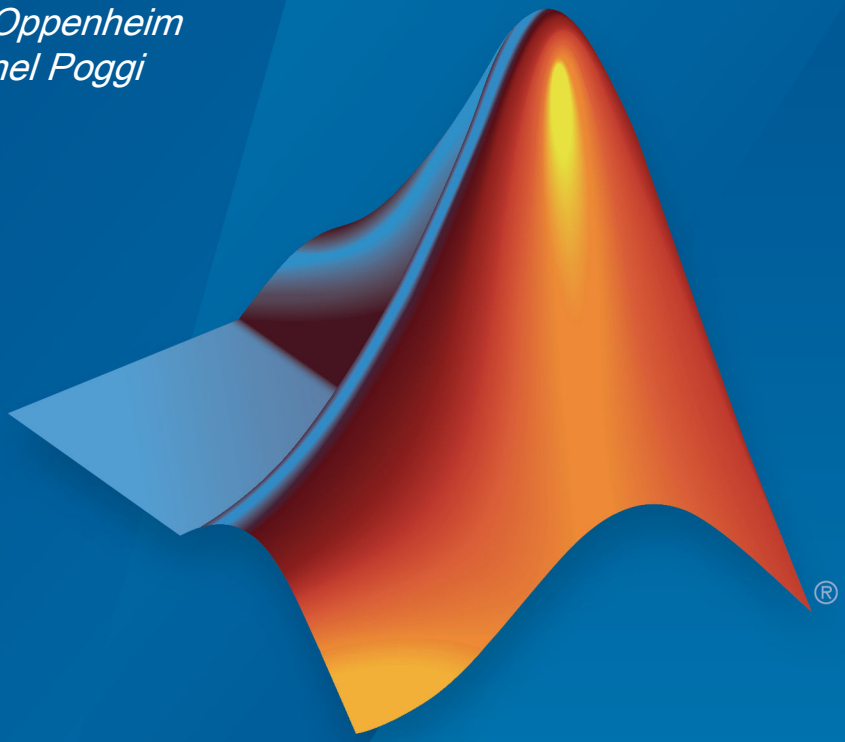
Wavelet Toolbox™

Reference

Michel Misiti
Yves Misiti
Georges Oppenheim
Jean-Michel Poggi



MATLAB®

R2017b

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

Technical support: www.mathworks.com/support/contact_us

Phone: 508-647-7000

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

Wavelet Toolbox™ Reference

Revision History

March 1997	First printing	New for Version 1.0
September 2000	Second printing	Revised for Version 2.0 (Release 12)
June 2001	Online only	Revised for Version 2.1 (Release 12.1)
July 2002	Online only	Revised for Version 2.2 (Release 13)
June 2004	Online only	Revised for Version 3.0 (Release 14)
July 2004	Third printing	Revised for Version 3.0
October 2004	Online only	Revised for Version 3.0.1 (Release 14SP1)
March 2005	Online only	Revised for Version 3.0.2 (Release 14SP2)
June 2005	Fourth printing	Minor revision for Version 3.0.2
September 2005	Online only	Minor revision for Version 3.0.3 (Release R14SP3)
March 2006	Online only	Minor revision for Version 3.0.4 (Release 2006a)
September 2006	Online only	Revised for Version 3.1 (Release 2006b)
March 2007	Online only	Revised for Version 4.0 (Release 2007a)
October 2007	Fifth printing	Revised for Version 4.1
September 2007	Online only	Revised for Version 4.1 (Release 2007b)
March 2008	Online only	Revised for Version 4.2 (Release 2008a)
October 2008	Online only	Revised for Version 4.3 (Release 2008b)
March 2009	Online only	Revised for Version 4.4 (Release 2009a)
September 2009	Online only	Minor revision for Version 4.4.1 (Release 2009b)
March 2010	Online only	Revised for Version 4.5 (Release 2010a)
September 2010	Online only	Revised for Version 4.6 (Release 2010b)
April 2011	Online only	Revised for Version 4.7 (Release 2011a)
September 2011	Online only	Revised for Version 4.8 (Release 2011b)
March 2012	Online only	Revised for Version 4.9 (Release 2012a)
September 2012	Online only	Revised for Version 4.10 (Release 2012b)
March 2013	Online only	Revised for Version 4.11 (Release 2013a)
September 2013	Online only	Revised for Version 4.12 (Release 2013b)
March 2014	Online only	Revised for Version 4.13 (Release 2014a)
October 2014	Online only	Revised for Version 4.14 (Release 2014b)
March 2015	Online only	Revised for Version 4.14.1 (Release 2015a)
September 2015	Online only	Revised for Version 4.15 (Release 2015b)
March 2016	Online only	Revised for Version 4.16 (Release 2016a)
September 2016	Online only	Revised for Version 4.17 (Release 2016b)
March 2017	Online only	Revised for Version 4.18 (Release 2017a)
September 2017	Online only	Revised for Version 4.19 (Release 2017b)

