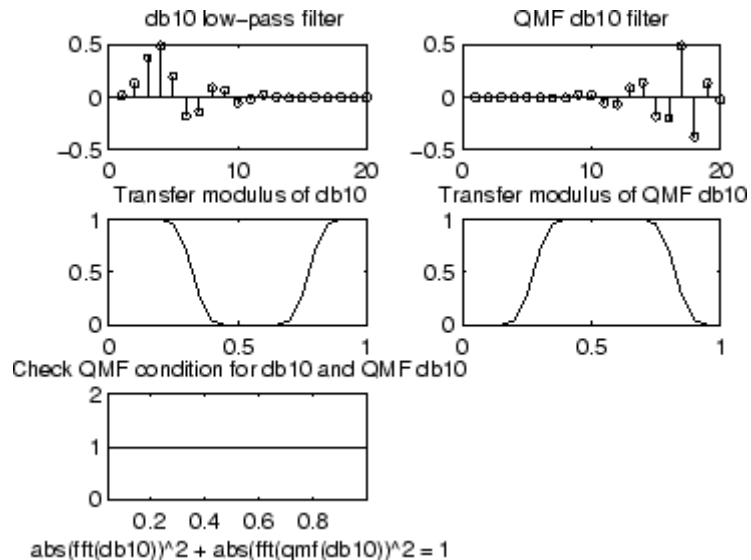
% Editing some graphical properties,
% the following figure is generated.
```



```
% Check for orthonormality.
df = [db10;qmfdb10]*sqrt(2);
id = df*df'
```

```
id =
    1.0000 0.0000
    0.0000 1.0000
```

**References**     Strang, G.; T. Nguyen (1996), *Wavelets and Filter Banks*,
                   Wellesley-Cambridge Press.

# rbiowavf

**Purpose**         Reverse biorthogonal spline wavelet filters

**Syntax**         [RF,DF] = rbiowavf(*W*)

**Description**    [RF,DF] = rbiowavf(*W*) returns the two scaling filters associated with the biorthogonal wavelet specified by the string *W*.

W = 'rbioNd.Nr' where possible values for Nd and Nr are

| | |
|---|---|
| Nd = 1 | Nr = 1 , 3 or 5 |
| Nd = 2 | Nr = 2 , 4 , 6 or 8 |
| Nd = 3 | Nr = 1 , 3 , 5 , 7 or 9 |
| Nd = 4 | Nr = 4 |
| Nd = 5 | Nr = 5 |
| Nd = 6 | Nr = 8 |

The output arguments are filters.

- RF is the reconstruction filter.
- DF is the decomposition filter.

**Reverse Biorthgonal Scaling Filter**

Obtain the reverse biorthogonal reconstruction and decomposition scaling filters for the 'rbio3.1' wavelet. The 'rbio3.1' wavelet has 3 vanishing moments for the decomposition (analysis) wavelet and 1 vanishing moment for the reconstruction (synthesis) wavelet.

```
[RF,DF] = rbiowavf('rbio3.1');
```

The reconstruction scaling filter, RF, and the decomposition filter, DF, are equal to the filters returned by wfilters scaled by $\sqrt{2}$.

```
[LoD,HiD,LoR,HiR] = wfilters('rbio3.1');
max(abs(sqrt(2)*DF-LoD))
max(abs(sqrt(2)*RF-LoR))
```

```
ans =

     0


ans =

     0
```

**See Also**      biorfilt | waveinfo

# read

| **Purpose** | Read values of WPTREE |
| --- | --- |

**Syntax**      VARARGOUT = read(T,VARARGIN)

**Description**   VARARGOUT = read(T,VARARGIN) is the most general syntax to read one
or more property values from the fields of a WPTREE object .

The different ways to call the read function are

```
PropValue = read(T,'PropName') or
PropValue = read(T,'PropName','PropParam')
```

or any combination of the previous syntaxes:

```
[PropValue1,PropValue2, ] =
read(T,'PropName1','PropParam1','PropName2','PropParam2', )
```

where '*PropParam*' is optional.

The valid choices for '*PropName*' and '*PropParam*' are listed in this
table.

| *PropName* | *PropParam* |
| --- | --- |
| 'ent', 'ento' or 'sizes' (see wptree) | Without '*PropParam*' or with '*PropParam*' = Vector of node indices, PropValue contains the entropy (or optimal entropy, or size) of the tree nodes in ascending node index order. |
| 'cfs' | With '*PropParam*' = One terminal node index. cfs = read(T,'cfs',NODE) is equivalent to cfs = read(T,'data',NODE) and returns the coefficients of the terminal node NODE. |

| *PropName* | *PropParam* |
|---|---|
| 'entName', 'entPar', 'wavName' (see wptree) or 'allcfs' | Without '*PropParam*'. cfs = read(T,'allcfs') is equivalent to cfs = read(T,'data'). PropValue contains the desired information in ascending node index order of the tree nodes. |
| 'wfilters' (see wfilters) | Without '*PropParam*' or with '*PropParam*' = 'd','r','l','h'. |
| 'data' | Without '*PropParam*' or with '*PropParam*' = One terminal node index or '*PropParam*' = Column vector of terminal node indices.In this last case, PropValue is a cell array. Without '*PropParam*', PropValue contains the coefficients of the tree nodes in ascending node index order. |

**Examples**

```
% Create a wavelet packet tree.
x = rand(1,512);
t = wpdec(x,3,'db3');
t = wpjoin(t,[4;5]);
plot(t);

% Click the node (3,0), (see the plot function).
l% Read values.

sAll = read(t,'sizes');
sNod = read(t,'sizes',[0,4,5]);
eAll = read(t,'ent');
eNod = read(t,'ent',[0,4,5]);
dAll = read(t,'data');
dNod = read(t,'data',[4;5]);
[lo_D,hi_D,lo_R,hi_R] = read(t,'wfilters');
```

```
[lo_D,lo_R,hi_D,hi_R] = read(t,'wfilters','l','wfilters','h');
[ent,ento,cfs4,cfs5]  = read(t,'ent','ento','cfs',4,'cfs',5);
```



data for node: (7) or (3,0).

**See Also**     disp | get | set | wptree | write

**Purpose**    Read wavelet packet decomposition tree from figure

**Syntax**    T = readtree(*F*)

**Description**    T = readtree(*F*) reads the wavelet packet decomposition tree from the figure whose handle is *F*.

**Examples**    
```
% Create a wavelet packet tree.
x   = sin(8*pi*[0:0.005:1]);
t   = wpdec(x,3,'db2');
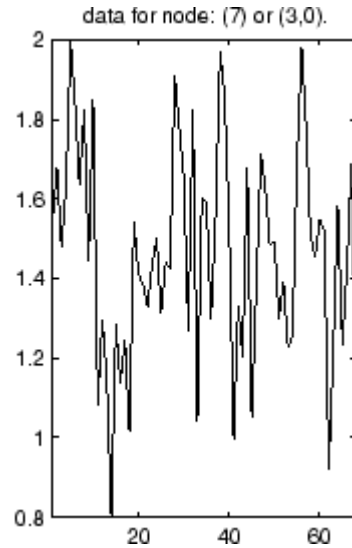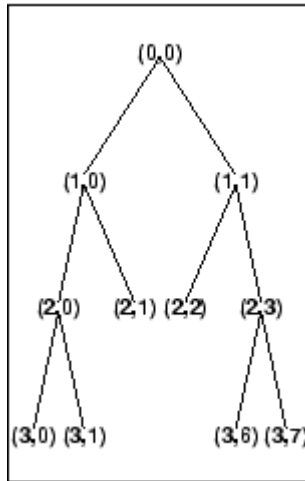
% Display the generated tree in a Wavelet Packet 1-D GUI window.
fig = drawtree(t);
```



```
%- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
% Use the GUI to split or merge Nodes.
%-----------------------------------
```



```
t = readtree(fig);
plot(t)
```

```
% Click the node (3,0), (see the plot function).
```

data for node: (7) or (3,0).

**See Also**     drawtree

# scal2frq

| | |
|---|---|
| **Purpose** | Scale to frequency |

**Syntax**

```
F = scal2frq(A,'wname',DELTA)
scal2frq(A,'wname')
scal2frq(A,'wname',1)
```

**Description**    `F = scal2frq(A,'wname',DELTA)` returns the pseudo-frequencies corresponding to the scales given by `A`, the wavelet function `'wname'` (see `wavefun` for more information) and the sampling period `DELTA`.

`scal2frq(A,'wname')` is equivalent to `scal2frq(A,'wname',1)`.

One of the most frequently asked questions is "How does one map a scale, for a given wavelet and a sampling period, to a kind of frequency?"

The answer can only be given in a broad sense and it's better to speak about the pseudo-frequency corresponding to a scale.

A way to do it is to compute the center frequency, $F_c$, of the wavelet and to use the following relationship.

$$F_a = \frac{F_c}{a \cdot \Delta}$$

where

- $a$ is a scale.
- $\Delta$ is the sampling period.
- $F_c$ is the center frequency of a wavelet in Hz.
- $F_a$ is the pseudo-frequency corresponding to the scale $a$, in Hz.

The idea is to associate with a given wavelet a purely periodic signal of frequency $F_c$. The frequency maximizing the `fft` of the wavelet modulus is $F_c$. The function `centfrq` can be used to compute the center frequency and it allows the plotting of the wavelet with the associated approximation based on the center frequency. The following figure (Center Frequencies for Real and Complex Wavelets on page 1-386) shows some examples generated using the `centfrq` function.

- Four real wavelets: Daubechies wavelets of order 2 and 7, coiflet of order 1, and the Gaussian derivative of order 4.

- Two complex wavelets: the complex Gaussian derivative of order 6 and a Shannon complex wavelet.

**Center Frequencies for Real and Complex Wavelets**

As you can see, the center frequency based approximation captures the main wavelet oscillations. So the center frequency is a convenient and simple characterization of the leading dominant frequency of the wavelet.

If we accept to associate the frequency $F_c$ to the wavelet function then, when the wavelet is dilated by a factor $a$, this center frequency becomes $F_c / a$. Lastly, if the underlying sampling period is $\Delta$, it is natural to associate to the scale $a$ the frequency

$$F_a = \frac{F_c}{a \cdot \Delta}$$

The function scal2frq computes this correspondence.

To illustrate the behavior of this procedure, let us consider the following simple test. We generate sine functions of sensible frequencies $F_0$. For each function, we shall try to detect this frequency by a wavelet decomposition followed by a translation of scale to frequency. More precisely, after a discrete wavelet decomposition, we identify the scale $a^*$ corresponding to the maximum value of the energy of the coefficients. The translated frequency $F^*$ is then given by

```
scal2frq(a_star,'wname',sampling_period)
```

The $F^*$ values are close to the chosen $F_0$. The plots at the end of example 2 presents the periods instead of frequencies. If we change slightly the $F_0$ values, the results remain satisfactory.

## Examples

### Example 1

```
% Set sampling period and wavelet name.
delta = 0.1; wname = 'coif3';

% Define scales.
amax = 7; a = 2.^[1:amax];

% Compute associated pseudo-frequencies.
```

```
f = scal2frq(a,wname,delta);

% Compute associated pseudo-periods.
per = 1./f;


% Display information.
disp('    Scale    Frequency  Period')
disp([a' f' per'])

    Scale    Frequency  Period

    2.0000    3.5294    0.2833
    4.0000    1.7647    0.5667
    8.0000    0.8824    1.1333
   16.0000    0.4412    2.2667
   32.0000    0.2206    4.5333
   64.0000    0.1103    9.0667
  128.0000    0.0551   18.1333
```

### Example 2

```
% Set sampling period and wavelet name.
delta = 0.1; wname = 'coif3';

% Define scales.
amax = 7;
a = 2.^[1:amax];

% Compute associated pseudo-frequencies.
f = scal2frq(a,wname,delta);

% Compute associated pseudo-periods.
per = 1./f;

% Plot pseudo-periods versus scales.
subplot(211), plot(a,per)
```

```
title(['Wavelet: ',wname, ', Sampling period: ',num2str(delta)])
xlabel('Scale')
ylabel('Computed pseudo-period')

% For each scale 2^i:
% - generate a sine function of period per(i);
% - perform a wavelet decomposition;
% - identify the highest energy level;
% - compute the detected pseudo-period.

for i = 1:amax
   % Generate sine function of period
   % per(i) at sampling period delta.
   t = 0:delta:100;
   x = sin((t.*2*pi)/per(i));
   % Decompose x at level 9.
   [c,l] = wavedec(x,9,wname);

   % Estimate standard deviation of detail coefficients.
   stdc = wnoisest(c,l,[1:amax]);
   % Compute identified period.
   [y,jmax] = max(stdc);
   idper(i) = per(jmax);
end

% Compare the detected and computed pseudo-periods.
subplot(212), plot(per,idper,'o',per,per)
title('Detected vs computed pseudo-period')
xlabel('Computed pseudo-period')
ylabel('Detected pseudo-period')
```

Wavelet: coif3, Sampling period: 0.1

Detected vs computed pseudo-period

### Example 3

This example demonstrates that, starting from the periodic function
x(t) = cos(5t), the scal2frq function translates the scale
corresponding to the maximum value of the CWT coefficients to a
pseudo-frequency (0.795), which is near to the true frequency (5/(2*pi)
=~ 0.796).

```
% Set wavelet name, interval and number of samples.
wname = 'db10';
A = -64; B = 64; P = 224;
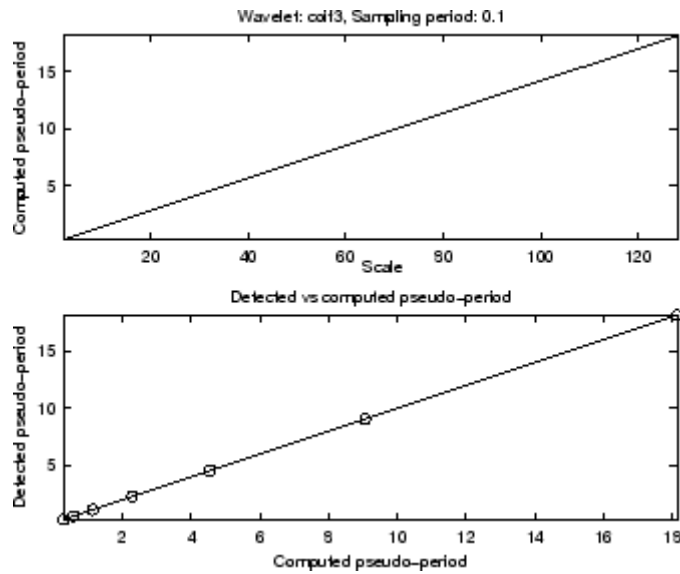
% Compute the sampling period and the sampled function,
% and the true frequency.
delta = (B-A)/(P-1);
t = linspace(A,B,P);
omega = 5; x = cos(omega*t);
freq  = omega/(2*pi);
```

```
% Set scales and use scal2frq to compute the array
% of pseudo-frequencies.
scales = [0.25:0.25:3.75];
TAB_PF = scal2frq(scales,wname,delta);

% Compute the nearest pseudo-frequency
% and the corresponding scale.
[dummy,ind] = min(abs(TAB_PF-freq));
freq_APP  = TAB_PF(ind);
scale_APP = scales(ind);

% Continuous analysis and plot.
str1 = ['224 samples of x = cos(5t) on [-64,64] - ' ...
        'True frequency = 5/(2*pi) =~ ' num2str(freq,3)];
str2 = ['Array of pseudo-frequencies and scales: '];
str3 = [num2str([TAB_PF',scales'],3)];
str4 = ['Pseudo-frequency = ' num2str(freq_APP,3)];
str5 = ['Corresponding scale = ' num2str(scale_APP,3)];
figure; cwt(x,scales,wname,'plot'); ax = gca; colorbar
axTITL = get(ax,'title');
axXLAB = get(ax,'xlabel');
set(axTITL,'String',str1)
set(axXLAB,'String',[str4,'  -  ' str5])
clc ; disp(char(' ',str1,' ',str2,str3,' ',str4,str5))

224 samples of x = cos(5t) on [-64,64] -
True frequency = 5/(2*pi) =~ 0.796

Array of pseudo-frequencies and scales:
 4.77      0.25
 2.38       0.5
 1.59      0.75
 1.19         1
0.954      1.25
0.795       1.5
0.681      1.75
0.596         2
```

```
 . . . .
0.341       3.5
0.318       3.75
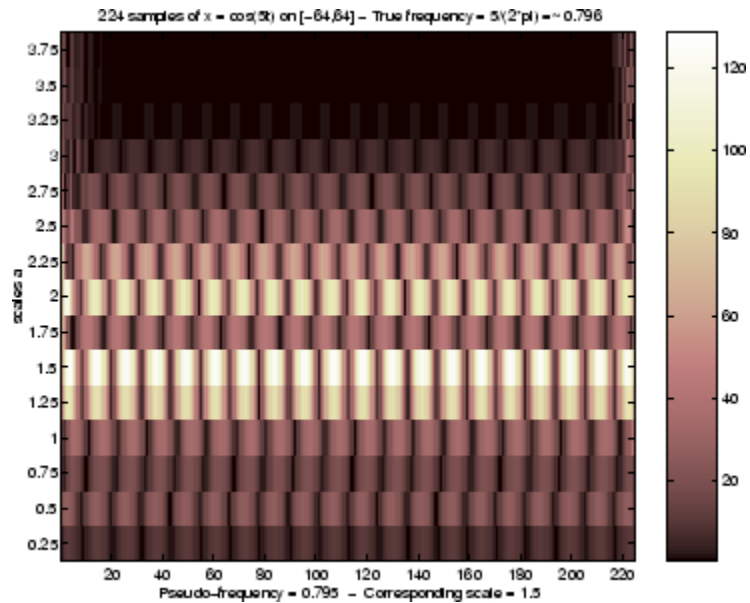

Pseudo-frequency = 0.795
Corresponding scale = 1.5
```



224 samples of x = cos(5t) on [-64,64] - True frequency = 5/(2*pi) =~ 0.796
Pseudo-frequency = 0.795 - Corresponding scale = 1.5

### Example 4

This example demonstrates that, starting from the periodic function
x(t) = 5*sin(5t)+3*sin(2t)+2*sin(t), the scal2frq function
translates the scales corresponding to the maximum values of the CWT
coefficients to pseudo-frequencies ([0.796 0.318 0.159]), which are
near to the true frequencies ([5 2 1] / (2*pi) =~ [0.796 0.318
0.159]).

```
% Set wavelet name,interval and number of samples.
wname  = 'morl';
A = 0; B = 64; P = 500;
```

```
% Compute the sampling period and the sampled function,
% and the true frequencies.
t = linspace(A,B,P);
delta = (B-A)/(P-1);
tab_OMEGA = [5,2,1];
tab_FREQ  = tab_OMEGA/(2*pi);
tab_COEFS = [5,3,2];
x = zeros(1,P);
for k = 1:3;
    x = x+tab_COEFS(k)*sin(tab_OMEGA(k)*t);
end

% Set scales and use scal2frq to compute the array
% of pseudo-frequencies.
scales = [1:1:60];
tab_PF = scal2frq(scales,wname,delta);

% Compute the nearest pseudo-frequencies
% and the corresponding scales.
for k=1:3
  [dummy,ind] = min(abs(tab_PF-tab_FREQ(k)));
  PF_app(k) = tab_PF(ind);
  SC_app(k) = scales(ind);
end

% Continuous analysis and plot.
str1 = char( ...
   '500 samples of x = 5*sin(5t)+3*sin(2t)+2*sin(t) on [0,64]',...
   ['True frequencies (in Hz): [5 2 1]/(2*pi) =~ [' ...
   num2str(tab_FREQ,3) ']' ]  ...
   );
str2 = ['Array of pseudo-frequencies and scales: '];
str3 = [num2str([tab_PF',scales'],3)];
str4 = ['Pseudo-frequencies   = ' num2str(PF_app,3)];
str5 = ['Corresponding scales = ' num2str(SC_app,3)];
figure; cwt(x,scales,wname,'plot'); ax = gca; colorbar
```

```
axTITL = get(ax,'title');
axXLAB = get(ax,'xlabel');
set(axTITL,'String',str1)
set(axXLAB,'String',char(str4, str5))
clc; disp(char(' ',str1,' ',str2,str3,' ',str4,str5))

500 samples of x = 5*sin(5t)+3*sin(2t)+2*sin(t) on [0,64]
True frequencies (in Hz): [5 2 1]/(2*pi) =~ [0.796   0.318    0.159]

Array of pseudo-frequencies and scales:
 6.33        1
 3.17        2
 2.11        3
 1.58        4
 1.27        5
 1.06        6
0.905        7
0.792        8
0.704        9
0.633       10


. . . .
. . . .

0.122       52
 0.12       53
0.117       54
0.115       55
0.113       56
0.111       57
0.109       58
0.107       59
0.106       60
Pseudo-frequencies  = 0.792     0.317      0.158
Corresponding scales = 8         20         40
```

500 samples of x = 5*sin(5t)+3*sin(2t)+2*sin(t) on [0,64]
True frequencies: [5 2 1]/(2*pi) =~ [0.796    0.318    0.159]

**References**   Abry, P. (1997), *Ondelettes et turbulence. Multirésolutions, algorithmes de décomposition, invariance d'échelles*, Diderot Editeur, Paris.

**See Also**   centfrq

**set**

| | |
|---|---|
| **Purpose** | WPTREE field contents |
| **Syntax** | T = set(T,'*FieldName1*',FieldValue1,'*FieldName2*',FieldValue2, ...) |
| **Description** | T = set(T,'*FieldName1*',FieldValue1,'*FieldName2*',FieldValue2, ...) sets the content of the specified fields for the WPTREE object T. |

For the fields that are objects or structures, you can set the subfield contents, giving the name of these subfields as '*FieldName*' values.

The valid choices for '*FieldName*' are

| | |
|---|---|
| 'dtree' | DTREE parent object |
| 'wavInfo' | Structure (wavelet information) |

The fields of the wavelet information structure, 'wavInfo', are also valid for '*FieldName*':

| | |
|---|---|
| 'wavName' | Wavelet name |
| 'Lo_D' | Low Decomposition filter |
| 'Hi_D' | High Decomposition filter |
| 'Lo_R' | Low Reconstruction filter |
| 'Hi_R' | High Reconstruction filter |

| | |
|---|---|
| 'entInfo' | Structure (entropy information) |

The fields of the entropy information structure, 'entInfo', are also valid for '*FieldName*':

| | |
|---|---|
| `'entName'` | Entropy name |
| `'entPar'` | Entropy parameter |

Or fields of DTREE parent object:

| | |
|---|---|
| `'ntree'` | NTREE parent object |
| `'allNI'` | All nodes information |
| `'terNI'` | Terminal nodes information |

Or fields of NTREE parent object:

| | |
|---|---|
| `'wtbo'` | WTBO parent object |
| `'order'` | Order of the tree |
| `'depth'` | Depth of the tree |
| `'spsch'` | Split scheme for nodes |
| `'tn'` | Array of terminal nodes of the tree |

Or fields of WTBO parent object:

| | |
|---|---|
| `'wtboInfo'` | Object information |
| `'ud'` | Userdata field |

**Caution** The set function should only be used to set the field '*ud*'.

**See Also**      disp | get | read | write

# shanwavf

**Purpose**     Complex Shannon wavelet

**Syntax**      [PSI,X] = shanwavf(LB,UB,N,FB,FC)

**Description**  [PSI,X] = shanwavf(LB,UB,N,FB,FC) returns values of the complex
Shannon wavelet defined by a bandwidth parameter FB, a wavelet
center frequency FC, and the expression

```
PSI(X) = (FB^0.5)*(sinc(FB*X).*exp(2*i*pi*FC*X))
```

on an N point regular grid in the interval [LB,UB].

FB and FC must be such that FC > 0 and FB > 0.

Output arguments are the wavelet function PSI computed on the grid X.

**Examples**   ```
% Set bandwidth and center frequency parameters.
fb = 1; fc = 1.5;

% Set effective support and grid parameters.
lb = -20; ub = 20; n = 1000;

% Compute complex Shannon wavelet shan1.5-1.
[psi,x] = shanwavf(lb,ub,n,fb,fc);

% Plot complex Shannon wavelet.
subplot(211)
plot(x,real(psi)),
title('Complex Shannon wavelet shan1.5-1')
xlabel('Real part'), grid
subplot(212)
plot(x,imag(psi))
xlabel('Imaginary part'), grid
```

Complex Shannon wavelet shan1.5−1
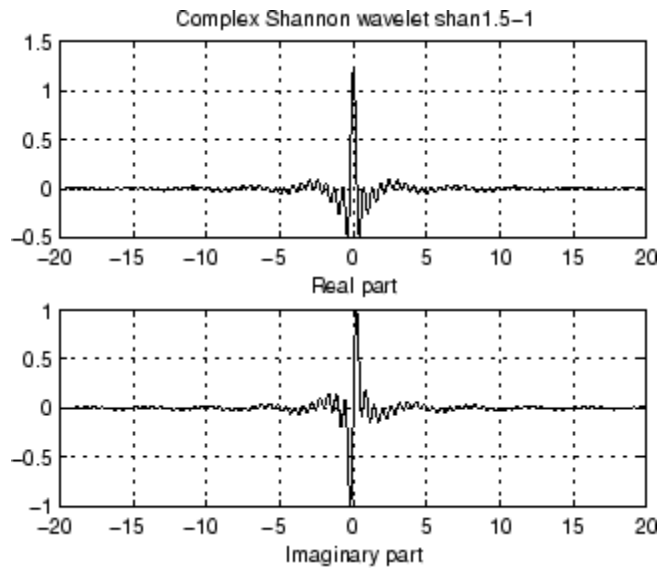


**References**     Teolis, A. (1998), *Computational signal processing with wavelets*, Birkäuser, p. 62.

**See Also**     `waveinfo`

| **Purpose** | Discrete stationary wavelet transform 1-D |
|---|---|

| **Syntax** | SWC = swt(X,N,'*wname*') |
|---|---|
| | SWC = swt(X,N,Lo_D,Hi_D) |

**Description**  swt performs a multilevel 1-D stationary wavelet decomposition using either a specific orthogonal wavelet ('*wname*', see wfilters for more information) or specific orthogonal wavelet decomposition filters.

SWC = swt(X,N,'*wname*') computes the stationary wavelet decomposition of the signal X at level N, using '*wname*'.

N must be a strictly positive integer (see wmaxlev for more information) and length(X) must be a multiple of $2^N$.

SWC = swt(X,N,Lo_D,Hi_D) computes the stationary wavelet decomposition as above, given these filters as input:

• Lo_D is the decomposition low-pass filter.

• Hi_D is the decomposition high-pass filter.

Lo_D and Hi_D must be the same length.

The output matrix SWC contains the vectors of coefficients stored row-wise:

For $1 \leq i \leq N$, the output matrix SWC(i,:) contains the detail coefficients of level i and SWC(N+1,:) contains the approximation coefficients of level N.

[SWA,SWD] = swt( ) computes approximations, SWA, and details, SWD, stationary wavelet coefficients.

The vectors of coefficients are stored row-wise:

For $1 \leq i \leq N$, the output matrix SWA(i,:) contains the approximation coefficients of level i and the output matrix SWD(i,:) contains the detail coefficients of level i.

---

**Note** swt is defined using dwt with periodic extension.

---

**Examples**

```
% Load original 1D signal.
load noisbloc; s = noisbloc;

% Perform SWT decomposition at level 3 of s using db1.
[swa,swd] = swt(s,3,'db1');

% Plots of SWT coefficients of approximations and details
% at levels 3 to 1.

% Using some plotting commands,
% the following figure is generated.
```

Original signal s.

Original signal s.

SWT Coefs: approx. at levels 3, 2 and 1    SWT Coefs: detail at levels 3, 2 and 1
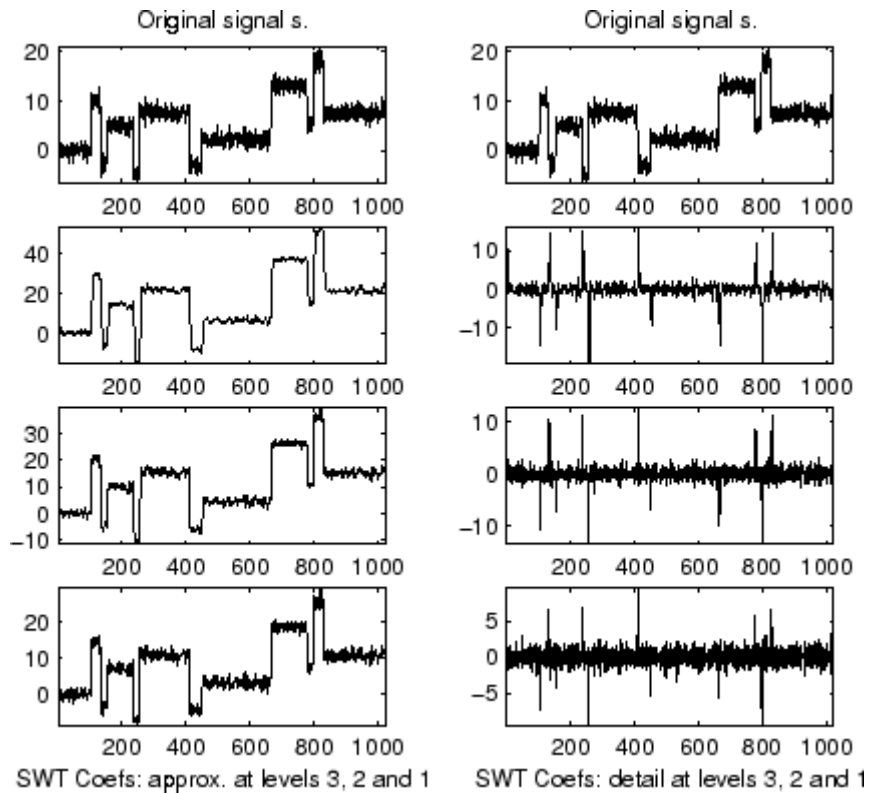
**Algorithms**     Given a signal *s* of length *N*, the first step of the SWT produces, starting from *s*, two sets of coefficients: approximation coefficients $cA_1$ and detail coefficients $cD_1$. These vectors are obtained by convolving *s* with the low-pass filter Lo_D for approximation, and with the high-pass filter Hi_D for detail.

More precisely, the first step is

---
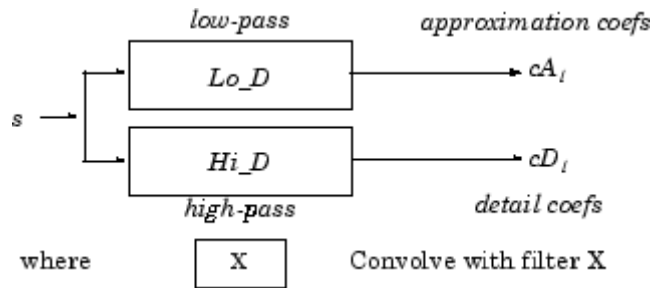
**Note** $cA_1$ and $cD_1$ are of length N instead of N/2 as in the DWT case.

---

The next step splits the approximation coefficients $cA_1$ in two parts using the same scheme, but with modified filters obtained by upsampling the filters used for the previous step and replacing $s$ by $cA_1$. Then, the SWT produces $cA_2$ and $cD_2$. More generally,
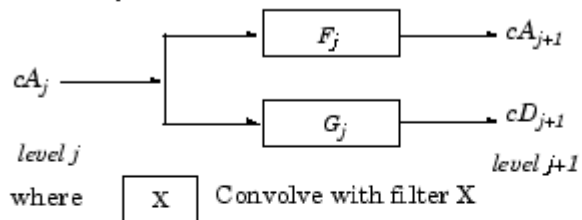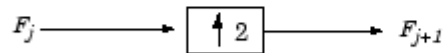


1-403

**References**    Nason, G.P.; B.W. Silverman (1995), "The stationary wavelet transform and some statistical applications," *Lecture Notes in Statistics*, 103, pp. 281–299.

Coifman, R.R.; Donoho, D.L. (1995), "Translation invariant de-noising," *Lecture Notes in Statistics*, 103, pp. 125–150.

Pesquet, J.C.; H. Krim, H. Carfatan (1996), "Time-invariant orthonormal wavelet representations," *IEEE Trans. Sign. Proc.*, vol. 44, 8, pp. 1964–1970.

**See Also**    dwt | wavedec

**Purpose**      Discrete stationary wavelet transform 2-D

**Syntax**       SWC = swt2(X,N,'*wname*')
                 [A,H,V,D] = swt2(X,N,'*wname*')
                 SWC = swt2(X,N,Lo_D,Hi_D)
                 [A,H,V,D] = swt2(X,N,Lo_D,Hi_D)

**Description**   swt2 performs a multilevel 2-D stationary wavelet decomposition using
                 either a specific orthogonal wavelet ('*wname*'— see wfilters for more
                 information) or specific orthogonal wavelet decomposition filters.

                 SWC = swt2(X,N,'*wname*') or [A,H,V,D] = swt2(X,N,'*wname*')
                 compute the stationary wavelet decomposition of the matrix X at level N,
                 using '*wname*'.

                 N must be a strictly positive integer (see wmaxlev for more information),
                 and $2^N$ must divide size(X,1) and size(X,2).

                 Outputs [A,H,V,D] are 3-D arrays, which contain the coefficients:

                 - For $1 \leq i \leq N$, the output matrix A(:,:,i) contains the coefficients
                   of approximation of level i.

                 - The output matrices H(:,:,i), V(:,:,i) and D(:,:,i) contain the
                   coefficients of details of level i (horizontal, vertical, and diagonal):

                   SWC = [H(:,:,1:N) ; V(:,:,1:N) ; D(:,:,1:N) ; A(:,:,N)]

                 SWC = swt2(X,N,Lo_D,Hi_D) or [A,H,V,D] = swt2(X,N,Lo_D,Hi_D),
                 computes the stationary wavelet decomposition as in the previous
                 syntax, given these filters as input:

                 - Lo_D is the decomposition low-pass filter.

                 - Hi_D is the decomposition high-pass filter.

                 Lo_D and Hi_D must be the same length.

                 ---

                 **Note** swt2 is defined using dwt with periodic extension.

                 ---

**Tips**        When X represents an indexed image, X is an m-by-n matrix and the
                output arrays SWC or cA,cH,cV, and cD are m-by-n-by-p arrays.

                When X represents a truecolor image, it becomes an m-by-n-by-3 array.
                This array is an m-by-n-by-3 array, where each m-by-n matrix represents
                a red, green, or blue color plane concatenated along the third dimension.
                The output arrays SWC or cA,cH,cV, and cD are m-by-n-by-p-by-3 arrays.

                For more information on image formats, see the `image` and `imfinfo`
                reference pages.

**Examples**
```
% Load original image.
load nbarb1;

% Image coding.
nbcol = size(map,1);
cod_X = wcodemat(X,nbcol);

% Visualize the original image.
subplot(221)
image(cod_X)
title('Original image');
colormap(map)

% Perform SWT decomposition
% of X at level 3 using sym4.
[ca,chd,cvd,cdd] = swt2(X,3,'sym4');

% Visualize the decomposition.

for k = 1:3
    % Images coding for level k.
    cod_ca  = wcodemat(ca(:,:,k),nbcol);
    cod_chd = wcodemat(chd(:,:,k),nbcol);
    cod_cvd = wcodemat(cvd(:,:,k),nbcol);
    cod_cdd = wcodemat(cdd(:,:,k),nbcol);
    decl = [cod_ca,cod_chd;cod_cvd,cod_cdd];
```

```
% Visualize the coefficients of the decomposition
% at level k.
subplot(2,2,k+1)
image(decl)

title(['SWT dec.: approx. ', ...
'and det. coefs (lev. ',num2str(k),')']);
colormap(map)
end
% Editing some graphical properties,
% the following figure is generated.
```
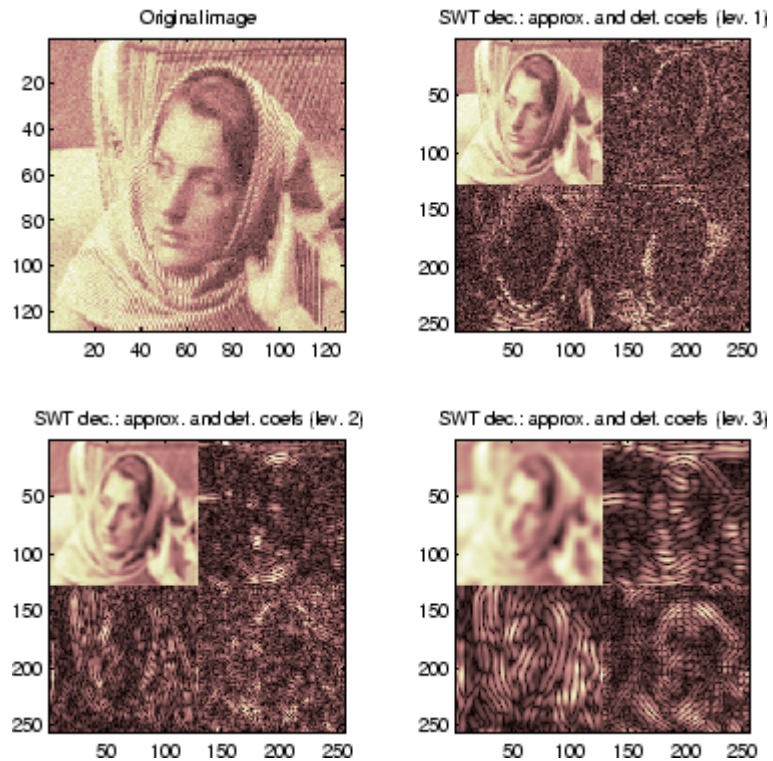
**Algorithms**    For images, an algorithm similar to the one-dimensional case is possible for two-dimensional wavelets and scaling functions obtained from one-dimensional ones by tensor product.

This kind of two-dimensional SWT leads to a decomposition of approximation coefficients at level $j$ in four components: the approximation at level $j+1$, and the details in three orientations (horizontal, vertical, and diagonal).

The following chart describes the basic decomposition step for images:

## Two-Dimensional SWT

**Decomposition step**
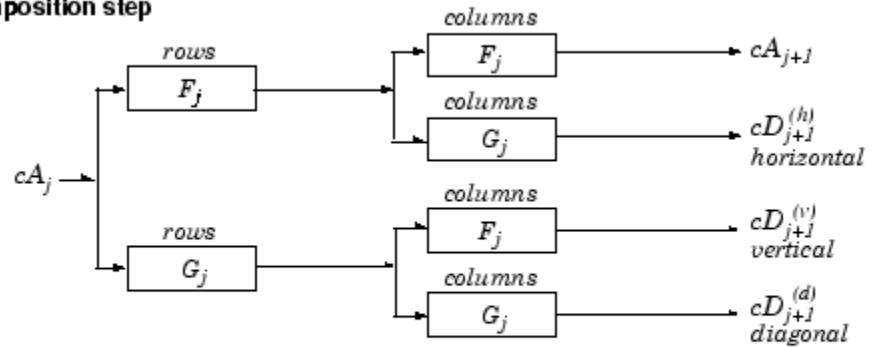


where

$\boxed{\text{X}}^{\;rows}$  Convolve with filter X the rows of the entry

$\boxed{\text{X}}^{\;columns}$  Convolve with filter X the columns of the entry

**Initialization**     $cA_0 = s$ for the decomposition initialization

$F_j \longrightarrow \boxed{\uparrow 2} \longrightarrow F_{j+1}$

Initialization $F_0 = Lo\_D$          where  $\boxed{\uparrow 2}$  Upsample

$G_j \longrightarrow \boxed{\uparrow 2} \longrightarrow G_{j+1}$

Initialization $G_0 = Hi\_D$

**Note**    $size(cA_j) = size(cD_j^{(h)}) = size(cD_j^{(v)}) = size(cD_j^{(d)}) = s$

where    $s = size\ of\ the\ analyzed\ image$

**References**    Nason, G.P.; B.W. Silverman (1995), "The stationary wavelet transform and some statistical applications," *Lecture Notes in Statistics*, 103, pp. 281–299.

Coifman, R.R.; Donoho, D.L. (1995), "Translation invariant de-noising," *Lecture Notes in Statistics*, 103, pp. 125–150.

Pesquet, J.C.; H. Krim, H. Carfatan (1996), "Time-invariant orthonormal wavelet representations," *IEEE Trans. Sign. Proc.*, vol. 44, 8, pp. 1964–1970.

**See Also**    dwt2 | iswt2 | wavedec2

**Purpose**   Symlet wavelet filter computation

**Syntax**    W = SYMAUX(N,SUMW)
              W = SYMAUX(N)
              W = SYMAUX(N,1)
              W = SYMAUX(N,0)
              W = SYMAUX(N,1)

**Description**   Symlets are the Äúleast asymmetricÄù Daubechies wavelets.

W = SYMAUX(N,SUMW) is the order N Symlet scaling filter such that
SUM(W) = SUMW. Possible values for N are 1, 2, 3, ...

---

**Note** Instability may occur when N is too large.

---

W = SYMAUX(N) is equivalent to W = SYMAUX(N,1).

W = SYMAUX(N,0) is equivalent to W = SYMAUX(N,1).

**Examples**   % Generate wdb4 the order 4 Daubechies scaling filter.
wdb4 = dbaux(4)

wdb4 =

  Columns 1 through 7

    0.1629    0.5055    0.4461   -0.0198   -0.1323    0.0218    0.0233

  Column 8

   -0.0075


% wdb4 is a solution of the equation: P = conv(wrev(w),w)*2,
% where P is the "Lagrange  trous" filter for N=4.

```
% wdb4 is a minimum phase solution of the previous equation,
% based on the roots of P (see dbaux).
P = conv(wrev(wdb4),wdb4)*2;

% Generate wsym4 the order 4 symlet scaling filter.
% The Symlets are the "least asymmetric" Daubechies'
% wavelets obtained from another choice between the roots of P.
wsym4 = symaux(4)

wsym4 =

  Columns 1 through 7

    0.0228   -0.0089   -0.0702    0.2106    0.5683    0.3519   -0.0210

  Column 8

   -0.0536


% Compute conv(wrev(wsym4),wsym4) * 2 and check that wsym4
% is another solution of the equation P = conv(wrev(w),w)*2.
Psym = conv(wrev(wsym4),wsym4)*2;
err = norm(P-Psym)

err =

  7.4988e-016
```

**See Also**    symwavf | wfilters

**Purpose**    Symlet wavelet filter

**Syntax**    F = symwavf(*W*)

**Description**    F = symwavf(*W*) returns the scaling filter associated with the symlet
wavelet specified by the string *W* where W = 'symN'. Possible values
for N are 2, 3, ..., 45.

**Examples**
```
% Compute the scaling filter corresponding to wavelet sym4.
w = symwavf('sym4')

w =
 Columns 1 through 7
    0.0228 -0.0089 -0.0702 0.2106 0.5683 0.3519 -0.0210
 Column 8
    -0.0536
```

**See Also**    symaux | waveinfo

# thselect

| **Purpose** | Threshold selection for de-noising |
| --- | --- |

**Syntax**      THR = thselect(X,TPTR)

**Description**  thselect is a one-dimensional de-noising oriented function.

THR = thselect(X,TPTR) returns threshold X-adapted value using selection rule defined by string TPTR.

Available selection rules are

- TPTR = 'rigrsure', adaptive threshold selection using principle of Stein's Unbiased Risk Estimate.

- TPTR = 'heursure', heuristic variant of the first option.

- TPTR = 'sqtwolog', threshold is sqrt(2*log(length(X))).

- TPTR = 'minimaxi', minimax thresholding.

Threshold selection rules are based on the underlying model $y = f(t) + e$ where $e$ is a white noise $N(0,1)$. Dealing with unscaled or nonwhite noise can be handled using rescaling output threshold THR (see SCAL parameter in wden for more information).

Available options are

- tptr = 'rigrsure' uses for the soft threshold estimator, a threshold selection rule based on SteinÄôs Unbiased Estimate of Risk (quadratic loss function). One gets an estimate of the risk for a particular threshold value (*t*). Minimizing the risks in (*t*) gives a selection of the threshold value.

- tptr = 'sqtwolog' uses a fixed-form threshold yielding minimax performance multiplied by a small factor proportional to log(length(X)).