Contents

Function Reference

addlift

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

Add Primal Lifting Step

This example shows how to start with the Haar lifting scheme and add a primal lifting step.

```
LSbegin = liftwave('haar');
```

Display the lifting scheme.

```
displs(LSbegin);
```

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

Create a primal lifting step.

```
pstep = { 'p', [-1 2 -1]/4 , 1 };
```

Add the primal lifting step.

```
LSend = addlift(LSbegin,pstep);
```

Display the final lifting scheme.

```
displs(LSend);
```

```
LSend = {...
'd'              [ -1.00000000]                            [0]
'p'              [  0.50000000]                            [0]
'p'              [ -0.25000000  0.50000000 -0.25000000]    [1]
[  1.41421356]   [  0.70710678]                            []
};
```

See Also

`liftfilt`

Introduced before R2006a

allnodes

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`.

Examples

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)
```

```
ans =

  31x2 table

    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
```

Introduced before R2006a

appcoef

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 character vector containing the wavelet name. Level *`N`* must be an integer such that `0 ≤ N ≤ 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).

Examples

Level 3 Approximation Coefficients

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

Load the signal consisting of electricity usage data.

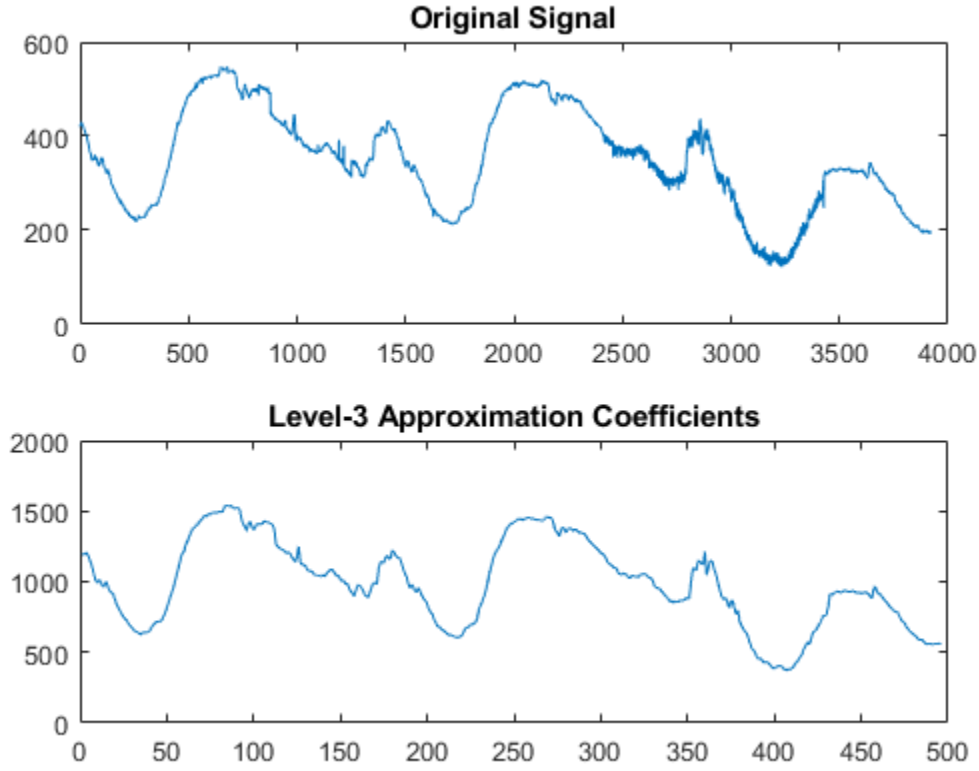
```
load leleccum;
sig = leleccum(1:3920);
```

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');
```



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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Variable-size data support must be enabled.

See Also

`detcoef` | `wavedec`

Introduced before R2006a

appcoef2

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 character vector containing the wavelet name. Level *`N`* must be an integer such that `0 ≤ N ≤ 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).

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
```

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.

Algorithms

The algorithm is built on the same principle as `appcoef`.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Variable-size data support must be enabled.

See Also

`detcoef2` | `wavedec2`

Introduced before R2006a

bestlevt

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.

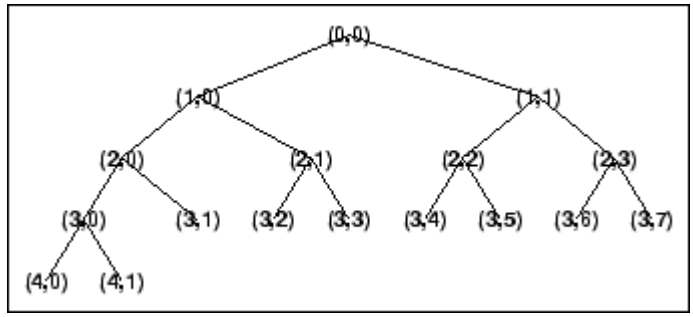
`[T,E] = bestlevt(T)` computes the best level tree *`T`*, and in addition, the best entropy value *`E`*.

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

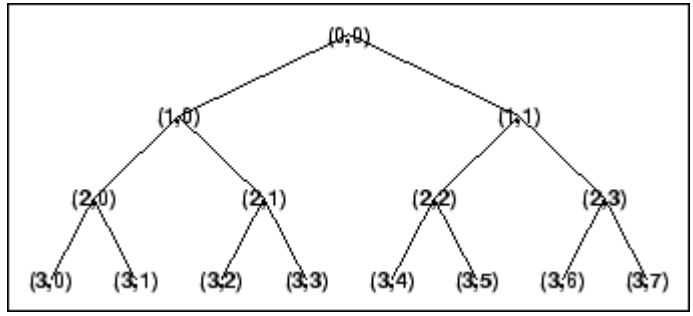
Examples

```
% 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`

Introduced before R2006a

besttree

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 $\alpha \geq 2^{N/2}$ (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.

Examples

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.