- tptr = 'heursure' is a mixture of the two previous options. As a result, if the signal to noise ratio is very small, the SURE estimate is very noisy. If such a situation is detected, the fixed form threshold is used.

- `tptr = 'minimaxi'` uses a fixed threshold chosen to yield minimax performance for mean square error against an ideal procedure. The minimax principle is used in statistics in order to design estimators. Since the de-noised signal can be assimilated to the estimator of the unknown regression function, the minimax estimator is the one that realizes the minimum of the maximum mean square error obtained for the worst function in a given set.

**Examples**

```
% The current extension mode is zero-padding (see dwtmode).
% Generate Gaussian white noise.
x = randn(1,1000);

% Find threshold for each selection rule.
% Adaptive threshold using SURE.
thr = thselect(x,'rigrsure')
% Fixed form threshold.
thr = thselect(x,'sqtwolog')
% Heuristic variant of the first option.
thr = thselect(x,'heursure')
% Minimax threshold.
thr = thselect(x,'minimaxi')
```

**References**

Donoho, D.L. (1993), "Progress in wavelet analysis and WVD: a ten minute tour," in *Progress in wavelet analysis and applications*, Y. Meyer, S. Roques, pp. 109–128. Frontières Ed.

Donoho, D.L., I.M. Johnstone (1994), "Ideal spatial adaptation by wavelet shrinkage," *Biometrika*, vol 81, pp. 425–455.

Donoho, D.L. (1995), "De-noising by soft-thresholding," *IEEE Trans. on Inf. Theory*, 41, 3, pp. 613–627.

**See Also**          wden

# tnodes

| | |
|---|---|
| **Purpose** | Determine terminal nodes |
| **Syntax** | N = tnodes(*T*) |
| | N = tnodes(*T*,'deppos') |
| | [N,K] = tnodes(*T*) |
| | [N,K] = tnodes(*T*,'deppos'), M = N(K) |
| **Description** | tnodes is a tree-management utility. |
| | N = tnodes(*T*) returns the indices of terminal nodes of the tree *T*. N is a column vector. |
| | The nodes are numbered from left to right and from top to bottom. The root index is 0. |
| | N = tnodes(*T*,'deppos') returns a matrix N, which contains the depths and positions of terminal nodes. |
| | N(i,1) is the depth of the i-th terminal node. N(i,2) is the position of the i-th terminal node. |
| | For [N,K] = tnodes(*T*) or [N,K] = tnodes(*T*,'deppos'), M = N(K) are the indices reordered as in tree *T*, from left to right. |
| **Examples** | ```
% Create initial tree.
ord = 2;
t = ntree(ord,3);      % Binary tree of depth 3.
t = nodejoin(t,5);
t = nodejoin(t,4);
plot(t)
``` |

```
% Change Node Label from Depth_Position to Index
% (see the plot function).
```



```
% List terminal nodes (index).
tnodes(t)
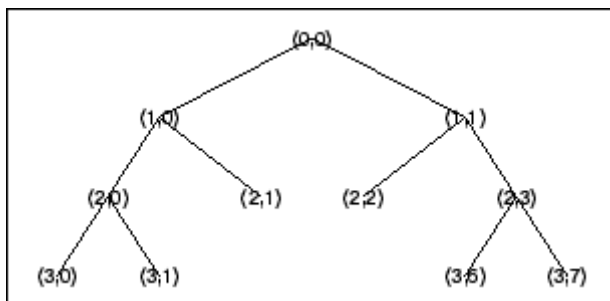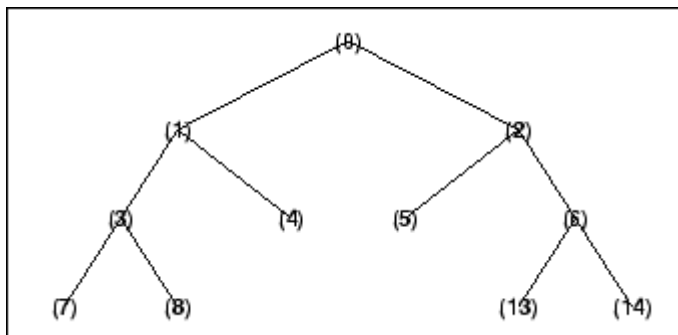
ans =
     4
     5
     7
     8
    13
    14
% List terminal nodes (Depth_Position).
tnodes(t,'deppos')
```

```
ans =
     2    1
     2    2
     3    0
     3    1
     3    6
     3    7
```

**See Also**        leaves | noleaves | wtreemgr

**Purpose**          Tree depth

**Syntax**           D = treedpth(*T*)

**Description**      treedpth is a tree-management utility.

D = treedpth(*T*) returns the depth D of the tree *T*.

**Examples**
```
% Create binary tree (tree of order 2) of depth 3.
t = ntree(2,3);
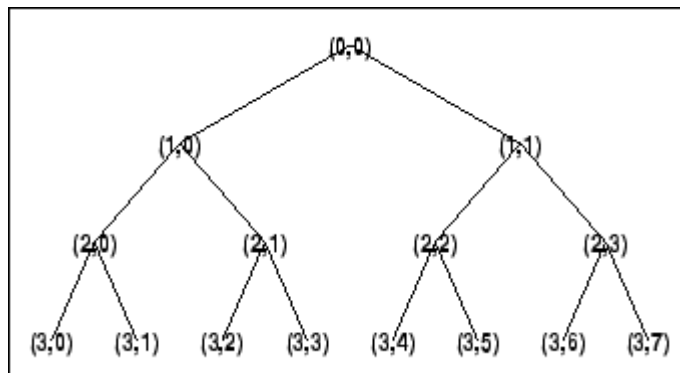
% Plot tree t.
plot(t)
```



```
% Tree depth.
treedpth(t)

ans =
    3
```

**See Also**         wtreemgr

# treeord

| | |
|---|---|
| **Purpose** | Tree order |
| **Syntax** | ORD = treeord(*T*) |
| **Description** | treeord is a tree-management utility. |
| | ORD = treeord(*T*) returns the order ORD of the tree *T*. |
| **Examples** | % Create binary tree (tree of order 2) of depth 3. |

```
t = ntree(2,3);
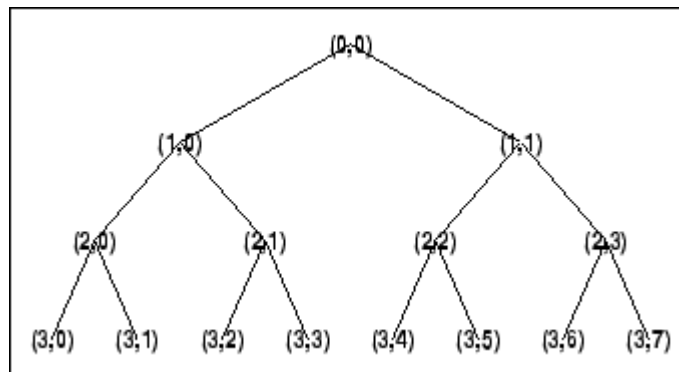
% Plot tree t.
plot(t)
```



```
% Tree order.
treeord(t)

ans =
    2
```

**See Also**    wtreemgr

**Purpose**          Direct reconstruction from 1-D wavelet coefficients

**Syntax**           Y = upcoef(O,X,'*wname*',N)
                     Y = upcoef(O,X,'*wname*',N,L)
                     Y = upcoef(O,X,Lo_R,Hi_R,N)
                     Y = upcoef(O,X,Lo_R,Hi_R,N,L)
                     Y = upcoef(O,X,'*wname*'')
                     Y = upcoef(O,X,'*wname*'',1)
                     Y = upcoef(O,X,Lo_R,Hi_R)
                     Y = upcoef(O,X,Lo_R,Hi_R,1)

**Description**      upcoef is a one-dimensional wavelet analysis function.

                     Y = upcoef(O,X,'*wname*',N) computes the N-step reconstructed
                     coefficients of vector X.

                     '*wname*' is a string containing the wavelet name. See wfilters for
                     more information.

                     N must be a strictly positive integer.

                     If O = 'a', approximation coefficients are reconstructed.

                     If O = 'd', detail coefficients are reconstructed.

                     Y = upcoef(O,X,'*wname*',N,L) computes the N-step reconstructed
                     coefficients of vector X and takes the length-L central portion of the
                     result.

                     Instead of giving the wavelet name, you can give the filters.

                     For Y = upcoef(O,X,Lo_R,Hi_R,N) or Y =
                     upcoef(O,X,Lo_R,Hi_R,N,L), Lo_R is the reconstruction low-pass filter
                     and Hi_R is the reconstruction high-pass filter.

                     Y = upcoef(O,X,'*wname*'') is equivalent to Y =
                     upcoef(O,X,'*wname*'',1).

                     Y = upcoef(O,X,Lo_R,Hi_R) is equivalent to Y =
                     upcoef(O,X,Lo_R,Hi_R,1).

# upcoef

**Examples**

```
% The current extension mode is zero-padding (see dwtmode).

% Approximation signals, obtained from a single coefficient
% at levels 1 to 6.
cfs = [1];  % Decomposition reduced a single coefficient.
essup = 10; % Essential support of the scaling filter db6.
figure(1)
for i=1:6
    % Reconstruct at the top level an approximation
    % which is equal to zero except at level i where only
    % one coefficient is equal to 1.
    rec = upcoef('a',cfs,'db6',i);

    % essup is the essential support of the
    % reconstructed signal.
    % rec(j) is very small when j is   essup.
    ax = subplot(6,1,i),h = plot(rec(1:essup));
    set(ax,'xlim',[1 325]);
    essup = essup*2;

end
subplot(611)
title(['Approximation signals, obtained from a single ' ...
    'coefficient at levels 1 to 6'])

% Editing some graphical properties,
% the following figure is generated.
```

Approximation signals, obtained from a single coefficient at levels 1 to 6



```
% The same can be done for details.
% Details signals, obtained from a single coefficient
% at levels 1 to 6.
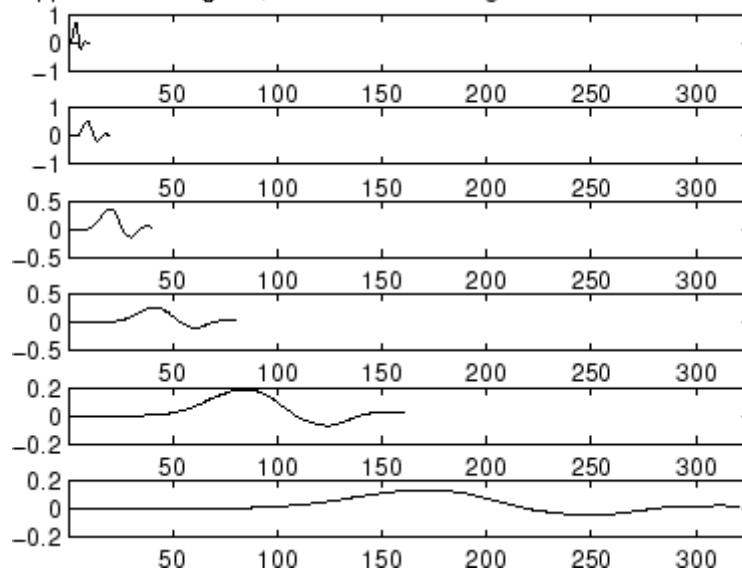
cfs = [1];
mi = 12; ma = 30;   % Essential support of
                    % the wavelet filter db6.
rec = upcoef('d',cfs,'db6',1);
figure(2)
subplot(611), plot(rec(3:12))
for i=2:6
    % Reconstruct at top level a single detail
    % coefficient at level i.
    rec = upcoef('d',cfs,'db6',i);
    subplot(6,1,i), plot(rec(mi*2^(i-2):ma*2^(i-2)))
end
subplot(611)
title(['Detail signals obtained from a single ' ...
```

# upcoef

```
          'coefficient at levels 1 to 6'])
    % Editing some graphical properties,
    % the following figure is generated.
```

Detail signals obtained from a single coefficient at levels 1 to 6



**Algorithms**   upcoef is equivalent to an N time repeated use of the inverse wavelet transform.

**See Also**   idwt

**Purpose**        Direct reconstruction from 2-D wavelet coefficients

**Syntax**         Y = upcoef2(O,X,'*wname*',N,S)
                   Y = upcoef2(O,X,Lo_R,Hi_R,N,S)
                   Y = upcoef2(O,X,'*wname*',N)
                   Y = upcoef2(O,X,Lo_R,Hi_R,N)
                   Y = upcoef2(O,X,'*wname*')
                   Y = upcoef2(O,X,'*wname*',1)
                   Y = upcoef2(O,X,Lo_R,Hi_R)
                   Y = upcoef2(O,X,Lo_R,Hi_R,1)

**Description**    upcoef2 is a two-dimensional wavelet analysis function.

                   Y = upcoef2(O,X,'*wname*',N,S) computes the N-step reconstructed
                   coefficients of matrix X and takes the central part of size S. '*wname*' is
                   a string containing the name of the wavelet. See wfilters for more
                   information.

                   If O = 'a', approximation coefficients are reconstructed; otherwise if
                   O = 'h' ('v' or 'd', respectively), horizontal (vertical or diagonal,
                   respectively) detail coefficients are reconstructed. N must be a strictly
                   positive integer.

                   Instead of giving the wavelet name, you can give the filters.

                   For Y = upcoef2(O,X,Lo_R,Hi_R,N,S) is the reconstruction low-pass
                   filter and Hi_R is the reconstruction high-pass filter.

                   Y = upcoef2(O,X,'*wname*',N) or Y = upcoef2(O,X,Lo_R,Hi_R,N)
                   returns the computed result without any truncation.

                   Y = upcoef2(O,X,'*wname*') is equivalent to Y =
                   upcoef2(O,X,'*wname*',1).

                   Y = upcoef2(O,X,Lo_R,Hi_R) is equivalent to
                   Y = upcoef2(O,X,Lo_R,Hi_R,1).

**Examples**       % The current extension mode is zero-padding (see dwtmode).

                   % Load original image.

```
load woman;
% X contains the loaded image.

% Perform decomposition at level 2
% of X using db4.
[c,s] = wavedec2(X,2,'db4');

% Reconstruct approximation and details
% at level 1, from coefficients.
% This can be done using wrcoef2, or
% equivalently using:
%
% Step 1: Extract coefficients from the
% decomposition structure [c,s].
%
% Step 2: Reconstruct using upcoef2.

siz = s(size(s,1),:);

ca1 = appcoef2(c,s,'db4',1);
a1 = upcoef2('a',ca1,'db4',1,siz);

chd1 = detcoef2('h',c,s,1);
hd1 = upcoef2('h',chd1,'db4',1,siz);

cvd1 = detcoef2('v',c,s,1);
vd1 = upcoef2('v',cvd1,'db4',1,siz);

cdd1 = detcoef2('d',c,s,1);
dd1 = upcoef2('d',cdd1,'db4',1,siz);
```

**Algorithms**   See upcoef.

**See Also**   idwt2

| | |
|---|---|
| **Purpose** | Single-level reconstruction of 1-D wavelet decomposition |
| **Syntax** | [NC,NL,cA] = upwlev(C,L,'*wname*') |
| **Description** | upwlev is a one-dimensional wavelet analysis function. |

[NC,NL,cA] = upwlev(C,L,'*wname*') performs the single-level reconstruction of the wavelet decomposition structure [C,L] giving the new one [NC,NL], and extracts the last approximation coefficients vector cA.

[C,L] is a decomposition at level n = length(L)-2, so [NC,NL] is the same decomposition at level n-1 and cA is the approximation coefficients vector at level n.

'*wname*' is a string containing the wavelet name, C is the original wavelet decomposition vector, and L the corresponding bookkeeping vector (for detailed storage information, see wavedec ).

Instead of giving the wavelet name, you can give the filters.

For [NC,NL,cA] = upwlev(C,L,Lo_R,Hi_R), Lo_R is the reconstruction low-pass filter and Hi_R is the reconstruction high-pass filter.

**Examples**

```
% The current extension mode is zero-padding (see dwtmode).

% Load original one-dimensional signal.
load sumsin; s = sumsin;

% Perform decomposition at level 3 of s using db1.
[c,l] = wavedec(s,3,'db1');
subplot(311); plot(s);
title('Original signal s.');
subplot(312); plot(c);
title('Wavelet decomposition structure, level 3')
xlabel(['Coefs for approx. at level 3 ' ...
        'and for det. at levels 3, 2 and 1'])

% One step reconstruction of the wavelet decomposition
```

```
% structure at level 3 [c,l], so the new structure [nc,nl]
% is the wavelet decomposition structure at level 2.
[nc,nl] = upwlev(c,l,'db1');
subplot(313); plot(nc);
title('Wavelet decomposition structure, level 2')
xlabel(['Coefs for approx. at level 2 ' ...
        'and for det. at levels 2 and 1'])
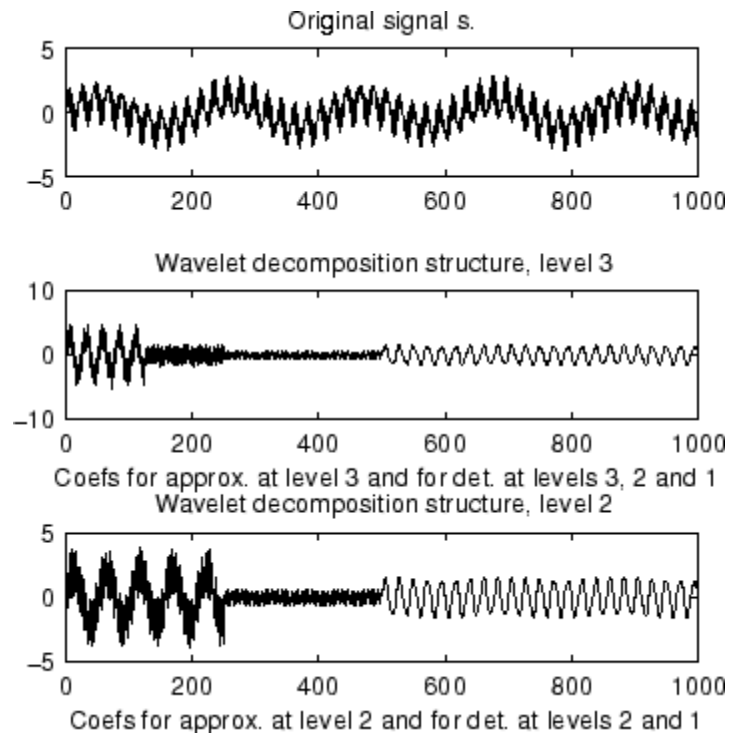
% Editing some graphical properties,
% the following figure is generated.
```

Original signal s.

Wavelet decomposition structure, level 3

Coefs for approx. at level 3 and for det. at levels 3, 2 and 1

Wavelet decomposition structure, level 2

Coefs for approx. at level 2 and for det. at levels 2 and 1

**See Also**     idwt

**How To**
- upcoef
- wavedec

# upwlev2

| | |
|---|---|
| **Purpose** | Single-level reconstruction of 2-D wavelet decomposition |
| **Syntax** | `[NC,NS,cA] = upwlev2(C,S,'wname')`<br>`[NC,NS,cA] = upwlev2(C,S,Lo_R,Hi_R)` |
| **Description** | upwlev2 is a two-dimensional wavelet analysis function. |

`[NC,NS,cA] = upwlev2(C,S,'wname')` performs the single-level reconstruction of wavelet decomposition structure [C,S] giving the new one [NC,NS], and extracts the last approximation coefficients matrix cA.

[C,S] is a decomposition at level n = `size(S,1)-2`, so [NC,NS] is the same decomposition at level n-1 and cA is the approximation matrix at level n.

'*wname*' is a string containing the wavelet name, C is the original wavelet decomposition vector, and S the corresponding bookkeeping matrix (for detailed storage information, see wavedec2).

Instead of giving the wavelet name, you can give the filters.

For `[NC,NS,cA] = upwlev2(C,S,Lo_R,Hi_R)`, Lo_R is the reconstruction low-pass filter and Hi_R is the reconstruction high-pass filter.

**Examples**

```
% The current extension mode is zero-padding (see dwtmode).

% Load original image.
load woman;
% X contains the loaded image.

% Perform decomposition at level 2
% of X using db1.
[c,s] = wavedec2(X,2,'db1');
sc = size(c)

sc =
     1    65536
```

```
val_s = s

val_s =
    64    64
    64    64
    128   128
    256   256

% One step reconstruction of wavelet
% decomposition structure [c,s].
[nc,ns] = upwlev2(c,s,'db1');
snc = size(nc)

snc =
    1    65536

val_ns = ns

val_ns =
    128   128
    128   128
    256   256
```

**See Also**      idwt2 | upcoef2 | wavedec2

# wave2lp

| | |
|---|---|
| **Purpose** | Laurent polynomials associated with wavelet |
| **Syntax** | [Hs,Gs,Ha,Ga] = wave2lp(*W*) |
| **Description** | [Hs,Gs,Ha,Ga] = wave2lp(*W*) returns the four Laurent polynomials associated with the wavelet *W* (see liftwave). |

The pairs (Hs,Gs) and (Ha,Ga) are the synthesis and the analysis pair respectively.

The H-polynomials (G-polynomials) are low-pass (high-pass) polynomials.

For an orthogonal wavelet, Hs = Ha and Gs = Ga.

**Examples**

```
% Get Laurent polynomials associated to the "lazy" wavelet.
[Hs,Gs,Ha,Ga] = wave2lp('lazy')

Hs(z) = 1

Gs(z) = z^(-1)

Ha(z) = 1

Ga(z) = z^(-1)

% Get Laurent polynomials associated to the db1 wavelet.
[Hs,Gs,Ha,Ga] = wave2lp('db1')

Hs(z) = + 0.7071 + 0.7071*z^(-1)

Gs(z) = - 0.7071 + 0.7071*z^(-1)

Ha(z) = + 0.7071 + 0.7071*z^(-1)

Ga(z) = - 0.7071 + 0.7071*z^(-1)

% Get Laurent polynomials associated to the bior1.3 wavelet.
[Hs,Gs,Ha,Ga] = wave2lp('bior1.3')
```

```
Hs(z) = + 0.7071 + 0.7071*z^(-1)

Gs(z) = ...
    + 0.08839*z^(+2) + 0.08839*z^(+1) - 0.7071 + 0.7071*z^(-1) -
0.08839*z^(-2)  ...
    - 0.08839*z^(-3)

Ha(z) = ...
    - 0.08839*z^(+2) + 0.08839*z^(+1) + 0.7071 + 0.7071*z^(-1) +
0.08839*z^(-2)  ...
    - 0.08839*z^(-3)

Ga(z) = - 0.7071 + 0.7071*z^(-1)
```

**See Also**      laurpoly

# wavedec

**Purpose**        Multilevel 1-D wavelet decomposition

**Syntax**          [C,L] = wavedec(X,N,'*wname*')
[C,L] = wavedec(X,N,Lo_D,Hi_D)

**Description**    wavedec performs a multilevel one-dimensional wavelet analysis using either a specific wavelet ('*wname*') or a specific wavelet decomposition filters (Lo_D and Hi_D, see wfilters).

> **Note** wavedec supports only Type 1 (orthogonal) or Type 2 (biorthogonal) wavelets.

[C,L] = wavedec(X,N,'*wname*') returns the wavelet decomposition of the signal X at level N, using '*wname*'. N must be a strictly positive integer (see wmaxlev for more information). The output decomposition structure contains the wavelet decomposition vector C and the bookkeeping vector L. The structure is organized as in this level-3 decomposition example.

**Decomposition:**



[C,L] = wavedec(X,N,Lo_D,Hi_D) returns the decomposition structure as above, given the low- and high-pass decomposition filters you specify.

**Examples**

```
% The current extension mode is zero-padding (see dwtmode).
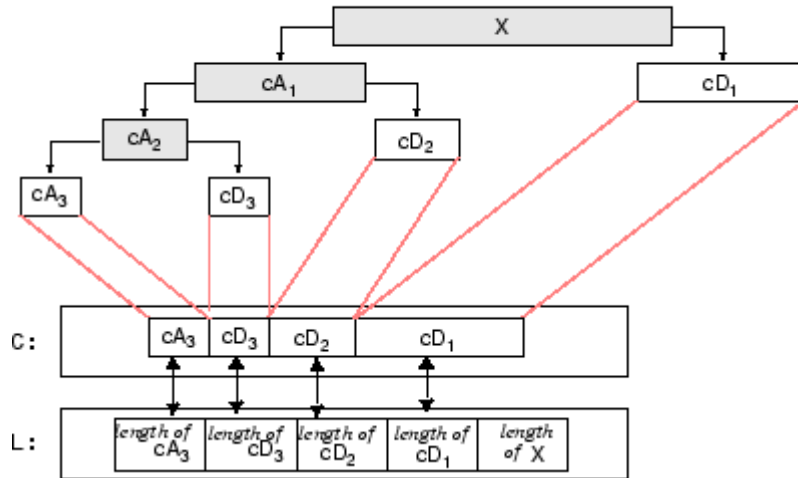
% Load original one-dimensional signal.
load sumsin; s = sumsin;
% Perform decomposition at level 3 of s using db1.
[c,l] = wavedec(s,3,'db1');
% Using some plotting commands,
% the following figure is generated.
```

Original signal s.

Wavelet decomposition structure

Coefs for approx. at level 3 and for det. at levels 3, 2 and 1

**Algorithms**    Given a signal *s* of length *N*, the DWT consists of $\log_2 N$ stages at most. The first step produces, starting from *s*, two sets of coefficients: approximation coefficients $CA_1$, and detail coefficients $CD_1$. These vectors are obtained by convolving *s* with the low-pass filter Lo_D for approximation, and with the high-pass filter Hi_D for detail, followed by dyadic decimation (downsampling).

More precisely, the first step is

The length of each filter is equal to 2*N*. If *n* = length(*s*), the signals *F* and *G* are of length $n + 2N - 1$ and the coefficients $cA_1$ and $cD_1$ are of length

$$\text{floor}\left(\frac{n-1}{2}\right) + N$$

The next step splits the approximation coefficients $cA_1$ in two parts using the same scheme, replacing *s* by $cA_1$, and producing $cA_2$ and $cD_2$, and so on

**One-Dimensional DWT**

**Decomposition step**

**Initialization**   $cA_0 = s$

The wavelet decomposition of the signal *s* analyzed at level *j* has the following structure: $[cA_j, cD_j, ..., cD_1]$.

This structure contains, for $J = 3$, the terminal nodes of the following tree:



**References**    Daubechies, I. (1992), *Ten lectures on wavelets*, CBMS-NSF conference series in applied mathematics. SIAM Ed.

Mallat, S. (1989), "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Pattern Anal. and Machine Intell.*, vol. 11, no. 7, pp 674–693.

Meyer, Y. (1990), *Ondelettes et opérateurs*, Tome 1, Hermann Ed. (English translation: *Wavelets and operators*, Cambridge Univ. Press. 1993.)

**See Also**      dwt | waveinfo | waverec | wfilters | wmaxlev

**Purpose**  Multilevel 2-D wavelet decomposition

**Syntax**  
```
[C,S] = wavedec2(X,N,'wname')
[C,S] = wavedec2(X,N,Lo_D,Hi_D)
```

**Description**  wavedec2 is a two-dimensional wavelet analysis function.

[C,S] = wavedec2(X,N,'*wname*') returns the wavelet decomposition of the matrix X at level N, using the wavelet named in string '*wname*' (see wfilters for more information).

Outputs are the decomposition vector C and the corresponding bookkeeping matrix S.

N must be a strictly positive integer (see wmaxlev for more information).

Instead of giving the wavelet name, you can give the filters.

For [C,S] = wavedec2(X,N,Lo_D,Hi_D), Lo_D is the decomposition low-pass filter and Hi_D is the decomposition high-pass filter.

Vector C is organized as

```
C = [ A(N) | H(N) | V(N) | D(N) | ...
H(N-1) | V(N-1) | D(N-1) | ... | H(1) | V(1) | D(1) ].
```

where A, H, V, D, are row vectors such that

- A = approximation coefficients

- H = horizontal detail coefficients

- V = vertical detail coefficients

- D = diagonal detail coefficients

- Each vector is the vector column-wise storage of a matrix.

Matrix S is such that

- S(1,:) = size of approximation coefficients(N).

- S(i,:) = size of detail coefficients(N-i+2) for i = 2, ...N+1 and
  S(N+2,:)  = size(X).

C (3n+1 sections)

S (n+2-by-2)

**Tips**    When X represents an indexed image, X, as well as the output arrays cA,cH,cV, and cD are m-by-n matrices. When X represents a truecolor image, it is an m-by-n-by-3 array, where each m-by-n matrix represents a red, green, or blue color plane concatenated along the third dimension. The size of vector C and the size of matrix S depend on the type of analyzed image.

For a truecolor image, the decomposition vector C and the corresponding bookkeeping matrix S can be represented as follows.



C (3n+1 sections)

S (n+2-by-3)

For more information on image formats, see the `image` and `imfinfo` reference pages.

**Examples**

```
% The current extension mode is zero-padding (see dwtmode).
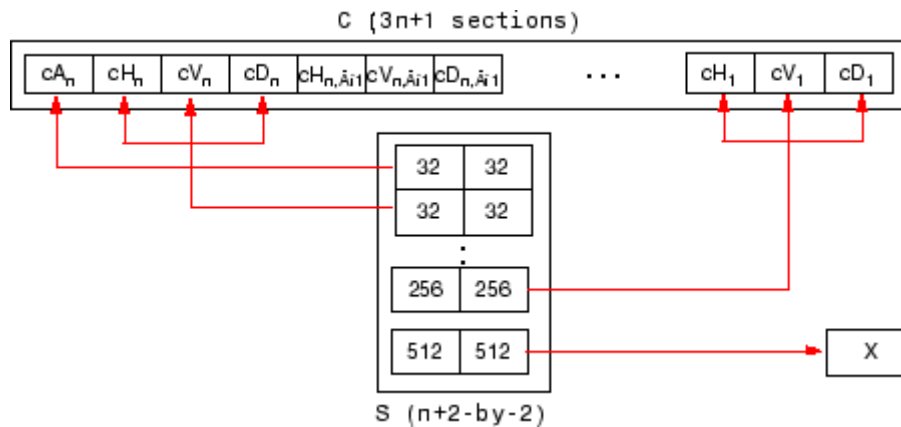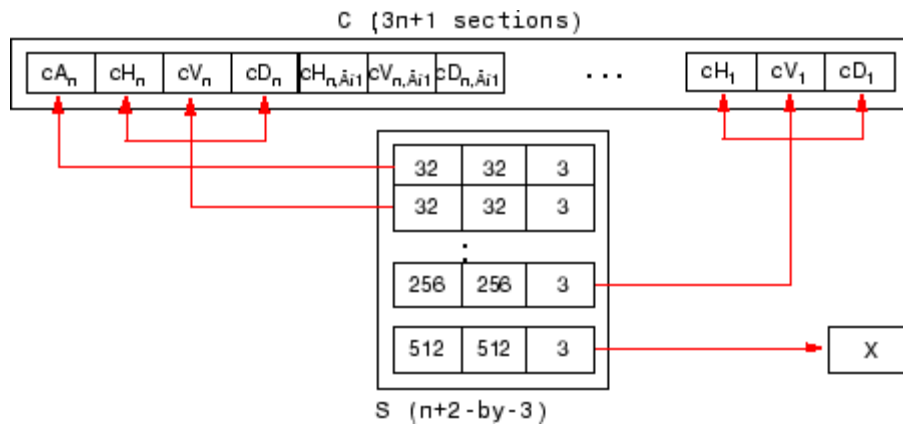
% Load original image.
load woman;
% X contains the loaded image.

% Perform decomposition at level 2
% of X using db1.
[c,s] = wavedec2(X,2,'db1');

% Decomposition structure organization.
sizex = size(X)

sizex =
    256  256
sizec = size(c)

sizec =
    1  65536
    val_s = s

val_s =
    64  64
    64  64
    128  128
    256  256
```

**Algorithms**

For images, an algorithm similar to the one-dimensional case is possible for two-dimensional wavelets and scaling functions obtained from one-dimensional ones by tensor product.

This kind of two-dimensional DWT leads to a decomposition of approximation coefficients at level $j$ in four components: the

approximation at level $j$+1, and the details in three orientations (horizontal, vertical, and diagonal).

The following chart describes the basic decomposition step for images:

**Two-Dimensional DWT**

**Decomposition step**



**Initialization**    $cA_0 = s$ for the decomposition initialization

So, for $J$=2, the two-dimensional wavelet tree has the form

$$s$$

$$cA_1 \quad cD_1^{\,h} \quad cD_1^{\,d} \quad cD_1^{\,v}$$

$$cA_2 \quad cD_2^{\,h} \quad cD_2^{\,d} \quad cD_2^{\,v}$$

**References**    Daubechies, I. (1992), *Ten lectures on wavelets*, CBMS-NSF conference series in applied mathematics. SIAM Ed.

Mallat, S. (1989), "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Pattern Anal. and Machine Intell.*, vol. 11, no. 7, pp. 674–693.

Meyer, Y. (1990), *Ondelettes et opérateurs*, Tome 1, Hermann Ed. (English translation: *Wavelets and operators*, Cambridge Univ. Press. 1993.)

**See Also**    dwt | waveinfo | waverec2 | wfilters | wmaxlev

# wavedec3

| **Purpose** | Multilevel 3-D wavelet decomposition |
|---|---|

**Syntax**
```
WDEC = wavedec3(X,N,'wname')
WDEC = wavedec3(X,N,'wname','mode','ExtM')
WDEC = wavedec3(X,N,{LoD,HiD,LoR,HiR})
```

**Description**    wavedec3 is a three-dimensional wavelet analysis function.

WDEC = wavedec3(X,N,'*wname*') returns the wavelet decomposition of the 3-D array X at level N, using the wavelet named in string '*wname*' or the particular wavelet filters you specify. It uses the default extension mode 'sym'. See dwtmode. N must be a positive integer.

WDEC = wavedec3(X,N,'*wname*','mode','*ExtM*') uses the specified DWT extension mode .

WDEC = wavedec3(X,N,{LoD,HiD,LoR,HiR}) uses the decomposition and reconstruction filters you specify in a cell array.

WDEC is the output decomposition structure, with the following fields:

| | |
|---|---|
| sizeINI | Size of the three-dimensional array X |
| level | Level of the decomposition |
| mode | Name of the wavelet transform extension mode |
| filters | Structure with 4 fields, LoD, HiD, LoR, HiR, which contain the filters used for DWT. |

| | |
|---|---|
| dec | N x 1 cell array containing the coefficients of the decomposition. N is equal to 7*WDEC.level+1. |
| | dec{1} contains the lowpass component (approximation) at the level of the decomposition. The approximation is equivalent to the filtering operations 'LLL'. |
| | dec{k+2},...,dec{k+8} with k = 0,7,14,...,7*(WDEC.level-1) contain the 3-D wavelet coefficients for the multiresolution starting with the coarsest level when k=0. |
| | For example, if WDEC.level=3, dec{2},...,dec{8} contain the wavelet coefficients for level 3 (k=0), dec{9},...,dec{15} contain the wavelet coefficients for level 2 (k=7), and dec{16},...,dec{22} contain the wavelet coefficients for level 1 (k=7*(WDEC.level-1)). |
| | At each level, the wavelet coefficients in dec{k+2},...,dec{k+8} are in the following order: 'HLL','LHL','HHL','LLH','HLH','LHH','HHH'. |
| | The strings give the order in which the separable filtering operations are applied from left to right. For example, 'LHH' means that the lowpass (scaling) filter with downsampling is applied to the rows of X, followed by the highpass (wavelet) filter with downsampling applied to the columns of X. Finally, the highpass filter with downsampling is applied to the 3rd dimension of X. |
| sizes | Successive sizes of the decomposition components |

**Examples**    **3-D Wavelet Transform**

Find the 3-D DWT of a volume.

Construct 8-by-8-by-8 matrix. Obtain the 3-D discrete wavelet
transform at level 1 using the Haar wavelet and the default whole-point
symmetric extension mode.

```
% Matrix of integers 1:64
M = magic(8);
% Make data 3-D
X = repmat(M,[1 1 8]);
% Decompose X at level 1 using db1.
wd1 = wavedec3(X,1,'db1');
```

**3-D Wavelet Transform Using Specified Decomposition and
Reconstruction Filters**

Specify the decomposition and reconstruction filters as a cell array.

Construct 8-by-8-by-8 matrix. Obtain the 3-D discrete wavelet
transform down to level 2 using the Daubechies extremal phase
wavelet with two vanishing moments. Input the decomposition and
reconstruction filters as a cell array. Use the periodic extension mode.

```
% Matrix of integers 1:64
M = magic(8);
% Make data 3-D
X = repmat(M,[1 1 8]);
[LoD,HiD,LoR,HiR] = wfilters('db2');
wd2 = wavedec3(X,2,{LoD,HiD,LoR,HiR},'mode','per');
```

**Coefficient Order in 3-D Wavelet Transform**

Compare the output of wavedec3 and dwt3 to illustrate the ordering of
the 3-D wavelet coefficients described in the dec field description.

```
X = reshape(1:512,8,8,8);
dwtOut = dwt3(X,'db1','mode','per');
wdec = wavedec3(X,1,'db1','mode','per');
```

```
max(abs((wdec.dec{4}(:)-dwtOut.dec{2,2,1}(:))))
max(abs((wdec.dec{5}(:)-dwtOut.dec{1,1,2}(:))))
```

**See Also**     dwt3 | dwtmode | waveinfo | waverec3 | wfilters | wmaxlev

# wavedemo

**Purpose**     Wavelet Toolbox software examples

**Syntax**

**Description**     wavedemo opens a GUI that allows you to choose between several
Wavelet Toolbox examples.

**Purpose**      Wavelet and scaling functions

**Syntax**       [PHI,PSI,XVAL] = wavefun('*wname*',ITER)
                 [PHI1,PSI1,PHI2,PSI2,XVAL] = wavefun('*wname*',ITER)
                 [PHI,PSI,XVAL] = wavefun('*wname*',ITER)
                 [PSI,XVAL] = wavefun('*wname*',ITER)
                 [...] = wavefun(*wname*,A,B)
                 [...] = wavefun('*wname*',max(A,B))
                 [...] = wavefun('*wname*',0)
                 [...] = wavefun('*wname*',8,0)
                 [...] = wavefun('*wname*')
                 [...] = wavefun('*wname*',8)

**Description**  The function wavefun returns approximations of the wavelet function
                '*wname*' and the associated scaling function, if it exists. The positive
                integer ITER determines the number of iterations computed; thus, the
                refinement of the approximations.

                *For an orthogonal wavelet*:

                [PHI,PSI,XVAL] = wavefun('*wname*',ITER) returns the scaling and
                wavelet functions on the points grid XVAL.

                *For a biorthogonal wavelet*:

                [PHI1,PSI1,PHI2,PSI2,XVAL] = wavefun('*wname*',ITER) returns the
                scaling and wavelet functions both for decomposition (PHI1,PSI1) and
                for reconstruction (PHI2,PSI2).

                *For a Meyer wavelet*:

                [PHI,PSI,XVAL] = wavefun('*wname*',ITER)

                *For a wavelet without scaling function (e.g., Morlet, Mexican Hat,
                Gaussian derivatives wavelets or complex wavelets)*:

                [PSI,XVAL] = wavefun('*wname*',ITER)

                [...] = wavefun(*wname*,A,B), where A and B are positive integers, is
                equivalent to [...] = wavefun('*wname*',max(A,B)), and draws plots.

                When A is set equal to the special value 0,

- [...] = wavefun('*wname*',0) is equivalent to

- [...] = wavefun('*wname*',8,0).

- [...] = wavefun('*wname*') is equivalent to

- [...] = wavefun('*wname*',8).

The output arguments are optional.

**Examples**    On the following graph, 10 piecewise linear approximations of the sym4 wavelet obtained after each iteration of the cascade algorithm are shown.

```
% Set number of iterations and wavelet name.
iter = 10;
wav = 'sym4';

% Compute approximations of the wavelet function using the
% cascade algorithm.
for i = 1:iter
    [phi,psi,xval] = wavefun(wav,i);
    plot(xval,psi);
    hold on
end
title(['Approximations of the wavelet ',wav, ...
        ' for 1 to ',num2str(iter),' iterations']);
hold off
```

Approximations of the wavelet sym4 for 1 to 10 iterations



**Algorithms**

For compactly supported wavelets defined by filters, in general no closed form analytic formula exists.

The algorithm used is the cascade algorithm. It uses the single-level inverse wavelet transform repeatedly.

Let us begin with the scaling function $\varphi$.

Since $\varphi$ is also equal to $\varphi_{0,0}$, this function is characterized by the following coefficients in the orthogonal framework:

- $\langle\varphi, \varphi_{0,n}\rangle = 1$ only if $n = 0$ and equal to 0 otherwise

- $\langle\varphi, \psi_{-j,k}\rangle = 0$ for positive $j$, and all $k$.

This expansion can be viewed as a wavelet decomposition structure. Detail coefficients are all zeros and approximation coefficients are all zeros except one equal to 1.

Then we use the reconstruction algorithm to approximate the function $\varphi$ over a dyadic grid, according to the following result:

For any dyadic rational of the form $x = n2^{-j}$ in which the function is continuous and where $j$ is sufficiently large, we have pointwise convergence and

$$\left| \phi(x) - 2^{\frac{j}{2}} \langle \phi, \phi_{-j, n\, 2^{j}-x} \rangle \right| \leq C.2^{-j\alpha}$$

where $C$ is a constant, and $\alpha$ is a positive constant depending on the wavelet regularity.

Then using a good approximation of $\varphi$ on dyadic rationals, we can use piecewise constant or piecewise linear interpolations $\eta$ on dyadic intervals, for which uniform convergence occurs with similar exponential rate:

$$\| \phi - \eta \|_{\infty} \leq C.2^{-j\alpha}$$

So using a $J$-step reconstruction scheme, we obtain an approximation that converges exponentially towards $\varphi$ when $J$ goes to infinity.

Approximations are computed over a grid of dyadic rationals covering the support of the function to be approximated.

Since a scaled version of the wavelet function $\psi$ can also be expanded on the $(\varphi_{-1,n})_n$, the same scheme can be used, after a single-level reconstruction starting with the appropriate wavelet decomposition structure. Approximation coefficients are all zeros and detail coefficients are all zeros except one equal to 1.

For biorthogonal wavelets, the same ideas can be applied on each of the two multiresolution schemes in duality.

---

**Note** This algorithm may diverge if the function to be approximated is not continuous on dyadic rationals.

---

**References**
Daubechies, I., *Ten lectures on wavelets*, CBMS, SIAM, 1992, pp. 202Äì213.

Strang, G.; T. Nguyen (1996), *Wavelets and Filter Banks*,
Wellesley-Cambridge Press.

**See Also**   `intwave | waveinfo | wfilters`

# wavefun2

**Purpose**       Wavelet and scaling functions 2-D

**Syntax**        `[PHI,PSI,XVAL] = wavefun('wname',ITER)`
                  `[S,W1,W2,W3,XYVAL] = wavefun2('wname',ITER,'plot')`
                  `[S,W1,W2,W3,XYVAL] = wavefun2(wname,A,B)`
                  `[S,W1,W2,W3,XYVAL] = wavefun2('wname',max(A,B))`
                  `[S,W1,W2,W3,XYVAL] = wavefun2('wname',0)`
                  `[S,W1,W2,W3,XYVAL] = wavefun2('wname',4,0)`
                  `[S,W1,W2,W3,XYVAL] = wavefun2('wname')`
                  `[S,W1,W2,W3,XYVAL] = wavefun2('wname',4)`

**Description**   For an orthogonal wavelet `'wname'`, `wavefun2` returns the scaling
                  function and the three wavelet functions resulting from the tensor
                  products of the one-dimensional scaling and wavelet functions.

                  If `[PHI,PSI,XVAL] = wavefun('wname',ITER)`, the scaling function `S`
                  is the tensor product of `PHI` and `PSI`.

                  The wavelet functions `W1`, `W2`, and `W3` are the tensor products (`PHI`,`PSI`),
                  (`PSI`,`PHI`), and (`PSI`,`PSI`), respectively.

                  The two-dimensional variable `XYVAL` is a $2^{\text{ITER}}$ x $2^{\text{ITER}}$ points grid
                  obtained from the tensor product (`XVAL`,`XVAL`).

                  The positive integer `ITER` determines the number of iterations computed
                  and thus, the refinement of the approximations.

                  `[S,W1,W2,W3,XYVAL] = wavefun2('wname',ITER,'plot')` computes
                  and also plots the functions.

                  `[S,W1,W2,W3,XYVAL] = wavefun2(wname,A,B)`, where `A` and `B` are
                  positive integers, is equivalent to
                  `[S,W1,W2,W3,XYVAL] = wavefun2('wname',max(A,B))`. The resulting
                  functions are plotted.

                  When `A` is set equal to the special value 0,

                  - `[S,W1,W2,W3,XYVAL] = wavefun2('wname',0)` is equivalent to
                    `[S,W1,W2,W3,XYVAL] = wavefun2('wname',4,0)`.

- [S,W1,W2,W3,XYVAL] = wavefun2('*wname*') is equivalent to
  [S,W1,W2,W3,XYVAL] = wavefun2('*wname*',4).

The output arguments are optional.

---

**Note** The wavefun2 function can only be used with an orthogonal
wavelet.

---

**Examples**    On the following graph, a linear approximation of the sym4 wavelet
obtained using the cascade algorithm is shown.