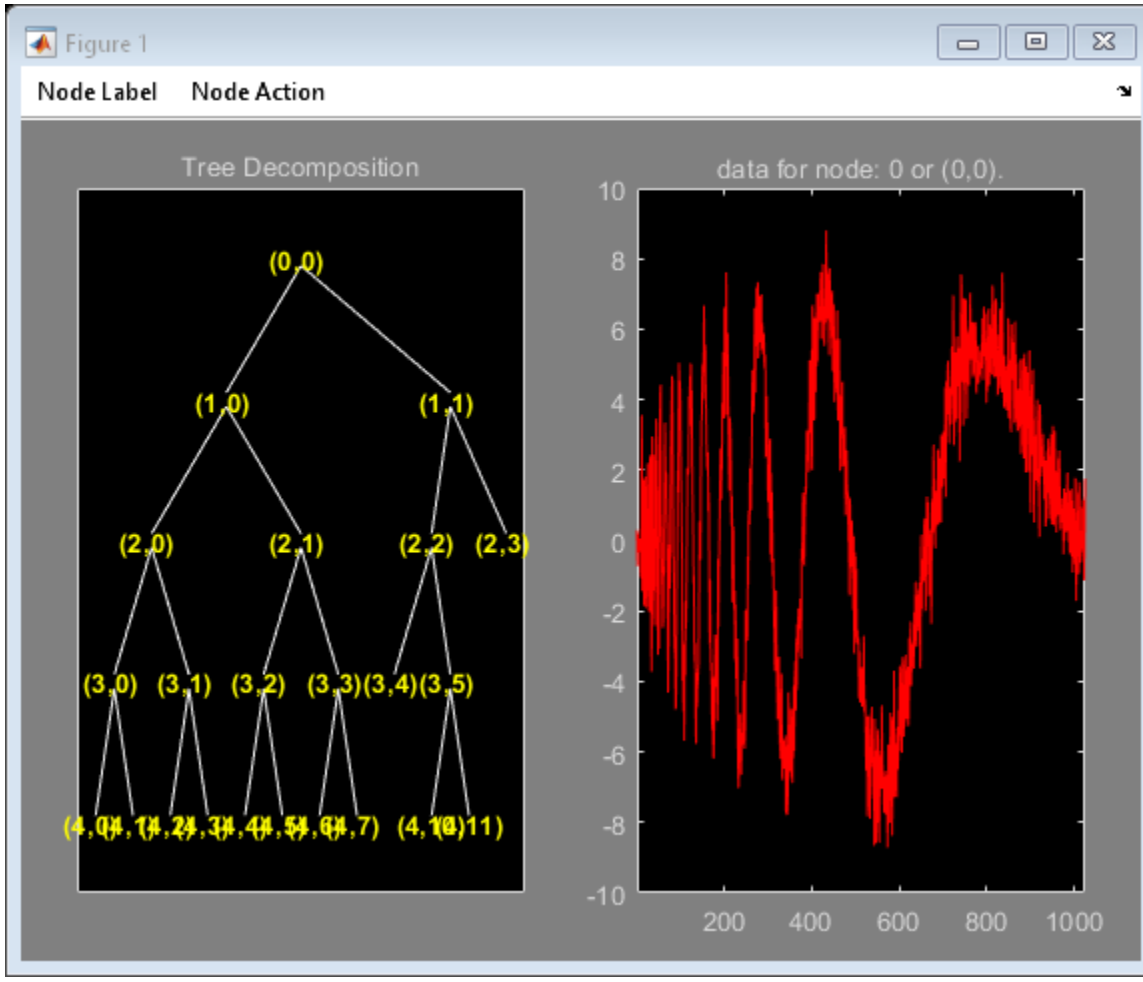
```
dwtmode('per');
```

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

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

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');
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!  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`

Topics

"Reconstructing a Signal Approximation from a Node"

Introduced before R2006a

biorfilt

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:

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

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

Examples

Biorthogonal Filters and Transfer Functions

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

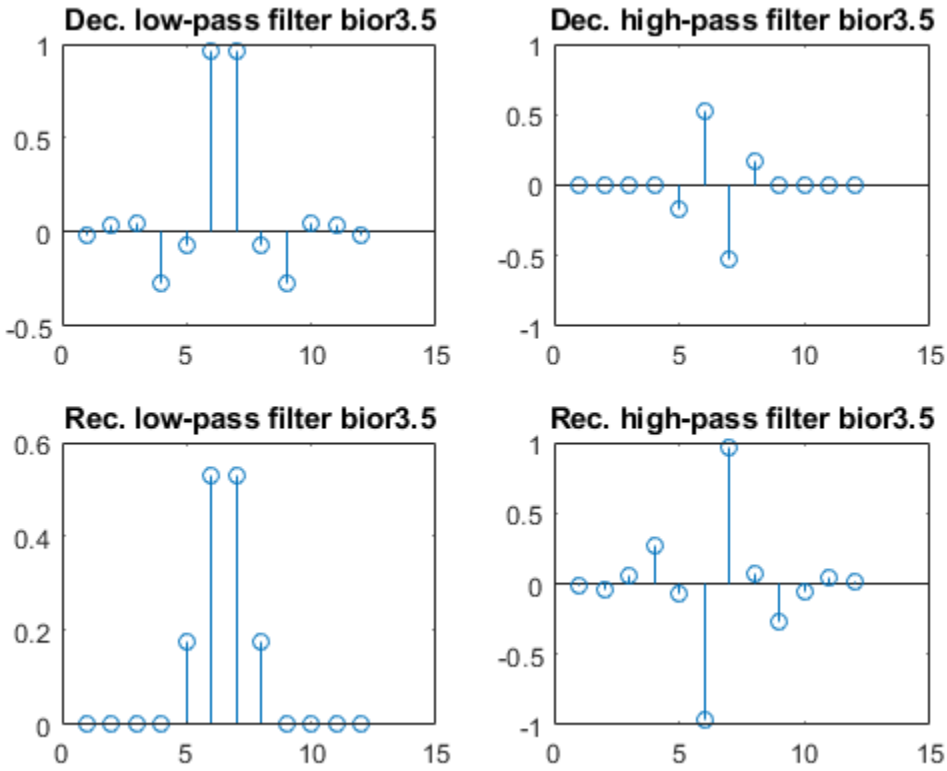
Determine the two scaling and wavelet filters associated with the `'bior3.5'` wavelet.

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

Plot the filter impulse responses.