```
% Set number of iterations and wavelet name.
iter = 4;
wav = 'sym4';

% Compute approximations of the wavelet and scale functions using
% the cascade algorithm and plot.
[s,w1,w2,w3,xyval] = wavefun2(wav,iter,0);
```

Scale function     Wave function (1)

Wavelet function (2)     Wavelet function (3)

**Algorithms**     See wavefun for more information.

**References**     Daubechies, I., *Ten lectures on wavelets*, CBMS, SIAM, 1992, pp. 202Äì213.

Strang, G.; T. Nguyen (1996), *Wavelets and Filter Banks*, Wellesley-Cambridge Press.

**See Also**     intwave | wavefun | waveinfo | wfilters

**Purpose**    Wavelets information

**Syntax**    waveinfo('*wname*')

**Description**    waveinfo provides information on all wavelets within the toolbox.

waveinfo('*wname*') provides information on the wavelet family whose short name is specified by the string '*wname*'. Available family short names are listed in the table below.

| Wavelet Family Short Name | Wavelet Family Name |
|---|---|
| 'haar' | Haar wavelet |
| 'db' | Daubechies wavelets |
| 'sym' | Symlets |
| 'coif' | Coiflets |
| 'bior' | Biorthogonal wavelets |
| 'rbio' | Reverse biorthogonal wavelets |
| 'meyr' | Meyer wavelet |
| 'dmey' | Discrete approximation of Meyer wavelet |
| 'gaus' | Gaussian wavelets |
| 'mexh' | Mexican hat wavelet |
| 'morl' | Morlet wavelet |
| 'cgau' | Complex Gaussian wavelets |
| 'shan' | Shannon wavelets |
| 'fbsp' | Frequency B-Spline wavelets |
| 'cmor' | Complex Morlet wavelets |

The family short names can also be user-defined ones (see `wavemngr` for more information).

`waveinfo('wsys')` provides information on wavelet packets.

**Examples**
```
waveinfo('db')
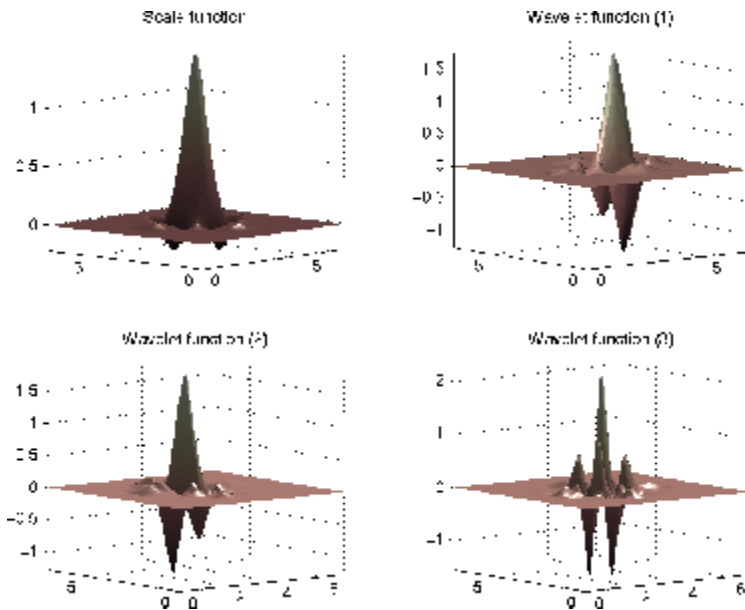
DBINFO Information on Daubechies wavelets.
Daubechies Wavelets
General characteristics: Compactly supported
wavelets with extremal phase and highest
number of vanishing moments for a given
support width. Associated scaling filters are
minimum-phase filters.

Family          Daubechies
Short name      db
Order N         N strictly positive integer
Examples        db1 or haar, db4, db15

Orthogonal      yes
Biorthogonal    yes
Compact support yes
DWT             possible
CWT             possible

Support width   2N-1
Filters length  2N
Regularity      about 0.2 N for large N
Symmetry        far from
Number of vanishing moments for psi     N

Reference:  I. Daubechies,
Ten lectures on wavelets CBMS, SIAM, 61, 1994, 194-202.
```

**See Also**  `wavemngr`

**Purpose**        Wavelet families and family members

**Syntax**         waveletfamilies('*f*')
                   waveletfamilies('*n*')
                   waveletfamilies('*a*')

**Description**    waveletfamilies or waveletfamilies('*f*') displays the names of all
                   available wavelet families.

                   waveletfamilies('*n*') displays the names of all available wavelets
                   in each family.

                   waveletfamilies('*a*') displays all available wavelet families with
                   their corresponding properties.

**Examples**       waveletfamilies

```
===================================
Haar                haar
Daubechies          db
Symlets             sym
Coiflets            coif
BiorSplines         bior
ReverseBior         rbio
Meyer               meyr
DMeyer              dmey
Gaussian            gaus
Mexican_hat         mexh
Morlet              morl
Complex Gaussian    cgau
Shannon             shan
Frequency B-Spline  fbsp
Complex Morlet      cmor
===================================
```

                   waveletfamilies('n')

```
==================================
Haar                    haar
==================================
Daubechies              db
----------------------------------
db1 db2 db3 db4
db5 db6 db7 db8
db9 db10    db**
==================================
Symlets                 sym
----------------------------------
sym2    sym3    sym4    sym5
sym6    sym7    sym8    sym**
==================================
Coiflets                coif
----------------------------------
coif1   coif2   coif3   coif4
coif5
==================================
BiorSplines             bior
----------------------------------
bior1.1 bior1.3 bior1.5 bior2.2
bior2.4 bior2.6 bior2.8 bior3.1
bior3.3 bior3.5 bior3.7 bior3.9
bior4.4 bior5.5 bior6.8
==================================
ReverseBior             rbio
----------------------------------
rbio1.1 rbio1.3 rbio1.5 rbio2.2
rbio2.4 rbio2.6 rbio2.8 rbio3.1
rbio3.3 rbio3.5 rbio3.7 rbio3.9
rbio4.4 rbio5.5 rbio6.8
==================================
Meyer                   meyr
==================================
DMeyer                  dmey
==================================
```

```
Gaussian                gaus
------------------------------
gaus1   gaus2   gaus3   gaus4
gaus5   gaus6   gaus7   gaus8
gaus**
==================================
Mexican_hat            mexh
==================================
Morlet                 morl
==================================
Complex Gaussian       cgau
------------------------------
cgau1   cgau2   cgau3   cgau4
cgau5   cgau**
==================================
Shannon                shan
------------------------------
shan1-1.5   shan1-1 shan1-0.5   shan1-0.1
shan2-3 shan**
==================================
Frequency B-Spline     fbsp
------------------------------
fbsp1-1-1.5 fbsp1-1-1   fbsp1-1-0.5 fbsp2-1-1
fbsp2-1-0.5 fbsp2-1-0.1 fbsp**
==================================
Complex Morlet         cmor
------------------------------
cmor1-1.5   cmor1-1 cmor1-0.5   cmor1-1
cmor1-0.5   cmor1-0.1   cmor**
==================================

waveletfamilies('a')

Type of Wavelets
----------------
type = 1   - orthogonals wavelets       (F.I.R.)
type = 2   - biorthogonals wavelets     (F.I.R.)
```

```
type = 3   - with scale function
type = 4   - without scale function
type = 5   - complex wavelet.
---------------------------------------------------------------

-----------------------
Family Name : Haar
haar
1
no
no
dbwavf

-----------------------
Family Name : Daubechies
db
1
1 2 3 4 5 6 7 8 9 10 **
integer
dbwavf

-----------------------
Family Name : Symlets
sym
1
2 3 4 5 6 7 8 **
integer
symwavf

-----------------------
Family Name : Coiflets
coif
1
1 2 3 4 5
integer
coifwavf
```

```
-----------------------
Family Name : BiorSplines
bior
2
1.1 1.3 1.5 2.2 2.4 2.6 2.8 3.1 3.3 3.5 3.7 3.9 4.4 5.5 6.8
real
biorwavf

-----------------------
Family Name : ReverseBior
rbio
2
1.1 1.3 1.5 2.2 2.4 2.6 2.8 3.1 3.3 3.5 3.7 3.9 4.4 5.5 6.8
real
rbiowavf

-----------------------
Family Name : Meyer
meyr
3
no
no
meyer
-8 8
-----------------------
Family Name : DMeyer
dmey
1
no
no
dmey.mat

-----------------------
Family Name : Gaussian
gaus
4
1 2 3 4 5 6 7 8 **
```

```
integer
gauswavf
-5 5
-----------------------
Family Name : Mexican_hat
mexh
4
no
no
mexihat
-8 8
-----------------------
Family Name : Morlet
morl
4
no
no
morlet
-8 8
-----------------------
Family Name : Complex Gaussian
cgau
5
1 2 3 4 5 **
integer
cgauwavf
-5 5
-----------------------
Family Name : Shannon
shan
5
1-1.5 1-1 1-0.5 1-0.1 2-3 **
string
shanwavf
-20 20
-----------------------
Family Name : Frequency B-Spline
```

```
fbsp
5
1-1-1.5 1-1-1 1-1-0.5 2-1-1 2-1-0.5 2-1-0.1 **
string
fbspwavf
-20 20
-----------------------
Family Name : Complex Morlet
cmor
5
1-1.5 1-1 1-0.5 1-1 1-0.5 1-0.1 **
string
cmorwavf
-8 8
-----------------------
```

**See Also**　　　wavemngr

**Purpose**        Wavelet Toolbox GUI tools

**Syntax**

**Description**     wavemenu opens a menu for accessing the various graphical tools
                   provided in the Wavelet Toolbox software.

**Examples**       wavemenu

**Purpose**        Wavelet manager

**Syntax**         wavemngr('add',FN,FSN,WT,NUMS,FILE)
                   wavemngr('add',FN,FSN,WT,NUMS,FILE,B)
                   wavemngr('add',FN,FSN,WT,{NUMS,TYPNUMS},FILE)
                   wavemngr('add',FN,FSN,WT,{NUMS,TYPNUMS},FILE,B)

**Description**    wavemngr is a type of wavelets manager. It allows you to add, delete, restore, or read wavelets.

                   wavemngr('add',FN,FSN,WT,NUMS,FILE) or
                   wavemngr('add',FN,FSN,WT,NUMS,FILE,B) or
                   wavemngr('add',FN,FSN,WT,{NUMS,TYPNUMS},FILE) or
                   wavemngr('add',FN,FSN,WT,{NUMS,TYPNUMS},FILE,B), add a new wavelet family to the toolbox.

                   FN = Family Name (string)

                   FSN = Family Short Name (string of length equal or less than four characters)

                   WT defines the wavelet type:

                   • WT = 1, for orthogonal wavelets

                   • WT = 2, for biorthogonal wavelets

                   • WT = 3, for wavelet with scaling function

                   • WT = 4, for wavelet without scaling function

                   • WT = 5, for complex wavelet without scaling function

                   If the family contains a single wavelet, NUMS = ' '.

                   Examples:

                   mexh              j

                   morl

                   If the wavelet is member of a finite family of wavelets, NUMS is a

string containing a blank separated list of items representing wavelet parameters.

Example:

```
 bior              NUMS = '1.1  1.3  ...   4.4  5.5  6.8'
```

If the wavelet is part of an infinite family of wavelets, NUMS is a string containing a blank separated list of items representing wavelet parameters, terminated by the special sequence **.

Examples:

```
 db                NUMS = '1  2  3  4  5  6  7  8  9  10
                   **'
 shan              NUMS = '1-1.5  1-1  1-0.5  1-0.1  2-3
                   **'
```

In these last two cases, TYPNUMS specifies the wavelet parameter input format: 'integer' or 'real' or 'string'; the default value is 'integer'.

Examples:

```
 db               TYPNUMS = 'integer'
 bior             TYPNUMS = 'real'
 shan             TYPNUMS = 'string'
```

FILE = MAT-file or code file name (string). See usage in the "Examples" section.

B = [lb ub] specifies lower and upper bounds of effective support for wavelets of type = 3, 4, or 5.

This option is fully documented in "Adding Your Own Wavelets" in the User's Guide.

wavemngr('del',N), deletes a wavelet or a wavelet family. N is the
Family Short Name or the Wavelet Name (in the family). N is a string.

wavemngr('restore') or wavemngr('restore',IN2) restores previous
or initial wavelets. If nargin = 1, the previous wavelets.asc
ASCII-file is restored; otherwise the initial wavelets.asc ASCII-file
is restored. Here IN2 is a dummy argument.

OUT1 = wavemngr('read') returns all wavelet family names.

OUT1 = wavemngr('read',IN2) returns all wavelet names, IN2 is a
dummy argument.

OUT1 = wavemngr('read_asc') reads wavelets.asc ASCII-file and
returns all wavelets information.

**Examples**

```
% List initial wavelets families.
    wavemngr('read')

ans =
==================================
Haar                haar
Daubechies          db
Symlets             sym
Coiflets            coif
BiorSplines         bior
ReverseBior         rbio
Meyer               meyr
DMeyer              dmey
Gaussian            gaus
Mexican_hat         mexh
Morlet              morl
Complex Gaussian    cgau
Shannon             shan
Frequency B-Spline  fbsp
Complex Morlet      cmor
==================================
% List all wavelets.
wavemngr('read',1)
```

```
ans =

==================================
Haar              haar
==================================
Daubechies        db
----------------------------------
db1   db2   db3   db4
db5   db6   db7   db8
db9   db10  db**
==================================
Symlets           sym
----------------------------------
sym2  sym3  sym4  sym5
sym6  sym7  sym8  sym**
==================================
Coiflets          coif
----------------------------------
coif1  coif2  coif3  coif4
coif5
==================================
BiorSplines       bior
----------------------------------
bior1.1   bior1.3   bior1.5   bior2.2
bior2.4   bior2.6   bior2.8   bior3.1
bior3.3   bior3.5   bior3.7   bior3.9
bior4.4   bior5.5   bior6.8
==================================
ReverseBior       rbio
----------------------------------
rbio1.1   rbio1.3   rbio1.5   rbio2.2
rbio2.4   rbio2.6   rbio2.8   rbio3.1
rbio3.3   rbio3.5   rbio3.7   rbio3.9
rbio4.4   rbio5.5   rbio6.8
==================================
Meyer             meyr
```

```
=================================
DMeyer              dmey
=================================
Gaussian            gaus
---------------------------------
gaus1   gaus2   gaus3   gaus4
gaus5   gaus6   gaus7   gaus8
gaus**
=================================
Mexican_hat         mexh
=================================
Morlet              morl
=================================
Complex Gaussian    cgau
---------------------------------
cgau1   cgau2   cgau3   cgau4
cgau5   cgau**
=================================
Shannon             shan
---------------------------------
shan1-1.5   shan1-1   shan1-0.5   shan1-0.1
shan2-3   shan**
=================================
Frequency B-Spline    fbsp
---------------------------------
fbsp1-1-1.5   fbsp1-1-1   fbsp1-1-0.5   fbsp2-1-1
fbsp2-1-0.5   fbsp2-1-0.1   fbsp**
=================================
Complex Morlet        cmor
---------------------------------
cmor1-1.5   cmor1-1   cmor1-0.5   cmor1-1
cmor1-0.5   cmor1-0.1   cmor**
=================================
```

In the following example, new compactly supported orthogonal
wavelets are added to the toolbox. These wavelets, which are a slight
generalization of the Daubechies wavelets, are based on the use

of Bernstein polynomials and are due to Kateb and Lemarié in an unpublished work.

---

**Note** The files used in this example can be found in the wavedemo folder.

---

```
% Add new family of orthogonal wavelets.
% You must define:
%
%     Family Name:        Lemarie
%     Family Short Name:  lem
%     Type of wavelet:    1 (orth)
%     Wavelets numbers:   1 2 3 4 5
%     File driver:        lemwavf
%
%     The function lemwavf.m must be as follows:
%     function w = lemwavf(wname)
%     where the input argument wname is a string:
%     wname = 'lem1' or 'lem2' ... i.e.,
%     wname = sh.name + number
%     and w the corresponding scaling filter.
%     The addition is obtained using:

wavemngr('add','Lemarie','lem',1,'1 2 3 4 5','lemwavf');

% The ascii file 'wavelets.asc' is saved as
% 'wavelets.prv', then it is modified and
% the MAT file 'wavelets.inf' is generated.

% List wavelets families.
wavemngr('read')

ans =
===================================
Haar                haar
Daubechies          db
```

```
Symlets            sym
Coiflets           coif
BiorSplines        bior
ReverseBior        rbio
Meyer              meyr
DMeyer             dmey
Gaussian           gaus
Mexican_hat        mexh
Morlet             morl
Complex Gaussian   cgau
Shannon            shan
Frequency B-Spline fbsp
Complex Morlet     cmor
Lemarie            lem
==================================
% Remove the added family.
wavemngr('del','Lemarie');

% List wavelets families.
wavemngr('read')

ans =
==================================
Haar               haar
Daubechies         db
Symlets            sym
Coiflets           coif
BiorSplines        bior
ReverseBior        rbio
Meyer              meyr
DMeyer             dmey
Gaussian           gaus
Mexican_hat        mexh
Morlet             morl
Complex Gaussian   cgau
Shannon            shan
Frequency B-Spline fbsp
```

```
Complex Morlet      cmor
==================================
% Restore the previous ascii file
% 'wavelets.prv', then build
% the MAT-file 'wavelets.inf'.
wavemngr('restore');

% List restored wavelets.
wavemngr('read',1)

ans =
==================================
Haar                haar
==================================
Daubechies          db
----------------------------------
db1    db2    db3    db4
db5    db6    db7    db8
db9    db10   db**
==================================
Symlets             sym
----------------------------------
sym2   sym3   sym4   sym5
sym6   sym7   sym8   sym**
==================================
Coiflets            coif
----------------------------------
coif1   coif2   coif3   coif4
coif5
==================================
BiorSplines         bior
----------------------------------
bior1.1   bior1.3   bior1.5   bior2.2
bior2.4   bior2.6   bior2.8   bior3.1
bior3.3   bior3.5   bior3.7   bior3.9
bior4.4   bior5.5   bior6.8
==================================
```

```
ReverseBior          rbio
------------------------------
rbio1.1   rbio1.3   rbio1.5   rbio2.2
rbio2.4   rbio2.6   rbio2.8   rbio3.1
rbio3.3   rbio3.5   rbio3.7   rbio3.9
rbio4.4   rbio5.5   rbio6.8
==================================
Meyer                meyr
==================================
DMeyer               dmey
==================================
Gaussian             gaus
------------------------------
gaus1   gaus2   gaus3   gaus4
gaus5   gaus6   gaus7   gaus8
gaus**
==================================
Mexican_hat          mexh
==================================
Morlet               morl
==================================
Complex Gaussian     cgau
------------------------------
cgau1   cgau2   cgau3   cgau4
cgau5   cgau**
==================================
Shannon              shan
------------------------------
shan1-1.5   shan1-1   shan1-0.5   shan1-0.1
shan2-3   shan**
==================================
Frequency B-Spline   fbsp
------------------------------
fbsp1-1-1.5   fbsp1-1-1   fbsp1-1-0.5   fbsp2-1-1
fbsp2-1-0.5   fbsp2-1-0.1   fbsp**
==================================
Complex Morlet       cmor
```

```
                        ------------------------------
                        cmor1-1.5    cmor1-1    cmor1-0.5    cmor1-1
                        cmor1-0.5    cmor1-0.1    cmor**
                        ================================
                        Lemarie      lem
                        ------------------------------
                        lem1    lem2    lem3    lem4    lem5
                        ================================
                        % Restore initial wavelets.
                        %
                        % Restore the initial ascii file
                        % 'wavelets.ini' and initial
                        % MAT-file 'wavelets.bin'.
                        wavemngr('restore',0);

                        % List wavelets families.
                        wavemngr('read')

                        ans =
                        ================================
                        Haar                haar
                        Daubechies          db
                        Symlets             sym
                        Coiflets            coif
                        BiorSplines         bior
                        ReverseBior         rbio
                        Meyer               meyr
                        DMeyer              dmey
                        Gaussian            gaus
                        Mexican_hat         mexh
                        Morlet              morl
                        Complex Gaussian    cgau
                        Shannon             shan
                        Frequency B-Spline  fbsp
                        Complex Morlet      cmor
                        ================================
                        % Add new family of orthogonal wavelets.
```

```
wavemngr('add','Lemarie','lem',1,'1 2 3','lemwavf');

% All command line capabilities are available for
% the new wavelets.
%
% Example 1: compute the four associated filters.
[Lo_D,Hi_D,Lo_R,Hi_R] = wfilters('lem3');

% Example 2: compute scale and wavelet functions.
[phi,psi,xval] = wavefun('lem3');

% Add a new family of orthogonal wavelets: special form
% for the GUI mode.
%
% The file lemwavf allows you to compute the filter for
% any order. If you want to get a popup of the form
% 1 2 3 **, associated with the family, then wavelets are
% appended for GUI mode using:

wavemngr('restore',O);
wavemngr('add','Lemarie','lem',1,'1 2 3 **','lemwavf');

% After this sequence, all GUI capabilities are available for
% the new wavelets.
% Note that the last command allows a short cut in the
% order definition only if possible orders are integers.
```

**Caution** wavemngr works on the current folder. If you add a new wavelet family, it is available in this folder only. Refer to, "Adding Your Own Wavelets", in the User's Guide.

**Limitations** wavemngr allows you to add a new wavelet. You must verify that it is truly a wavelet. No check is performed either about this point or about the type of the new wavelet.

# wavenames

**Purpose**          Wavelet names for LWT

**Syntax**          W = wavenames(*T*)

**Description**          W = wavenames(*T*) returns a cell array that contains the name of all wavelets of type *T*. The valid values for *T* are

- `'all'` — all wavelets
- `'lazy'` — "lazy" wavelet
- `'orth'` — orthogonal wavelets
- `'bior'` — biorthogonal wavelets

W = wavenames is equivalent to W = wavenames(`'all'`).

**Purpose**    Multilevel 1-D wavelet reconstruction

**Syntax**
```
X = waverec(C,L,Lo_R,Hi_R)
X = waverec(C,L,'wname')
X = appcoef(C,L,'wname',0)
```

**Description**    waverec performs a multilevel one-dimensional wavelet reconstruction using either a specific wavelet ('*wname*', see wfilters) or specific reconstruction filters (Lo_R and Hi_R). .

> **Note** waverec supports only Type 1 (orthogonal) or Type 2 (biorthogonal) wavelets.

X = waverec(C,L,'*wname*') reconstructs the signal X based on the multilevel wavelet decomposition structure [C,L] and wavelet '*wname*'. (For information about the decomposition structure, see wavedec.)

X = waverec(C,L,Lo_R,Hi_R) reconstructs the signal X as above, using the reconstruction filters you specify. Lo_R is the reconstruction low-pass filter and Hi_R is the reconstruction high-pass filter.

Note that X = waverec(C,L,'*wname*') is equivalent to X = appcoef(C,L,'*wname*',0).

**Examples**
```
% The current extension mode is zero-padding (see dwtmode).

% Load original one-dimensional signal.
load leleccum; s = leleccum(1:3920); ls = length(s);

% Perform decomposition of signal at level 3 using db5.
[c,l] = wavedec(s,3,'db5');

% Reconstruct s from the wavelet decomposition structure [c,l].
a0 = waverec(c,l,'db5');
```

```
% Check for perfect reconstruction.
err = norm(s-a0)
err =
    3.2079e-09
```

**See Also**   appcoef | idwt | wavedec

# waverec2

| | |
|---|---|
| **Purpose** | Multilevel 2-D wavelet reconstruction |

**Syntax**
```
X = waverec2(C,S,'wname')
X = waverec2(C,S,Lo_R,Hi_R)
waverec2(wavedec2(X,N,'wname'),'wname')
X = waverec2(C,S,'wname')
X = appcoef2(C,S,'wname',0)
```

**Description**    X = waverec2(C,S,'*wname*') performs a multilevel wavelet reconstruction of the matrix X based on the wavelet decomposition structure [C,S]. For detailed storage information, see wavedec2. '*wname*' is a string containing the name of the wavelet. See wfilters for more information.

Instead of specifying the wavelet name, you can specify the filters.

- X = waverec2(C,S,Lo_R,Hi_R), Lo_R is the reconstruction low-pass filter

- Hi_R is the reconstruction high-pass filter.

waverec2 is the inverse function of wavedec2 in the sense that the abstract statement waverec2(wavedec2(X,N,'*wname*'),'*wname*') returns X.

X = waverec2(C,S,'*wname*') is equivalent to X = appcoef2(C,S,'*wname*',0).

**Tips**    If C and S are obtained from an indexed image analysis or a truecolor image analysis, X is an m-by-n matrix or an m-by-n-by-3 array, respectively.

For more information on image formats, see the image and imfinfo reference pages.

**Examples**
```
% The current extension mode is zero-padding (see dwtmode).
% Load original image.
load woman;
% X contains the loaded image.
```

```
% Perform decomposition at level 2
% of X using sym4.
[c,s] = wavedec2(X,2,'sym4');
% Reconstruct X from the wavelet
% decomposition structure [c,s].
a0 = waverec2(c,s,'sym4');
% Check for perfect reconstruction.
max(max(abs(X-a0)))
ans =
    2.5565e-10
```

**See Also**        appcoef2 | idwt2 | wavedec2

**Purpose**    Multilevel 3-D wavelet reconstruction

**Syntax**
```
X = waverec3(WDEC)
C = waverec3(WDEC,TYPE,N)
X = waverec3(WDEC,'a',0)
X = waverec3(WDEC,'ca',0)
C = waverec3(WDEC,TYPE)
C = waverec3(WDEC,TYPE,N)
```

**Description**    waverec3 performs a multilevel 3-D wavelet reconstruction starting from a multilevel 3-D wavelet decomposition.

X = waverec3(WDEC) reconstructs the 3-D array X based on the multilevel wavelet decomposition structure WDEC. You can also use waverec3 to extract coefficients from a 3-D wavelet decomposition.

WDEC is a structure with the fields shown in the table.

C = waverec3(WDEC,TYPE,N) reconstructs the multilevel components at level N of a 3-D wavelet decomposition. N must be a positive integer less than or equal to the level of the decomposition.

Valid values for TYPE are:

- A group of three characters 'xyz', one per direction, with 'x','y' and 'z' selected in the set {'a', 'd', 'l', 'h'} or in the corresponding uppercase set {'A','D', 'L', 'H'}), where 'A' (or 'L') is a low-pass filter and 'D' (or 'H') is a high-pass filter.

- The char 'd' (or 'h' or 'D' or 'H') gives the sum of all the components different from the low-pass.

- The char 'a' (or 'l' or 'A' or 'L') gives the low-pass component (the approximation at level N).

For extraction, the valid values for TYPE are the same but prefixed by 'c' or 'C'.

X = waverec3(WDEC,'a',0) or X = waverec3(WDEC,'ca',0) is equivalent to X = waverec3(WDEC). X is a reconstruction of the coefficients in WDEC at level 0.

C = waverec3(WDEC,TYPE) is equivalent to C = waverec3(WDEC,TYPE,N) with N equal to the level of the decomposition.

| sizeINI | Size of the three-dimensional array X |
|---------|----------------------------------------|
| level | Level of the decomposition |
| mode | Name of the wavelet transform extension mode |
| filters | Structure with 4 fields, LoD, HiD, LoR, HiR, which contain the filters used for DWT |
| dec | N x 1 cell array containing the coefficients of the decomposition. N is equal to 7*WDEC.level+1. |
| | dec{1} contains the lowpass component (approximation) at the level of the decomposition. The approximation is equivalent to the filtering operations 'LLL'. |
| | dec{k+2},...,dec{k+8} with k = 0,7,14,...,7*(WDEC.level-1) contain the 3-D wavelet coefficients for the multiresolution starting with the coarsest level when k=0. |
| | For example, if WDEC.level=3, dec{2},...,dec{8} contain the wavelet coefficients for level 3 (k=0), dec{9},...,dec{15} contain the wavelet coefficients for level 2 (k=7), and dec{16},...,dec{22} contain the wavelet coefficients for level 1 (k=7*(WDEC.level-1)). |
| | At each level, the wavelet coefficients in dec{k+2},...,dec{k+8} are in the following order: 'HLL','LHL','HHL','LLH','HLH','LHH','HHH'. |
| | The strings give the order in which the separable filtering operations are applied from left to right. For example, 'LHH' means that the lowpass (scaling) filter with downsampling is |

| | applied to the rows of X, followed by the highpass (wavelet) filter with downsampling applied to the columns of X. Finally, the highpass filter with downsampling is applied to the 3rd dimension of X. |
|---|---|
| sizes | Successive sizes of the decomposition components |

**Examples**　　**Perfect Reconstruction with 3-D Discrete Wavelet Transform**

Construct a 3-D matrix, obtain the wavelet transform down to level 2 using the db2 wavelet, and reconstruct the matrix to verify perfect reconstruction.

Create 3-D matrix.

```
M = magic(8);
X = repmat(M,[1 1 8]);
```

Obtain the 3-D discrete wavelet transform of the matrix and reconstruct the input based on the 3-D approximation and detail coefficients.

```
wd = wavedec3(X,2,'db2');
XR = waverec3(wd);
```

Verify perfect reconstruction using the wavelet decomposition down to level 2.

```
err1 = max(abs(X(:)-XR(:)))
```

Verify that the data matrix is the sum of the approximation and the details from levels 2 and 1.

```
A = waverec3(wd,'LLL');
% Reconstruct the sum of components different from
% the lowpass component.
D = waverec3(wd,'d');
% Check that X = A + D.
```

```
err2 = max(abs(X(:)-A(:)-D(:)))
```

**Compare `waverec3` and `idwt3`**

Compare level-1 reconstructions based on the filtering operations `'LLH'` using `idwt3` and `waverec3`.

```
dwtOut = dwt3(X,'db2');
Xr = idwt3(dwtOut,'LLH');
Xrec = waverec3(wd,'LLH',1);
norm(Xr(:)-Xrec(:))
```

**See Also**     idwt3 | waveinfo | wavedec3

**Purpose**       Wavelet support

**Syntax**        [LB,UB] = wavsupport(wname)

**Description**   [LB,UB] = wavsupport(wname) returns the lower bound, LB, and upper
                 bound, UB, of the support for the wavelet specified by wname. wname is
                 any valid wavelet. For real-valued wavelets with and without scaling
                 functions and complex-valued wavelets without scaling functions
                 (wavelets type 3,4, and 5), the bounds indicate the effective support of
                 the wavelet. For orthogonal and biorthogonal wavelets (type 1 and
                 type 2), the lower and upper bounds are -0.5*(LF-1) and 0.5*(LF-1),
                 where LF is the length of the wavelet filter.

**Examples**     Support of Haar wavelet:

                 ```
                 [LB, UB] = wavsupport('haar');
                 LowerBound = -0.5*(2-1);
                 UpperBound = 0.5*(2-1);
                 % Compare [LB,UB] and [LowerBound, UpperBound]
                 ```

                 Effective support of complex-valued Gaussian wavelet:

                 ```
                 [LB,UB] = wavsupport('cgau3');
                 ```

**See Also**     wavemngr

# wbmpen

| | |
|---|---|
| **Purpose** | Penalized threshold for wavelet 1-D or 2-D de-noising |
| **Syntax** | THR = wbmpen(C,L,SIGMA,ALPHA)<br>wbmpen(C,L,SIGMA,ALPHA,ARG) |

**Description**    THR = wbmpen(C,L,SIGMA,ALPHA) returns global threshold THR for de-noising. THR is obtained by a wavelet coefficients selection rule using a penalization method provided by Birgé-Massart.

[C,L] is the wavelet decomposition structure of the signal or image to be de-noised.

SIGMA is the standard deviation of the zero mean Gaussian white noise in de-noising model (see wnoisest for more information).

ALPHA is a tuning parameter for the penalty term. It must be a real number greater than 1. The sparsity of the wavelet representation of the de-noised signal or image grows with ALPHA. Typically ALPHA = 2.

THR minimizes the penalized criterion given by

let t\* be the minimizer of

crit(t) = -sum(c(k)^2,k t) + 2*SIGMA^2*t*(ALPHA + log(n/t))

where c(k) are the wavelet coefficients sorted in decreasing order of their absolute value and n is the number of coefficients; then THR = |c(t\*)|.

wbmpen(C,L,SIGMA,ALPHA,ARG) computes the global threshold and, in addition, plots three curves:

- 2*SIGMA^2*t*(ALPHA + log(n/t))

- sum(c(k)^2,k‹ t)

- crit(t)

**Examples**
```
% Example 1: Signal de-noising.
% Load noisy bumps signal.
load noisbump; x = noisbump;
```

```
% Perform a wavelet decomposition of the signal
% at level 5 using sym6.
wname = 'sym6'; lev = 5;
[c,l] = wavedec(x,lev,wname);
% Estimate the noise standard deviation from the
% detail coefficients at level 1, using wnoisest.
sigma = wnoisest(c,l,1);

% Use wbmpen for selecting global threshold
% for signal de-noising, using the tuning parameter.
alpha = 2;
thr = wbmpen(c,l,sigma,alpha)
thr =

    2.7681

% Use wdencmp for de-noising the signal using the above
% threshold with soft thresholding and approximation kept.
keepapp = 1;
xd = wdencmp('gbl',c,l,wname,lev,thr,'s',keepapp);

% Plot original and de-noised signals.
figure(1)
subplot(211), plot(x), title('Original signal')
subplot(212), plot(xd), title('De-noised signal')
```

Original signal

De—noised signal

```
% Example 2: Image de-noising.
% Load original image.
load noiswom;
nbc = size(map,1);

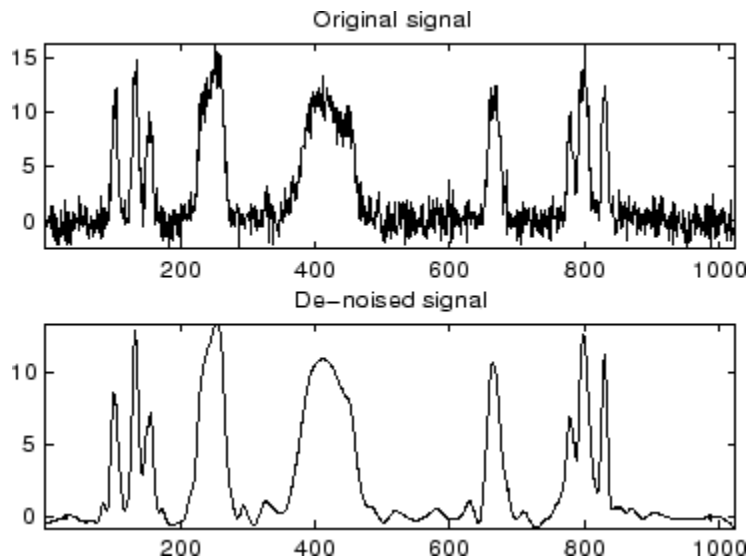% Perform a wavelet decomposition of the image
% at level 3 using coif2.
wname = 'coif2'; lev = 3;
[c,s] = wavedec2(X,lev,wname);

% Estimate the noise standard deviation from the
% detail coefficients at level 1.
det1 = detcoef2('compact',c,s,1);
sigma = median(abs(det1))/0.6745;

% Use wbmpen for selecting global threshold
% for image de-noising.
alpha = 1.2;
thr = wbmpen(c,l,sigma,alpha)
```

```
thr =

   36.0621

% Use wdencmp for de-noising the image using the above
% thresholds with soft thresholding and approximation kept.
keepapp = 1;
xd = wdencmp('gbl',c,s,wname,lev,thr,'s',keepapp);

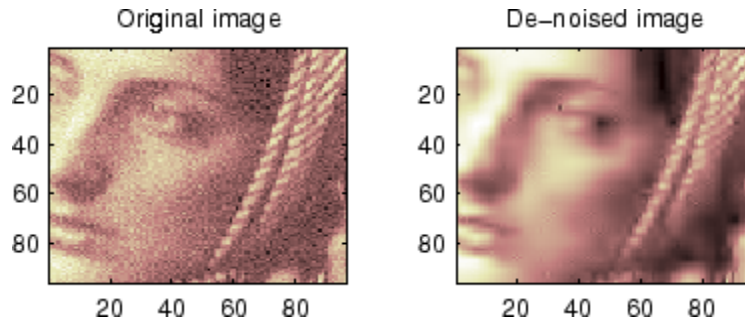% Plot original and de-noised images.
figure(2)
colormap(pink(nbc));
subplot(221), image(wcodemat(X,nbc))
title('Original image')
subplot(222), image(wcodemat(xd,nbc))
title('De-noised image')
```



**See Also**    wden | wdencmp | wpbmpen | wpdencmp

# wcodemat

**Purpose**      Extended pseudocolor matrix scaling

**Syntax**       Y = wcodemat(X)
                 Y = wcodemat(X,NBCODES)
                 Y = wcodemat(X,NBCODES,OPT)
                 Y = wcodemat(X,NBCODES,OPT,ABSOL)

**Description**  wcodemat rescales an input matrix to a specified range for display. If the specified range is the full range of the current colormap, wcodemat is similar in behavior to imagesc.

Y = wcodemat(X) rescales the matrix X to integers in the range [1,16].

Y = wcodemat(X,NBCODES) rescales the input X as integers in the range [1,NBCODES] . The default value of NBCODES is 16.

Y = wcodemat(X,NBCODES,OPT) rescales the matrix along the dimension specified by OPT. Valid strings for OPT are: 'column' (or 'c'), 'row' (or 'r'), and 'mat' (or 'm'). 'rows' scales X row-wise, 'column' scales X column-wise, and 'mat' scales X globally. The default value of OPT is 'mat'.

Y = wcodemat(X,NBCODES,OPT,ABSOL) rescales the input matrix X based on the absolute values of the entries in X if ABSOL is nonzero, or based on the signed values of X if ABSOL is equal to zero. The default value of ABSOL is 1.

**Examples**    Scale level-one approximation coefficients globally to the full range of the colormap.