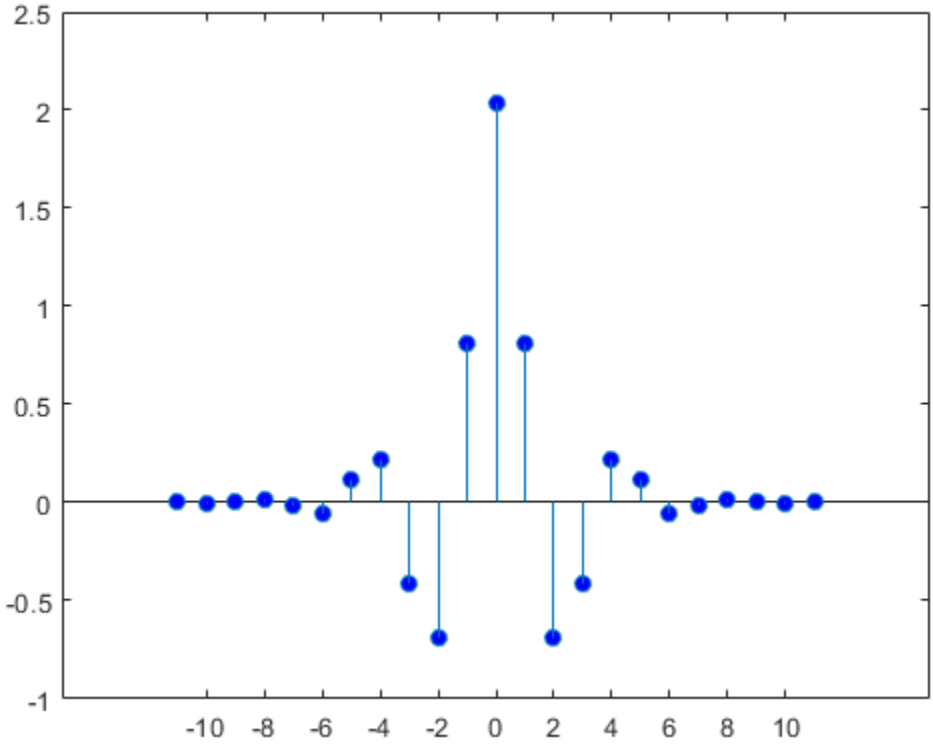
```
subplot(221); stem(LoD);
title('Dec. low-pass filter bior3.5');
subplot(222); stem(HiD);
title('Dec. high-pass filter bior3.5');
```

```
subplot(223); stem(LoR);
title('Rec. low-pass filter bior3.5');
subplot(224); stem(HiR);
title('Rec. high-pass filter bior3.5');
```



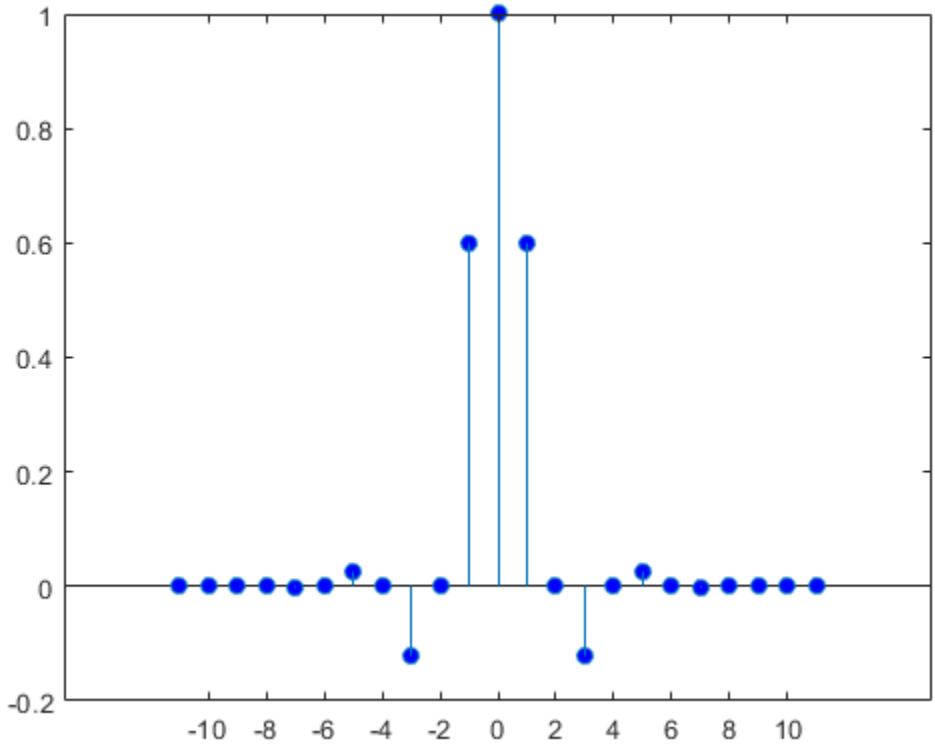
Demonstrate that autocorrelations at even lags are only zero for dual pairs of filters. Examine the autocorrelation sequence for the lowpass decomposition filter.

```
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.

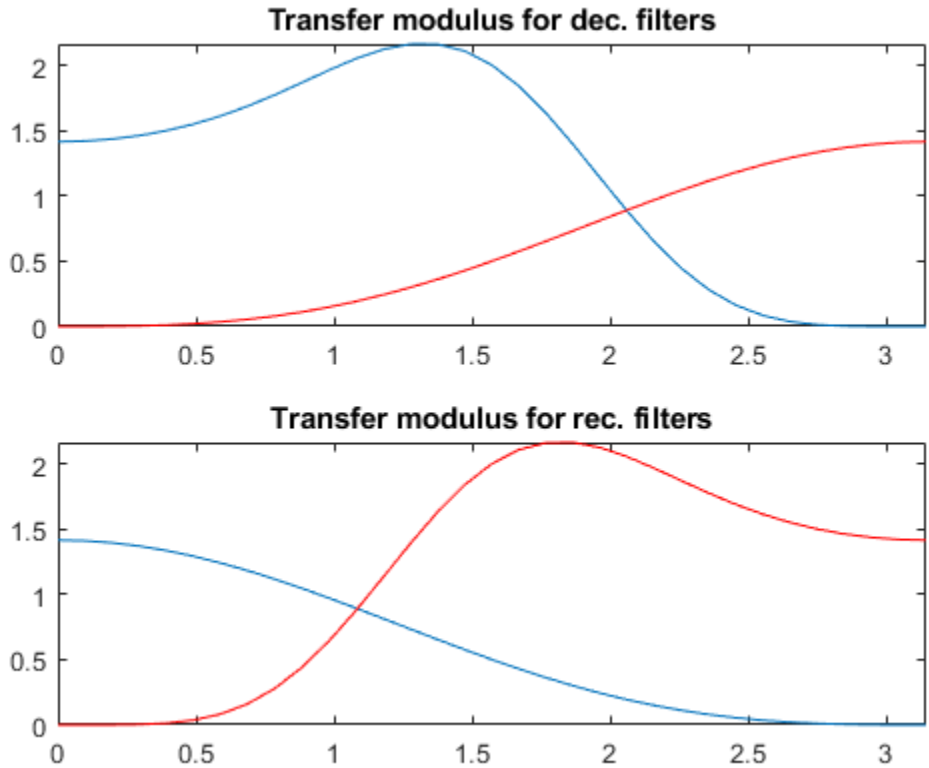
```
npad = 2*length(LoD)-1;
xcr = fftshift(ifft(fft(LoD,npad).*conj(fft(LoR,npad))));
lags = -floor(npad/2):floor(npad/2);
stem(lags,xcr,'markerfacecolor',[0 0 1])
set(gca,'xtick',-10:2:10)
```



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')
```