```
load woman;
% Get the range of the colormap
NBCOL = size(map,1);
% Obtain the 2D dwt using the Haar wavelet
[cA1,cH1,cV1,cD1] = dwt2(X,'db1');
% Display without scaling
image(cA1);
colormap(map);
title('Unscaled Image');
```

```
figure;
% Display with scaling
image(wcodemat(cA1,NBCOL));
colormap(map);
title('Scaled Image');
```

# wcoher

| | |
|---|---|
| **Purpose** | Wavelet coherence |

**Syntax**

```
WCOH = wcoher(Sig1,Sig2,Scales,wname)
WCOH = wcoher(...,Name,Value)
[WCOH,WCS] = wcoher(...)
[WCOH,WCS,CWT_S1,CWT_S2] = wcoher(...)
[...] = wcoh(...,'plot')
```

**Description**
WCOH = wcoher(Sig1,Sig2,Scales,wname) returns the wavelet coherence for the input signals Sig1 and Sig2 using the wavelet specified in wname at the scales in Scales. The input signals must be real-valued and equal in length.

WCOH = wcoher(...,Name,Value) returns the wavelet coherence with additional options specified by one or more Name,Value pair arguments.

[WCOH,WCS] = wcoher(...) returns the wavelet cross spectrum.

[WCOH,WCS,CWT_S1,CWT_S2] = wcoher(...) returns the continuous wavelet transforms of Sig1 and Sig2.

[...] = wcoh(...,'plot') displays the modulus and phase of the wavelet cross spectrum.

**Input Arguments**

**Sig1**

A real-valued one-dimensional input signal. Sig1 is a row or column vector.

**Sig2**

A real-valued one-dimensional input signal. Sig2 is a row or column vector.

**Scales**

Scales is a vector of real-valued, positive scales at which to compute the wavelet coherence.

**wname**

Wavelet used in the wavelet coherence. wname is any valid wavelet name.

## Name-Value Pair Arguments

**'asc'**

Scale factor for arrows in quiver plot. wcoher represents the phase using quiver. asc corresponds to the scale input argument in quiver.

**Default:** 1

**'nas'**

Number of arrows in scale. Together with the number of scales, nas determines the spacing between the y coordinates in the input to quiver. The y input to quiver is 1:length(Scales)/(nas-1):Scales(end)

**Default:** 20

**'nsw'**

Length of smoothing window in scale. nsw is a positive integer that specifies the length of a moving average filter in scale.

**Default:** 1

**'ntw'**

Length of smoothing window in time. ntw is a positive integer that specifies the length of a moving average filter in time.

**Default:** min[20,0.05*length(Sig1)]

**'plot'**

Type of plot. plot is one of the following strings:

- 'cwt'

Displays the continuous wavelet transforms of signals 1 and 2.

- `'wcs'`

  Displays the wavelet cross spectrum.

- `'wcoh'`

  Displays the phase of the wavelet cross spectrum.

- `'all'`

  Displays all plots in separate figures.

**Output Arguments**

**WCOH**

Wavelet coherence.

**WCS**

Wavelet cross spectrum.

**CWT_S1**

Continuous wavelet transform of signal 1.

**CWT_S2**

Continuous wavelet transform of signal 2.

**Definitions**

**Wavelet Cross Spectrum**

The wavelet cross spectrum of two time series, *x* and *y* is:

$$C_{xy}(a,b) = S(C_x^*(a,b)C_y(a,b))$$

where $C_x(a,b)$ and $C_y(a,b)$ denote the continuous wavelet transforms of *x* and *y* at scales *a* and positions *b*. The superscript * is the complex conjugate and *S* is a smoothing operator in time and scale.

For real-valued time series, the wavelet cross spectrum is real-valued if you use a real-valued analyzing wavelet, and complex-valued if you use a complex-valued analyzing wavelet.

### Wavelet Coherence

The wavelet coherence of two time series *x* and *y* is:

$$\frac{S(C_x^*(a,b)C_y(a,b))}{\sqrt{S(|C_x(a,b)|^2)}\sqrt{S(|C_y(a,b)|^2)}}$$

where $C_x(a,b)$ and $C_y(a,b)$ denote the continuous wavelet transforms of *x* and *y* at scales *a* and positions *b*. The superscript * is the complex conjugate and *S* is a smoothing operator in time and scale.

For real-valued time series, the wavelet coherence is real-valued if you use a real-valued analyzing wavelet, and complex-valued if you use a complex-valued analyzing wavelet.

**Examples**    Wavelet coherence of sine waves in noise with delay:

```
t  = linspace(0,1,2048);
x = sin(16*pi*t)+0.5*randn(1,2048);
y = sin(16*pi*t+pi/4)+0.5*randn(1,2048);
wname  = 'cgau3';
scales = 1:512;
ntw = 21; % smoothing parameter
% Display the modulus and phased of the wavelet cross spectrum.
wcoher(x,y,scales,wname,'ntw',ntw,'plot');
```

Sine wave and Doppler signal:

```
t  = linspace(0,1,1024);
x = -sin(8*pi*t) + 0.4*randn(1,1024);
x = x/max(abs(x));
y = wnoise('doppler',10);
```

# wcoher

```
wname  = 'cgau3';
scales = 1:512;
ntw = 21; % smoothing parameter
% Display of the CWT of the two signals.
wcoher(x,y,scales,wname,'ntw',ntw,'plot','cwt');
% Display of the wavelet cross spectrum.
wcoher(x,y,scales,wname,'ntw',ntw,'nsw',1,'plot','wcs');
% Display of the modulus and phased of the wavelet cross spectrum.
wcoher(x,y,scales,wname,'ntw',ntw,'plot');
```

**References**    Grinsted, A, J.C. Moore, and S. Jevrejeva. "Application of the cross wavelet transform and wavelet coherence to geophysical time series. *Nonlinear Processes in Geophysics*. 11, 2004, pp. 561-566.

Torrence. C., and G. Compo. "A Practical Guide to Wavelet Analysis". *Bulletin of the American Meteorological Society*, 79, pp. 61-78.

**See Also**    cwt

**Tutorials**    • "1-D Continuous Wavelet Analysis"

• "One-Dimensional Complex Continuous Wavelet Analysis"

**How To**    • Wavelet Coherence

| | |
|---|---|
| **Purpose** | True compression of images using wavelets |
| **Syntax** | wcompress('c',X,SAV_FILENAME,COMP_METHOD)<br>wcompress(...,'ParName1',ParVal1,'ParName2',ParVal2,...)<br>[COMPRAT,BPP] = wcompress('c',...)<br>XC = wcompress('u',SAV_FILENAME)<br>XC = wcompress('u',SAV_FILENAME,'plot')<br>XC = wcompress('u',SAV_FILENAME,'step') |
| **Description** | The wcompress command performs either compression or uncompression of grayscale or truecolor images. |

More theoretical information on true compression is in "True Compression for Images" of the Wavelet Toolbox User's Guide.

### Compression

wcompress('c',X,SAV_FILENAME,COMP_METHOD) compresses the image X using the compression method COMP_METHOD.

The compressed image is saved in the file SAV_FILENAME. X can be either a 2-D array containing an indexed image or a 3-D array of uint8 containing a truecolor image.

wcompress('c',FILENAME,...) loads the image X from the file FILENAME which is a MATLAB Supported Format (MSF) file: MAT-file or other image files (see imread).

wcompress('c',I,...) converts the indexed image X = I{1} to a truecolor image Y using the colormap map = I{2} and then compresses Y.

The valid compression methods are divided in three categories.

**1** Progressive Coefficients Significance Methods (**PCSM**):

| MATLAB Name | Compression Method Name |
|---|---|
| 'ezw' | Embedded Zerotree Wavelet |
| 'spiht' | Set Partitioning In Hierarchical Trees |

| MATLAB Name | Compression Method Name |
|---|---|
| 'stw' | Spatial-orientation Tree Wavelet |
| 'wdr' | Wavelet Difference Reduction |
| 'aswdr' | Adaptively Scanned Wavelet Difference Reduction |
| 'spiht_3d' | Set Partitioning In Hierarchical Trees 3D for truecolor images |

For more details on these methods, see the references and especially Walker and also Said and Pearlman.

**2** Coefficients Thresholding Methods (**CTM-1**):

| MATLAB Name | Compression Method Name |
|---|---|
| 'lvl_mmc' | Subband thresholding of coefficients and Huffman encoding |

For more details on this method, see the Strang and Nguyen reference.

**3** Coefficients Thresholding Methods (**CTM-2**):

| MATLAB Name | Compression Method Name |
|---|---|
| 'gbl_mmc_f' | Global thresholding of coefficients and fixed encoding |
| 'gbl_mmc_h' | Global thresholding of coefficients and Huffman encoding |

> **Note** The Discrete Wavelet Transform uses the periodized extension
> mode. Each of the two dimensions of the image must be a power of 2.

All the compression methods use parameters which have default values.
You can change these values using the following syntax:

```
wcompress(...,'ParName1',ParVal1,'ParName2',ParVal2,...)
```

Some of the parameters are related to display or to data transform
functionalities. The others are linked to the compression process itself.

### Data transform parameters

- `'ParName'` = `'wname'` or `'WNAME'` sets the wavelet name.

  `ParVal` is a string (see `waveletfamilies`). The default for is *bior4.4*

- `'ParName'` = `'level'` or `'LEVEL'` sets the level of decomposition.

  `ParVal` is an integer such that: $1 \leq$ `level` $\leq$ `levmax` which is the
  maximum possible level (see `wmaxlev`).

  The default level depends on the method:

    - for **PCSM** methods `level` is equal to `levmax`.

    - for **CTM** methods `level` is equal to `fix(levmax/2)`

- `ParName'` = `'it'` or `'IT'` sets Image type Transform.

  `ParVal` must be one of the following strings:

  `'n'` : no transformation (default), image type (truecolor or grayscale)
  is automatically detected.

  `'g'` : grayscale transformation type.

  `'c'` : color transformation type (RGB `uint8`).

- `'ParName'` = `'cc'` or `'CC'` sets Color Conversion parameter if X is
  a truecolor image.

  `ParVal` must be one of the following strings:

'rgb' or 'none' : No conversion (default).

'yuv' : YUV color space transform.

'klt' : Karhunen-Loeve transform.

'yiq' : YIQ color space transform.

'xyz' : CIEXYZ color space transform.

### Parameter for Progressive Coefficients Significance Methods (PCSM)

- 'ParName' = 'maxloop' or 'MAXLOOP' sets the maximum number of steps for the compression algorithm.

  ParVal must be a positive integer or Inf (default is 10).

### Parameters for Coefficients Thresholding Methods (CTM-1)

Either of the following parameters may be used:

- 'ParName' = 'bpp' or 'BPP' sets the bit-per-pixel ratio.

  ParVal must be such that $0 \leq \texttt{ParVal} \leq 8$ (grayscale) or 24 (truecolor).

- 'ParName' = 'comprat' or 'COMPRAT' sets the compression ratio.

  ParVal must be such that $0 \leq \texttt{ParVal} \leq 100$.

### Parameters for Coefficients Thresholding Methods (CTM-2)

Two parameters may be used. The first is related to the threshold and the second is the number of classes for quantization.

The first one may be chosen among the five following parameters:

- 'ParName' = 'threshold' or 'THRESHOLD' sets the threshold value for compression.

  ParVal must be a positive (or zero) real number.

- 'ParName' = 'nbcfs' or 'NBCFS' sets the number of preserved coefficients in the wavelet decomposition.

ParVal must be an integer such that: $0 \leq$ ParVal $\leq$ total number of coefficients of wavelet decomposition.

- 'ParName' = 'percfs' or 'PERCFS' sets the percentage of preserved coefficients in the wavelet decomposition.

  ParVal must be a real number such that: $0 \leq$ ParVal $\leq$ 100.

- 'ParName' = 'bpp' or 'BPP' sets the bit-per-pixel ratio.

  ParVal must be such that: $0 \leq$ ParVal $\leq 8$ (grayscale) or 24 (truecolor)

- 'ParName' = 'comprat' or 'COMPRAT' sets the compression ratio.

  ParVal must be such that: $0 \leq$ ParVal $\leq$ 100.

The second parameter sets the number of classes for quantization:

- 'ParName' = 'nbclas' or 'NBCLAS' sets the number of classes.

  ParVal must be a real number such that: $2 \leq$ ParVal $\leq$ 200.

### Display parameter

- 'ParName' = 'plotpar' or 'PLOTPAR' sets the plot parameter.

  ParVal must be one of the following strings or numbers:

  'plot' or 0: plots only the compressed image.

  'step' or 1: displays each step of the encoding process (only for **PCSM** methods).

[COMPRAT,BPP] = wcompress('c',...) returns the compression ratio COMPRAT and the bit_per_pixel ratio BPP.

### Uncompression

XC = wcompress('u',SAV_FILENAME) uncompresses the file SAV_FILENAME and returns the image XC. Depending on the initial compressed image, XC can be a 2-D array containing either an indexed image or a 3-D array of uint8 containing a truecolor image.

XC = wcompress('u',SAV_FILENAME,'plot') plots the uncompressed image.

XC = wcompress('u',SAV_FILENAME,'step') shows the step-by-step uncompression, only for **PCSM** methods.

**Examples**

```
% Example 1: Compression and uncompression using
% basic parameters.
%
% This example demonstrates first how to compress the jpeg
% image arms.jpg using the 'stw' compression method and
% save it to the file: 'comp_arms.wtc'.

wcompress('c','arms.jpg','comp_arms.wtc','stw');

% Then, it shows how to load the stored image from
% the file 'comp_arms.wtc' and to display the step by
% step uncompression leading to the final image below.

wcompress('u','comp_arms.wtc','step');
```

Compressed Image

```
% Example 2: Compression and uncompression using
% advanced parameters.
%
% This example demonstrates how to compress a jpeg
% image using the 'aswdr' compression method and
% save it to the file: 'woodstatue.wtc'.
% During the compression process 3 parameters are used:
% - Conversion color (cc) set to Karhunen-Loeve transform 'klt'
% - Maximum number of loops (maxloop) set to 11
% - Plot type (plotpar) set to step by step display
% By the way two performance indicators are displayed:
% the compression ratio (cr) and the bit-per-pixel ratio (bpp).
```

```
[cr,bpp] = wcompress('c','woodstatue.jpg','woodstatue.wtc', ...
            'aswdr','cc','klt','maxloop',11,'plotpar','step')

cr =

    3.0701

bpp =
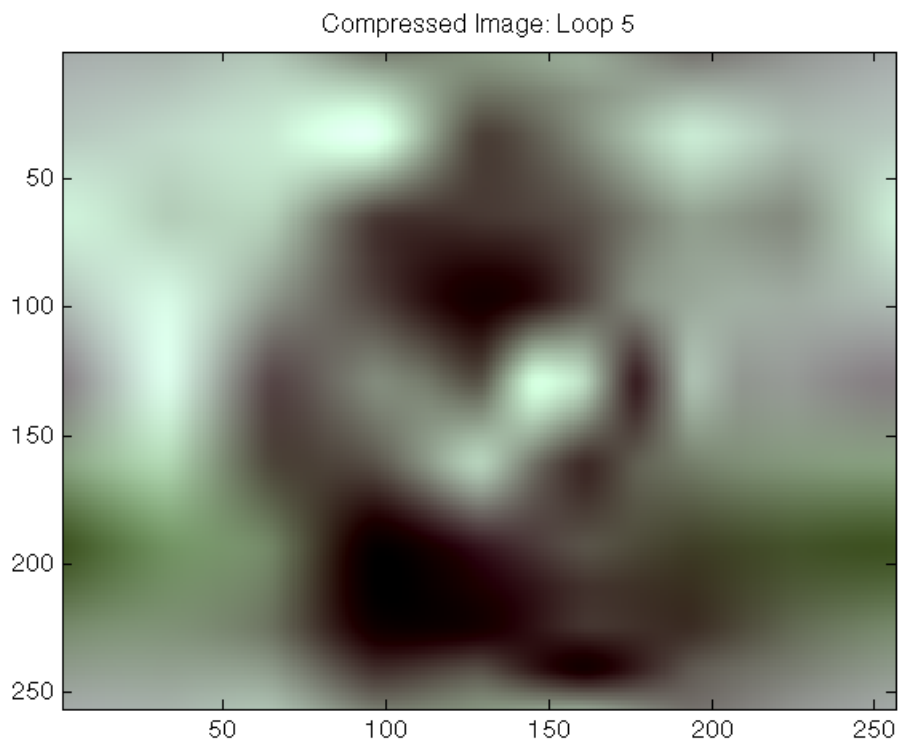
    0.7368
```

Compressed Image: Loop 5

Compressed Image: Loop 8

Compressed Image



```
% Then, it shows how to load the stored image from the
% file 'woodstatue.wtc' and to display the step by step
% uncompression process.
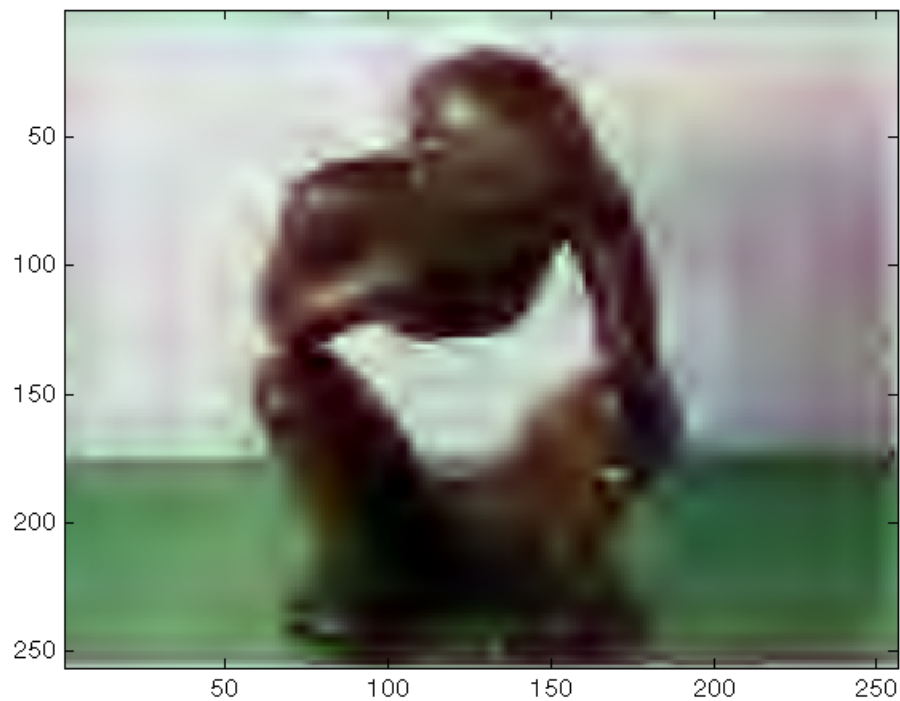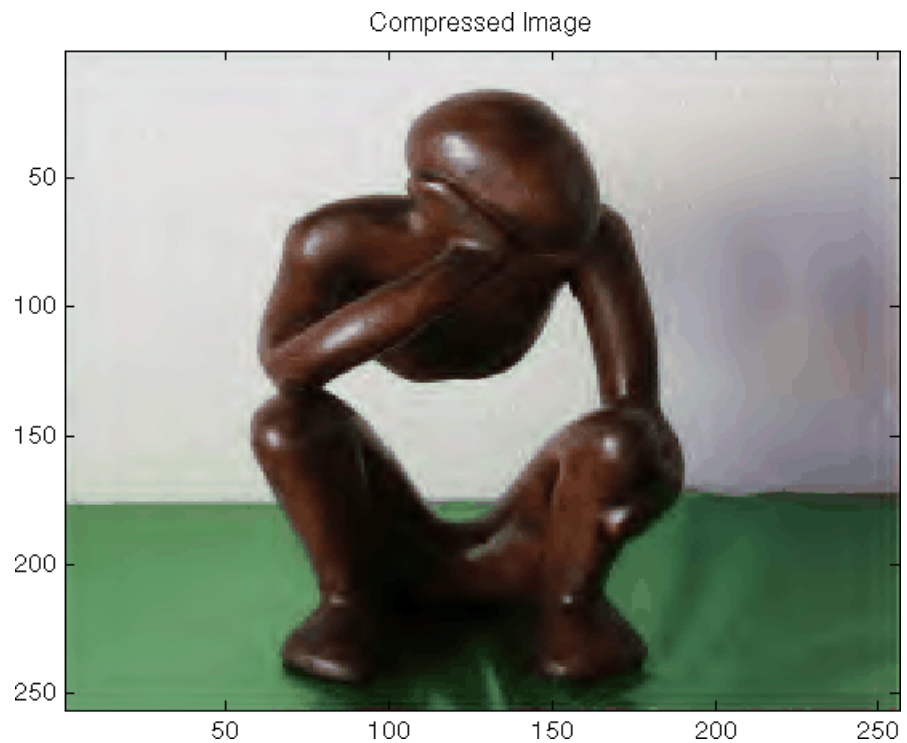
wcompress('u','woodstatue.wtc','step');
```

Compressed Image



```
delete('woodstatue.wtc')

% Example 3: Compression and uncompression of a grayscale image
% and computed MSE and PSNR error values.
%
% Two measures are commonly used to quantify the error between
% two images: the Mean Square Error(MSE) and the Peak Signal
% to Noise Ratio (PSNR) which is expressed in decibels.
%
% This example demonstrates how to compress the mask image using
% the 'spiht' compression method and save it to the 'mask.wtc'
% file.
```

```
load mask;
[cr,bpp] = wcompress('c',X,'mask.wtc','spiht','maxloop',12)

cr =
    2.8336

bpp =

    0.2267

% Then, it shows how to load the stored image from the file
% 'mask.wtc', uncompress it and delete the file 'mask.wtc'.

Xc = wcompress('u','mask.wtc');
delete('mask.wtc')

% The orginal and compressed images are displayed.
colormap(pink(255))
subplot(1,2,1); image(X);  title('Original image')
axis square
subplot(1,2,2); image(Xc); title('Compressed image')
axis square
```

```
% Finally the MSE and the PSNR are computed.
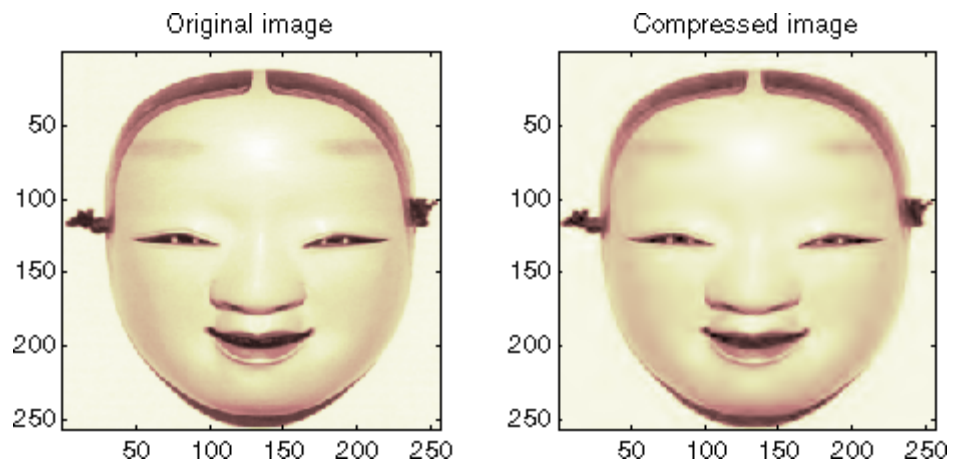
D = abs(X-Xc).^2;
mse  = sum(D(:))/numel(X)

mse =
   33.6564

psnr = 10*log10(255*255/mse)

psnr =

   32.8601
% Example 4: Compression and uncompression of a truecolor image
% and computed MSE and PSNR error values.
% Compression parameters are the same as those used for example 3,
% but using the 'spiht_3d' method give better performance yet.

X = imread('wpeppers.jpg');
[cr,bpp] = wcompress('c',X,'wpeppers.wtc','spiht','maxloop',12)

cr =
    1.6527

bpp =
    0.3966

Xc = wcompress('u','wpeppers.wtc');
delete('wpeppers.wtc')
subplot(1,2,1); image(X);  title('Original image'), axis square
subplot(1,2,2); image(Xc); title('Compressed image'), axis square
```

Original image                                Compressed image

```
D = abs(double(X)-double(Xc)).^2;
mse  = sum(D(:))/numel(X)
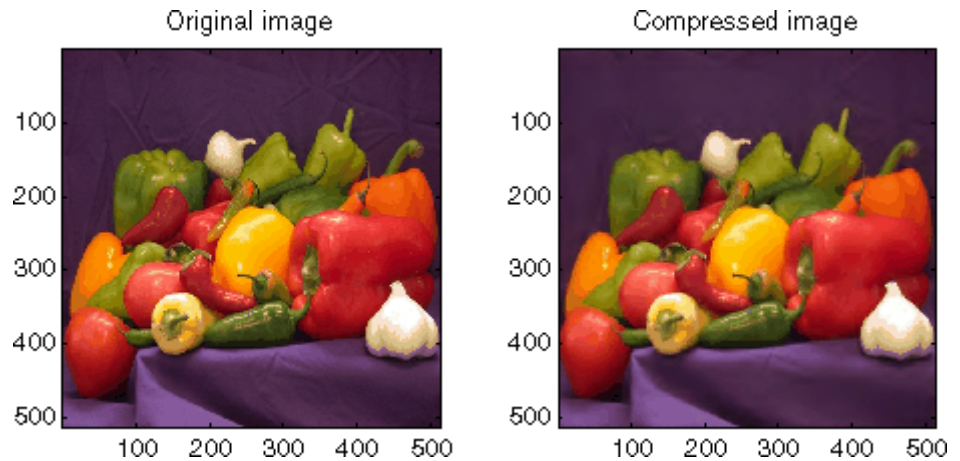
mse =

    26.7808

psnr = 10*log10(255*255/mse)

psnr =

    33.8526
```

**References**     Christophe, E., C. Mailhes, P. Duhamel (2006), "Adaptation of zerotrees using signed binary digit representations for 3 dimensional image coding," EURASIP *Journal on Image and Video Processing*, 2007, to appear in the special issue on Wavelets in Source Coding, Communications, and Networks, Paper ID 54679.

Misiti, M., Y. Misiti, G. Oppenheim, J.-M. Poggi (2007), *Wavelets and their applications*, ISTE DSP Series.

Said A., W.A. Pearlman (1996), "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. on Circuits and Systems for Video Technology*, Vol. 6, No. 3, pp. 243–250.

Shapiro J.M. (1993), "Embedded image coding using zerotrees of wavelet coefficients",P *IEEE Trans. Signal Proc.*, Vol. 41, No. 12, pp. 3445–3462.

Strang, G.; T. Nguyen (1996), *Wavelets and Filter Banks*, Wellesley-Cambridge Press.

Walker J.S. (1999), "Wavelet-Based Image Compression," University of Wisconsin, Eau Claire, Wisconsin, USA, , Sub-chapter of CRC Press book: *Transform and Data Compression. A Primer on Wavelets and Their Scientific Applications*.

**See Also**     `imread | imwrite | wmaxlev`

# wdcbm

**Purpose**      Thresholds for wavelet 1-D using Birgé-Massart strategy

**Syntax**       [THR,NKEEP] = wdcbm(C,L,ALPHA,M)
                 wdcbm(C,L,ALPHA)
                 wdcbm(C,L,ALPHA,L(1))

**Description**  [THR,NKEEP] = wdcbm(C,L,ALPHA,M) returns level-dependent
                 thresholds THR and numbers of coefficients to be kept NKEEP, for
                 de-noising or compression. THR is obtained using a wavelet coefficients
                 selection rule based on the Birgé-Massart strategy.

                 [C,L] is the wavelet decomposition structure of the signal to be
                 de-noised or compressed, at level j = length(L)-2. ALPHA and M must
                 be real numbers greater than 1.

                 THR is a vector of length j; THR(i) contains the threshold for level i.

                 NKEEP is a vector of length j; NKEEP(i) contains the number of
                 coefficients to be kept at level i.

                 j, M and ALPHA define the strategy:

                 • At level j+1 (and coarser levels), everything is kept.

                 • For level i from 1 to j, the $n_i$ largest coefficients are kept with
                   $n_i$ = M / $(j+2-i)^{ALPHA}$.

                 Typically ALPHA = 1.5 for compression and ALPHA = 3 for de-noising.

                 A default value for M is M = L(1), the number of the coarsest
                 approximation coefficients, since the previous formula leads for i = j+1,
                 to $n_{j+1}$ = M = L(1). Recommended values for M are from L(1) to 2*L(1).

                 wdcbm(C,L,ALPHA) is equivalent to wdcbm(C,L,ALPHA,L(1)).

**Examples**     ```
                 % Load electrical signal and select a part of it.
                 load leleccum; indx = 2600:3100;
                 x = leleccum(indx);

                 % Perform a wavelet decomposition of the signal
                 % at level 5 using db3.
                 ```

```
wname = 'db3'; lev = 5;
[c,l] = wavedec(x,lev,wname);

% Use wdcbm for selecting level dependent thresholds
% for signal compression using the adviced parameters.
alpha = 1.5; m = l(1);
[thr,nkeep] = wdcbm(c,l,alpha,m)

thr =
   19.5569   17.1415   20.2599   42.8959   15.0049

nkeep =
     1     2     3     4     7

% Use wdencmp for compressing the signal using the above
% thresholds with hard thresholding.
[xd,cxd,lxd,perf0,perfl2] = ...
                   wdencmp('lvd',c,l,wname,lev,thr,'h');

% Plot original and compressed signals.
subplot(211), plot(indx,x), title('Original signal');
subplot(212), plot(indx,xd), title('Compressed signal');
xlab1 = ['2-norm rec.: ',num2str(perfl2)];
xlab2 = [' %  -- zero cfs: ',num2str(perf0), ' %'];
xlabel([xlab1 xlab2]);
```

Original signal / Compressed signal

2-norm rec.: 99.9549 %  — zero cfs: 92.9524 %

**References**   Birgé, L.; P. Massart (1997), "From model selection to adaptive estimation," in D. Pollard (ed), *Festchrift for L. Le Cam*, Springer, pp. 55–88.

**See Also**   wden | wdencmp | wpdencmp

**Purpose**      Thresholds for wavelet 2-D using Birgé-Massart strategy

**Syntax**       [THR,NKEEP] = wdcbm2(C,S,ALPHA,M)
                 wdcbm2(C,S,ALPHA)
                 wdcbm2(C,S,ALPHA,prod(S(1,:)))

**Description**  [THR,NKEEP] = wdcbm2(C,S,ALPHA,M) returns level-dependent
                 thresholds THR and numbers of coefficients to be kept NKEEP, for
                 de-noising or compression. THR is obtained using a wavelet coefficients
                 selection rule based on the Birgé-Massart strategy.

                 [C,S] is the wavelet decomposition structure of the image to be
                 de-noised or compressed, at level j = size(S,1)-2.

                 ALPHA and M must be real numbers greater than 1.

                 THR is a matrix 3 by j; THR(:,i) contains the level dependent thresholds
                 in the three orientations: horizontal, diagonal, and vertical, for level i.

                 NKEEP is a vector of length j; NKEEP(i) contains the number of
                 coefficients to be kept at level i.

                 j, M and ALPHA define the strategy:

                 • At level j+1 (and coarser levels), everything is kept.

                 • For level i from 1 to j, the $n_i$ largest coefficients are kept with
                   $n_i = M \ (j+2-i)^{\text{ALPHA}}$.

                 Typically ALPHA = 1.5 for compression and ALPHA = 3 for de-noising.

                 A default value for M is M = prod(S(1,:)), the length of the coarsest
                 approximation coefficients, since the previous formula leads for i = j+1,
                 to $n_{j+1} = M = \text{prod}(S(1,:))$.

                 Recommended values for M are from prod(S(1,:)) to 6*prod(S(1,:)).

                 wdcbm2(C,S,ALPHA) is equivalent to
                 wdcbm2(C,S,ALPHA,prod(S(1,:))).

**Examples**     ```
                 % Load original image.
                 load detfingr;
                 ```

```
nbc = size(map,1);

% Perform a wavelet decomposition of the image
% at level 3 using sym4.
wname = 'sym4'; lev = 3;
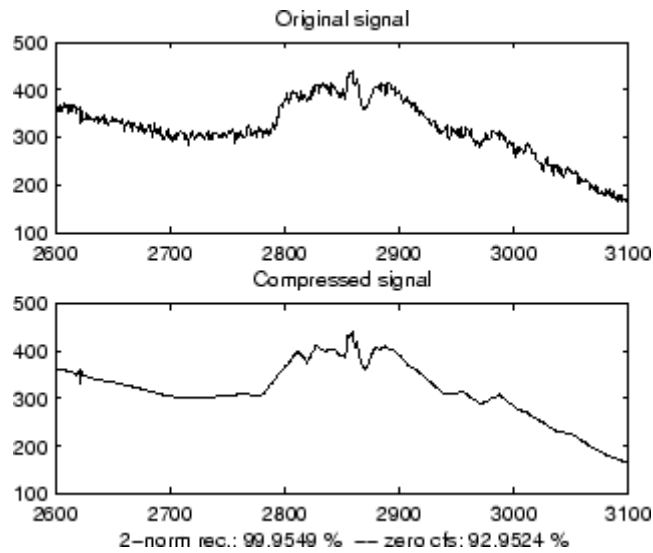[c,s] = wavedec2(X,lev,wname);

% Use wdcbm2 for selecting level dependent thresholds
% for image compression using the adviced parameters.
alpha = 1.5; m = 2.7*prod(s(1,:));
[thr,nkeep] = wdcbm2(c,s,alpha,m)

thr =
   21.4814    46.8354    40.7907
   21.4814    46.8354    40.7907
   21.4814    46.8354    40.7907

nkeep =
         624         961        1765

% Use wdencmp for compressing the image using the above
% thresholds with hard thresholding.
[xd,cxd,sxd,perf0,perfl2] = ...
                   wdencmp('lvd',c,s,wname,lev,thr,'h');

% Plot original and compressed images.
colormap(pink(nbc));
subplot(221), image(wcodemat(X,nbc)),
title('Original image')
subplot(222), image(wcodemat(xd,nbc)),
title('Compressed image')
xlab1 = ['2-norm rec.: ',num2str(perfl2)];
xlab2 = [' %  -- zero cfs: ',num2str(perf0), ' %'];
xlabel([xlab1 xlab2]);
```

Original image | Compressed image

2-norm rec.: 98.0065 % —— zero cfs: 94.4

**References**    Birgé, L.; P. Massart (1997). "From model selection to adaptive estimation," in D. Pollard (ed), *Festchrift for L. Le Cam*, Springer, pp. 55–88.

**See Also**    wdencmp | wpdencmp

# wdecenergy

**Purpose**      Multisignal 1-D decomposition energy distribution

**Syntax**       [E,PEC,PECFS] = wdecenergy(DEC)
                 [E,PEC,PECFS,IDXSORT,LONGS] = wdecenergy(DEC,'sort')
                 [E,PEC,PECFS] = wdecenergy(DEC,OPTSORT,IDXSIG)
                 [E,PEC,PECFS,IDXSORT,LONGS] = wdecenergy(DEC,OPTSORT,IDXSIG)

**Description**  [E,PEC,PECFS] = wdecenergy(DEC) computes the vector E that
                 contains the energy (L2-Norm) of each decomposed signal, the matrix
                 PEC that contains the percentage of energy for each wavelet component
                 (approximation and details) of each signal, and the matrix PECFS that
                 contains the percentage of energy for each coefficient.

- E(i) is the energy (L2-norm) of the ith signal.

- PEC(i,1) is the percentage of energy for the approximation of level
  MAXLEV = DEC.level of the ith signal.

- PEC(i,j), j=2,...,MAXLEV+1 is the percentage of energy for the detail
  of level (MAXLEV+1-j) of the ith signal.

- PECFS(i,j), is the percentage of energy for jth coefficients of the ith
  signal.

[E,PEC,PECFS,IDXSORT,LONGS] = wdecenergy(DEC,'sort') returns
PECFS sorted (by row) in ascending order and an index vector
IDXSORT.

- Replacing 'sort' by 'ascend' returns the same result.

- Replacing 'sort' by 'descend' returns PECFS sorted in descending
  order.

LONGS is a vector containing the lengths of each family of coefficients.

[E,PEC,PECFS] = wdecenergy(DEC,OPTSORT,IDXSIG) returns the
values for the signals whose indices are given by the IDXSIG vector.

[E,PEC,PECFS,IDXSORT,LONGS] =
wdecenergy(DEC,OPTSORT,IDXSIG) returns the values for the
signals whose indices are given by the IDXSIG vector, the index
vector IDXSORT, and LONGS, which is a vector containing the

lengths of each family of coefficients. Valid values for OPTSORT are 'none', 'sort', 'ascend', 'descend'.

**Examples**

```
% Load original 1D-multisignal.
load thinker
% Perform a decomposition at level 2 using wavelet db2.
dec = mdwtdec('r',X,2,'db2');
% Compute the energy distribution.
[E,PEC,PECFS] = wdecenergy(dec);
% Display the total energy and the distribution of energy
% for each wavelet component (A2, D2, D1).
E31 = E(31)
perA2D2D1 = PEC(31,:)
% Compare the coefficient energy distribution
% for signal 1 and signal 31.
PECFS_1  = PECFS(1,:);
PECFS_31 = PECFS(31,:);
figure;
plot(PECFS_1,'--r','linewidth',2); hold on
plot(PECFS_31,'b','linewidth',2);
grid; set(gca,'Xlim',[1,size(PECFS,2)])
title('PECFS1 dashed line and PECFS31 solid line')
```

# wdecenergy



PECFS1 dashed line and PECFS31 solid line

**See Also**          mdwtdec | mdwtrec

**Purpose**      Automatic 1-D de-noising

**Syntax**       [XD,CXD,LXD] = wden(X,TPTR,SORH,SCAL,N,'*wname*')
                 [XD,CXD,LXD] = wden(C,L,TPTR,SORH,SCAL,N,'*wname*')

**Description**   wden is a one-dimensional de-noising function.

wden performs an automatic de-noising process of a one-dimensional signal using wavelets.

[XD,CXD,LXD] = wden(X,TPTR,SORH,SCAL,N,'*wname*') returns a de-noised version XD of input signal X obtained by thresholding the wavelet coefficients.

Additional output arguments [CXD,LXD] are the wavelet decomposition structure (see wavedec for more information) of the de-noised signal XD.

TPTR string contains the threshold selection rule:

- 'rigrsure' uses the principle of Stein's Unbiased Risk.

- 'heursure' is an heuristic variant of the first option.

- 'sqtwolog' for the universal threshold $\sqrt{2\ln(\bullet)}$

- 'minimaxi' for minimax thresholding (see thselect for more information)

SORH ('s' or 'h') is for soft or hard thresholding (see wthresh for more information).

SCAL defines multiplicative threshold rescaling:

'one' for no rescaling

'sln' for rescaling using a single estimation of level noise based on first-level coefficients

'mln' for rescaling done using level-dependent estimation of level noise

Wavelet decomposition is performed at level N and '*wname*' is a string containing the name of the desired orthogonal wavelet (see wmaxlev and wfilters for more information).

[XD,CXD,LXD] = wden(C,L,TPTR,SORH,SCAL,N,'*wname*') returns the same output arguments, using the same options as above, but obtained directly from the input wavelet decomposition structure [C,L] of the signal to be de-noised, at level N and using '*wname*' orthogonal wavelet.

The underlying model for the noisy signal is basically of the following form:

$$s(n) = f(n) + \sigma e(n)$$

where time $n$ is equally spaced.

In the simplest model, suppose that $e(n)$ is a Gaussian white noise $N(0,1)$ and the noise level $\sigma$ a is supposed to be equal to 1.

The de-noising objective is to suppress the noise part of the signal $s$ and to recover $f$.

The de-noising procedure proceeds in three steps:

**1** Decomposition. Choose a wavelet, and choose a level N. Compute the wavelet decomposition of the signal s at level N.

**2** Detail coefficients thresholding. For each level from 1 to N, select a threshold and apply soft thresholding to the detail coefficients.

**3** Reconstruction. Compute wavelet reconstruction based on the original approximation coefficients of level N and the modified detail coefficients of levels from 1 to N.

More details about threshold selection rules are in "Denoising and Nonparametric Function Estimation", in the User's Guide, and in the help of the thselect function. Let us point out that

- The detail coefficients vector is the superposition of the coefficients of $f$ and the coefficients of $e$, and that the decomposition of $e$ leads to detail coefficients that are standard Gaussian white noises.

- Minimax and SURE threshold selection rules are more conservative and are more convenient when small details of function $f$ lie in the

noise range. The two other rules remove the noise more efficiently. The option `'heursure'` is a compromise.

In practice, the basic model cannot be used directly. This section examines the options available, to deal with model deviations. The remaining parameter scal has to be specified. It corresponds to threshold rescaling methods.

- Option scal = `'one'` corresponds to the basic model.

- In general, you can ignore the noise level that must be estimated. The detail coefficients $CD_1$ (the finest scale) are essentially noise coefficients with standard deviation equal to σ. The median absolute deviation of the coefficients is a robust estimate of σ. The use of a robust estimate is crucial because if level 1 coefficients contain *f* details, these details are concentrated in few coefficients to avoid signal end effects, which are pure artifacts due to computations on the edges.

- The option scal = `'sln'` handles threshold rescaling using a single estimation of level noise based on the first-level coefficients.

- When you suspect a nonwhite noise *e*, thresholds must be rescaled by a level-dependent estimation of the level noise. The same kind of strategy is used by estimating $σ_{lev}$ level by level. This estimation is implemented in the file wnoisest, which handles the wavelet decomposition structure of the original signal *s* directly.

- The option scal = `'mln'` handles threshold rescaling using a level-dependent estimation of the level noise.

**Examples**

```
% The current extension mode is zero-padding (see dwtmode).

% Set signal to noise ratio and set rand seed.
snr = 3; init = 2055615866;

% Generate original signal and a noisy version adding
% a standard Gaussian white noise.
[xref,x] = wnoise(3,11,snr,init);
```

```
% De-noise noisy signal using soft heuristic SURE thresholding
% and scaled noise option, on detail coefficients obtained
% from the decomposition of x, at level 5 by sym8 wavelet.
lev = 5;
xd = wden(x,'heursure','s','one',lev,'sym8');

% Plot signals.
subplot(611), plot(xref), axis([1 2048 -10 10]);
title('Original signal');
subplot(612), plot(x), axis([1 2048 -10 10]);
title(['Noisy signal - Signal to noise ratio = ',...
num2str(fix(snr))]);
subplot(613), plot(xd), axis([1 2048 -10 10]);
title('De-noised signal - heuristic SURE');

% De-noise noisy signal using soft SURE thresholding
xd = wden(x,'heursure','s','one',lev,'sym8');

% Plot signal.
subplot(614), plot(xd), axis([1 2048 -10 10]);
title('De-noised signal - SURE');

% De-noise noisy signal using fixed form threshold with
% a single level estimation of noise standard deviation.
xd = wden(x,'sqtwolog','s','sln',lev,'sym8');

% Plot signal.
subplot(615), plot(xd), axis([1 2048 -10 10]);
title('De-noised signal - Fixed form threshold');

% De-noise noisy signal using minimax threshold with
% a multiple level estimation of noise standard deviation.
xd = wden(x,'minimaxi','s','sln',lev,'sym8');

% Plot signal.
subplot(616), plot(xd), axis([1 2048 -10 10]);
title('De-noised signal - Minimax');
```

```
% If many trials are necessary, it is better to perform
% decomposition once and threshold it many times:

% decomposition.
[c,l] = wavedec(x,lev,'sym8');

% threshold the decomposition structure [c,l].
xd = wden(c,l,'minimaxi','s','sln',lev,'sym8');
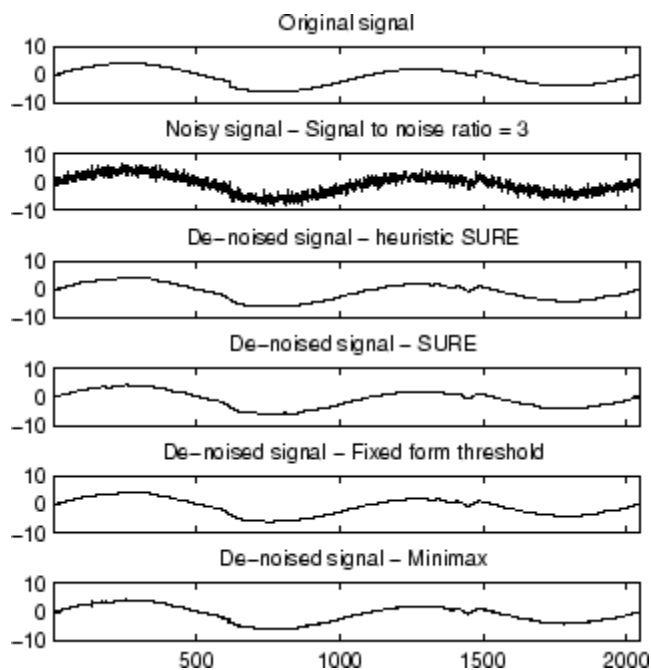
% Editing some graphical properties,
% the following figure is generated.
```

# wden

**References**     Antoniadis, A.; G. Oppenheim, Eds. (1995), *Wavelets and statistics*, 103, Lecture Notes in Statistics, Springer Verlag.

Donoho, D.L. (1993), "Progress in wavelet analysis and WVD: a ten minute tour," in *Progress in wavelet analysis and applications*, Y. Meyer, S. Roques, pp. 109–128. Frontières Ed.

Donoho, D.L.; I.M. Johnstone (1994), "Ideal spatial adaptation by wavelet shrinkage," *Biometrika*, Vol. 81, pp. 425–455.

Donoho, D.L. (1995), "De-noising by soft-thresholding," *IEEE Trans. on Inf. Theory*, 42 3, pp. 613– 627.

Donoho, D.L.; I.M. Johnstone, G. Kerkyacharian, D. Picard (1995), "Wavelet shrinkage: asymptotia," *Jour. Roy. Stat. Soc.*, *series B*, Vol. 57, No. 2, pp. 301–369.

**See Also**     thselect | wavedec | wdencmp | wfilters | wthresh

**Purpose**       De-noising or compression

**Syntax**        ```
[XC,CXC,LXC,PERF0,PERFL2] =
wdencmp('gbl',X,'wname',N,THR,SORH,
    KEEPAPP)
wdencmp('gbl',C,L,'wname',N,THR,SORH,KEEPAPP)
[XC,CXC,LXC,PERF0,PERFL2] =
wdencmp('lvd',X,'wname',N,THR,SORH)
[XC,CXC,LXC,PERF0,PERFL2] =
wdencmp('lvd',C,L,'wname',N,THR,SORH)
[XC,CXC,LXC,PERF0,PERFL2] =
wdencmp('lvd',X,'wname',N,THR,SORH)
[XC,CXC,LXC,PERF0,PERFL2] =
wdencmp('lvd',C,L,'wname',N,THR,SORH)
```

**Description**   wdencmp is a one- or two-dimensional de-noising and
compression-oriented function.

wdencmp performs a de-noising or compression process of a signal or
an image, using wavelets.

`[XC,CXC,LXC,PERF0,PERFL2] =
wdencmp('gbl',X,'wname',N,THR,SORH, KEEPAPP)` returns a
de-noised or compressed version XC of input signal X (one- or
two-dimensional) obtained by wavelet coefficients thresholding using
global positive threshold THR.

Additional output arguments [CXC,LXC] are the wavelet decomposition
structure of XC (see wavedec or wavedec2 for more information). PERF0
and PERFL2 are $L^2$-norm recovery and compression score in percentage.

PERFL2 = 100 * (vector-norm of CXC / vector-norm of C)$^2$ if [C,L] denotes
the wavelet decomposition structure of X.

If X is a one-dimensional signal and 'wname' an orthogonal wavelet,
PERFL2 is reduced to

$$\frac{100\|XC\|^2}{\|X\|^2}$$

Wavelet decomposition is performed at level N and '*wname*' is a string containing wavelet name (see wmaxlev and wfilters for more information). SORH ('s' or 'h') is for soft or hard thresholding (see wthresh for more information). If KEEPAPP = 1, approximation coefficients cannot be thresholded, otherwise it is possible.

wdencmp('gbl',C,L,'*wname*',N,THR,SORH,KEEPAPP) has the same output arguments, using the same options as above, but obtained directly from the input wavelet decomposition structure [C,L] of the signal to be de-noised or compressed, at level N and using '*wname*' wavelet.

For the one-dimensional case and 'lvd' option, [XC,CXC,LXC,PERF0,PERFL2] = wdencmp('lvd',X,'*wname*',N,THR,SORH) or [XC,CXC,LXC,PERF0,PERFL2] = wdencmp('lvd',C,L,'*wname*',N,THR,SORH) have the same output arguments, using the same options as above, but allowing level-dependent thresholds contained in vector THR (THR must be of length N). In addition, the approximation is kept. Note that, with respect to wden (automatic de-noising), wdencmp allows more flexibility and you can implement your own de-noising strategy.

For the two-dimensional case and 'lvd' option, [XC,CXC,LXC,PERF0,PERFL2] = wdencmp('lvd',X,'*wname*',N,THR,SORH) or [XC,CXC,LXC,PERF0,PERFL2] = wdencmp('lvd',C,L,'*wname*',N,THR,SORH).

THR must be a matrix 3 by N containing the level-dependent thresholds in the three orientations, horizontal, diagonal, and vertical.

Like denoising, the compression procedure contains three steps:

**1** Decomposition.

**2** Detail coefficient thresholding. For each level from 1 to N, a threshold is selected and hard thresholding is applied to the detail coefficients.

**3** Reconstruction.

The difference with the denoising procedure is found in step 2.

**Examples**

### Denoise Image Using Default Global Threshold

Denoise an image in additive white Gaussian noise using the Donoho-Johnstone universal threshold.

Load the image and add white Gaussian noise.

```
load sinsin;
Y = X + 18*randn(size(X));
```

Use ddencmp to obtain the threshold and denoise the image. Plot the original image, noisy image, and denoised result.

```
[thr,sorh,keepapp] = ddencmp('den','wv',Y);
xd = wdencmp('gbl',Y,'sym4',2,thr,sorh,keepapp);
subplot(221)
imagesc(X); title('Original Image');
subplot(222);
imagesc(Y); title('Noisy Image');
subplot(223)
imagesc(xd); title('Denoised Image');
```

### Denoise 1-D Signal Using Default Global Threshold

Denoise 1-D electricity consumption data using the Donoho-Johnstone global threshold.

Load the signal and select a segment for denoising.

```
load leleccum; indx = 2600:3100;
x = leleccum(indx);
```

1-531

# wdencmp

Use ddencmp to determine the default global threshold and denoise the signal. Plot the original and denoised signals.

```
[thr,sorh,keepapp] = ddencmp('den','wv',x);
xd = wdencmp('gbl',x,'db3',2,thr,sorh,keepapp);
subplot(211)
plot(x); title('Original Signal');
subplot(212)
plot(xd); title('Denoised Signal');
```

**References**  DeVore, R.A.; B. Jawerth, B.J. Lucier (1992), "Image compression through wavelet transform coding," *IEEE Trans. on Inf. Theory*, vol. 38, No 2, pp. 719–746.

Donoho, D.L. (1993), "Progress in wavelet analysis and WVD: a ten minute tour," in Progress in wavelet analysis and applications, Y. Meyer, S. Roques, pp. 109–128. Frontières Ed.

Donoho, D.L.; I.M. Johnstone (1994), "Ideal spatial adaptation by wavelet shrinkage," *Biometrika*, vol. 81, pp. 425–455.

Donoho, D.L.; I.M. Johnstone, G. Kerkyacharian, D. Picard (1995), "Wavelet shrinkage: asymptopia," *Jour. Roy. Stat. Soc.*, *series B*, vol. 57 no. 2, pp. 301–369.

Donoho, D.L.; I.M. Johnstone, "Ideal de-noising in an orthonormal basis chosen from a library of bases," C.R.A.S. Paris, t. 319, Ser. I, pp. 1317–1322.

Donoho, D.L. (1995), "De-noising by soft-thresholding," *IEEE Trans. on Inf. Theory*, 41, 3, pp. 613–627.

**See Also**  ddencmp | wavedec | wavedec2 | wbmpen | wcompress | wdcbm2 | wden | wpdencmp | wthresh

**Purpose**        Energy for 1-D wavelet or wavelet packet decomposition

**Syntax**         [Ea,Ed] = wenergy(C,L)
                   E = wenergy(*T*)

**Description**    For a one-dimensional wavelet decomposition [C,L] (see wavedec for
                   details), [Ea,Ed] = wenergy(C,L) returns Ea, which is the percentage
                   of energy corresponding to the approximation and Ed, which is the
                   vector containing the percentages of energy corresponding to the details.

                   For a wavelet packet tree *T* (see wptree, wpdec, wpdec2), E =
                   wenergy(*T*) returns a vector E, which contains the percentages of
                   energy corresponding to the terminal nodes of the tree *T*. In this case,
                   wenergy is a method of the wptree object *T*, which overloads the
                   previous wenergy function.

**Examples**       ```
                   % Example 1: 1-D wavelet decomposition
                   %-----------------------------------
                   load noisbump
                   [C,L] = wavedec(noisbump,4,'sym4');
                   [Ea,Ed] = wenergy(C,L)

                   Ea =

                      88.2860

                   Ed =

                       2.1560    1.2286    1.4664    6.8630

                   % Example 2: 1-D wavelet packet decomposition
                   %-------------------------------------------
                   load noisbump
                   T = wpdec(noisbump,3,'sym4');
                   E = wenergy(T)

                   E =
                   ```

```
95.0329    1.4664    0.6100    0.6408    0.5935    0.5445    0.5154
0.5965
```

**Purpose**      Energy for 2-D wavelet decomposition

**Syntax**       ```
[Ea,Eh,Ev,Ed] = wenergy2(C,S)
[Ea,EDetail] = wenergy2(C,S)
```

**Description**  For a two-dimensional wavelet decomposition [C,S] (see wavedec2 for details), [Ea,Eh,Ev,Ed] = wenergy2(C,S) returns Ea, which is the percentage of energy corresponding to the approximation, and vectors Eh, Ev, Ed, which contain the percentages of energy corresponding to the horizontal, vertical, and diagonal details, respectively.

[Ea,EDetail] = wenergy2(C,S) returns Ea, and EDetail, which is the sum of vectors Eh, Ev, and Ed.

**Examples**     ```
load detail
[C,S] = wavedec2(X,2,'sym4');
[Ea,Eh,Ev,Ed] = wenergy2(C,S)

Ea =
   89.3520

Eh =
    1.8748    2.7360

Ev =
    1.5860    2.6042

Ed =
    0.7539    1.0932

[Ea,EDetails] = wenergy2(C,S)

Ea =
   89.3520

EDetails =
    4.2147    6.4334
```

# wentropy

**Purpose**        Entropy (wavelet packet)

**Syntax**        E = wentropy(X,*T*,P)
E = wentropy(X,*T*)
E = wentropy(X,*T*,0)

**Description**    E = wentropy(X,*T*,P) returns the entropy E of the vector or matrix input X. In both cases, output E is a real number.

E = wentropy(X,*T*) is equivalent to E = wentropy(X,*T*,0).

*T* is a string containing the type of entropy and P is an optional parameter depending on the value of *T*.

| Entropy Type Name (*T*) | Parameter (P) | Comments |
|---|---|---|
| 'shannon' | | P is not used. |
| 'log energy' | | P is not used. |
| 'threshold' | 0    P | P is the threshold. |
| 'sure' | 0    P | P is the threshold. |
| 'norm' | 1    P | P is the power. |
| 'user' | string | P is a string containing the file name of your own entropy function, with a single input X. |
| FunName | No constraints on P | FunName is any other string except those used for the previous Entropy Type Names listed above.<br><br>FunName contains the file name of your own entropy function, with X as input and P as additional parameter to your entropy function. |

> **Note** The `'user'` option is historical and still kept for compatibility, but it is obsoleted by the last option described in the table above. The `FunName` option do the same as the `'user'` option and in addition gives the possibility to pass a parameter to your own entropy function.

Functionals verifying an additive-type property are well suited for efficient searching of binary-tree structures and the fundamental splitting property of the wavelet packets decomposition. Classical entropy-based criteria match these conditions and describe information-related properties for an accurate representation of a given signal. Entropy is a common concept in many fields, mainly in signal processing. The following example lists different entropy criteria. Many others are available and can be easily integrated. In the following expressions, $s$ is the signal and $(s_i)_i$ the coefficients of $s$ in an orthonormal basis.

The entropy E must be an additive cost function such that $E(0) = 0$ and

$$E(s) = \sum_i E(s_i)$$

- The (nonnormalized) Shannon entropy.

$$E1(s_i) = s_i^2 \log(s_i^2) \quad \text{so} \quad E1(s) = -\sum_i s_i^2 \log(s_i^2)$$

with the convention $0\log(0) = 0$.

- The concentration in $l^p$ norm entropy with $1 \le p$.

$$E2(s_i) = |s_i|^p \quad \text{so} \quad E2(s) = \sum_i |s_i|^p = \|s\|_p^p$$

- The "log energy" entropy.

$$E3(s_i) = \log(s_i^2) \text{ so } E3(s) = \sum_i \log(s_i^2)$$

  with the convention $log(0) = 0$.

- The threshold entropy.

  $E4(s_i) = 1$ if $|s_i| > p$ and 0 elsewhere so $E4(s) = \#\{i$ such that $|s_i| > p\}$ is the number of time instants when the signal is greater than a threshold $p$.

- The "SURE" entropy.

$$E5(s) = n - \#\{i \text{ such that } |s_i| \le p\} + \sum_i min(s_i^2, p^2)$$

  For more information, see the section "Wavelet Packets for Compression and De-Noising" of the User's Guide.

**Examples**

```
% The current extension mode is zero-padding (see dwtmode).

% Generate initial signal.
x = randn(1,200);

% Compute Shannon entropy of x.
e = wentropy(x,'shannon')

e =
    -142.7607

% Compute log energy entropy of x.
e = wentropy(x,'log energy')
e =
    -281.8975
```

```
% Compute threshold entropy of x
% with threshold equal to 0.2.
e = wentropy(x,'threshold',0.2)
e =
    162

% Compute Sure entropy of x
% with threshold equal to 3.
e = wentropy(x,'sure',3)
e =
 -0.6575

% Compute norm entropy of x with power equal to 1.1.
e = wentropy(x,'norm',1.1)
e =
 160.1583

% Compute user entropy of x with a user defined
% function: userent for example.
% This function must be a code file, with first line
% of the following form:
%
%     function e = userent(x)
%
% where x is a vector and e is a real number.
% Then a new entropy is defined and can be used typing:
%
% e = wentropy(x,'user','userent')
%
% or more directly
%
% e = wentropy(x,'userent')
```

**References**    Coifman, R.R.; M.V. Wickerhauser (1992), "Entropy-based Algorithms for best basis selection," *IEEE Trans. on Inf. Theory*, vol. 38, 2, pp. 713–718.

Donoho, D.L.; I.M. Johnstone, "Ideal de-noising in an orthonormal basis chosen from a library of bases," *C.R.A.S. Paris*, *Ser. I*, t. 319, pp. 1317–1322.

**Purpose**     Extend vector or matrix

**Syntax**

**Description**     The valid extension types (TYPE) are listed in the table below.

| TYPE | Description |
|------|-------------|
| 1, '1', '1d' or '1D' | 1-D extension |
| 2, '2', '2d' or '2D' | 2-D extension |
| 'ar' or 'addrow' | Add rows |
| 'ac' or 'addcol' | Add columns |

The valid extension modes (MODE) are listed in the table below.

| MODE | Description |
|------|-------------|
| 'zpd' | Zero extension |
| 'sp0' | Smooth extension of order 0 |
| 'spd' (or 'sp1') | Smooth extension of order 1 |
| 'sym' or 'symh' | Symmetric-padding (half-point): boundary value symmetric replication |
| 'symw' | Symmetric-padding (whole-point): boundary value symmetric replication |
| 'asym' or 'asymh' | Antisymmetric-padding (half-point): boundary value antisymmetric replication |
| 'asymw' | Antisymmetric-padding (whole-point): boundary value antisymmetric replication |

| MODE | Description |
|------|-------------|
| `'ppd'` | Periodized extension (1) |
| `'per'` | Periodized extension (2): |
|  | If the signal length is odd, wextend adds an extra sample, equal to the last value, on the right and performs extension using the `'ppd'` mode. Otherwise, `'per'` reduces to `'ppd'`. The same kind of rule stands for images. |

With TYPE = {1, '1', '1d' or '1D'}:

- LOC = 'l' (or 'u') for left (or up) extension.

- LOC = 'r' (or 'd') for right (or down) extension.

- LOC = 'b' for extension on both sides.

- LOC = 'n' null extension.

- The default is LOC = 'b'.

- L is the length of the extension.

With TYPE = {'ar', 'addrow'}:

- LOC is a 1D extension location.

- The default is LOC = 'b'.

- L is the number of rows to add.

With TYPE = {'ac', 'addcol'}:

- LOC is a 1D extension location.

- The default is LOC = 'b'.

- L is the number of columns to add.

With TYPE = {2, '2', '2d' or '2D'}:

- LOC = [LOCROW,LOCCOL] where LOCROW and LOCCOL are 1D extension locations or 'n' (none).

- The default is LOC = 'bb'.

- L = [LROW,LCOL] where LROW is the number of rows to add and LCOL is the number of columns to add.

For more information on symmetric extension modes see "References".

**Examples**
```
% Original signal.
x = [1 2 3]

x =

     1     2     3

% 1-D extension length.
l = 2;

% Zero-padding extensions 1-D.
xextzpd1 = wextend('1','zpd',x,l)
xextzpd1 =

     0     0     1     2     3     0     0

xextzpd2 = wextend('1D','zpd',x,l,'b')

xextzpd2 =

     0     0     1     2     3     0     0

% Symmetric extension 1-D.
xextsym = wextend('1D','sym',x,l)

xextsym =

     2     1     1     2     3     3     2
```

# wextend

```
% Periodic extension 1-D.
xextper = wextend('1D','per',x,l)

xextper =

    3    3    1    2    3    3    1    2

% Original image.
X = [1 2 3;4 5 6]

X =
    1    2    3
    4    5    6

% 2-D extension length.
l = 2;

% Zero-padding extension 2-D.
Xextzpd = wextend(2,'zpd',X,l)

Xextzpd =
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0
    0    0    1    2    3    0    0
    0    0    4    5    6    0    0
    0    0    0    0    0    0    0
    0    0    0    0    0    0    0

% Symmetric extension 2-D.
Xextsym = wextend('2D','sym',X,l)

Xextsym =
    5    4    4    5    6    6    5
    2    1    1    2    3    3    2
    2    1    1    2    3    3    2
    5    4    4    5    6    6    5
```

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| 5 | 4 | 4 | 5 | 6 | 6 | 5 |
| 2 | 1 | 1 | 2 | 3 | 3 | 2 |

**References**     Strang, G.; T. Nguyen (1996), *Wavelets and filter banks*, Wellesley-Cambridge Press.

# wfbm

| | |
|---|---|
| **Purpose** | Fractional Brownian motion synthesis |

**Syntax**
```
FBM = wfbm(H,L)
FBM = wfbm(H,L,'plot')
FBM = wfbm(H,L,NS,W)
FBM = wfbm(H,L,W,NS)
wfbm(H,L,'plot',NS)
wfbm(H,L,'plot',W)
wfbm(H,L,'plot',NS,W)
wfbm(H,L,'plot',W,NS)
```

**Description**    FBM = wfbm(H,L) returns a fractional Brownian motion signal FBM of the Hurst parameter H (0 < H < 1) and length L, following the algorithm proposed by Abry and Sellan.

FBM = wfbm(H,L,'plot') generates and plots the FBM signal.

FBM = wfbm(H,L,NS,W) or FBM = wfbm(H,L,W,NS) returns the FBM using NS reconstruction steps and the sufficiently regular orthogonal wavelet W.

wfbm(H,L,'plot',NS) or wfbm(H,L,'plot',W) or
wfbm(H,L,'plot',NS,W) or wfbm(H,L,'plot',W,NS) generates and plots the FBM signal.

wfbm(H,L) is equivalent to WFBM(H,L,6,'db10').

wfbm(H,L,NS) is equivalent to WFBM(H,L,NS,'db10').

wfbm(H,L,W) is equivalent to WFBM(H,L,W,6).

A fractional Brownian motion (fBm) is a continuous-time Gaussian process depending on the Hurst parameter 0 < H < 1. It generalizes the ordinary Brownian motion corresponding to H = 0.5 and whose derivative is the white noise. The fBm is self-similar in distribution and the variance of the increments is given by

```
Var(fBm(t)-fBm(s)) = v |t-s|^(2H)
```

where v is a positive constant.

**Examples**     According to the value of H, the fBm exhibits for H > 0.5, long-range
dependence and for H < 0.5, short or intermediate dependence. This
example shows each situation using the wfbm file, which generates a
sample path of this process.

```
% Generate fBm for H = 0.3 and H = 0.7

% Set the parameter H and the sample length
H = 0.3; lg = 1000;
% Generate and plot wavelet-based fBm for H = 0.3
fBm03 = wfbm(H,lg,'plot');

H = 0.7;
% Generate and plot wavelet-based fBm for H = 0.7
fBm07 = wfbm(H,lg,'plot');

% The last step is equivalent to
% Define wavelet and level of decomposition
% w = ' db10'; ns = 6;
% Generate
% fBm07 = wfbm(H,lg,'plot',w,ns);
```

fBm07 clearly exhibits a stronger low-frequency component and has,
locally, less irregular behavior.

**Algorithms**     Starting from the expression of the fBm process as a fractional integral
of the white noise process, the idea of the algorithm is to build a
biorthogonal wavelet depending on a given orthogonal one and adapted
to the parameter H.

Then the generated sample path is obtained by the reconstruction
using the new wavelet starting from a wavelet decomposition at a given
level designed as follows: details coefficients are independent random
Gaussian realizations and approximation coefficients come from a
fractional ARIMA process.

This method was first proposed by Meyer and Sellan and
implementation issues were examined by Abry and Sellan.

Nevertheless, the samples generated following this original scheme exhibit too many high-frequency components. To circumvent this undesirable behavior Bardet et al. propose downsampling the obtained sample by a factor 10.

Two internal parameters `delta = 10` (the downsampling factor) and a threshold `prec = 1E-4`, to evaluate series by truncated sums, can be modified by the user for extreme values of `H`.

A complete overview of long-range dependence process generators is available in Bardet et al.

**References**    Abry, P.; F. Sellan (1996), "The wavelet-based synthesis for the fractional Brownian motion proposed by F. Sellan and Y. Meyer: Remarks and fast implementation," *Appl. and Comp. Harmonic Anal.*, 3(4), pp. 377–383.

Bardet, J.-M.; G. Lang, G. Oppenheim, A. Philippe, S. Stoev, M.S. Taqqu (2003), "Generators of long-range dependence processes: a survey," Theory and applications of long-range dependence, Birkhäuser, pp. 579–623.

**See Also**    wfbmesti

**Purpose**      Parameter estimation of fractional Brownian motion

**Syntax**       HEST = wfbmesti(X)

**Description**  HEST = wfbmesti(X) returns a row vector HEST which contains three
estimates of the fractal index H of the signal X supposed to come from
a fractional Brownian motion of parameter H.

The two first estimates are based on second order discrete derivative,
the second one is wavelet-based.

The third estimate is based on the linear regression in loglog plot, of
the variance of detail versus level.

A fractional Brownian motion (fBm) is a continuous-time Gaussian
process depending on the so-called Hurst parameter 0 < H < 1. It
generalizes the ordinary Brownian motion corresponding to H = 0.5
and whose derivative is the white noise. The fBm is self-similar in
distribution and the variance of the increments is given by

Var(fBm(t)-fBm(s)) = v |t-s|^(2H)

where v is a positive constant.

This special form of the variance of the increments suggests various
ways to estimate the parameter H. One can find in Bardet et al. a
survey of such methods. The wfbmesti file provides three different
estimates. The first one, due to Istas and Lang, is based on the discrete
second-order derivative. The second one is a wavelet-based adaptation
and has similar properties. The third one, proposed by Flandrin,
estimates H using the slope of the loglog plot of the detail variance
versus the level. A more recent extension can be found in Abry et al.

**Examples**     This example shows a statistical comparison of the three estimators by
a short Monte-Carlo study.

% Initialize the randn generator

% Set parameter H to 0.6 and sample length

```
H = 0.6; lg = 10000;
% Generate 100 wavelet-based fBm realizations for H = 0.6
% and compute the three estimates for each of them
n = 100; Hest = zeros(n,3);
for i = 1:n
 fBm06 = wfbm(H,lg);
 Hest(i,:) = wfbmesti(fBm06);
end

% Compare empirical distributions
subplot(311), hist(Hest(:,1));
title('Discrete second derivative estimator DSOD')
subplot(312), hist(Hest(:,2));
title('Wavelet version of DSOD')
subplot(313), hist(Hest(:,3));
title('Wavelet details regression estimator')
xlabel('True value of the parameter H = 0.6')
```

For these experimental conditions, the two first methods give similar results with smaller dispersion than the third one. The third one is clearly slightly biased and has greater dispersion.

These experimental results depend on H and on the various experimental conditions. For a complete study, see Bardet et al.

**References**

Abry, P.; P. Flandrin, M.S. Taqqu, D. Veitch (2003), "Self-similarity and long-range dependence through the wavelet lens," Theory and applications of long-range dependence, Birkhäuser, pp. 527–556.

Bardet, J.-M.; G. Lang, G. Oppenheim, A. Philippe, S. Stoev, M.S. Taqqu (2003), "Semi-parametric estimation of the long-range dependence parameter: a survey," Theory and applications of long-range dependence, Birkhäuser, pp. 557–577.

Flandrin, P. (1992), "Wavelet analysis and synthesis of fractional Brownian motion," *IEEE Trans. on Inf. Th.*, 38, pp. 910–917.

Istas, J.; G. Lang (1994), "Quadratic variations and estimation of the local Hölder index of a Gaussian process," *Ann. Inst. Poincaré*, 33, pp. 407–436.

**See Also**     wfbm

# wfilters

**Purpose**     Wavelet filters

**Syntax**      [Lo_D,Hi_D,Lo_R,Hi_R] = wfilters('*wname*')
                [F1,F2] = wfilters('*wname*','*type*')

**Description**  [Lo_D,Hi_D,Lo_R,Hi_R] = wfilters('*wname*') computes four filters
                associated with the orthogonal or biorthogonal wavelet named in the
                string '*wname*'.

                The four output filters are

                • Lo_D, the decomposition low-pass filter

                • Hi_D, the decomposition high-pass filter

                • Lo_R, the reconstruction low-pass filter

                • Hi_R, the reconstruction high-pass filter

                Available orthogonal or biorthogonal wavelet names '*wname*' are listed
                in the table below.

| Wavelet Families | Wavelets |
|---|---|
| Daubechies | 'db1' or 'haar', 'db2', ... ,'db10', ... , 'db45' |
| Coiflets | 'coif1', ... , 'coif5' |
| Symlets | 'sym2', ... , 'sym8', ... ,'sym45' |
| Discrete Meyer | 'dmey' |
| Biorthogonal | 'bior1.1', 'bior1.3', 'bior1.5' 'bior2.2', 'bior2.4', 'bior2.6', 'bior2.8' 'bior3.1', 'bior3.3', 'bior3.5', 'bior3.7' 'bior3.9', 'bior4.4', 'bior5.5', 'bior6.8' |
| Reverse Biorthogonal | 'rbio1.1', 'rbio1.3', 'rbio1.5' 'rbio2.2', 'rbio2.4', 'rbio2.6', 'rbio2.8' 'rbio3.1', 'rbio3.3', 'rbio3.5', 'rbio3.7' 'rbio3.9', 'rbio4.4', 'rbio5.5', 'rbio6.8' |

[F1,F2] = wfilters('*wname*','*type*') returns the following filters:

| | | |
|---|---|---|
| Lo_D and Hi_D | (Decomposition filters) | If 'type' = 'd' |
| Lo_R and Hi_R | (Reconstruction filters) | If 'type' = 'r' |
| Lo_D and Lo_R | (Low-pass filters) | If 'type' = 'l' |
| Hi_D and Hi_R | (High-pass filters) | If 'type' = 'h' |

**Examples**

```
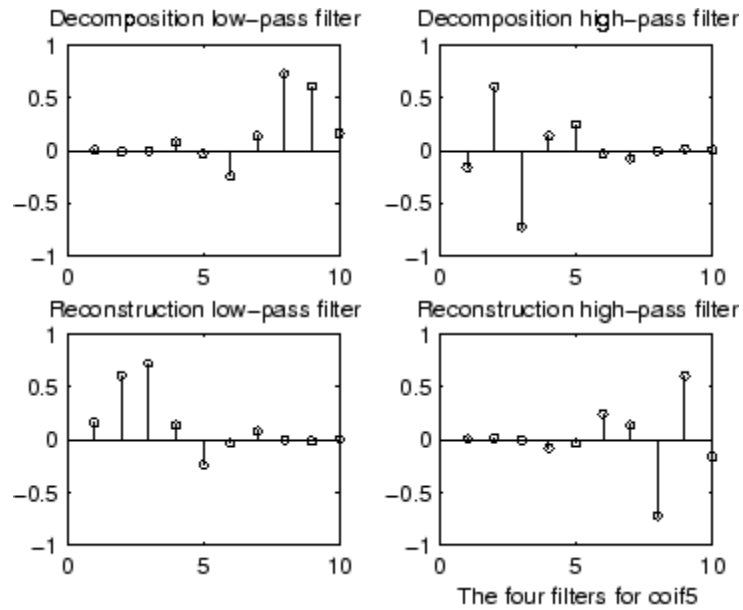% Set wavelet name.
wname = 'db5';

% Compute the four filters associated with wavelet name given
% by the input string wname.
[Lo_D,Hi_D,Lo_R,Hi_R] = wfilters(wname);
subplot(221); stem(Lo_D);
title('Decomposition low-pass filter');
subplot(222); stem(Hi_D);
title('Decomposition high-pass filter');
subplot(223); stem(Lo_R);
title('Reconstruction low-pass filter');
subplot(224); stem(Hi_R);
title('Reconstruction high-pass filter');
xlabel('The four filters for db5')

% Editing some graphical properties,
% the following figure is generated.
```

The four filters for coif5

**References**  Daubechies, I. (1992), *Ten lectures on wavelets*, CBMS-NSF conference series in applied mathematics. SIAM Ed.

Mallat, S. (1989), "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Pattern Anal. and Machine Intell.*, vol. 11, no. 7, pp. 674–693.

**See Also**  biorfilt | orthfilt | waveinfo

**Purpose**        Fusion of two images

**Syntax**         XFUS = wfusimg(X1,X2,WNAME,LEVEL,AFUSMETH,DFUSMETH)
                   [XFUS,TXFUS,TX1,TX2] = wfusimg(X1,X2,WNAME,LEVEL,AFUSMETH,DFUSMETH)
                   wfusimg(X1,X2,WNAME,LEVEL,AFUSMETH,DFUSMETH,FLAGPLOT)

**Description**    The principle of image fusion using wavelets is to merge the wavelet
                   decompositions of the two original images using fusion methods applied
                   to approximations coefficients and details coefficients (see Zeeuw and
                   Misiti et al.).

                   XFUS = wfusimg(X1,X2,WNAME,LEVEL,AFUSMETH,DFUSMETH) returns
                   the fused image XFUS obtained by fusion of the two original images
                   X1 and X2. Each fusion method, defined by AFUSMETH and DFUSMETH,
                   merges in a specific way detailed below, the decompositions of X1 and
                   X2, at level LEVEL and using wavelet WNAME.

                   AFUSMETH and DFUSMETH define the fusion method for approximations
                   and details, respectively.

                   [XFUS,TXFUS,TX1,TX2] =
                   wfusimg(X1,X2,WNAME,LEVEL,AFUSMETH,DFUSMETH) returns, in
                   addition to matrix XFUS, three objects of the class WDECTREE
                   associated with XFUS, X1, and X2 respectively (see @WDECTREE).
                   wfusimg(X1,X2,WNAME,LEVEL,AFUSMETH,DFUSMETH,FLAGPLOT) also
                   plots the objects TXFUS, TX1, and TX2.

                   Fusmeth denotes AFUSMETH or DFUSMETH. Available fusion methods are

                   • Simple — Fusmeth can be 'max', 'min', 'mean', 'img1', 'img2' or
                     'rand', which merges the two approximations or details structures
                     obtained from X1 and X2 elementwise by taking the maximum, the
                     minimum, the mean, the first element, the second element, or a
                     randomly chosen element

                   • Parameter-dependent — Fusmeth is of the following form

                     Fusmeth = struct('name',nameMETH,'param',paramMETH)

                   where nameMETH can be

| | |
|---|---|
| `'linear'` | |
| `'UD_fusion'` | Up-down fusion |
| `'DU_fusion'` | Down-up fusion |
| `'RL_fusion'` | Right-left fusion |
| `'UserDEF'` | User-defined fusion |

For the description of these options and the `paramMETH` parameter, see `wfusmat`.

**Tips**
X1 and X2 must be of same size (see `wextend` to resize images) and represent indexed images or truecolor images, which are m-by-n matrices or m-by-n-by-3 arrays, respectively.

For more information on image formats, see the `image` and `imfinfo` reference pages.

**Examples**
The following three examples examine the process of image fusion

- The first example merges two different images leading to a new image

- The second example restores an image from two fuzzy versions of an original image.

- The third example shows how to make an image fusion using a user defined fusion method.

```
% Example 1: Fusion of two different images

% Load two original images: a mask and a bust
load mask; X1 = X;
load bust; X2 = X;

% Merge the two images from wavelet decompositions at level 5
% using db2 by taking two different fusion methods

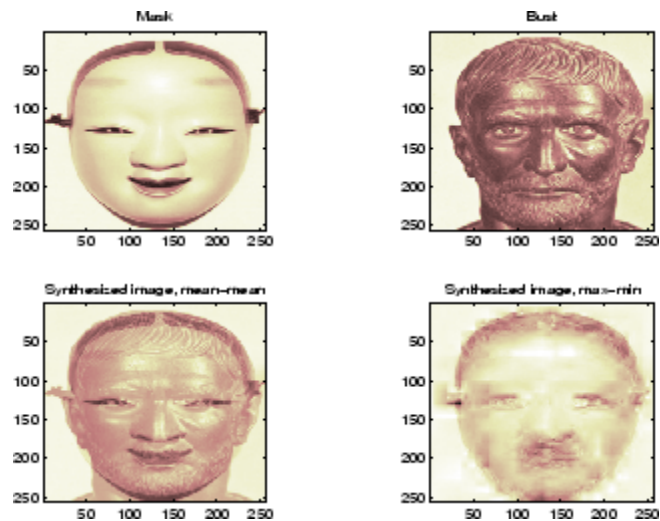% fusion by taking the mean for both approximations and details
```

```
XFUSmean = wfusimg(X1,X2,'db2',5,'mean','mean');

% fusion by taking the maximum for approximations and the
% minimum for the details
XFUSmaxmin = wfusimg(X1,X2,'db2',5,'max','min');

% Plot original and synthesized images
colormap(map);
subplot(221), image(X1), axis square, title('Mask')
subplot(222), image(X2), axis square, title('Bust')
subplot(223), image(XFUSmean), axis square,
title('Synthesized image, mean-mean')
subplot(224), image(XFUSmaxmin), axis square,
title('Synthesized image, max-min')
```



```
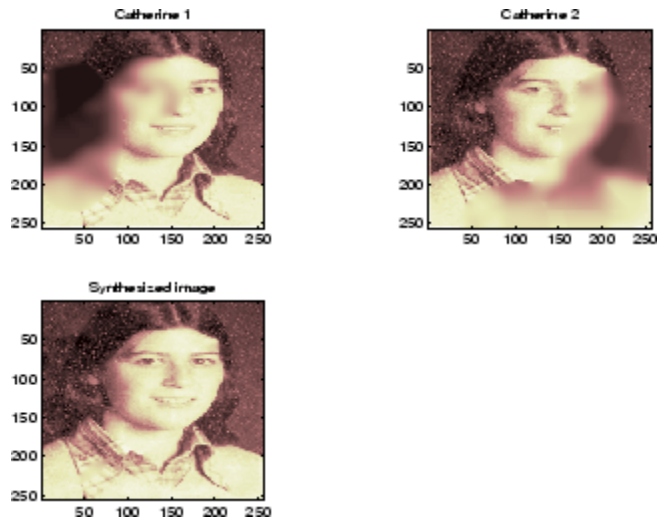% Example 2: Restoration by fusion of fuzzy images

% Load two fuzzy versions of an original image
load cathe_1; X1 = X;
load cathe_2; X2 = X;
```

```
% Merge the two images from wavelet decompositions at level 5
% using sym4 by taking the maximum of absolute value of the
% coefficients for both approximations and details
XFUS = wfusimg(X1,X2,'sym4',5,'max','max');

% Plot original and synthesized images
colormap(map);
subplot(221), image(X1), axis square,
title('Catherine 1')
subplot(222), image(X2), axis square,
title('Catherine 2')
subplot(223), image(XFUS), axis square,
title('Synthesized image')
```



```
% The synthesized image is a restored version of good
% quality of the common underlying original image.

% Example 3: Fusion using a user defined fusion method.
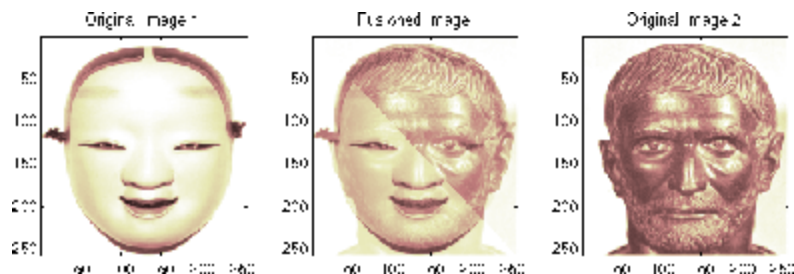% This example calls a user fusion method defined by the
```

```
% file myfus_FUN.m which is listed below at the end of
% the example.

% load two images of the same size.
load mask; A = X;
load bust; B = X;

% Define the fusion method and call the fusion function
Fus_Method = struct('name','userDEF','param','myfus_FUN');
C = wfusmat(A,B,Fus_Method);

figure;
colormap(pink(220))
subplot(1,3,1), image(A), title('Original Image 1'), axis square
subplot(1,3,2), image(C), title('Fusioned Image'), axis square
subplot(1,3,3), image(B), title('Original Image 2'), axis square
```



```
%*****************************
% User defined fusion method. *
%*****************************
 function C = myfus_FUN(A,B)

D = logical(triu(ones(size(A)))); t = 0.3;
C = A;
C(D)  = t*A(D)+(1-t)*B(D);
C(~D) = t*B(~D)+(1-t)*A(~D);
```

# wfusimg

**References**    Zeeuw, P.M. (1998), "Wavelet and image fusion," CWI, Amsterdam, March 1998, `http:/www.cwi.nl/~pauldz/`

Misiti, M.; Y. Misiti, G. Oppenheim, J.-M. Poggi (2003), "Les ondelettes et leurs applications," Hermes.

**See Also**    `wfusmat | wextend`

**Purpose**　　　Fusion of two matrices or arrayz

**Syntax**　　　`C = wfusmat(A,B,METHOD)`

**Description**　`C = wfusmat(A,B,METHOD)` returns the fused matrix `C` obtained from the matrices `A` and `B` using the fusion method defined by `METHOD`.

The matrices `A` and `B` must be of the same size. The output matrix `C` is of the same size as `A` and `B`.

Available fusion methods are

- Simple, where METHOD is

  - `'max'` : `D = abs(A) ≥ abs(B) ; C = A(D) + B(~D)`

  - `'min'` : `D = abs(A) ≤ abs(B) ; C = A(D) + B(~D)`

  - `'mean'` : `C = (A+B) / 2 ; D = ones(size(A))`

  - `'rand'` : C = A(D) + B(~D); D is a Boolean random matrix

  - `'img1'` : C = A

  - `'img2'` : C = B

- Parameter-dependent, where METHOD is of the following form:

  `METHOD = struct('name',nameMETH,'param',paramMETH)`

  where nameMETH can be

  - `'linear'` : `C = A*paramMETH + B*(1-paramMETH)`,

    where $0 ≤ paramMETH ≤ 1$

  - `'UD_fusion'`: Up-down fusion, with paramMETH $≥ 0$

    ```
    x = linspace(0,1,size(A,1));
    P = x.^paramMETH;
    ```

  Then each row of C is computed with

  `C(i,:) = A(i,:)*(1-P(i)) + B(i,:)*P(i);`

So `C(1,:) = A(1,:)` and `C(end,:) = A(end,:)`

- `'DU_fusion'`: Down-up fusion

- `'LR_fusion'`: Left-right fusion (columnwise fusion)

- `'RL_fusion'`: Right-left fusion (columnwise fusion)

- `'UserDEF'`: User-defined fusion, `paramMETH` is a string `'userFUNCTION'` containing a function name such that `C = userFUNCTION(A,B)`.

In addition, `[C,D] = wfusmat(A,B,METHOD)` returns the Boolean matrix `D` when defined, or an empty matrix otherwise.

**Purpose**      Keep part of vector or matrix

**Syntax**
```
Y = wkeep(X,L,OPT)
Y = wkeep(X,L,FIRST)
Y = wkeep(X,L)
Y = wkeep(X,L,'c')
Y = wkeep(X,S,[FIRSTR FIRSTC])
```

**Description**    wkeep is a general utility.

For a vector, Y = wkeep(X,L,OPT) extracts the vector Y from the vector X. The length of Y is L.

If OPT = 'c' ('l', 'r', respectively), Y is the central (left, right, respectively) part of X.

Y = wkeep(X,L,FIRST) returns the vector X(FIRST:FIRST+L-1).

Y = wkeep(X,L) is equivalent to Y = wkeep(X,L,'c').

For a matrix, Y = wkeep(X,S) extracts the central part of the matrix X. The size of Y is S.

Y = wkeep(X,S,[FIRSTR FIRSTC]) extracts the submatrix of matrix X, of size S and starting from X(FIRSTR,FIRSTC).

**Examples**
```
% For a vector.
x = 1:10;
y = wkeep(x,6,'c')
y =
    3    4    5    6    7    8

y = wkeep(x,6)
y =
    3    4    5    6    7    8

y = wkeep(x,7,'c')
y =
    2    3    4    5    6    7    8
```

```
y = wkeep(x,6,'l')
y =
    1     2     3     4     5     6

y = wkeep(x,6,'r')
y =
    5     6     7     8     9     10

% For a matrix.
x = magic(5)
x =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9

y = wkeep(x,[3 2])
y =
     5     7
     6    13
    12    19
```

**Purpose**      Maximum wavelet decomposition level

**Syntax**       L = wmaxlev(S,'*wname*')

**Description**  wmaxlev is a one- or two-dimensional wavelet or wavelet packets oriented function.

wmaxlev can help you avoid unreasonable maximum level values. L = wmaxlev(S,'*wname*') returns the maximum level decomposition of signal or image of size S using the wavelet named in the string '*wname*' (see wfilters for more information).

wmaxlev gives the maximum allowed level decomposition, but in general, a smaller value is taken.

Usual values are 5 for the one-dimensional case, and 3 for the two-dimensional case.

**Examples**    ```
% For a 1-D signal.
s = 2^10;
w = 'db1';

% Compute maximum level decomposition.
% The rule is the last level for which at least
% one coefficient is correct.
l = wmaxlev(s,w)

l =
    10

% Change wavelet.
w = 'db7';

% Compute maximum level decomposition.
l = wmaxlev(s,w)

l =
    6
```

```
% For a 2-D signal.
s = [2^9 2^7];
w = 'db1';

% Compute maximum level decomposition.
l = wmaxlev(s,w)

l =
    7

% which is the same as:
l = wmaxlev(min(s),w)

l =
    7

% Change wavelet.
w = 'db7';

% Compute maximum level decomposition.
l = wmaxlev(s,w)

l =
    3
```

**See Also**    wavedec | wavedec2 | wpdec | wpdec2

**Purpose**       Matching pursuit

**Syntax**        
```
YFIT = wmpalg(MPALG,Y,MPDICT)
[YFIT,R] = wmpalg(...)
[YFIT,R,COEFF] = wmpalg(...)
[YFIT,R,COEFF,IOPT] = wmpalg(...)
[YFIT,R,COEFF,IOPT,QUAL] = wmpalg(...)
[YFIT,R,COEFF,IOPT,QUAL,X] = wmpalg(...)
[YFIT,R,COEFF,IOPT,QUAL,X] = wmpalg(...,Name,Value)
```

**Description**   YFIT = wmpalg(MPALG,Y,MPDICT) returns an adaptive greedy approximation, YFIT, of the input signal, Y, in the dictionary, MPDICT. The adaptive greedy approximation uses the matching pursuit algorithm, MPALG. The dictionary, MPDICT, is typically an overcomplete set of vectors constructed using wmpdictionary.

[YFIT,R] = wmpalg(...) returns the residual, R, which is the difference vector between Y and YFIT at the termination of the matching pursuit.

[YFIT,R,COEFF] = wmpalg(...) returns the expansion coefficients, COEFF. The number of expansion coefficients depends on the number of iterations in the matching pursuit.

[YFIT,R,COEFF,IOPT] = wmpalg(...) returns the column indices of the retained atoms, IOPT. The length of IOPT equals the length of COEFF and is determined by the number of iterations in the matching pursuit.

[YFIT,R,COEFF,IOPT,QUAL] = wmpalg(...) returns the proportion of retained signal energy, QUAL, for each iteration of the matching pursuit. QUAL is the ratio of the $\ell^2$ squared norm of the expansion coefficient vector, COEFF, to the $\ell^2$ squared norm of the input signal, Y.

[YFIT,R,COEFF,IOPT,QUAL,X] = wmpalg(...) returns the normalized dictionary, X. X contains the unit vectors in the $\ell^2$ norm corresponding to the columns of MPDICT.

# wmpalg

[YFIT,R,COEFF,IOPT,QUAL,X] = wmpalg(...,Name,Value) returns an adaptive greedy approximation with additional options specified by one or more Name,Value pair arguments.

**Input Arguments**

**MPALG**

Matching pursuit algorithm as a string. Valid entries are:

- 'BMP' — Basic matching pursuit
- 'OMP' — Orthogonal matching pursuit
- 'WMP' — Weak orthogonal matching pursuit

See "Matching Pursuit Algorithms".

   **Default:** 'BMP'

**MPDICT**

Matching pursuit dictionary. MPDICT is a N-by-P matrix where N is equal to the length of the input signal, Y. You can construct MPDICT using wmpdictionary. In matching pursuit, MPDICT is commonly a frame, or overcomplete set of vectors. You may use the Name-Value pair 'lstcpt' to specify a dictionary instead of using MPDICT. If you specify a value for 'lstcpt', wmpalg calls wmpdictionary.

**Y**

Signal for matching pursuit. Y is 1-D, real-valued row or column vector. The row dimension of MPDICT must match the length of Y.

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

**'itermax'**

Positive integer fixing the maximum number of iterations of the matching pursuit algorithm. If you do not specify a `'maxerr'` value, the number of expansion coefficients, `COEFF`, the number of dictionary vector indices, `IOPT`, and the length of the `QUAL` vector equal the value of `'itermax'`.

> **Default:** 25

**'lstcpt'**

A cell array of cell arrays with valid subdictionaries. This name-value pair is only valid if you do not input a dictionary in `MPDICT`. Each cell array describes one subdictionary. Valid subdictionaries are:

- A valid Wavelet Toolbox orthogonal or biorthogonal wavelet family short name with the number of vanishing moments and an optional decomposition level and extension mode. For example, `{'sym4',5}` denotes the Daubechies least-asymmetric wavelet with 4 vanishing moments at level 5 and the default extension mode `'per'`. If you do not specify the optional number level and extension mode, the decomposition level defaults to 5 and the extension mode to `'per'`.

- A valid Wavelet Toolbox orthogonal or biorthogonal wavelet family short name preceded by `wp` with the number of vanishing moments and an optional decomposition level and extension mode. For example, `{'wpsym4',5}` denotes the Daubechies least-asymmetric wavelet packet with 4 vanishing moments at level 5. If you do not specify the optional number level and extension mode, the decomposition level defaults to 5 and the extension mode to `'per'`.

- `'dct'` Discrete cosine transform-II basis. The DCT-II orthonormal basis is:

$$
\phi_k(n) = \begin{cases} \dfrac{1}{\sqrt{N}} & k = 0 \\ \sqrt{\dfrac{2}{N}} \cos(\tfrac{\pi}{N}(n + \tfrac{1}{2})k) & k = 1, 2, \ldots, N-1 \end{cases}
$$

- `'sin'` Sine subdictionary. The sine subdictionary is:

$$\phi_k(t) = \sin(2\pi kt) \quad k = 1, 2, \ldots, \left\lceil \frac{N}{2} \right\rceil \quad 0 \le t \le 1$$

- `'cos'` Cosine subdictionary. The cosine subdictionary is

$$\phi_k(t) = \cos(2\pi kt) \quad k = 1, 2, \ldots, \left\lceil \frac{N}{2} \right\rceil \quad 0 \le t \le 1$$

- `'poly'` Polynomial subdictionary. The polynomial subdictionary is:

$$p_n(t) = t^{n-1} \quad n = 1, 2, \ldots 20 \qquad 0 \le t \le 1$$

- `'RnIdent'` The shifted Kronecker delta subdictionary. The shifted Kronecker delta subdictionary is:

$$\phi_k(n) = \delta(n - k) \quad k = 0, 1, \ldots N$$

If you use the `'lstcpt'` name-value pair to generate your dictionary, you can use the additional `'addbeg'` and `'addend'` name-value pairs to append and addend dictionary atoms. See `wmpdictionary` for details.

**'maxerr'**

Cell array containing the name of the norm and the maximum relative error in the norm expressed as a percentage. Valid norms are `'L1'`, `'L2'`, and `'Linf'`. The relative error expressed as a percentage is

$$100 \frac{||R||}{||Y||}$$

where $R$ is the residual at each iteration and $Y$ is the input signal. For example, `{'L1',10}` sets maximum acceptable ratio of the L1 norms of the residual to the input signal to 0.10.

If you specify `'maxerr'`, the matching pursuit terminates when the first of the following conditions is satisfied:

- The number of iterations reaches the minimum of the length of the input signal, Y, or 500:
  `min(length(Y),500)`

- The relative error falls below the percentage you specify with the `'maxerr'` name-value pair.

### 'stepplot'

Number of iterations between successive plots. `'stepplot'` requires a positive integer. This name-value pair is only valid when `'typeplot'` is 2 or 3 (`'movie'` or `'stepwise'`).

### 'typeplot'

Type of plot to produce during the progression of matching pursuit. Valid entries for `'typeplot'` are: 0 or `'none'`, 1 or `'one'`, 2 or `'movie'`, 3 or `'stepwise'`. When `'typeplot'` is `'movie'` or `'stepwise'`, the plot updates based on the value of `'stepplot'`.

> **Default:** 0 or `'none'`

### 'wmpcfs'

Optimality factor for weak orthogonal matching pursuit. The optimality factor is a real number in the interval (0,1]. This name-value pair is only valid when MPALG is `'WMP'`.

> **Default:** 0.6

**Output Arguments**

### YFIT

Adaptive greedy approximation of the input signal, Y, in the dictionary

### R

Residual after matching pursuit terminates

### COEFF

Expansion coefficients in the dictionary. The selected dictionary atoms weighted by the expansion coefficients yield the approximated signal, YFIT.

### IOPT

Column indices of the selected dictionary atoms. Using the column indices in IOPT with the expansion coefficients in COEFF, you can form the approximated signal, YFIT.

### QUAL

Proportion of retained signal energy for each iteration in the matching pursuit. QUAL is a vector with each element equal to

$$\frac{||\alpha_k||_2^2}{||Y||_2^2}$$

where $a_k$ is the vector of expansion coefficients after the $k$-th iteration.

### X

The normalized matching pursuit dictionary. X is an N-by-P matrix where N is the length of the input signal, Y. The columns of X have unit norm.

**Examples**

### Adaptive Approximation using Orthogonal Matching Pursuit

Approximate the cuspamax signal with the dictionary using orthogonal matching pursuit.

Use a dictionary consisting of sym4 wavelet packets and the DCT-II basis.