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`

Introduced before R2006a

biorwavf

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 character vector `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

Biorthogonal Spline Wavelet Filter

Return the biorthogonal spline wavelet scaling filters with two vanishing moments.

```
wname = 'bior2.2';
[RF,DF] = biorwavf(wname)

RF =

    0.2500    0.5000    0.2500


DF =

   -0.1250    0.2500    0.7500    0.2500   -0.1250
```

See Also

`biorfilt` | `waveinfo`

Introduced before R2006a

bswfun

Biorthogonal scaling and wavelet functions

Syntax

```
[PHIS,PSIS,PHIA,PSIA,XVAL] = bswfun(LoD,HiD,LoR,HiR)
bswfun(LoD,HiD,LoR,HiR,ITER)
bswfun(LoD,HiD,LoR,HiR,'plot')
bswfun(LoD,HiD,LoR,HiR,ITER,'plot')
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 biorthogonal scaling and wavelet functions. `PHIS` and `PSIS` are the scaling and wavelet functions constructed from the decomposition filters, `LoD` and `HiD`. `PHIA` and `PSIA` are the scaling and wavelet functions constructed from the reconstruction filters, `LoR` and `HiR`.

`bswfun(LoD,HiD,LoR,HiR,ITER)` computes the two pairs of scaling and wavelet functions using ITER iterations.

`bswfun(LoD,HiD,LoR,HiR,'plot')` or `bswfun(LoD,HiD,LoR,HiR,ITER,'plot')` or `bswfun(LoD,HiD,LoR,HiR,'plot',ITER)` computes and plots the functions.

Examples

Biorthogonal Scaling and Wavelet from Lifting Scheme

This example shows how to obtain the biorthogonal scaling and wavelet functions corresponding to a lifting scheme. Obtain the lifting scheme for the CDF 3/1 wavelet.

```
lscdf = liftwave('cdf3.1');
```

Display the lifting scheme, which consists of two primal and one dual step.