```
load cuspamax;
mpdict = wmpdictionary(length(cuspamax),'LstCpt',{{'wpsym4',2},'dct'});
yfit = wmpalg('OMP',cuspamax,mpdict);
plot(cuspamax,'k'); hold on;
plot(yfit,'linewidth',2); legend('Original Signal','Matching Pursuit');
```

### Return Residual, Expansion Coefficients, Selected Atoms, and Approximation Quality

Obtain the expansion coefficients in the dictionary, the column indices of the selected dictionary atoms, and the proportion of retained signal energy.

Create a dictionary consisting of sym4 wavelet packets and the DCT-II basis. Approximate the cuspamax signal with the dictionary using orthogonal matching pursuit.

```
load cuspamax;
mpdict = wmpdictionary(length(cuspamax),'LstCpt',{{'wpsym4',2},'dct'})
[yfit,r,coeff,iopt,qual] = wmpalg('OMP',cuspamax,mpdict);
```

### Specify the Maximum Number of Iterations

This example shows how to set the maximum number of iterations of the orthogonal matching pursuit to 50.

```
load cuspamax;
lstcpt = {{'wpsym4',1},{'wpsym4',2},'dct'};
mpdict = wmpdictionary(length(cuspamax),'LstCpt',lstcpt);
[yfit,r,coeff,iopt,qual] = wmpalg('OMP',cuspamax,mpdict,'itermax',50);
```

### Stepwise Plot of Weak Orthogonal Matching Pursuit

This example shows how to allow for a suboptimal choice in the update of the orthogonal matching pursuit.

Relax the requirement to be 0.8 times the optimal assignment. Plot the results stepwise and update the plot every 5 iterations.

```
load cuspamax;
lstcpt = {{'wpsym4',1},{'wpsym4',2},'dct'};
mpdict = wmpdictionary(length(cuspamax),'LstCpt',lstcpt);
[yfit,r,coeff,iopt,qual] = wmpalg('WMP',cuspamax,mpdict,'wmpcfs',0.8,.
    'typeplot','stepwise','stepplot',5);
```

### Matching Pursuit of Electricity Consumption Data

Obtain a matching pursuit of electricity consumption measured every minute over a 24-hour period.

Load and plot data. The data shows electricity consumption sampled every minute over a 24-hour period. Because the data is centered, the actual usage values are not interpretable.

```
load elec35_nor;
y = signals(32,:);
plot(y); xlabel('Minutes'); ylabel('Usage');
set(gca,'xlim',[1 1440]);
```

Construct a dictionary for matching pursuit consisting of the Daubechies' extremal–phase wavelet with 2 vanishing moments at level 2, the Daubechies' least-asymmetric wavelet with 4 vanishing moments at levels 1 and 4, the discrete cosine transform-II basis, and the sine basis.

```
dictionary = {{'db4',2},'dct','sin',{'sym4',1},{'sym4',4}};
[mpdict,nbvect] = wmpdictionary(length(y),'lstcpt',dictionary);
```

Implement orthogonal matching pursuit to obtain a signal approximation in the dictionary. Use 35 iterations. Plot the result.

```
[yfit,r,coef,iopt,qual] = wmpalg('OMP',y,mpdict,'itermax',35);
plot(y); hold on;
plot(yfit,'r'); xlabel('Minutes'); ylabel('Usage');
legend('Original Signal','OMP','Location','NorthEast');
set(gca,'xlim',[1 1440]);
```

Using the expansion coefficients in coef and the atom indices in iopt, construct the signal approximation, yhat, directly from the dictionary. Compare yhat with yfit returned by wmpalg.

```
[~,I] = sort(iopt);
X = mpdict(:,iopt(I));
yhat = X*coef(I);
```

```
max(abs(yfit-yhat))
```

**References**      [1] Cai, T.T. and Wang,L. "Orthogonal Matching Pursuit for Sparse Signal Recovery with Noise". *IEEE® Transactions on Information Theory*, vol. 57, 7, 4680–4688, 2011.

[2] Donoho, D., Elad, M., and Temlyakov, V. "Stable Recovery of Sparse Overcomplete Representations in the Presence of Noise". *IEEE Transactions on Information Theory*. Vol. 52, 1, 6–18, 2004.

[3] Mallat, S. and Zhang, Z. "Matching Pursuits with Time-Frequency Dictionaries". *IEEE Transactions on Signal Processing*, vol. 41, 12, 3397–3415, 1993

[4] bTropp, J.A. "Greed is good: Algorithmic results for sparse approximation". *IEEE Transactions on Information Theory*, 50, pp. 2231–2242, 2004.

**See Also**       wavemenu **|** wmpdictionary

**Related Examples**
- "Matching Pursuit — Command Line"
- "Matching Pursuit — Interactive Analysis"

**Concepts**
- "Sparse Representation in Redundant Dictionaries"
- "Matching Pursuit Algorithms"

# wmpdictionary

| **Purpose** | Dictionary for matching pursuit |
|---|---|

**Syntax**

```
MPDICT = wmpdictionary(N)
[MPDICT,NBVECT] = wmpdictionary(N)
[MPDICT,NBVECT]= wmpdictionary(N,Name,Value)
[MPDICT,NBVECT,LST] = wmpdictionary(N,Name,Value)
[MPDICT,NBVECT,LST,LONGS] = wmpdictionary(N,Name,Value)
```

**Description**

MPDICT = wmpdictionary(N) returns the N-by-P dictionary, MPDICT, for the default subdictionaries {{'sym4',5},{'wpsym4',5},'dct','sin'}. The column dimension of MPDICT depends on N.

[MPDICT,NBVECT] = wmpdictionary(N) returns the row vector, NBVECT, which contains the number of vectors in each subdictionary. The order of the elements in NBVECT corresponds to the order of the subdictionaries and any prepended or appended subdictionaries. The sum of the elements in NBVECT is the column dimension of MPDICT.

[MPDICT,NBVECT]= wmpdictionary(N,Name,Value) returns the dictionary, MPDICT, using additional options specified by one or more Name,Value pair arguments.

[MPDICT,NBVECT,LST] = wmpdictionary(N,Name,Value) returns the cell array, LST, with descriptions of the subdictionaries.

[MPDICT,NBVECT,LST,LONGS] = wmpdictionary(N,Name,Value) returns the cell array, LONGS, containing the number of vectors in each subdictionary. LONGS is only useful for wavelet subdictionaries. In wavelet subdictionaries, the corresponding element in LONGS gives the number of scaling functions at the coarsest level and wavelet functions by level. See "Visualize Haar Wavelet Dictionary" on page 1-580 for an example using LONGS.

**Input Arguments**

**N**

A positive integer equal to the length of your input signal. The dictionary atoms are constructed to have N elements. N equals the row dimension of the dictionary, MPDICT.

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

#### 'addbeg'

Prepended subdictionary. The prepended subdictionary is an N-by-M matrix where N is the length of the input signal. `wmpdictionary` does not check that the M column vectors of the prepended dictionary form a basis. If you do not specify a value for `lstcpt`, the subdictionary is prepended to the default dictionary. The column vectors in the prepended subdictionary do not have to be unit-norm.

#### 'addend'

Appended subdictionary. The appended subdictionary is a N-by-M matrix where N is the length of the input signal. `wmpdictionary` does not check that the M column vectors of the prepended dictionary form a basis. If you do not specify a value for `lstcpt`, the subdictionary is appended to the default dictionary. The column vectors in the appended subdictionary do not have to be unit-norm.

#### 'lstcpt'

A cell array of cell arrays with valid subdictionaries. Each cell array describes one subdictionary. Valid subdictionaries are:

- A valid Wavelet Toolbox orthogonal or biorthogonal wavelet family short name with the number of vanishing moments and an optional decomposition level and extension mode. For example, {`'sym4'`,5} denotes the Daubechies least-asymmetric wavelet with 4 vanishing moments at level 5 and the default extension mode `'per'`. If you do not specify the optional level and extension mode, the decomposition level defaults to 5 and the extension mode to `'per'`.

- A valid Wavelet Toolbox orthogonal or biorthogonal wavelet family short name preceded by wp with the number of vanishing moments and an optional decomposition level and extension mode. For example, {'wpsym4',5} denotes the Daubechies least-asymmetric wavelet packet with 4 vanishing moments at level 5. If you do not specify the optional level and extension mode, the decomposition level defaults to 5 and the extension mode to 'per'.

- 'dct' Discrete cosine transform-II basis. The DCT-II orthonormal basis is:

$$\phi_k(n) = \begin{cases} \dfrac{1}{\sqrt{N}} & k = 0 \\[2mm] \sqrt{\dfrac{2}{N}} \cos(\tfrac{\pi}{N}(n + \tfrac{1}{2})k) & k = 1, 2, \ldots, N-1 \end{cases}$$

- 'sin' Sine subdictionary. The sine subdictionary is

$$\phi_k(t) = \sin(2\pi k t) \quad k = 1, 2, \ldots, \left\lceil \tfrac{N}{2} \right\rceil \quad 0 \le t \le 1$$

where $t$ is a linearly-spaced $N$-point vector.

- 'cos' Cosine subdictionary. The cosine subdictionary is

$$\phi_k(t) = \cos(2\pi k t) \quad k = 1, 2, \ldots, \left\lceil \tfrac{N}{2} \right\rceil \quad 0 \le t \le 1$$

where $t$ is a linearly-spaced $N$-point vector.

- 'poly' Polynomial subdictionary. The polynomial subdictionary is:

$$p_n(t) = t^{n-1} \quad n = 1, 2, \ldots 20 \quad\quad 0 \le t \le 1$$

where $t$ is a linearly-spaced $N$-point vector.

- 'RnIdent' The shifted Kronecker delta subdictionary. The shifted Kronecker delta subdictionary is:

$$\phi_k(n) = \delta(n - k) \quad k = 0, 1, \ldots N$$

**Default:** {{'sym4',5},{'wpsym4',5},'dct','sin'}

**Output Arguments**

**MPDICT**

Matching pursuit dictionary. MPDICT is an N-by-P matrix with the row dimension, N, equal to the length of the input signal. The column dimension of the matrix depends on the size of the concatenated subdictionaries.

**NBVECT**

Number of vectors in subdictionaries. NBVECT is a row vector containing the number of elements in each subdictionary. The order of the elements in NBVECT corresponds to the order of the subdictionaries and any prepended or appended subdictionaries.

**LST**

Cell array describing the dictionary. LST is a 1-by-N cell array where N is the number of subdictionaries. Each element of the cell array contains a description of a subdictionary. If you specify a prepended or appended subdictionary, the first element of LST is 'AddBeg' or 'AddEnd'. If you specify a level for the wavelet or wavelet packet, the corresponding element of LST is a 1-by-2 cell array containing the wavelet or wavelet packet name in the first element and the level in the second element.

**LONGS**

Cell array containing the number of elements for each subdictionary. LONGS is useful only for wavelet subdictionaries. If you specify a wavelet subdictionary, the corresponding element of LONGS provides the number of scaling functions at the coarsest level and the number of wavelets at each level. See "Visualize Haar Wavelet Dictionary" on page 1-580 for an example using LONGS.

**Definitions**

**Matching Pursuit**

Matching pursuit refers to a number of greedy or weak-greedy algorithms for computing an adaptive nonlinear expansion of a signal

in a *dictionary*. In the majority of matching pursuit applications, a dictionary is an overcomplete set of vectors. The elements of the dictionary are referred to as *atoms* and are typically constructed to have certain time/frequency or time/scale properties. Matching pursuit takes the NP-hard problem of finding the best nonlinear expansion in a dictionary and implements it in an energy-perserving formulation that guarantees convergence. See "Matching Pursuit Algorithms" for more details.

## Examples

### Default Dictionary

Create the default dictionary to represent a signal of length 100.

```
mpdict = wmpdictionary(100);
```

### Discrete Cosine Transform and Kronecker Delta Dictionary

Create a DCT and shifted Kronecker delta dictionary to represent a signal of length 100.

```
mpdict = wmpdictionary(100,'lstcpt',{'dct','RnIdent'});
```

### Haar Wavelet Packets and Discrete Cosine Transform Dictionary

Create a Haar wavelet packet (level 2) and DCT dictionary. Return the number of atoms in each subdictionary.

```
[mpdict,nbvect] = wmpdictionary(100,'lstcpt',{{'wphaar',2},'dct'});
```

### Visualize Haar Wavelet Dictionary

Use the output argument, LONGS, to visualize a dictionary.

Create a Haar wavelet dictionary consisting of level-2 scaling functions and level-1 and level-2 wavelet functions.

```
[mpdict,~,~,longs] = wmpdictionary(100,'lstcpt',{{'haar',2}});
for nn = 1:size(mpdict,2)

    if (nn <= longs{1}(1))
```

```
    plot(mpdict(:,nn),'k','linewidth',2); grid on;
xlabel('Translation');
    title('Haar Scaling Function - Level 2');
elseif (nn>longs{1}(1) & nn<= longs{1}(1)+longs{1}(2))
     plot(mpdict(:,nn),'r','linewidth',2); grid on;
     xlabel('Translation');
    title('Haar Wavelet - Level 2');
else
    title('Haar Wavelet - Level 1');
    plot(mpdict(:,nn),'b','linewidth',2); grid on;
    title('Haar Wavelet - Level 1');
    xlabel('Translation');
end
    pause(0.2);
end
```

**References**

[1] Cai, T.T. and L. Wang "Orthogonal Matching Pursuit for Sparse Signal Recovery with Noise". *IEEE Transactions on Information Theory*, vol. 57, 7, 4680–4688, 2011.

[2] Donoho, D., M. Elad, and V. Temlyakov "Stable Recovery of Sparse Overcomplete Representations in the Presence of Noise". *IEEE Transactions on Information Theory*, 52,1, 6–18, 2004.

[3] Mallat, S. and Z. Zhang "Matching Pursuits with Time-Frequency Dictionaries". *IEEE Transactions on Signal Processing*, vol. 41, 12, 3397–3415, 1993

[4] Tropp, J.A. "Greed is good: Algorithmic results for sparse approximation". *IEEE Transactions on Information Theory*, 50, pp. 2231–2242, 2004.

**See Also**      wavemenu **|** wmpalg

**Related Examples**
- "Matching Pursuit — Command Line"
- "Matching Pursuit — Interactive Analysis"

# wmpdictionary

**Concepts**
- "Sparse Representation in Redundant Dictionaries"
- "Matching Pursuit Algorithms"

| **Purpose** | Multiscale Principal Component Analysis |
|---|---|

**Syntax**
```
[X_SIM,QUAL,NPC,DEC_SIM,PCA_Params] = wmspca(X,LEVEL,WNAME,NPC)
[...] = wmspca(X,LEVEL,WNAME,'mode',EXTMODE,NPC)
[...] = wmspca(DEC,NPC)
[...] = wmspca(X,LEVEL,WNAME,'mode',EXTMODE,NPC)
```

**Description** `[X_SIM,QUAL,NPC,DEC_SIM,PCA_Params] = wmspca(X,LEVEL,WNAME,NPC)` or `[...] = wmspca(X,LEVEL,WNAME,'mode',EXTMODE,NPC)` returns a simplified version `X_SIM` of the input matrix `X` obtained from the wavelet-based multiscale principal component analysis (PCA).

The input matrix `X` contains `P` signals of length `N` stored columnwise ($N > P$).

### Wavelet Decomposition Parameters

The wavelet decomposition is performed using the decomposition level `LEVEL` and the wavelet `WNAME`.

`EXTMODE` is the extended mode for the DWT (See `dwtmode`).

If a decomposition `DEC` obtained using `mdwtdec` is available, you can use

`[...]  = wmspca(DEC,NPC)` instead of

`[...]  = wmspca(X,LEVEL,WNAME,'mode',EXTMODE,NPC)`.

### Principal Components Parameter: NPC

If `NPC` is a vector, then it must be of length `LEVEL+2`. It contains the number of retained principal components for each PCA performed:

- `NPC(d)` is the number of retained noncentered principal components for details at level `d`, for $1 <= d <= \text{LEVEL}$.

- `NPC(LEVEL+1)` is the number of retained non-centered principal components for approximations at level `LEVEL`.

- `NPC(LEVEL+2)` is the number of retained principal components for final PCA after wavelet reconstruction.

NPC must be such that $0 <= $ `NPC(d)` $ <= $ `P` for $1 <= $ `d` $ <= $ `LEVEL`+2.

If `NPC = 'kais'` (respectively, `'heur'`), then the number of retained principal components is selected automatically using Kaiser's rule (or the heuristic rule).

- Kaiser's rule keeps the components associated with eigenvalues greater the mean of all eigenvalues.

- The heuristic rule keeps the components associated with eigenvalues greater than 0.05 times the sum of all eigenvalues.

If `NPC = 'nodet'`, then the details are "killed" and all the approximations are retained.

**Output Parameters**

`X_SIM` is a simplified version of the matrix `X`.

`QUAL` is a vector of length `P` containing the quality of column reconstructions given by the relative mean square errors in percent.

`NPC` is the vector of selected numbers of retained principal components.

`DEC_SIM` is the wavelet decomposition of `X_SIM`.

`PCA_Params` is a structure array of length `LEVEL`+2 such that:

- `PCA_Params(d).pc` is a P-by-P matrix of principal components.

  The columns are stored in descending order of the variances.

- `PCA_Params(d).variances` is the principal component variances vector.

- `PCA_Params(d).npc = NPC`

**Examples**     **Wavelet Principal Component Analysis of Noisy Multivariate Signal**

Use wavelet multiscale principal component analysis to denoise a multivariate signal.

Load the dataset consisting of 4 signals of length 1024. Plot the original signals and the signals with additive noise.

```
load ex4mwden;
kp = 0;
for i = 1:4
    subplot(4,2,kp+1), plot(x_orig(:,i)); axis tight;
    title(['Original signal ',num2str(i)])
    subplot(4,2,kp+2), plot(x(:,i)); axis tight;
    title(['Noisy signal ',num2str(i)])
    kp = kp + 2;
end
```

Perform the first multiscale wavelet PCA using the Daubechies' least-asymmetric wavelet with 4 vanishing moments, sym4. Obtain the multiresolution decomposition down to level 5. Use the heuristic rule to decide how many principal components to retain.

```
level = 5;
wname = 'sym4';
npc = 'heur';
[x_sim, qual, npc] = wmspca(x,level,wname,npc);
```

Plot the result and examine the quality of the approximation.

```
qual
kp = 0;
for i = 1:4
    subplot(4,2,kp+1), plot(x(:,i)); axis tight;
    title(['Noisy signal ',num2str(i)])
    subplot(4,2,kp+2), plot(x_sim(:,i)); axis tight;
    title(['First PCA ',num2str(i)])
    kp = kp + 2;
end
```

The quality results are all close to 100%. The npc vector gives the number of principal components retained at each level.

Suppress the noise by removing the principal components at levels 1–3. Perform the multiscale PCA again.

```
npc(1:3) = zeros(1,3);
[x_sim, qual, npc] = wmspca(x,level,wname,npc);
```

Plot the result.

```
kp = 0;
for i = 1:4
    subplot(4,2,kp+1), plot(x(:,i)); axis tight;
    title(['Noisy signal ',num2str(i)])
    subplot(4,2,kp+2), plot(x_sim(:,i)); axis tight;
    title(['Second PCA ',num2str(i)])
    kp = kp + 2;
end
```

**Algorithms**    The multiscale principal components generalizes the usual PCA of a multivariate signal seen as a matrix by performing simultaneously a PCA on the matrices of details of different levels. In addition, a PCA is performed also on the coarser approximation coefficients matrix in the wavelet domain as well as on the final reconstructed matrix. By selecting conveniently the numbers of retained principal components, interesting simplified signals can be reconstructed.

**References**    Aminghafari, M.; Cheze, N.; Poggi, J-M. (2006), "Multivariate de-noising using wavelets and principal component analysis," *Computational Statistics & Data Analysis*, 50, pp. 2381–2398.

Bakshi, B. (1998), "Multiscale PCA with application to MSPC monitoring," *AIChE J.*, 44, pp. 1596–1610.

**See Also**    wmulden

| | |
|---|---|
| **Purpose** | Wavelet multivariate de-noising |
| **Syntax** | `[X_DEN,NPC,NESTCOV,DEC_DEN,PCA_Params,DEN_Params] = ...`<br>`  wmulden(X,LEVEL,WNAME,NPC_APP,NPC_FIN,TPTR,SORH)`<br>`[...] = wmulden(X,LEVEL,WNAME,'mode',EXTMODE,NPC_APP,...)`<br>`[...] = wmulden(DEC,NPC_APP)`<br>`[...] = wmulden(X,LEVEL,WNAME,'mode',EXTMODE,NPC_APP)`<br>`[DEC,PCA_Params] = wmulden('estimate',DEC,NPC_APP,NPC_FIN)`<br>`[X_DEN,NPC,DEC_DEN,PCA_Params] =`<br>`wmulden('execute',DEC,PC_Params)` |

**Description**  `[X_DEN,NPC,NESTCOV,DEC_DEN,PCA_Params,DEN_Params] = ...`
`wmulden(X,LEVEL,WNAME,NPC_APP,NPC_FIN,TPTR,SORH)` or
`[...] = wmulden(X,LEVEL,WNAME,'mode',EXTMODE,NPC_APP,...)`
returns a de-noised version `X_DEN` of the input matrix `X`. The strategy
combines univariate wavelet de-noising in the basis where the
estimated noise covariance matrix is diagonal with noncentered
Principal Component Analysis (PCA) on approximations in the wavelet
domain or with final PCA.

The input matrix `X` contains `P` signals of length `N` stored columnwise
where `N > P`.

### Wavelet Decomposition Parameters

The wavelet decomposition is performed using the decomposition level
`LEVEL` and the wavelet `WNAME`.

`EXTMODE` is the extended mode for the DWT (See `dwtmode`).

If a decomposition `DEC` obtained using `mdwtdec` is available, you can use

`[...]  = wmulden(DEC,NPC_APP)` instead of

`[...]  = wmulden(X,LEVEL,WNAME,'mode',EXTMODE,NPC_APP)`.

### Principal Components Parameters: NPC_APP and NPC_FIN

The input selection methods `NPC_APP` and `NPC_FIN` define the way
to select principal components for approximations at level `LEVEL` in

the wavelet domain and for final PCA after wavelet reconstruction, respectively.

If `NPC_APP` (or `NPC_FIN`) is an integer, it contains the number of retained principal components for approximations at level `LEVEL` (or for final PCA after wavelet reconstruction).

`NPC_XXX` must be such that $0 <= \text{NPC\_XXX} <= P$

`NPC_APP` or `NPC_FIN` = `'kais'` (or `'heur'`) selects the number of retained principal components using Kaiser's rule (or the heuristic rule) automatically.

- Kaiser's rule keeps the components associated with eigenvalues greater than the mean of all eigenvalues.

- Heuristic rule keeps the components associated with eigenvalues greater than 0.05 times the sum of all eigenvalues.

`NPC_APP` or `NPC_FIN` = `'none'` is equivalent to `NPC_APP` or `NPC_FIN` = P.

### De-noising Parameters: TPTR and SORH

The default values for the de-noising parameters TPTR and SORH are:

`TPTR = 'sqtwolog'` and `SORH = 's'`

- Valid values for `TPTR` are

  `'rigsure'`, `'heursure'`, `'sqtwolog'`, `'minimaxi'`,
  `'penalhi'`, `'penalme'`, `'penallo'`

- Valid values for SORH are:

  `'s'` (soft) or `'h'` (hard)

For additional information, see `wden` and `wbmpen`.

### Output Parameters

`X_DEN` is a de-noised version of the input matrix `X`.

`NPC` is the vector of selected numbers of retained principal components.

NESTCOV is the estimated noise covariance matrix obtained using the minimum covariance determinant (MCD) estimator.

DEC_DEN is the wavelet decomposition of X_DEN.

PCA_Params is a structure such that:

```
PCA_Params.NEST = {pc_NEST,var_NEST,NESTCOV}
PCA_Params.APP  = {pc_APP,var_APP,npc_APP}
PCA_Params.FIN  = {pc_FIN,var_FIN,npc_FIN}
```

where:

- pc_XXX is a P-by-P matrix of principal components.

  The columns are stored in descending order of the variances.

- var_XXX is the principal component variances vector.

- NESTCOV is the covariance matrix estimate for detail at level 1.

DEN_Params is a structure such that:

- DEN_Params.thrVAL is a vector of length LEVEL which contains the threshold values for each level.

- DEN_Params.thrMETH is a string containing the name of the de-noising method (TPTR).

- DEN_Params.thrTYPE is a character variable containing the type of the thresholding (SORH).

## Special Cases

[DEC,PCA_Params] = wmulden('estimate',DEC,NPC_APP,NPC_FIN) returns the wavelet decomposition DEC and the Principal Components Estimates PCA_Params.

[X_DEN,NPC,DEC_DEN,PCA_Params] = wmulden('execute',DEC,PC_Params) uses the principal components estimates PCA_Params previously computed.

The input value DEC can be replaced by X, LEVEL, and WNAME.

# wmulden

```
% Load a multivariate signal x together with
% the original signals (x_orig) and true noise
% covariance matrix (covar).

load ex4mwden

% Set the de-noising method parameters.
level = 5;
wname = 'sym4';
tptr  = 'sqtwolog';
sorh  = 's';

% Set the PCA parameters to select the number of
% retained principal components automatically by
% Kaiser's rule.

npc_app = 'kais';
npc_fin = 'kais';

% Perform multivariate de-noising.
[x_den, npc, nestco] = wmulden(x, level, wname, npc_app, ...
                                   npc_fin, tptr, sorh);

% Display the original and de-noised signals.
kp = 0;
for i = 1:4
    subplot(4,3,kp+1), plot(x_orig(:,i));
    title(['Original signal ',num2str(i)])
    subplot(4,3,kp+2), plot(x(:,i));
    title(['Observed signal ',num2str(i)])
    subplot(4,3,kp+3), plot(x_den(:,i));
    title(['De-noised signal ',num2str(i)])
    kp = kp + 3;
end
```
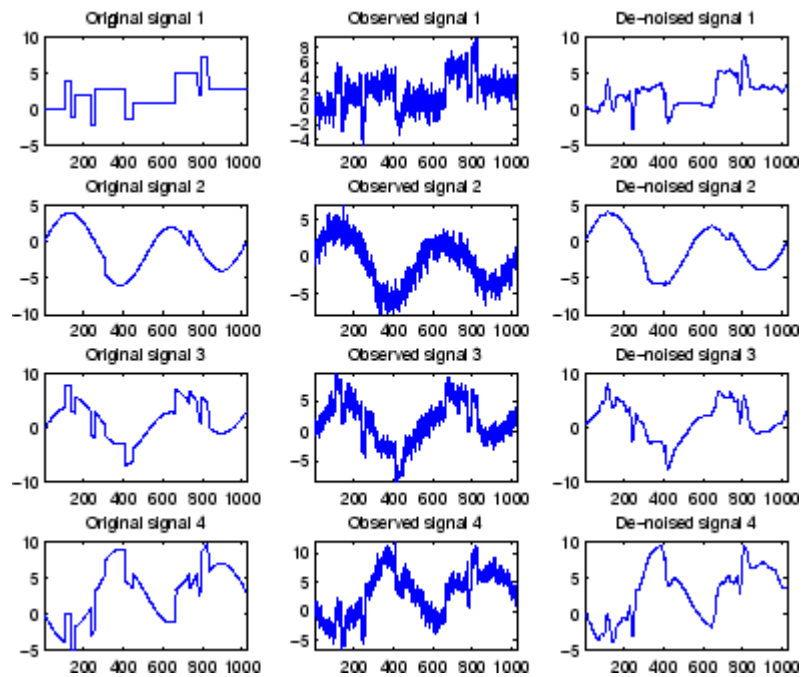
```
% The results are good: the first function, which is
% irregular, is correctly recovered while the second
% function, more regular, is well de-noised.

% The second output argument gives the numbers
% of retained principal components for PCA for
% approximations and for final PCA.

npc

npc =

     2     2

% The third output argument contains the estimated
```

```
% noise covariance matrix using the MCD based
% on the matrix of finest details.

nestco

nestco =

    1.0784    0.8333    0.6878    0.8141
    0.8333    1.0025    0.5275    0.6814
    0.6878    0.5275    1.0501    0.7734
    0.8141    0.6814    0.7734    1.0967

% The estimation is satisfactory since the values are close
% to the true values given by covar.

covar

covar =

    1.0000    0.8000    0.6000    0.7000
    0.8000    1.0000    0.5000    0.6000
    0.6000    0.5000    1.0000    0.7000
    0.7000    0.6000    0.7000    1.0000
```

**Algorithms**    The multivariate de-noising procedure is a generalization of the one-dimensional strategy. It combines univariate wavelet de-noising in the basis where the estimated noise covariance matrix is diagonal and non-centered Principal Component Analysis (PCA) on approximations in the wavelet domain or with final PCA.

The robust estimate of the noise covariance matrix given by the minimum covariance determinant estimator based on the matrix of finest details.

**References**    Aminghafari, M.; Cheze, N.; Poggi, J-M. (2006), "Multivariate de-noising using wavelets and principal component analysis," *Computational Statistics & Data Analysis*, 50, pp. 2381–2398.

Rousseeuw, P.; Van Driessen, K. (1999), "A fast algorithm for the minimum covariance determinant estimator," *Technometrics,* 41, pp. 212–223.

**See Also**      wmspca

# wnoise

| | |
|---|---|
| **Purpose** | Noisy wavelet test data |
| **Syntax** | X = wnoise(FUN,N) |
| | [X,XN] = wnoise(FUN,N,SQRT_SNR) |
| | [X,XN] = wnoise(FUN,N,SQRT_SNR,INIT) |

**Description**    X = wnoise(FUN,N) returns values of the test signal given by FUN, on a $2^N$ grid of [0,1].

[X,XN] = wnoise(FUN,N,SQRT_SNR) returns a test vector X as above, rescaled such that std(X) = SQRT_SNR. The returned vector XN contains the same test vector corrupted by additive Gaussian white noise $N(0,1)$. Then, XN has a signal-to-noise ratio of SNR = $(SQRT\_SNR)^2$.

[X,XN] = wnoise(FUN,N,SQRT_SNR,INIT) returns previous vectors X and XN, but the generator seed is set to INIT value.

The six functions below are due to Donoho and Johnstone (See "References").

| FUN = 1 | or | 'blocks' |
|---|---|---|
| FUN = 2 | or | 'bumps' |
| FUN = 3 | or | 'heavy sine' |
| FUN = 4 | or | 'doppler' |
| FUN = 5 | or | 'quadchirp' |
| FUN = 6 | or | 'mishmash' |

**Examples**
```
% Generate 2^10 samples of 'Heavy sine' (item 3).
x = wnoise(3,10);

% Generate 2^10 samples of 'Doppler' (item 4) and of
% noisy 'Doppler' with a square root of signal-to-noise
% ratio equal to 7.
[x,noisyx] = wnoise(4,10,7);
```

```
% To introduce your own rand seed, a fourth
% argument is allowed:
init = 2055415866;
[x,noisyx] = wnoise(4,10,7,init);

% Plot all the test functions.
ind = linspace(0,1,2^10);
for i = 1:6
    x = wnoise(i,10);
    subplot(6,1,i), plot(ind,x)
end

% Editing some graphical properties,
% the following figure is generated.
```



**References**   Donoho, D.L.; I.M. Johnstone (1994), "Ideal spatial adaptation by wavelet shrinkage," *Biometrika*, vol. 81, pp. 425–455.

Donoho, D.L.; I.M. Johnstone (1995), "Adapting to unknown smoothness via wavelet shrinkage via wavelet shrinkage," *JASA*, vol. 90, 432, pp. 1200–1224.

**See Also**     wden

**Purpose**     Estimate noise of 1-D wavelet coefficients

**Syntax**      STDC = wnoisest(C,L,S)
                STDC = wnoisest(C)
                STDC = wnoisest(C)

**Description**  STDC = wnoisest(C,L,S) returns estimates of the detail coefficients'
                standard deviation for levels contained in the input vector S. [C,L]
                is the input wavelet decomposition structure (see wavedec for more
                information).

                If C is a one dimensional cell array, STDC = wnoisest(C) returns a
                vector such that STDC(k) is an estimate of the standard deviation of
                C{k}.

                If C is a numeric array, STDC = wnoisest(C) returns a vector such that
                STDC(k) is an estimate of the standard deviation of C(k,:).

                The estimator used is Median Absolute Deviation / 0.6745, well suited
                for zero mean Gaussian white noise in the de-noising one-dimensional
                model (see thselect for more information).

**Examples**    **Estimate Noise Standard Deviation in The Presence of
                Outliers**

                Estimate of the noise standard deviation in an N(0,1) white Gaussian
                noise vector with outliers.

                Create an N(0,1) noise vector with 10 randomly-placed outliers.

```
rng default;
x = randn(1000,1);
P = randperm(length(x));
indices = P(1:10);
x(indices(1:5)) = 10;
x(indices(6:end)) = -10;
```

                Obtain the discrete wavelet transform down to level 2 using the
                Daubechies' extremal phase wavelet with 3 vanishing moments.

```
[c,l] = wavedec(x,2,'db3');
stdc = wnoisest(c,l,1:2)
```

In spite of the outliers, wnoisest provides a robust estimate of the standard deviation.

**References**    Donoho, D.L.; I.M. Johnstone (1994), "Ideal spatial adaptation by wavelet shrinkage," *Biometrika*, vol 81, pp. 425–455.

Donoho, D.L.; I.M. Johnstone (1995), "Adapting to unknown smoothness via wavelet shrinkage via wavelet shrinkage," *JASA*, vol 90, 432, pp. 1200–1224.

**See Also**    thselect | wavedec | wden

**Purpose**      Extract wavelet tree from wavelet packet tree

**Syntax**       T = wp2wtree(*T*)

**Description**  wp2wtree is a one- or two-dimensional wavelet packet analysis function.

T = wp2wtree(*T*) computes the modified wavelet packet tree *T* corresponding to the wavelet decomposition tree.

**Examples**
```
% The current extension mode is zero-padding (see dwtmode).

% Load signal.
   load noisdopp; x = noisdopp;

% Decompose x at depth 3 with db1 wavelet packets
% using shannon entropy.
wpt = wpdec(x,3,'db1');

% Plot wavelet packet tree wpt.
plot(wpt)
```
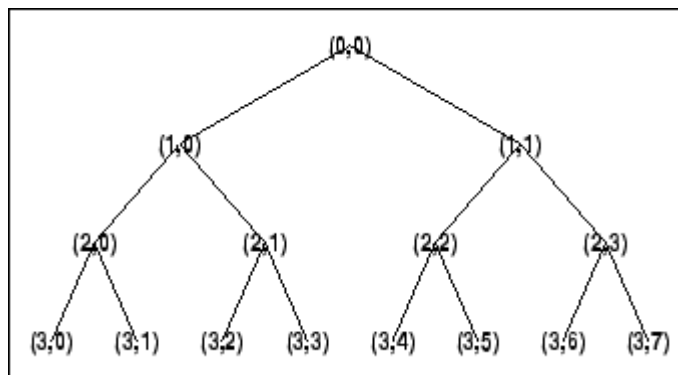


```
% Compute wavelet tree.
wt = wp2wtree(wpt);
```

```
% Plot wavelet tree wt.
plot(wt)
```
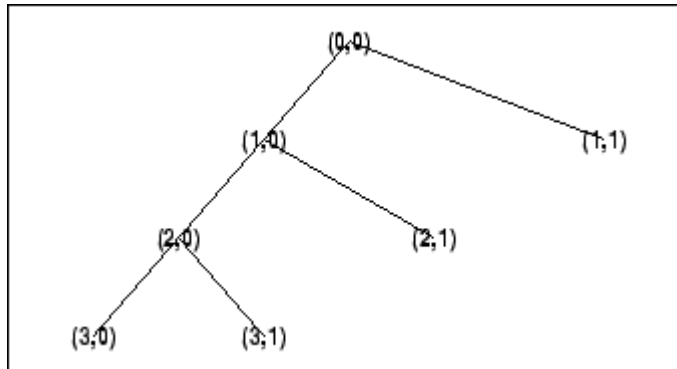


**See Also**    wpdec | wpdec2

**Purpose**        Penalized threshold for wavelet packet de-noising

**Syntax**          THR = wpbmpen(T,SIGMA,ALPHA)
wpbmpen(T,SIGMA,ALPHA,ARG)

**Description**    THR = wpbmpen(T,SIGMA,ALPHA) returns a global threshold THR for de-noising. THR is obtained by a wavelet packet coefficients selection rule using a penalization method provided by Birge-Massart.

T is a wavelet packet tree corresponding to the wavelet packet decomposition of the signal or image to be de-noised.

SIGMA is the standard deviation of the zero mean Gaussian white noise in the de-noising model (see wnoisest for more information).

ALPHA is a tuning parameter for the penalty term. It must be a real number greater than 1. The sparsity of the wavelet packet representation of the de-noised signal or image grows with ALPHA. Typically ALPHA = 2.

THR minimizes the penalized criterion given by

let $t^*$ be the minimizer of

crit(t) = -sum(c(k)^2,k t) + 2*SIGMA^2*t*(ALPHA + log(n/t))

where c(k) are the wavelet packet coefficients sorted in decreasing order of their absolute value and n is the number of coefficients, then THR = |c(t$^*$)|.

wpbmpen(T,SIGMA,ALPHA,ARG) computes the global threshold and, in addition, plots three curves:

- 2*SIGMA^2*t*(ALPHA + log(n/t))

- sum(c(k)^2,k£t)

- crit(t)

**Examples**     % Example 1: Signal de-noising.
% Load noisy chirp signal.

```
load noischir; x = noischir;

% Perform a wavelet packet decomposition of the signal
% at level 5 using sym6.
wname = 'sym6'; lev = 5;
tree = wpdec(x,lev,wname);

% Estimate the noise standard deviation from the
% detail coefficients at level 1,
% corresponding to the node index 2.
det1 = wpcoef(tree,2);
sigma = median(abs(det1))/0.6745;

% Use wpbmpen for selecting global threshold
% for signal de-noising, using the recommended parameter.
alpha = 2;
thr = wpbmpen(tree,sigma,alpha)

thr =

    4.5740


% Use wpdencmp for de-noising the signal using the above
% threshold with soft thresholding and keeping the
% approximation.
keepapp = 1;
xd = wpdencmp(tree,'s','nobest',thr,keepapp);

% Plot original and de-noised signals.
figure(1)
subplot(211), plot(x),
title('Original signal')
subplot(212), plot(xd)
title('De-noised signal')
```
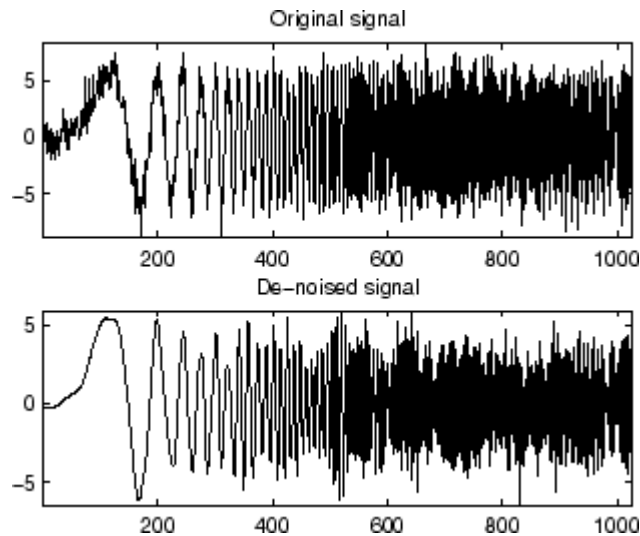
Original signal



De-noised signal

```
% Example 2: Image de-noising.
% Load original image.
load noiswom;
nbc = size(map,1);

% Perform a wavelet packet decomposition of the image
% at level 3 using coif2.
wname = 'coif2'; lev = 3;
tree = wpdec2(X,lev,wname);

% Estimate the noise standard deviation from the
% detail coefficients at level 1.
det1 = [wpcoef(tree,2) wpcoef(tree,3) wpcoef(tree,4)];
sigma = median(abs(det1(:)))/0.6745;

% Use wpbmpen for selecting global threshold
% for image de-noising.
alpha = 1.1;
thr = wpbmpen(tree,sigma,alpha)
```
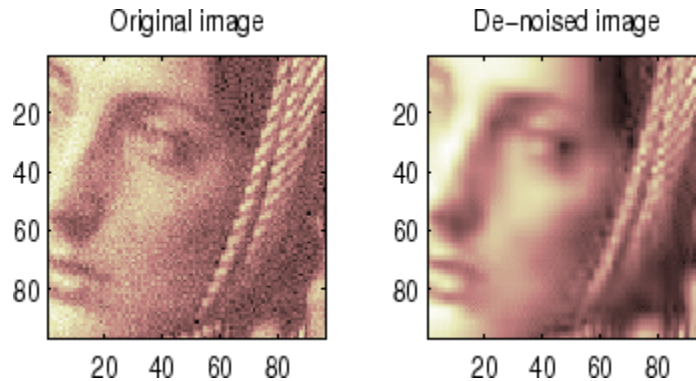
```
thr =

   38.5125


% Use wpdencmp for de-noising the image using the above
% thresholds with soft thresholding and keeping the
% approximation.
keepapp = 1;
xd = wpdencmp(tree,'s','nobest',thr,keepapp);

% Plot original and de-noised images.
figure(2)
colormap(pink(nbc));
subplot(221), image(wcodemat(X,nbc))
title('Original image')
subplot(222), image(wcodemat(xd,nbc))
title('De-noised image')
```



**See Also**   wbmpen | wden | wdencmp | wpdencmp

**Purpose**      Wavelet packet coefficients

**Syntax**       X = wpcoef(*T*,N)
                 X = wpcoef(*T*)

**Description**  wpcoef is a one- or two-dimensional wavelet packet analysis function.

X = wpcoef(*T*,N) returns the coefficients associated with the node N of
the wavelet packet tree *T*. If N doesn't exist, X = [ ];

X = wpcoef(*T*) is equivalent to X = wpcoef(*T*,0).

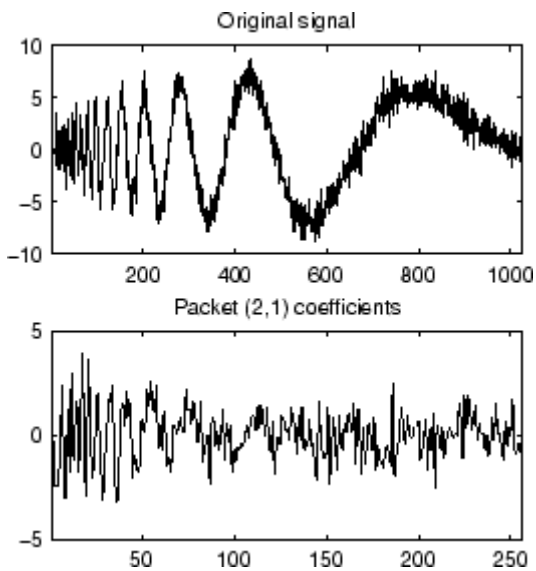**Examples**     % The current extension mode is zero-padding (see dwtmode).

                 % Load signal.
                 load noisdopp; x = noisdopp;

                 figure(1); subplot(211);
                 plot(x); title('Original signal');

                 % Decompose x at depth 3 with db1 wavelet packets
                 % using Shannon entropy.
                 wpt = wpdec(x,3,'db1');

                 % Plot wavelet packet tree wpt.
                 plot(wpt)
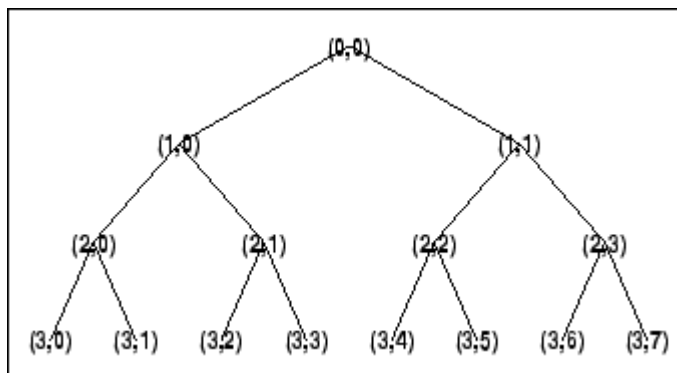
```
% Read packet (2,1) coefficients.
cfs = wpcoef(wpt,[2 1]);

figure(1); subplot(212);
plot(cfs); title('Packet (2,1) coefficients');
```

**See Also**  wpcoef | wpdec | wpdec2 | wprcoef

**How To**  • "Reconstructing a Signal Approximation from a Node"

# wpcutree

| | |
|---|---|
| **Purpose** | Cut wavelet packet tree |
| **Syntax** | T = wpcutree(*T*,L)<br>T<br>[T,RN] = wpcutree(*T*,L) |
| **Description** | wpcutree is a one- or two-dimensional wavelet packet analysis function.<br><br>T = wpcutree(*T*,L) cuts the tree T at level L.<br><br>[T,RN] = wpcutree(*T*,L) returns the same arguments as above and, in addition, the vector RN contains the indices of the reconstructed nodes. |
| **Examples** | % The current extension mode is zero-padding (see dwtmode).<br><br>% Load signal.<br>load noisdopp; x = noisdopp;<br><br>% Decompose x at depth 3 with db1 wavelet packets<br>% using Shannon entropy.<br>wpt = wpdec(x,3,'db1');<br><br>% Plot wavelet packet tree wpt.<br>plot(wpt) |
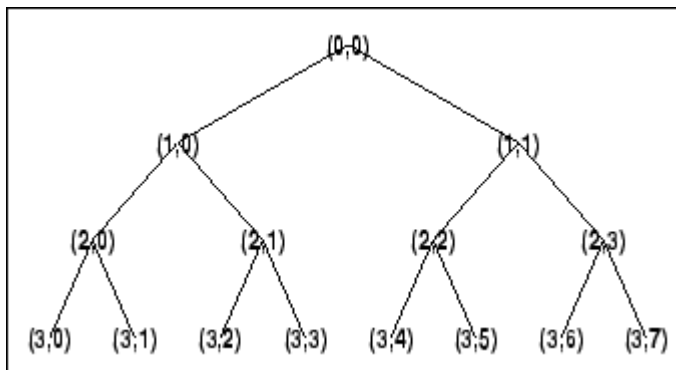
```
% Cut wavelet packet tree at level 2.
nwpt = wpcutree(wpt,2);

% Plot new wavelet packet tree nwpt.
plot(nwpt)
```
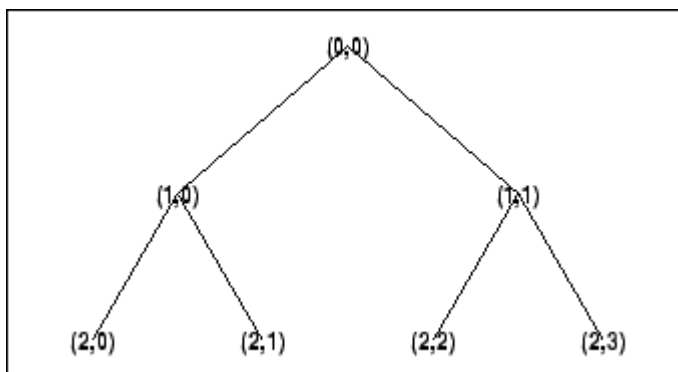


**See Also**     wpdec | wpdec2

# wpdec

| | |
|---|---|
| **Purpose** | Wavelet packet decomposition 1-D |

**Syntax**

```
T = wpdec(X,N,'wname',E,P)
T = wpdec(X,N,'wname')
T = wpdec(X,N,'wname','shannon')
T
```

**Description**    wpdec is a one-dimensional wavelet packet analysis function.

T = wpdec(X,N,'*wname*',E,P) returns a wavelet packet tree *T* corresponding to the wavelet packet decomposition of the vector X at level N, with a particular wavelet ('*wname*', see wfilters for more information).

T = wpdec(X,N,'*wname*') is equivalent to T = wpdec(X,N,'*wname*','shannon').

E is a string containing the type of entropy and *P* is an optional parameter depending on the value of T (see wentropy for more information).

| Entropy Type Name (E) | Parameter (P) | Comments |
|---|---|---|
| 'shannon' | | P is not used. |
| 'log energy' | | P is not used. |
| 'threshold' | $0 \leq P$ | P is the threshold. |
| 'sure' | $0 \leq P$ | P is the threshold. |
| 'norm' | $1 \leq P$ | P is the power. |

| Entropy Type Name (E) | Parameter (P) | Comments |
|---|---|---|
| `'user'` | `string` | `P` is a string containing the file name of your own entropy function, with a single input `X`. |
| `FunName` | `No constraints on P` | `FunName` is any other string except those used for the previous Entropy Type Names listed above. `FunName` contains the file name of your own entropy function, with `X` as input and `P` as additional parameter to your entropy function. |

**Note** The `'user'` option is historical and still kept for compatibility, but it is obsoleted by the last option described in the table above. The `FunName` option do the same as the `'user'` option and in addition gives the possibility to pass a parameter to your own entropy function.

The wavelet packet method is a generalization of wavelet decomposition that offers a richer signal analysis. Wavelet packet atoms are waveforms indexed by three naturally interpreted parameters: position and scale as in wavelet decomposition, and frequency.

For a given orthogonal wavelet function, a library of wavelet packets bases is generated. Each of these bases offers a particular way of coding signals, preserving global energy and reconstructing exact features. The wavelet packets can then be used for numerous expansions of a given signal.

# wpdec

Simple and efficient algorithms exist for both wavelet packets decomposition and optimal decomposition selection. Adaptive filtering algorithms with direct applications in optimal signal coding and data compression can then be produced.

In the orthogonal wavelet decomposition procedure, the generic step splits the approximation coefficients into two parts. After splitting we obtain a vector of approximation coefficients and a vector of detail coefficients, both at a coarser scale. The information lost between two successive approximations is captured in the detail coefficients. The next step consists in splitting the new approximation coefficient vector; successive details are never re-analyzed.

In the corresponding wavelet packets situation, each detail coefficient vector is also decomposed into two parts using the same approach as in approximation vector splitting. This offers the richest analysis: the complete binary tree is produced in the one-dimensional case or a quaternary tree in the two-dimensional case.

**Examples**

```
% The current extension mode is zero-padding (see dwtmode).

% Load signal.
load noisdopp; x = noisdopp;

% Decompose x at depth 3 with db1 wavelet packets
% using Shannon entropy.
wpt = wpdec(x,3,'db1','shannon');

% The result is the wavelet packet tree wpt.

% Plot wavelet packet tree (binary tree, or tree of order 2).
plot(wpt)
```
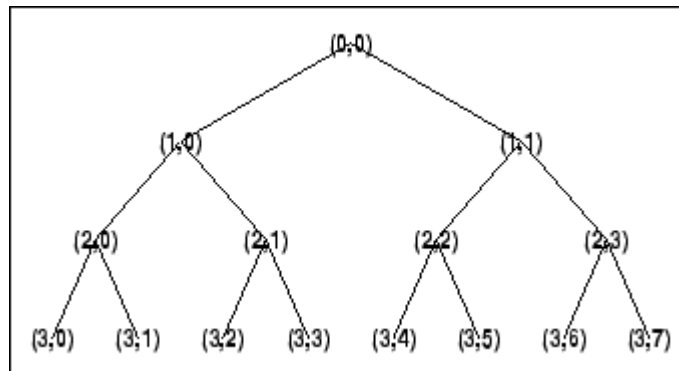
**Algorithms**    The algorithm used for the wavelet packets decomposition follows the same line as the wavelet decomposition process (see dwt and wavedec for more information).

**References**    Coifman, R.R.; M.V. Wickerhauser, (1992), "Entropy-based Algorithms for best basis selection," *IEEE Trans. on Inf. Theory*, vol. 38, 2, pp. 713–718.

Meyer, Y. (1993), *Les ondelettes. Algorithmes et applications*, Colin Ed., Paris, 2nd edition. (English translation: *Wavelets: Algorithms and Applications*, SIAM).

Wickerhauser, M.V. (1991), "INRIA lectures on wavelet packet algorithms," *Proceedings ondelettes et paquets d'ondes*, 17–21 June, Rocquencourt, France, pp. 31–99.

Wickerhauser, M.V. (1994), *Adapted wavelet analysis from theory to software algorithms*, A.K. Peters.

**See Also**    wavedec | waveinfo | wenergy | wpdec | wprec

# wpdec2

| | |
|---|---|
| **Purpose** | Wavelet packet decomposition 2-D |
| **Syntax** | T = wpdec2(X,N,*'wname'*,E,P)<br>T = wpdec2(X,N,*'wname'*)<br>T = wpdec2(X,N,*wnam*,'shannon') |
| **Description** | wpdec2 is a two-dimensional wavelet packet analysis function. |

T = wpdec2(X,N,*'wname'*,E,P) returns a wavelet packet tree T corresponding to the wavelet packet decomposition of the matrix X, at level N, with a particular wavelet (*'wname'*, see wfilters for more information).

T = wpdec2(X,N,*'wname'*) is equivalent to T = wpdec2(X,N,*wnam*,'shannon').

E is a string containing the type of entropy and P is an optional parameter depending on the value of T (see wentropy for more information).

| Entropy Type Name (E) | Parameter (P) | Comments |
|---|---|---|
| 'shannon' | | P is not used. |
| 'log energy' | | P is not used. |
| 'threshold' | $0 \leq P$ | P is the threshold. |
| 'sure' | $0 \leq P$ | P is the threshold. |
| 'norm' | $1 \leq P$ | P is the power. |

| Entropy Type Name (E) | Parameter (P) | Comments |
|---|---|---|
| 'user' | string | P is a string containing the file name of your own entropy function, with a single input X. |
| STR | No constraints on P | STR is any other string except those used for the previous Entropy Type Names listed above. STR contains the file name of your own entropy function, with X as input and P as additional parameter to your entropy function. |

**Note** The 'user' option is historical and still kept for compatibility, but it is obsoleted by the last option described in the preceding table. The FunName option does the same as the 'user' option and in addition, allows you to pass a parameter to your own entropy function.

See wpdec for a more complete description of the wavelet packet decomposition.

**Tips**    When X represents an indexed image, X is an m-by-n matrix. When X represents a truecolor image, it is an m-by-n-by-3 array, where each m-by-n matrix represents a red, green, or blue color plane concatenated along the third dimension.

For more information on image formats, see the image and imfinfo reference pages.

**Examples**    % The current extension mode is zero-padding (see dwtmode).

```
% Load image.
load tire
% X contains the loaded image.

% For an image the decomposition is performed using:
t = wpdec2(X,2,'db1');
% The default entropy is shannon.

% Plot wavelet packet tree
% (quarternary tree, or tree of order 4).
plot(t)
```
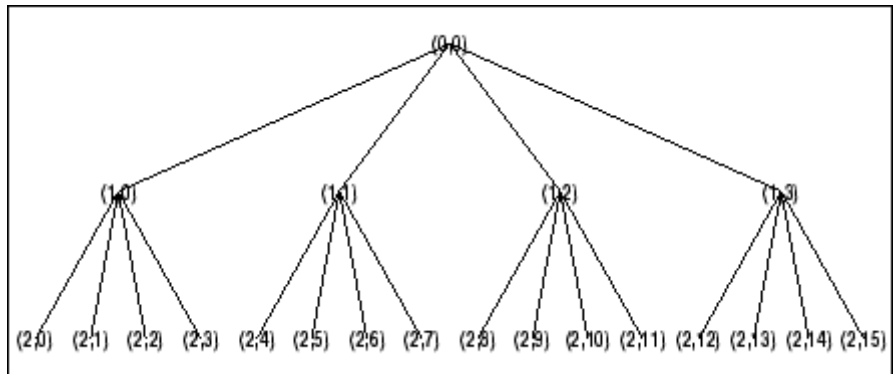


**Algorithms**   The algorithm used for the wavelet packets decomposition follows the same line as the wavelet decomposition process (see dwt2 and wavedec2 for more information).

**References**   Coifman, R.R.; M.V. Wickerhauser (1992), "Entropy-based algorithms for best basis selection," *IEEE Trans. on Inf. Theory*, vol. 38, 2, pp. 713–718.

Meyer, Y. (1993), *Les ondelettes. Algorithmes et applications*, Colin Ed., Paris, 2nd edition. (English translation: *Wavelets: Algorithms and Applications*, SIAM).

Wickerhauser, M.V. (1991), "INRIA lectures on wavelet packet algorithms," *Proceedings ondelettes et paquets d'ondes*, 17–21 June, Rocquencourt, France, pp. 31–99.

Wickerhauser, M.V. (1994), *Adapted wavelet analysis from theory to software Algorithms*, A.K. Peters.

**See Also**  wavedec2 | waveinfo | wenergy | wpdec | wprec2

# wpdencmp

**Purpose**　　　De-noising or compression using wavelet packets

**Syntax**　　　`[XD,TREED,PERF0,PERFL2] =`
`wpdencmp(X,SORH,N,'`*`wname`*`',CRIT,PAR,KEEPAPP)`
`[XD,TREED,PERF0,PERFL2] = wpdencmp(TREE,SORH,CRIT,PAR,KEEPAPP)`

**Description**　　`wpdencmp` is a one- or two-dimensional de-noising and compression oriented function.

`wpdencmp` performs a de-noising or compression process of a signal or an image, using wavelet packet. The ideas and the procedures for de-noising and compression using wavelet packet decomposition are the same as those used in the wavelets framework (see `wden` and `wdencmp` for more information).

`[XD,TREED,PERF0,PERFL2] =`
`wpdencmp(X,SORH,N,'`*`wname`*`',CRIT,PAR,KEEPAPP)` returns a de-noised or compressed version `XD` of input signal `X` (one- or two-dimensional) obtained by wavelet packets coefficients thresholding.

The additional output argument `TREED` is the wavelet packet best tree decomposition (see `besttree` for more information) of `XD`. `PERFL2` and `PERF0` are $L^2$ energy recovery and compression scores in percentages.

`PERFL2 = 100 *` (vector-norm of WP-cfs of `XD` / vector-norm of WP-cfs of `X`)$^2$.

If `X` is a one-dimensional signal and '`*wname*`' an orthogonal wavelet, `PERFL2` is reduced to

$$\frac{100\|XD\|^2}{\|X\|^2}$$

`SORH` (`'s'` or `'h'`) is for soft or hard thresholding (see `wthresh` for more information).

Wavelet packet decomposition is performed at level `N` and '`*wname*`' is a string containing the wavelet name. Best decomposition is performed using entropy criterion defined by string `CRIT` and parameter `PAR` (see

wentropy for more information). Threshold parameter is also PAR. If KEEPAPP = 1, approximation coefficients cannot be thresholded; otherwise, they can be.

```
[XD,TREED,PERF0,PERFL2] =
wpdencmp(TREE,SORH,CRIT,PAR,KEEPAPP) has the same output
```
arguments, using the same options as above, but obtained directly from the input wavelet packet tree decomposition TREE (see wpdec for more information) of the signal to be de-noised or compressed.

In addition if CRIT = 'nobest' no optimization is done and the current decomposition is thresholded.

**Examples**

```
% The current extension mode is zero-padding (see dwtmode).

% Load original signal.
load sumlichr; x = sumlichr;

% Use wpdencmp for signal compression.
% Find default values (see ddencmp).
[thr,sorh,keepapp,crit] = ddencmp('cmp','wp',x)

thr =
    0.5193

sorh =
h

keepapp =
    1

crit =
    threshold

% De-noise signal using global thresholding with
% threshold best basis.
[xc,wpt,perf0,perfl2] = ...
wpdencmp(x,sorh,3,'db2',crit,thr,keepapp);
```
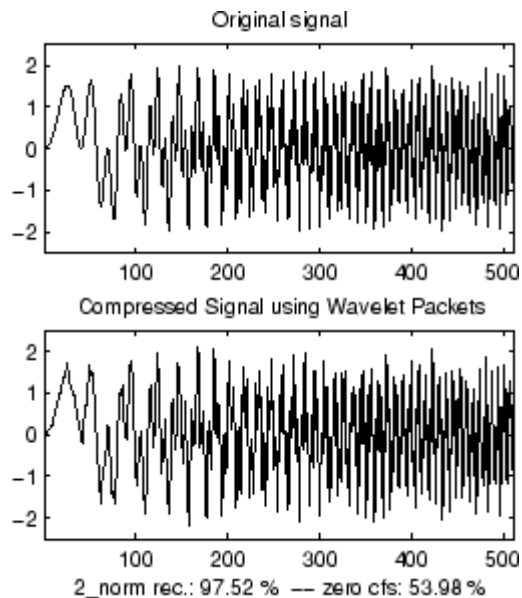
```
% Using some plotting commands,
% the following figure is generated.
```

Original signal



Compressed Signal using Wavelet Packets



2_norm rec.: 97.52 % -- zero cfs: 53.98 %

```
% Load original image.
load sinsin

% Generate noisy image.
x = X/18 + randn(size(X));

% Use wpdencmp for image de-noising.
% Find default values (see ddencmp).
[thr,sorh,keepapp,crit] = ddencmp('den','wp',x)

thr =
    4.9685

sorh =
```

```
h
keepapp =
     1

crit =
sure
% De-noise image using global thresholding with
% SURE best basis.
xd = wpdencmp(x,sorh,3,'sym4',crit,thr,keepapp);

% Using some plotting commands,
% the following figure is generated.

% Generate heavy sine and a noisy version of it.
init = 1000;
[xref,x] = wnoise(5,11,7,init);

% Use wpdencmp for signal de-noising.
n = length(x);
thr = sqrt(2*log(n*log(n)/log(2)));
xwpd = wpdencmp(x,'s',4,'sym4','sure',thr,1);

% Compare with wavelet-based de-noising result.
xwd = wden(x,'rigrsure','s','one',4,'sym4');
```

**References**    Antoniadis, A.; G. Oppenheim, Eds. (1995), *Wavelets and statistics*, Lecture Notes in Statistics, 103, Springer Verlag.

Coifman, R.R.; M.V. Wickerhauser (1992), "Entropy-based algorithms for best basis selection," *IEEE Trans. on Inf. Theory*, vol. 38, 2, pp. 713–718.

DeVore, R.A.; B. Jawerth, B.J. Lucier (1992), "Image compression through wavelet transform coding," *IEEE Trans. on Inf. Theory*, vol. 38, No 2, pp. 719–746.

Donoho, D.L. (1993), "Progress in wavelet analysis and WVD: a ten minute tour," in Progress in wavelet analysis and applications, Y. Meyer, S. Roques, pp. 109–128. Frontières Ed.

Donoho, D.L.; I.M. Johnstone (1994), "Ideal spatial adaptation by wavelet shrinkage," *Biometrika*, vol. 81, pp. 425–455.

Donoho, D.L.; I.M. Johnstone, G. Kerkyacharian, D. Picard (1995), "Wavelet shrinkage: asymptopia," *Jour. Roy. Stat. Soc.*, series B, vol. 57 no. 2, pp. 301–369.

**See Also**    besttree | ddencmp | wdencmp | wenergy | wpbmpen | wpdec | wpdec2 | wthresh

**Purpose**     Wavelet packet functions

**Syntax**
```
[WPWS,X] = wpfun('wname',NUM,PREC)
[WPWS,X] = wpfun('wname',NUM)
[WPWS,X] = wpfun('wname',NUM,7)
```

**Description**     wpfun is a wavelet packet analysis function.

[WPWS,X] = wpfun('wname',NUM,PREC) computes the wavelet packets for a wavelet 'wname' (see wfilters for more information), on dyadic intervals of length $2^{-PREC}$.

PREC must be a positive integer. Output matrix WPWS contains the $W$ functions of index from 0 to NUM, stored row-wise as [ $W_0$; $W_1$; ... ; $W_{NUM}$ ]. Output vector X is the corresponding common X-grid vector.

[WPWS,X] = wpfun('wname',NUM) is equivalent to
[WPWS,X] = wpfun('wname',NUM,7).

The computation scheme for wavelet packets generation is easy when using an orthogonal wavelet. We start with the two filters of length $2N$, denoted $h(n)$ and $g(n)$, corresponding to the wavelet.

Now by induction let us define the following sequence of functions $(W_n(x)$ , $n = 0,1,2,...)$ by

$$W_{2n}(x) = \sqrt{2} \sum_{k=0,\ldots,2N-1} h(k)W_n(2x-k)$$
$$W_{2n+1}(x) = \sqrt{2} \sum_{k=0,\ldots,2N-1} g(k)W_n(2x-k)$$

where $W_0(x) = \varphi(x)$ is the scaling function and $W_1(x) = \psi(x)$ is the wavelet function.

For example for the Haar wavelet we have

$$N = 1, h(0) = h(1) = \frac{1}{\sqrt{2}}$$

and

$$g(0) = -g(1) = \frac{1}{\sqrt{2}}$$

The equations become

$$W_{2n}(x) = W_n(2x) + W_n(2x-1)$$

and

$$(W_{2n+1}(x) = W_n(2x) - W_n(2x-1))$$

$W_0(x) = \varphi(x)$ is the haar scaling function and $W_1(x) = \psi(x)$ is the haar wavelet, both supported in [0,1].

Then we can obtain $W_{2\,n}$ by adding two 1/2-scaled versions of $W_n$ with distinct supports [0,1/2] and [1/2,1], and obtain $W_{2n+1}$ by subtracting the same versions of $W_n$.

Starting from more regular original wavelets, using a similar construction, we obtain smoothed versions of this system of *W*-functions, all with support in the interval [0, 2*N*-1].

**Examples**

```
% Compute the db2 Wn functions for n = 0 to 7, generating
% the db2 wavelet packets.
[wp,x] = wpfun('db2',7);

% Using some plotting commands,
% the following figure is generated.
```
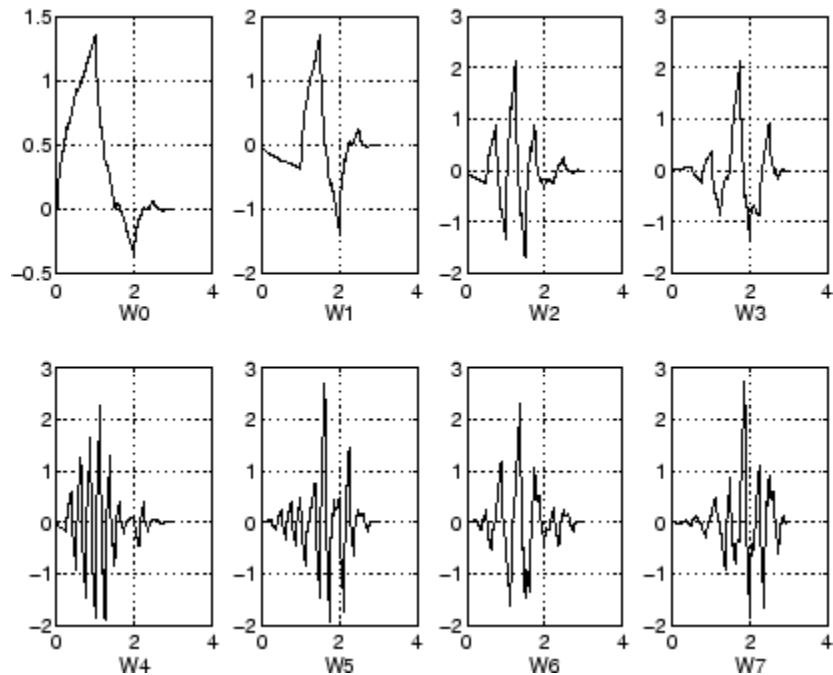
**References**
Coifman, R.R.; M.V. Wickerhauser (1992), "Entropy-based Algorithms for best basis selection," *IEEE Trans. on Inf. Theory*, vol. 38, 2, pp. 713–718.

Meyer, Y. (1993), *Les ondelettes. Algorithmes et applications*, Colin Ed., Paris, 2nd edition. (English translation: *Wavelets: Algorithms and applications*, SIAM).

Wickerhauser, M.V. (1991), "INRIA lectures on wavelet packet algorithms," *Proceedings ondelettes et paquets d'ondes*, 17–21 June, Rocquencourt, France, pp. 31–99.

Wickerhauser, M.V. (1994), *Adapted wavelet analysis from theory to software algorithms*, A.K. Peters.

**See Also**    wavefun | waveinfo

# wpjoin

| | |
|---|---|
| **Purpose** | Recompose wavelet packet |
| **Syntax** | T = wpjoin(*T*,N)<br>[T,X] = wpjoin(*T*,N)<br>T = wpjoin(*T*)<br>T = wpjoin(*T*,0)<br>[T,X] = wpjoin(*T*)<br>[T,X] = wpjoin(*T*,0) |
| **Description** | wpjoin is a one- or two-dimensional wavelet packet analysis function.<br><br>wpjoin updates the wavelet packet tree after the recomposition of a node.<br><br>The nodes are numbered from left to right and from top to bottom. The root index is 0.<br><br>T = wpjoin(*T*,N) returns the modified wavelet packet tree T corresponding to a recomposition of the node N.<br><br>[T,X] = wpjoin(*T*,N) also returns the coefficients of the node.<br><br>T = wpjoin(*T*) is equivalent to T = wpjoin(*T*,0).<br><br>[T,X] = wpjoin(*T*) is equivalent to [T,X] = wpjoin(*T*,0). |
| **Examples** | % The current extension mode is zero-padding (see dwtmode).<br><br>% Load signal.<br>load noisdopp; x = noisdopp;<br><br>% Decompose x at depth 3 with db1 wavelet packets.<br>wpt = wpdec(x,3,'db1');<br><br>% Plot wavelet packet tree wpt.<br>plot(wpt) |
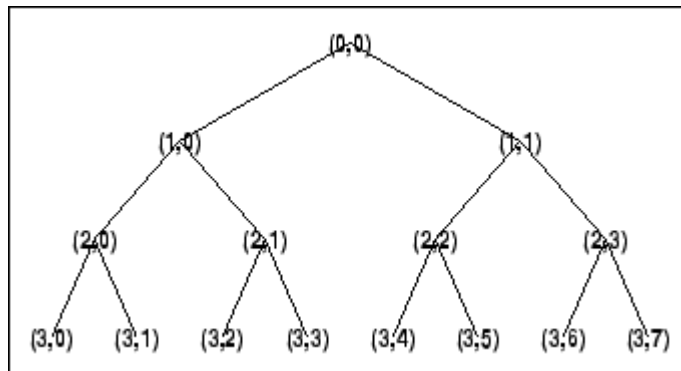
```
% Recompose packet (1,1) or 2
wpt = wpjoin(wpt,[1 1]);

% Plot wavelet packet tree wpt.
plot(wpt)
```
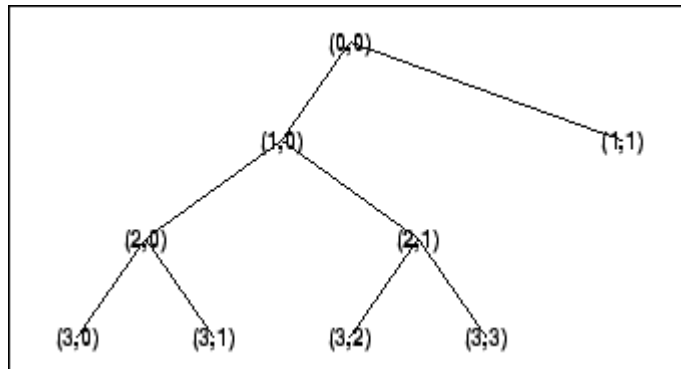


**See Also**   wpdec | wpdec2 | wpsplt

# wprcoef

| **Purpose** | Reconstruct wavelet packet coefficients |
|---|---|

**Syntax**

```
X = wprcoef(T,N)
X = wprcoef(T)
X = wprcoef(T,0)
```

**Description**

wprcoef is a one- or two-dimensional wavelet packet analysis function.

X = wprcoef(T,N) computes reconstructed coefficients of the node N of the wavelet packet tree *T*.

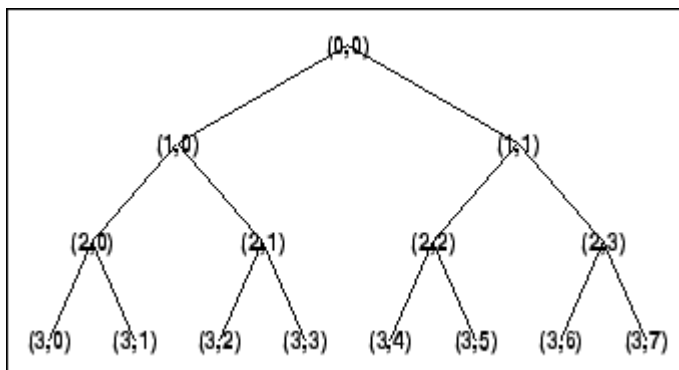X = wprcoef(*T*) is equivalent to X = wprcoef(*T*,0).

**Examples**

```
% The current extension mode is zero-padding (see dwtmode)

% Load signal.
load noisdopp; x = noisdopp;

figure(1); subplot(211);
plot(x); title('Original signal');

% Decompose x at depth 3 with db1 wavelet packets
% using Shannon entropy.
t = wpdec(x,3,'db1','shannon');

% Plot wavelet packet tree.
plot(t)
```
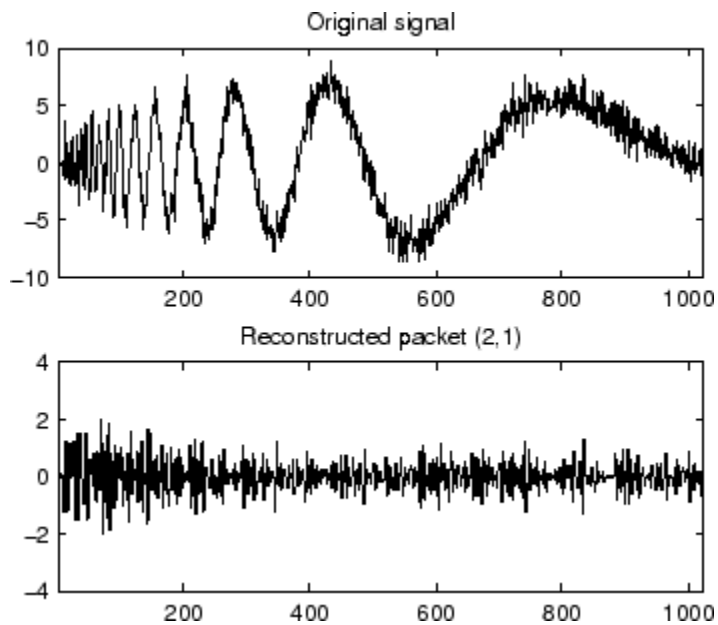
```
% Reconstruct packet (2,1).
rcfs = wprcoef(t,[2 1]);

figure(1); subplot(212);
plot(rcfs); title('Reconstructed packet (2,1)');
```

# wprcoef



Original signal

Reconstructed packet (2,1)

**See Also**     wpdec | wpdec2 | wprec | wprec2

**How To**     • "Reconstructing a Signal Approximation from a Node"

**Purpose**      Wavelet packet reconstruction 1-D

**Syntax**       X = wprec(*T*)
                 wprec(wpdec(X,'*wname*'))

**Description**  wprec is a one-dimensional wavelet packet analysis function.

                 X = wprec(*T*) returns the reconstructed vector X corresponding to a
                 wavelet packet tree *T*.

                 wprec is the inverse function of wpdec in the sense that the abstract
                 statement wprec(wpdec(X,'*wname*')) would give back X.

**See Also**     wpdec | wpdec2 | wpjoin | wprec2 | wpsplt

# wprec2

| | |
|---|---|
| **Purpose** | Wavelet packet reconstruction 2-D |
| **Syntax** | X = wprec2(*T*)<br>wprec2(wpdec2(X,'*wname*')) |
| **Description** | wprec2 is a two-dimensional wavelet packet analysis function. |
| | X = wprec2(*T*) returns the reconstructed matrix X corresponding to a wavelet packet tree *T*. |
| | wprec2 is the inverse function of wpdec2 in the sense that the abstract statement wprec2(wpdec2(X,'*wname*')) would give back X. |
| **Tips** | If *T* is obtained from an indexed image analysis or a truecolor image analysis, X is an m-by-n matrix or an m-by-n-by-3 array, respectively. |
| | For more information on image formats, see the image and imfinfo reference pages. |
| **See Also** | wpdec | wpdec2 | wpjoin | wprec | wpsplt |

**Purpose**      Wavelet packet spectrum

**Syntax**       [SPEC,TIMES,FREQ] = wpspectrum(WPT,Fs)
                 [...] = wpspectrum(WPT,Fs,'plot')
                 [...,TNFO] = wpspectrum(...)

**Description**  [SPEC,TIMES,FREQ] = wpspectrum(WPT,Fs) returns a matrix of
                 wavelet packet spectrum estimates, SPEC, for the binary wavelet packet
                 tree object, WPT. Fs is the sampling frequency in Hertz. SPEC is a
                 $2^J$-by-$N$ matrix where $J$ is the level of the wavelet packet transform and
                 $N$ is the length of the time series. TIMES is a 1-by-$N$ vector of times and
                 FREQ is a 1-by-$2^J$ vector of frequencies.

                 [...] = wpspectrum(WPT,Fs,'plot') displays the wavelet packet
                 spectrum.

                 [...,TNFO] = wpspectrum(...) returns the terminal nodes of the
                 wavelet packet tree in frequency order.

**Input Arguments**

**WPT**

WPT is a binary wavelet packet tree of class wptree.

**Fs**

Sampling frequency in Hertz as a scalar of class double.

> **Default:** 1

**plot**

The string 'plot' displays the wavelet packet spectrum. Enter 'plot'
after Fs to produce a plot of the wavelet packet spectrum.

**Output Arguments**

**SPEC**

Wavelet packet spectrum. SPEC is a $2^J$-by-$N$ matrix where $J$ is the
level of the wavelet packet transform and $N$ is the length of node 0
in the wavelet packet tree object.

# wpspectrum

The frequency spacing between the rows of SPEC is $Fs/2^{J+1}$.

### TIMES

Time vector. TIMES is a vector of times in seconds equal in length to node 0 of the wavelet packet tree object. The time spacing between elements is $1/Fs$.

### FREQ

Frequency vector. FREQ is a vector of frequencies of length $2^J$ where $J$ is the level of the wavelet packet tree object. The frequency spacing in FREQ is $Fs/2^{J+1}$.

### TNFO

Terminal nodes. TNFO is a vector of the terminal nodes of the wavelet packet tree object in frequency order.

**Definitions**    **Wavelet Packet Spectrum**

The wavelet packet spectrum contains the absolute values of the coefficients from the frequency-ordered terminal nodes of the input binary wavelet packet tree. The terminal nodes provide the finest level of frequency resolution in the wavelet packet transform. If $J$ denotes the level of the wavelet packet transform and $Fs$ is the sampling frequency, the terminal nodes approximate bandpass filters of the form:

$$[\frac{nFs}{2^{J+1}}, \frac{(n+1)Fs}{2^{J+1}}) \quad n = 0,1,2,3,\ldots 2^J - 1$$

At the terminal level of the wavelet packet tree, the transform divides the interval from 0 to the Nyquist frequency into bands of approximate width $Fs / 2^{J+1}$.

**Examples**    Wavelet packet spectrum for signal consisting of two sinusoids with disjoint support:

```
fs = 500;
t = 0:1/fs:4;
y = sin(32*pi*t).*(t<2) + sin(128*pi*t).*(t>=2);
subplot(2,1,1);
plot(t,y);
axis tight
title('Analyzed Signal');

% Wavelet packet spectrum
level = 6;
wpt = wpdec(y,level,'sym6');
subplot(2,1,2);
[S,T,F] = wpspectrum(wpt,fs,'plot');
```

Wavelet packet spectrum of chirp:

```
fs = 1000;
t = 0:1/fs:2;
% create chirp signal
y = sin(256*pi*t.^2);

% Plot the analyzed signal
subplot(2,1,1);
plot(t,y);
axis tight
title('Analyzed Signal');

% Wavelet packet spectrum
level = 6;
wpt = wpdec(y,level,'sym8');
subplot(2,1,2);
[S,T,F] = wpspectrum(wpt,fs,'plot');
```

# wpspectrum

**Algorithms**    wpspectrum computes the wavelet packet spectrum as follows:

- Extract the wavelet packet coefficients corresponding to the terminal nodes. Take the absolute value of the coefficients.

- Order the wavelet packet coefficients by frequency ordering.

- Determine the time extent on the original time axis corresponding to each wavelet packet coefficient. Repeat each wavelet packet coefficient to fill in the time gaps between neighboring wavelet packet coefficients and create a vector equal in length to node 0 of the wavelet packet tree object.

**References**    Wickerhauser, M.V. *Lectures on Wavelet Packet Algorithms*, Technical Report, Washington University, Department of Mathematics, 1992.

**See Also**      otnodes | wpdec

**How To**        • "Wavelet Packet Spectrum"

**Purpose**    Split (decompose) wavelet packet

**Syntax**    T = wpsplt(*T*,N)
[T,cA,cD] = wpsplt(*T*,N)
[T,cA,cH,cV,cD] = wpsplt(*T*,N)

**Description**    wpsplt is a one- or two-dimensional wavelet packet analysis function.

wpsplt updates the wavelet packet tree after the decomposition of a node.

T = wpsplt(*T*,N) returns the modified wavelet packet tree *T* corresponding to the decomposition of the node N.

For a one-dimensional decomposition,

[T,cA,cD] = wpsplt(*T*,N) with cA = approximation and cD = detail of node N.

For a two-dimensional decomposition,

[T,cA,cH,cV,cD] = wpsplt(*T*,N) with cA = approximation and cH,cV,c = horizontal, vertical, and diagonal details of node N.

**Examples**    % The current extension mode is zero-padding (see dwtmode).

% Load signal.
load noisdopp;
x = noisdopp;

% Decompose x at depth 3 with db1 wavelet packets.
wpt = wpdec(x,3,'db1');

% Plot wavelet packet tree wpt.
plot(wpt)
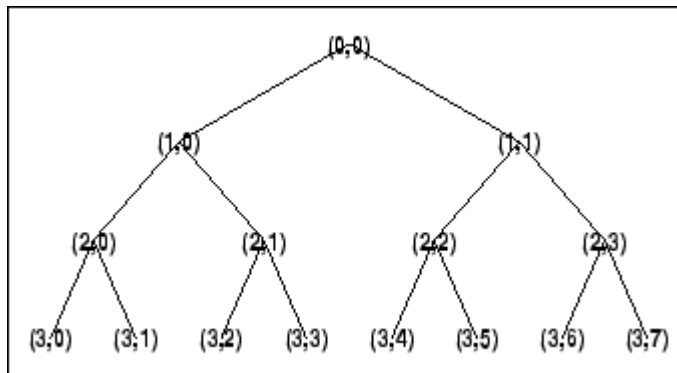
```
% Decompose packet (3,0).
wpt = wpsplt(wpt,[3 0]);
% or equivalently wpsplt(wpt,7).