```
Sc = displs(lscdf);
Sc
```

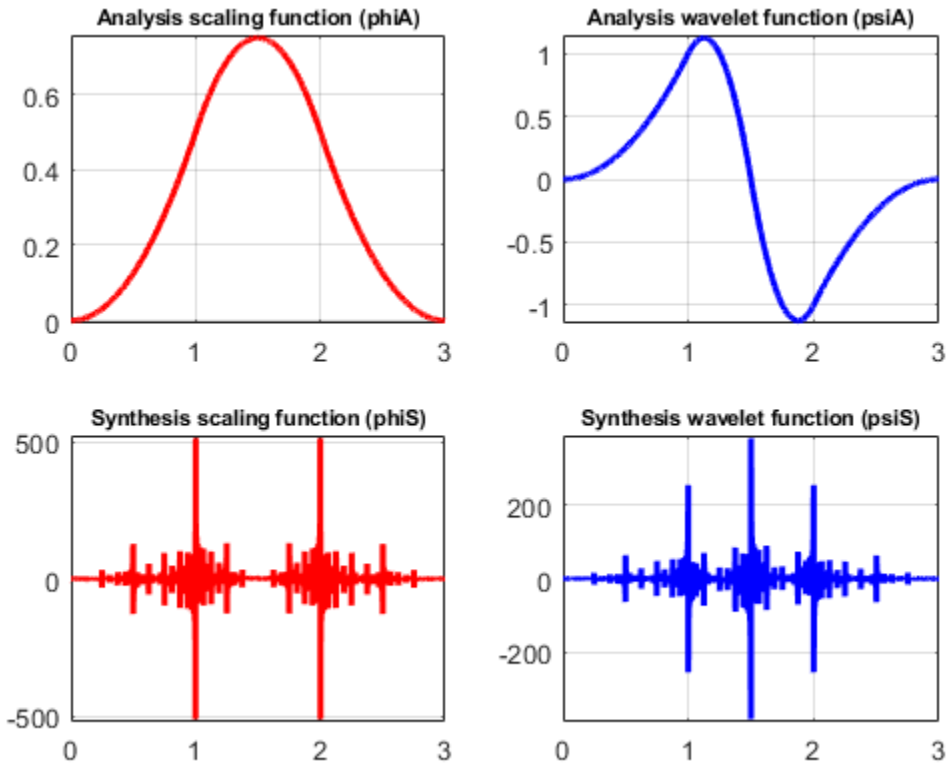
```
Sc =

  6x50 char array

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

Obtain the decomposition and reconstruction filters from the lifting scheme.

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

Visualize the scaling and wavelet function and their duals.

```
bswfun(LoD,HiD,LoR,HiR,'plot');
```



Algorithms

This function uses the cascade algorithm.

See Also

`wavefun`

Introduced before R2006a

centfrq

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 hertz 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