% Plot wavelet packet tree wpt.
plot(wpt)
```
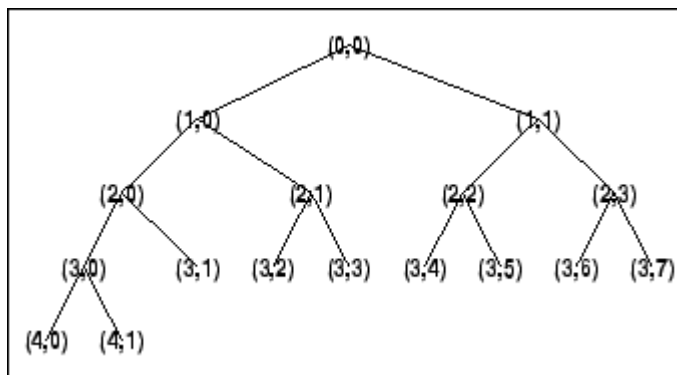


**See Also**        wavedec | wavedec2 | wpdec | wpdec2 | wpjoin

**Purpose**      Wavelet packet coefficients thresholding

**Syntax**       NT = wpthcoef(*T*,KEEPAPP,SORH,THR)

**Description**   wpthcoef is a one- or two-dimensional de-noising and compression utility.

NT = wpthcoef(*T*,KEEPAPP,SORH,THR) returns a new wavelet packet tree NT obtained from the wavelet packet tree *T* by coefficients thresholding.

If KEEPAPP = 1, approximation coefficients are not thresholded; otherwise, they can be thresholded.

If SORH = 's', soft thresholding is applied; if SORH = 'h', hard thresholding is applied (see wthresh for more information).

THR is the threshold value.

**See Also**     wpdec | wpdec2 | wpdencmp | wthresh

# wptree

| | |
|---|---|
| **Purpose** | WPTREE constructor |
| **Syntax** | T = wptree(ORDER,DEPTH,X,WNAME,ENT_TYPE,PARAMETER)<br>T = wptree(ORDER,DEPTH,X,WNAME)<br>T = wptree(ORDER,DEPTH,X,WNAME,'shannon')<br>T = wptree(ORDER,DEPTH,X,WNAME,ENT_TYPE,ENT_PAR,USERDATA) |

**Description**  T = wptree(ORDER,DEPTH,X,WNAME,ENT_TYPE,PARAMETER) returns a complete wavelet packet tree T.

ORDER is an integer representing the order of the tree (the number of "children" of each non terminal node). ORDER must be equal to 2 or 4.

If ORDER = 2, T is a WPTREE object corresponding to a wavelet packet decomposition of the vector (signal) X, at level DEPTH with a particular wavelet WNAME.

If ORDER = 4, T is a WPTREE object corresponding to a wavelet packet decomposition of the matrix (image) X, at level DEPTH with a particular wavelet WNAME.

ENT_TYPE is a string containing the entropy type and ENT_PAR is an optional parameter used for entropy computation ( see wentropy, wpdec, or wpdec2 for more information).

T = wptree(ORDER,DEPTH,X,WNAME) is equivalent to T = wptree(ORDER,DEPTH,X,WNAME,'shannon')

With  T = wptree(ORDER,DEPTH,X,WNAME,ENT_TYPE,ENT_PAR,USERDATA) you may set a userdata field.

The function wptree returns a WPTREE object.

For more information on object fields, see the get function or type

```
help wptree/get
```

Class WPTREE (Parent class: DTREE)

**Fields**

| | |
|---|---|
| 'dtree' | DTREE parent object |
| 'wavInfo' | Structure (wavelet information) |
| 'entInfo' | Structure (entropy information) |

The wavelet information structure, 'wavInfo', contains

| | |
|---|---|
| 'wavName' | Wavelet name |
| 'Lo_D' | Low Decomposition filter |
| 'Hi_D' | High Decomposition filter |
| 'Lo_R' | Low Reconstruction filter |
| 'Hi_R' | High Reconstruction filter |

The entropy information structure, 'entInfo', contains

| | |
|---|---|
| 'entName' | Entropy name |
| 'entPar' | Entropy parameter |

Fields from the DTREE parent object:

| | |
|---|---|
| 'allNI' | All nodes information |

'allNI' is an array of size nbnode by 5, which contains

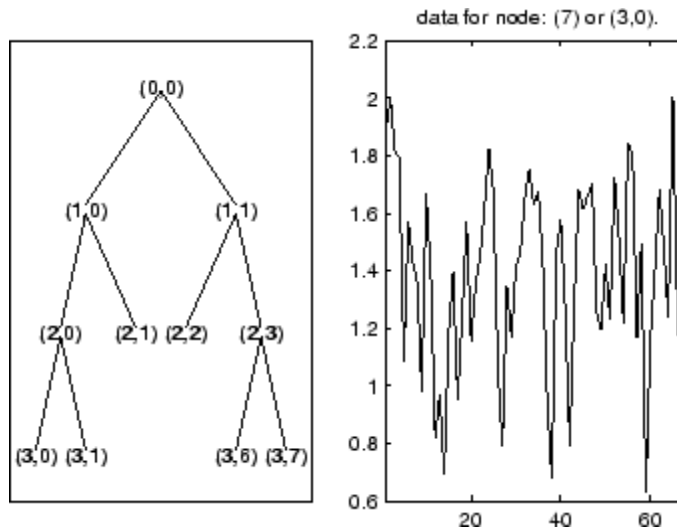| | |
|---|---|
| ind | Index |
| size | Size of data |
| ent | Entropy |
| ento | Optimal entropy |

Each line is built based on the following scheme:



**Examples**

```
% Create a wavelet packet tree.
x = rand(1,512);
t = wptree(2,3,x,'db3');
t = wpjoin(t,[4;5]);

% Plot tree t4.
plot(t);

% Click the node (3,0), (see the plot function).
```



**See Also**     dtree | ntree

**Purpose**        Plot wavelet packets colored coefficients

**Syntax**         wpviewcf(*T*,CMODE)
                   wpviewcf(*T*,CMODE,NBCOL)

**Description**    wpviewcf(*T*,CMODE) plots the colored coefficients for the terminal nodes of the tree *T*.

                   *T* is a wavelet packet tree and CMODE is an integer, which represents the color mode. The color modes are listed in the table below.

| Color Mode | Description |
|---|---|
| 1 | Frequency order – Global coloration – Absolute values |
| 2 | Frequency order – By level – Absolute values |
| 3 | Frequency order – Global coloration – Values |
| 4 | Frequency order – By level coloration – Values |
| 5 | Natural order – Global coloration – Absolute values |
| 6 | Natural order – By level – Absolute values |
| 7 | Natural order – Global coloration – Values |
| 8 | Natural order – By level coloration – Values |

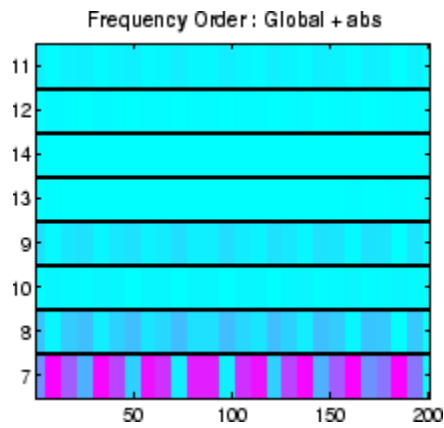wpviewcf(*T*,CMODE,NBCOL) uses NBCOL colors.

**Examples**
```
% Create a wavelet packet tree.
x = sin(8*pi*[0:0.005:1]);
t = wpdec(x,3,'db1');

% Plot tree t.
% Click the node (3,0), (see the plot function)
plot(t);
```

data for node: (7) or (3,0).
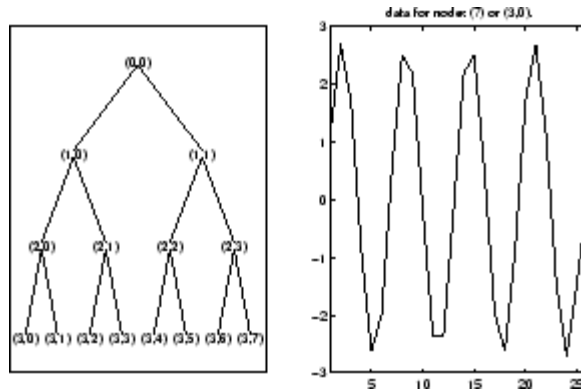
```
% Plot the colored wavelet packet coefficients.
wpviewcf(t,1);
```



Frequency Order: Global + abs

**See Also**      wpdec

**Purpose**    Reconstruct single branch from 1-D wavelet coefficients

**Syntax**     X = wrcoef('*type*',C,L,'*wname*',N)
X = wrcoef('*type*',C,L,Lo_R,Hi_R,N)
X = wrcoef('*type*',C,L,'*wname*')
X = wrcoef('*type*',C,L,Lo_R,Hi_R)

**Description**    wrcoef reconstructs the coefficients of a one-dimensional signal, given
a wavelet decomposition structure (C and L) and either a specified
wavelet ('*wname*', see wfilters for more information) or specified
reconstruction filters (Lo_R and Hi_R).

X = wrcoef('*type*',C,L,'*wname*',N) computes the vector of
reconstructed coefficients, based on the wavelet decomposition structure
[C,L] (see wavedec for more information), at level N. '*wname*' is a string
containing the wavelet name.

Argument '*type*' determines whether approximation ('*type*' = 'a')
or detail ('*type*' = 'd') coefficients are reconstructed. When '*type*'
= 'a', N is allowed to be 0; otherwise, a strictly positive number N is
required. Level N must be an integer such that N ≤ length(L)-2.

X = wrcoef('*type*',C,L,Lo_R,Hi_R,N) computes coefficients as
above, given the reconstruction filters you specify.

X = wrcoef('*type*',C,L,'*wname*') and X =
wrcoef('*type*',C,L,Lo_R,Hi_R) reconstruct coefficients of maximum
level N = length(L)-2.

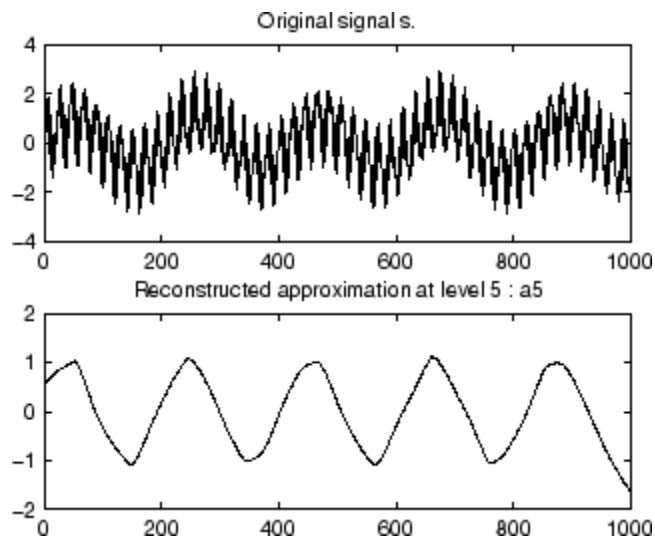**Examples**    % The current extension mode is zero-padding (see dwtmode).

% Load a one-dimensional signal.
load sumsin; s = sumsin;

% Perform decomposition at level 5 of s using sym4.
[c,l] = wavedec(s,5,'sym4');

% Reconstruct approximation at level 5,

```
% from the wavelet decomposition structure [c,l].
a5 = wrcoef('a',c,l,'sym4',5);

% Using some plotting commands,
% the following figure is generated.
```



**See Also**     appcoef | detcoef | wavedec

**Purpose**      Reconstruct single branch from 2-D wavelet coefficients

**Syntax**       X = wrcoef2('*type*',C,S,'*wname*',N)
                 X = wrcoef2('*type*',C,S,Lo_R,Hi_R,N)
                 X = wrcoef2('*type*',C,S,'*wname*')
                 X = wrcoef2('*type*',C,S,Lo_R,Hi_R)

**Description**  wrcoef2 is a two-dimensional wavelet analysis function. wrcoef2
                 reconstructs the coefficients of an image.

                 X = wrcoef2('*type*',C,S,'*wname*',N) computes the matrix of
                 reconstructed coefficients of level N, based on the wavelet decomposition
                 structure [C,S] (see wavedec2 for more information).

                 '*wname*' is a string containing the name of the wavelet (see wfilters
                 for more information). If '*type*' = 'a', approximation coefficients are
                 reconstructed; otherwise if '*type*' = 'h' ('v' or 'd', respectively),
                 horizontal (vertical or diagonal, respectively) detail coefficients are
                 reconstructed.

                 Level N must be an integer such that $0 \leq N \leq$ size(S,1)-2 if '*type*' =
                 'a' and such that $1 \leq N \leq$ size(S,1)-2 if '*type*' = 'h', 'v', or 'd'.

                 Instead of giving the wavelet name, you can give the filters.

                 For X = wrcoef2('*type*',C,S,Lo_R,Hi_R,N), Lo_R is the
                 reconstruction low-pass filter and Hi_R is the reconstruction high-pass
                 filter.

                 X = wrcoef2('*type*',C,S,'*wname*') or X =
                 wrcoef2('*type*',C,S,Lo_R,Hi_R) reconstruct coefficients of maximum
                 level N = size(S,1)-2.

**Tips**         If C and S are obtained from an indexed image analysis (respectively
                 a truecolor image analysis) then X is an m-by-n matrix (respectively an
                 m-by-n-by-3 array).

                 For more information on image formats, see the reference pages of
                 image and imfinfo functions.

**Examples**

```
% The current extension mode is zero-padding (see dwtmode).

% Load an image.
load woman;
% X contains the loaded image.

% Perform decomposition at level 2
% of X using sym5.
[c,s] = wavedec2(X,2,'sym5');

% Reconstruct approximations at
% levels 1 and 2, from the wavelet
% decomposition structure [c,s].
a1 = wrcoef2('a',c,s,'sym5',1);
a2 = wrcoef2('a',c,s,'sym5',2);

% Reconstruct details at level 2,
% from the wavelet decomposition
% structure [c,s].
% 'h' is for horizontal,
% 'v' is for vertical,
% 'd' is for diagonal.
hd2 = wrcoef2('h',c,s,'sym5',2);
vd2 = wrcoef2('v',c,s,'sym5',2);
dd2 = wrcoef2('d',c,s,'sym5',2);

% All these images are of same size sX.
sX = size(X)

sX =
 256 256

sa1 = size(a1)

sa1 =
 256 256
```

```
shd2 = size(hd2)

shd2 =
 256 256
```

**See Also**     appcoef2 | detcoef2 | wavedec2

**wrev**

| | |
|---|---|
| **Purpose** | Flip vector |
| **Syntax** | `Y = wrev(X)` |
| **Description** | `wrev` is a general utility. |
| | `Y = wrev(X)` reverses the vector X. |
| **Examples** | `v = [1 2 3];`<br>`wrev(v)`<br>`wrev(v')` |
| **See Also** | `fliplr` \| `flipud` |

**Purpose**      Write values in WPTREE fields

**Syntax**        `T = write(T,'cfs',NODE,COEFS)`
              `T = write(T,'cfs',N1,CFS1,'cfs',N2,CFS2, ...)`

**Description**   `T = write(T,'cfs',NODE,COEFS)` writes coefficients for the terminal node `NODE`.

               `T = write(T,'cfs',N1,CFS1,'cfs',N2,CFS2, ...)` writes coefficients `CFS1, CFS2, ...` for the terminal nodes `N1, N2, ...`.

---

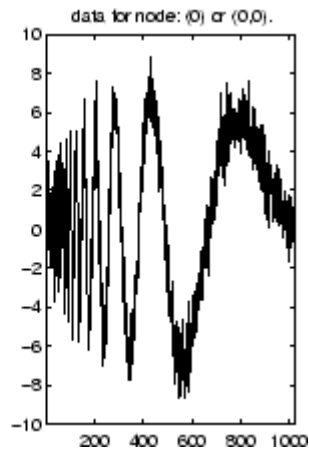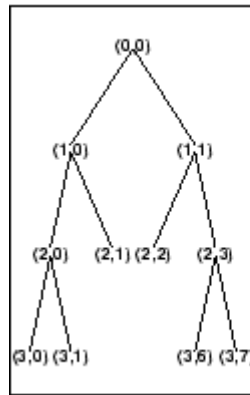**Caution** The coefficients values must have the suitable size. You can use `S = read(T,'sizes',NODE)` or `S = read(T,'sizes',[N1;N2; ...])` in order to get those sizes.

---

**Examples**    
```
% Create a wavelet packet tree.
load noisdopp; x = noisdopp;
t = wpdec(x,3,'db3');
t = wpjoin(t,[4;5]);

% Plot tree t and click the node (0,0) (see the plot function).
plot(t);
```
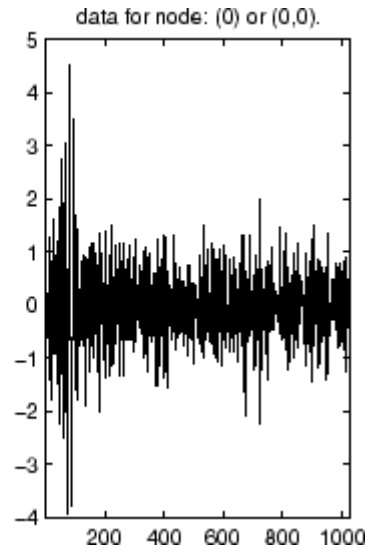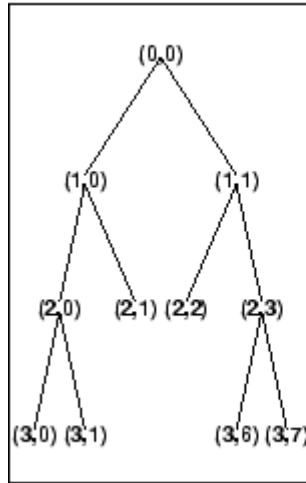
data for node: (0) or (0,0).

```
% Write values.
sNod = read(t,'sizes',[4,5,7]);
cfs4 = zeros(sNod(1,:));
cfs5 = zeros(sNod(2,:));
cfs7 = zeros(sNod(3,:));
t = write(t,'cfs',4,cfs4,'cfs',5,cfs5,'cfs',7,cfs7);

% Plot tree t and click the node (0,0) (see the plot function).
plot(t)
```

data for node: (0) or (0,0).

**See Also**     disp | get | read | set

# wscalogram

| | |
|---|---|
| **Purpose** | Scalogram for continuous wavelet transform |
| **Syntax** | SC = wscalogram(TYPEPLOT,COEFS)<br>SC = wscalogram(TYPEPLOT,COEFS,'PropName1',PropVal1,...) |

**Description**

SC = wscalogram(TYPEPLOT,COEFS) computes the scalogram SC which represents the percentage of energy for each coefficient. COEFS is the matrix of the continuous wavelet coefficients (see cwt).

The scalogram is obtained by computing:

```
S = abs(coefs.*coefs); SC = 100*S./sum(S(:))
```

When TYPEPLOT is equal to 'image', a scaled image of scalogram is displayed. When TYPEPLOT is equal to 'contour', a contour representation of scalogram is displayed. Otherwise, the scalogram is returned without plot representation.

SC = wscalogram(TYPEPLOT,COEFS,'PropName1',PropVal1,...) allows you to modify some properties. The valid choices for PropName are:

| | |
|---|---|
| 'scales' | Scales used for the CWT. |
| 'ydata' | Signal used for the CWT. |
| 'xdata' | *x* values corresponding to the signal values. |
| 'power' | Positive real value. Default value is zero. |

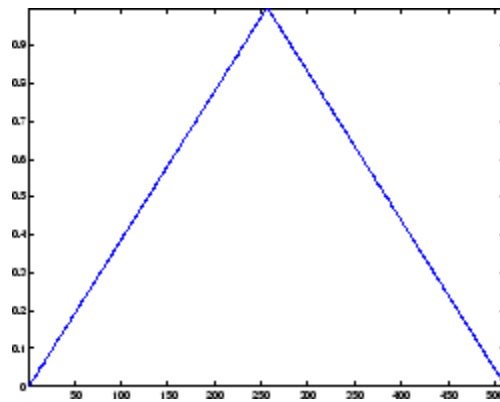If power > 0, coefficients are first normalized

```
coefs(k,:) = coefs(k,:)/(scales(k)^power)
```

and then the scalogram is computed as explained above.

**Examples**

```
% Compute signal s
t = linspace(-1,1,512);
```
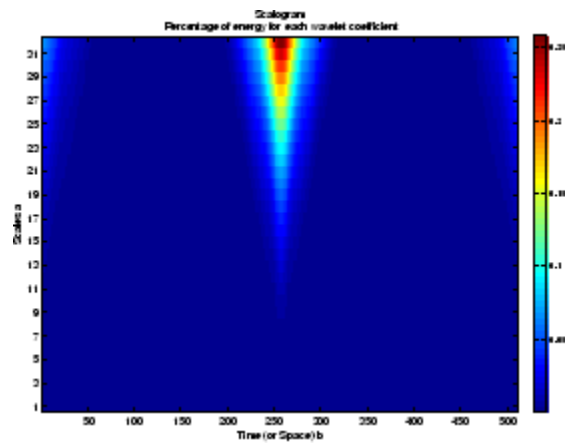
```
s = 1-abs(t);

% Plot signal s
figure;
plot(s), axis tight
```
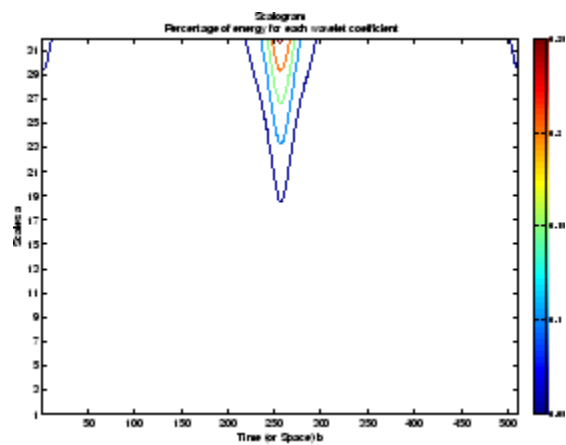


```
% Compute coefficients COEFS using cwt
COEFS = cwt(s,1:32,'cgau4');

% Compute and plot the scalogram (image option)
figure;
SC = wscalogram('image',COEFS);
```

# wscalogram



```
% Compute and plot the scalogram (contour option)
figure;
SC = wscalogram('contour',COEFS);
```



**See Also**    cwt

**Purpose**        WTBO constructor

**Syntax**         OBJ = wtbo
                   OBJ = wtbo(USERDATA)

**Description**     OBJ = wtbo returns a WTBO object. Any object in the Wavelet Toolbox
                   software is parented by a WTBO object.

                   With OBJ = wtbo(USERDATA) you can set a userdata field.

                   Class WTBO (Parent class: none)

**Fields**           wtboInfo           Object information (not used in the current
                                        version of the toolbox)

                     ud                 Userdata field

# wtbxmngr

| **Purpose** | Wavelet Toolbox manager |
| --- | --- |

**Syntax**

```
wtbxmngr(OPTION)
V = wtbxmngr('version')
```

**Description**  wtbxmngr or wtbxmngr('version') displays the current version of Wavelet Toolbox software.

wtbxmngr(OPTION) sets a toolbox option. Available options are

| Option | Description |
| --- | --- |
| 'LargeFonts' | Sets the size of future-created figures to use large fonts. |
| 'DefaultSize' | Restores the default figure size for future-created figures. |
| 'FigRatio' | Returns the current figure ratio value. |
| 'FigRatio',ratio | Changes the size of future-created figures by multiplying the default size by the specified ratio, where ratio must be between 0.75 and 1.25. |

V = wtbxmngr('version') saves the current version of the toolbox to variable V.

**Examples**

```
wtbxmngr('version')

*************************************
**  Wavelet Toolbox Version: V3.1  **
*************************************

wtbxmngr('FigRatio')      % Display the current figure ratio
wtbxmngr('FigRatio',1.25) % Set the figure ratio to 1.25
wtbxmngr('FigRatio')      % Display the current figure ratio
wtbxmngr('DefaultSize')   % Return to the default figure ratio
```
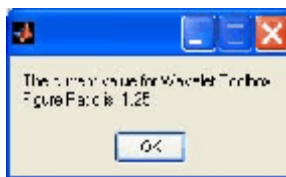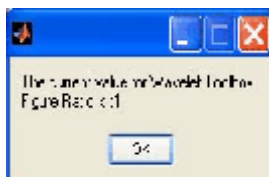
# wthcoef

| **Purpose** | 1-D wavelet coefficient thresholding |
|---|---|

**Syntax**

```
NC = wthcoef('d',C,L,N,P)
NC = wthcoef('d',C,L,N)
NC = wthcoef('a',C,L)
NC = wthcoef('t',C,L,N,T,SORH)
```

**Description**    wthcoef thresholds wavelet coefficients for the denoising or compression of a 1-D signal.

NC = wthcoef('d',C,L,N,P) returns coefficients obtained from the wavelet decomposition structure [C,L] (see wavedec for more information), by rate compression defined in vectors N and P. N contains the detail levels to be compressed and P the corresponding percentages of lower coefficients to be set to zero. N and P must be of same length. Vector N must be such that $1 \leq N(i) \leq length(L)-2$.

NC = wthcoef('d',C,L,N) returns coefficients obtained from [C,L] by setting all the coefficients of detail levels defined in N to zero.

NC = wthcoef('a',C,L) returns coefficients obtained by setting approximation coefficients to zero.

NC = wthcoef('t',C,L,N,T,SORH) returns coefficients obtained from the wavelet decomposition structure [C,L] by soft (if SORH ='s') or hard (if SORH ='h') thresholding (see wthresh for more information) defined in vectors N and T. N contains the detail levels to be thresholded and T the corresponding thresholds. N and T must be of the same length.

[NC,L] is the modified wavelet decomposition structure.

**See Also**    wavedec | wthresh

**Purpose**    Wavelet coefficient thresholding 2-D

**Syntax**
```
NC = wthcoef2('type',C,S,N,T,SORH)
NC = wthcoef2('type',C,S,N)
NC = wthcoef2('a',C,S)
NC = wthcoef2('t',C,S,N,T,SORH)
```

**Description**    wthcoef2 is a two-dimensional de-noising and compression oriented function.

For '*type*' = 'h' ('v' or 'd'), NC = wthcoef2('*type*',C,S,N,T,SORH) returns the horizontal (vertical or diagonal, respectively) coefficients obtained from the wavelet decomposition structure [C,S] (see wavedec2 for more information), by soft (if SORH ='s') or hard (if SORH ='h') thresholding defined in vectors N and T. N contains the detail levels to be thresholded and T the corresponding thresholds. N and T must be of the same length. The vector N must be such that $1 \leq N(i) \leq size(S,1)-2$.

For '*type*' = 'h' ('v' or 'd'), NC = wthcoef2('*type*',C,S,N) returns the horizontal (vertical or diagonal, respectively) coefficients obtained from [C,S] by setting all the coefficients of detail levels defined in N to zero.

NC = wthcoef2('a',C,S) returns the coefficients obtained by setting approximation coefficients to zero.

NC = wthcoef2('t',C,S,N,T,SORH) returns the detail coefficients obtained from the wavelet decomposition structure [C,S] by soft (if SORH ='s') or hard (if SORH ='h') thresholding (see wthresh for more information) defined in vectors N and T. N contains the detail levels to be thresholded and T the corresponding thresholds which are applied in the three detail orientations. N and T must be of the same length.

[NC,S] is the modified wavelet decomposition structure.

**See Also**    wavedec2 | wthresh

# wthresh

**Purpose**        Soft or hard thresholding

**Syntax**         Y = wthresh(X,SORH,T)
                   Y = wthresh(X,'s',T)
                   Y = wthresh(X,'h',T)

**Description**    Y = wthresh(X,SORH,T) returns the soft (if SORH = 's') or hard (if
                   SORH = 'h') thresholding of the input vector or matrix X. *T* is the
                   threshold value.

                   Y = wthresh(X,'s',T) returns $Y = sign(X) \cdot (|X| - T)_+$, soft
                   thresholding is wavelet shrinkage ( $(x)_+ = 0$ if $x < 0$; $(x)_+ = x$, if $x \geq 0$ ).

                   Y = wthresh(X,'h',T) returns $Y = X \cdot 1_{(|X| > T)}$, hard thresholding is
                   cruder.

**Examples**       % Generate signal and set threshold.
                   y = linspace(-1,1,100);
                   thr = 0.4;

                   % Perform hard thresholding.
                   ythard = wthresh(y,'h',thr);

                   % Perform soft thresholding.
                   ytsoft = wthresh(y,'s',thr);

                   % Using some plotting commands,
                   % the following figure is generated.
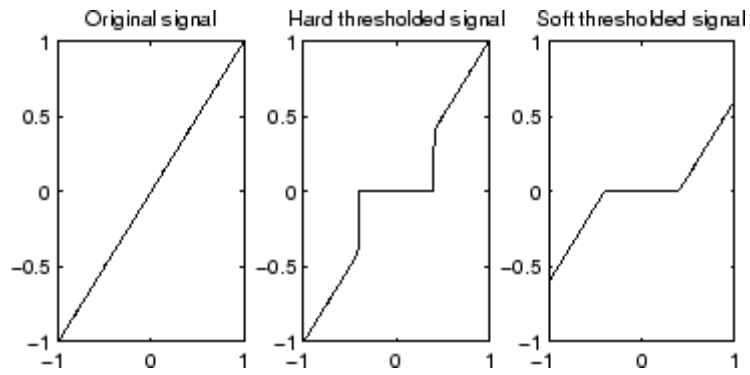
**See Also**        wden | wdencmp | wpdencmp

# wthrmngr

| | |
|---|---|
| **Purpose** | Threshold settings manager |
| **Syntax** | THR = wthrmngr(OPTION,METHOD,VARARGIN) |
| **Description** | THR = wthrmngr(OPTION,METHOD,VARARGIN) returns a global threshold or level dependent thresholds depending on OPTION. The inputs, VARARGIN, depend on the OPTION and METHOD values. |

This file returns the thresholds used throughout the Wavelet Toolbox software for de-noising and compression tools (command line files or GUI tools).

Valid options for the METHOD parameter are listed in the table below.

| METHOD | Description |
|---|---|
| 'scarcehi' | See wdcbm or wdcbm2 when used with 'high' predefined value of parameter M. |
| 'scarceme' | See wdcbm or wdcbm2 when used with 'medium' predefined value of parameter M. |
| 'scarcelo' | See wdcbm or wdcbm2 when used with 'low' predefined value of parameter M. |
| 'sqtwolog' | See 'sqtwolog' option in thselect, and see also wden. |
| 'sqtwologuwn' | See 'sqtwolog' option in thselect, and see also wden when used with 'sln' option. |
| 'sqtwologswn' | See 'sqtwolog' option in thselect, and see also wden when used with 'mln' option. |
| 'rigsure' | See 'rigsure' option in thselect, and see also wden. |
| 'heursure' | See 'heursure' option in thselect, and see also wden. |
| 'minimaxi' | See 'minimaxi' option in thselect, and see also wden. |

| METHOD | Description |
|--------|-------------|
| `'penalhi'` | See wbmpen or wpbmpen when used with `'high'` value of parameter ALPHA. |
| `'penalme'` | See wbmpen or wpbmpen when used with `'medium'` value of parameter ALPHA. |
| `'penallo'` | See wbmpen or wpbmpen when used with `'low'` value of parameter ALPHA. |
| `'rem_n0'` | This option returns a threshold close to 0. A typical THR value is median(abs(coefficients)). |
| `'bal_sn'` | This option returns a threshold such that the percentages of retained energy and number of zeros are the same. |
| `'sqrtbal_sn'` | This option returns a threshold equal to the square root of the value such that the percentages of retained energy and number of zeros are the same. |

### Discrete Wavelet 1-D Options

For 1–D wavelet transforms, the expansion coefficients are in the vector C and the lengths of the expansion coefficient vectors are stored in L.

#### Compression using a global threshold.

X is the signal to be compressed and [C,L] is the wavelet decomposition structure of the signal to be compressed.

```
THR = wthrmngr('dw1dcompGBL','rem_n0',X)
THR = wthrmngr('dw1dcompGBL','bal_sn',X)
```

#### Compression using level dependent thresholds.

X is the signal to be compressed and [C,L] is the wavelet decomposition structure of the signal to be compressed.

ALFA is a sparsity parameter (see wdcbm for more information).

```
THR = wthrmngr('dw1dcompLVL','scarcehi',C,L,ALFA)
      ALFA must be such that 2.5 < ALFA < 10
THR = wthrmngr('dw1dcompLVL','scarceme',C,L,ALFA)
      ALFA must be such that 1.5 < ALFA < 2.5
THR = wthrmngr('dw1dcompLVL','scarcelo',C,L,ALFA)
      ALFA must be such that 1 < ALFA < 2
```

**De-noising using level dependent thresholds.**
[C,L] is the wavelet decomposition structure of the signal to be
de-noised, SCAL defines the multiplicative threshold rescaling (see wden
for more information) and ALFA is a sparsity parameter (see wbmpen for
more information).

```
THR = wthrmngr('dw1ddenoLVL','sqtwolog',C,L,SCAL)
THR = wthrmngr('dw1ddenoLVL','rigrsure',C,L,SCAL)
THR = wthrmngr('dw1ddenoLVL','heursure',C,L,SCAL)
THR = wthrmngr('dw1ddenoLVL','minimaxi',C,L,SCAL)
THR = wthrmngr('dw1ddenoLVL','penalhi',C,L,ALFA)
      ALFA must be such that 2.5 < ALFA < 10
THR = wthrmngr('dw1ddenoLVL','penalme',C,L,ALFA)
      ALFA must be such that 1.5 < ALFA < 2.5
THR = wthrmngr('dw1ddenoLVL','penallo',C,L,ALFA)
      ALFA must be such that 1 < ALFA < 2
```

**Discrete Stationary Wavelet 1-D Options**

**De-noising using level dependent thresholds.**
SWTDEC is the stationary wavelet decomposition structure of the signal
to be de-noised, SCAL defines the multiplicative threshold rescaling
(see wden for more information) and ALFA is a sparsity parameter (see
wbmpen for more information).

```
THR = wthrmngr('sw1ddenoLVL',METHOD,SWTDEC,SCAL)
THR = wthrmngr('sw1ddenoLVL',METHOD,SWTDEC,ALFA)
```

The options for METHOD are the same as in the 'dw1ddenoLVL'case.

### Discrete Wavelet 2-D Options

For 2–D wavelet transforms, the expansion coefficients are in the vector C and the size of the coefficient matrices at each level is stored in S.

#### Compression using a global threshold.
X is the image to be compressed and [C,S] is the wavelet decomposition structure of the image to be compressed.

```
THR = wthrmngr('dw2dcompGBL','rem_n0',X)
THR = wthrmngr('dw2dcompGBL','bal_sn',C,S)
THR = wthrmngr('dw2dcompGBL','sqrtbal_sn',C,S)
```

#### Compression using level dependent thresholds.
X is the image to be compressed and [C,S] is the wavelet decomposition structure of the image to be compressed. ALFA is a sparsity parameter (see wdcbm2 for more information).

```
THR = wthrmngr('dw2dcompLVL','scarcehi',C,S,ALFA)
      ALFA must be such that 2.5 < ALFA < 10
THR = wthrmngr('dw2dcompLVL','scarceme',C,S,ALFA)
      ALFA must be such that 1.5 < ALFA < 2.5
THR = wthrmngr('dw2dcompLVL','scarcelo',C,S,ALFA)
      ALFA must be such that 1 < ALFA < 2
```

#### De-noising using level dependent thresholds.
[C,S] is the wavelet decomposition structure of the image to be de-noised, SCAL defines the multiplicative threshold rescaling (see wden for more information) and ALFA is a sparsity parameter (see wbmpen for more information).

```
THR = wthrmngr('dw2ddenoLVL','penalhi',C,S,ALFA)
      ALFA must be such that 2.5 < ALFA < 10
THR = wthrmngr('dw2ddenoLVL','penalme',C,S,ALFA)
      ALFA must be such that 1.5 < ALFA < 2.5
THR = wthrmngr('dw2ddenoLVL','penallo',C,S,ALFA)
      ALFA must be such that 1 < ALFA < 2
THR = wthrmngr('dw2ddenoLVL','sqtwolog',C,S,SCAL)
THR = wthrmngr('dw2ddenoLVL','sqrtbal_sn',C,S)
```

### Discrete Stationary Wavelet 2-D Options

#### De-noising using level dependent thresholds.

SWTDEC is the stationary wavelet decomposition structure of the image to be de-noised, SCAL defines the multiplicative threshold rescaling (see wden for more information) and ALFA is a sparsity parameter (see wbmpen for more information).

```
THR = wthrmngr('sw2ddenoLVL',METHOD,SWTDEC,SCAL)
THR = wthrmngr('sw2ddenoLVL',METHOD,SWTDEC,ALFA)
```

The options for METHOD are the same as in the 'dw2ddenoLVL' case.

### Discrete Wavelet Packet 1-D Options

#### Compression using a global threshold.

X is the signal to be compressed and WPT is the wavelet packet decomposition structure of the signal to be compressed.

```
THR = wthrmngr('wp1dcompGBL','bal_sn',WPT)
THR = wthrmngr('wp1dcompGBL','rem_n0',X)
```

#### De-noising using a global threshold.

WPT is the wavelet packet decomposition structure of the signal to be de-noised.

```
THR = wthrmngr('wp1ddenoGBL','sqtwologuwn',WPT)
THR = wthrmngr('wp1ddenoGBL','sqtwologswn',WPT)
THR = wthrmngr('wp1ddenoGBL','bal_sn',WPT)
THR = wthrmngr('wp1ddenoGBL','penalhi',WPT)
     see wbmpen with ALFA = 6.25
THR = wthrmngr('wp1ddenoGBL','penalme',WPT)
     see wbmpen with ALFA = 2
THR = wthrmngr('wp1ddenoGBL','penallo',WPT)
     see wbmpen with ALFA = 1.5
```

### Discrete Wavelet Packet 2-D Options

#### Compression using a global threshold.

X is the image to be compressed and WPT is the wavelet packet decomposition structure of the image to be compressed.

```
THR = wthrmngr('wp2dcompGBL','bal_sn',WPT)
THR = wthrmngr('wp2dcompGBL','rem_n0',X)
THR = wthrmngr('wp2dcompGBL','sqrtbal_sn',WPT)
```

#### De-noising using a global threshold.

WPT is the wavelet packet decomposition structure of the image to be de-noised.

```
THR = wthrmngr('wp2ddenoGBL','sqtwologuwn',WPT)
THR = wthrmngr('wp2ddenoGBL','sqtwologswn',WPT)
THR = wthrmngr('wp2ddenoGBL','sqrtbal_sn',WPT)
THR = wthrmngr('wp2ddenoGBL','penalhi',WPT)
    see wbmpen with ALFA = 6.25
THR = wthrmngr('wp2ddenoGBL','penalme',WPT)
    see wbmpen with ALFA = 2
THR = wthrmngr('wp2ddenoGBL','penallo',WPT)
    see wbmpen with ALFA = 1.5
```

**Examples**

### Level-Independent Threshold — Stationary Wavelet Transform

This example uses a level-independent threshold based on the finest-scale wavelet coefficients to implement hard thresholding with the stationary wavelet transform.

Load the noisy blocks signal. Obtain the stationary wavelet transform down to level 5 using the Haar wavelet.

```
load noisbloc;
L = 5;
swc = swt(noisbloc,L,'db1');
```

Make a copy of the wavelet transform coefficients. Determine the Donoho-Johnstone universal threshold based on the first-level detail coefficients. Using the `'sln'` option, wthrmngr returns a 1-by-L vector with every element equal to the same value. Take the mean of the vector to obtain a scalar threshold.

```
swcnew = swc;
ThreshSL = mean(wthrmngr('sw1ddenoLVL','sqtwolog',swc,'sln'));
```

Use the universal threshold to implement hard thresholding. The same threshold is applied to the wavelet coefficients at every level.

```
for jj = 1:L
swcnew(jj,:) = wthresh(swc(jj,:),'h',ThreshSL);
end
```

Invert the stationary wavelet transform on the thresholded coefficients, swcnew. Plot the original signal and the denoised signal for comparison.

```
noisbloc_denoised = iswt(swcnew,'db1');
plot(noisbloc); hold on;
plot(noisbloc_denoised,'r','linewidth',2);
```

### Level-Dependent Threshold — Stationary Wavelet Transform

This example uses a level-dependent threshold derived from the wavelet coefficients at each scale to implement hard thresholding with the stationary wavelet transform.

Load the noisy blocks signal. Obtain the stationary wavelet transform down to level 5 using the Haar wavelet.

```
load noisbloc;
L = 5;
swc = swt(noisbloc,L,'db1');
```

Make a copy of the wavelet transform coefficients. Determine the Donoho-Johnstone universal threshold based on the detail coefficients for each scale. Using the `'mln'` option, wthrmngr returns a 1-by-L

vector with each element of the vector equal to the universal threshold for the corresponding scale.

```
swcnew = swc;
ThreshML = wthrmngr('sw1ddenoLVL','sqtwolog',swc,'mln');
```

Use the universal thresholds to implement hard thresholding. The thresholds are applied in a scale-dependent manner.

```
for jj = 1:L
swcnew(jj,:) = wthresh(swc(jj,:),'h',ThreshML(jj));
end
```

Invert the stationary wavelet transform on the thresholded coefficients, swcnew. Plot the original signal and the denoised signal for comparison.

```
noisbloc_denoised = iswt(swcnew,'db1');
plot(noisbloc); hold on;
plot(noisbloc_denoised,'r','linewidth',2);
```

### Image Compression— Birgé-Massart Thresholds

This example compresses an image using the Birgé-Massart strategy.

Load the image and add white Gaussian noise.

```
load sinsin
x = X+18*randn(size(X));
```

Obtain the 2-D discrete wavelet transform down to level 2 using the Daubechies' least-asymmetric wavelet with 8 vanishing moments. Obtain the compression thresholds using the Birgé-Massart strategy with alpha equal to 2.

```
[C,L] = wavedec2(x,2,'sym8');
alpha = 2;
THR = wthrmngr('dw2dcompLVL','scarcehi',C,L,alpha);
```

Compress the image and display the result.

```
 Xd = wdencmp('lvd',X,'sym8',2,THR,'s');
image(X); title('Original image');
figure;
image(x); title('Noisy image');
figure;
image(Xd); title('Denoised image');
```

**Purpose**     NTREE manager

**Syntax**

**Description**     wtreemgr is a tree management utility.

This function returns information on the tree *T* depending on the value of the OPT parameter.

Allowed values for OPT are listed in the table below.

| | |
|---|---|
| 'allnodes' | Tree nodes |
| 'isnode' | True for existing node |
| 'istnode' | True for terminal nodes |
| 'nodeasc' | Node ascendants |
| 'nodedesc' | Node descendants |
| 'nodepar' | Node parent |
| 'ntnode' | Number of terminal nodes |
| 'tnodes' | Terminal nodes |
| 'leaves' | Terminal nodes |
| 'noleaves' | Not terminal nodes |
| 'order' | Tree order |
| 'depth' | Tree depth |

**See Also**     allnodes | | istnode | leaves | nodeasc | nodedesc | nodepar | noleaves | ntnode | tnodes | treedpth | treeord

# wvarchg

**Purpose**        Find variance change points

**Syntax**        `[PTS_OPT,KOPT,T_EST] = wvarchg(Y,K,D)`

**Description**    `[PTS_OPT,KOPT,T_EST] = wvarchg(Y,K,D)` computes estimated variance change points for the signal `Y` for j change points, with j = 0, 1, 2, ... , `K`.

Integer `D` is the minimum delay between two change points.

Integer `KOPT` is the proposed number of change points ($0 \leq$ `KOPT` $\leq$ `K`). The vector `PTS_OPT` contains the corresponding change points.

For $1 \leq k \leq$ `K`, `T_EST(k+1,1:k)` contains the k instants of the variance change points and then, if `KOPT > 0`, `PTS_OPT = T_EST(KOPT+1,1:KOPT)` else `PTS_OPT = []`.

`K` and `D` must be integers such that $1 <$ `K` $\ll$ length(`Y`) and $1 \leq$ `D` $\ll$ length(`Y`).

The signal `Y` should be zero mean.

`wvarchg(Y,K)` is equivalent to `wvarchg(Y,K,10)`.

`wvarchg(Y)` is equivalent to `wvarchg(Y,6,10)`.

**Examples**    **Detect Variance Change Points**

Add two variance change points to the blocks signal. Detect the variance change points using `wvarchg`.

Load the blocks signal. Add white noise with two variance change points located at index 180 and 600.

```
x = wnoise(1,10);
rng default;
bb = 1.5*randn(1,length(x));
cp1 = 180; cp2 = 600;
x = x + [bb(1:cp1),bb(cp1+1:cp2)/4,bb(cp2+1:end)];
```

Obtain the level-1 wavelet coefficients. Replace the top 2% of values with the mean value of the wavelet coefficients to remove all signal.

```
wname = 'db3'; lev = 1;
[c,l] = wavedec(x,lev,wname);
det = wrcoef('d',c,l,wname,1);
y = sort(abs(det));
v2p100 = y(fix(length(y)*0.98));
ind = find(abs(det)>v2p100);
det(ind) = mean(det);
```

Estimate the variance change points using the wavelet coefficients.

```
[pts_Opt,kopt,t_est] = wvarchg(det,5);
sprintf('The estimated change points are %d and %d\n',pts_Opt)
```

**References**     Lavielle, M. (1999), "Detection of multiple changes in a sequence of dependent variables," *Stoch. Proc. and their Applications*, 83, 2, pp. 79–102.