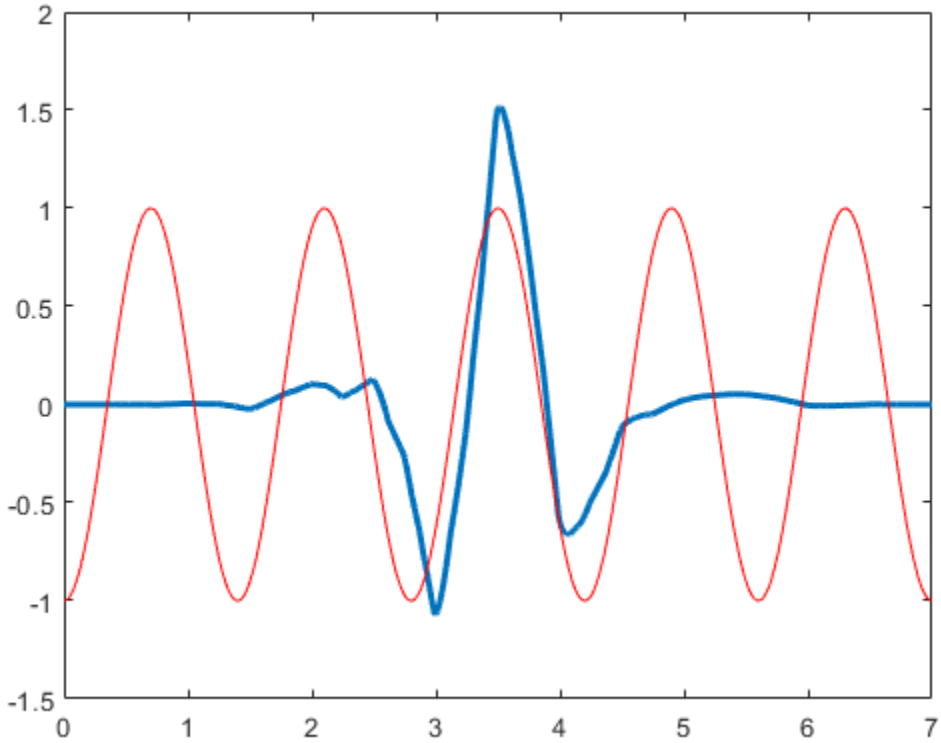
Determine Center Frequency

This example shows how to determine the center frequency in hertz for Daubechies' least-asymmetric wavelet with 4 vanishing moments.

```
cfreq = centfrq('sym4');
```

Obtain the wavelet and create a sine wave with a frequency equal to the center frequency, `cfreq`, of the wavelet. Use a starting phase of $-\pi$ for the sine wave to visualize how the oscillation in the sine wave matches the oscillation in the wavelet.

```
[~,psi,xval] = wavefun('sym4');
y = cos(2*pi*cfreq*xval-pi);
plot(xval,psi,'linewidth',2);
hold on;
plot(xval,y,'r');
```



Convert Scales to Frequencies

This example shows to convert scales to frequencies for the Morlet wavelet. There is an approximate inverse relationship between scale and frequency. Specifically, scale is

inversely proportional to frequency with the constant of proportionality being the center frequency of the wavelet.

Construct a vector of scales with 32 voices per octave over 5 octaves for data sampled at 1 kHz.

```
Fs = 1000;
numvoices = 32;
a0 = 2^(1/numvoices);
numoctaves = 5;
scales = a0.^(numvoices:1/numvoices:numvoices*numoctaves).*1/Fs;
```

Convert the scales to approximate frequencies in hertz for the Morlet wavelet.

```
Frq = centfrq('morl')./scales;
```

You can also use `scal2frq` to convert scales to approximate frequencies in hertz.

See Also

`scal2frq` | `wavefun`

Introduced before R2006a

cfs2wpt

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).

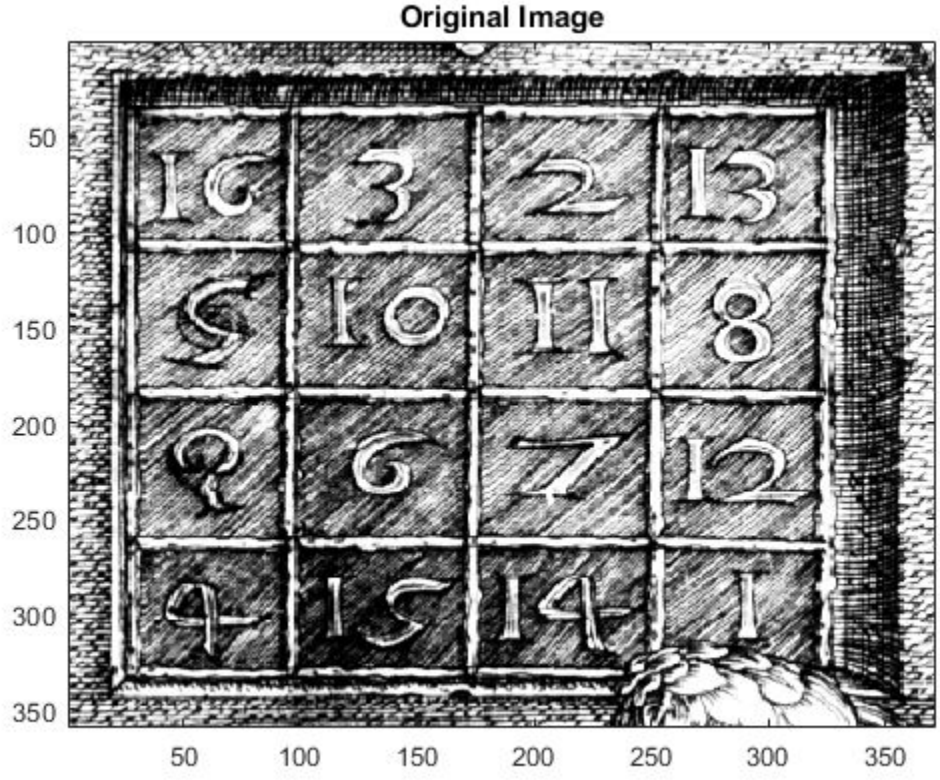
Examples

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.

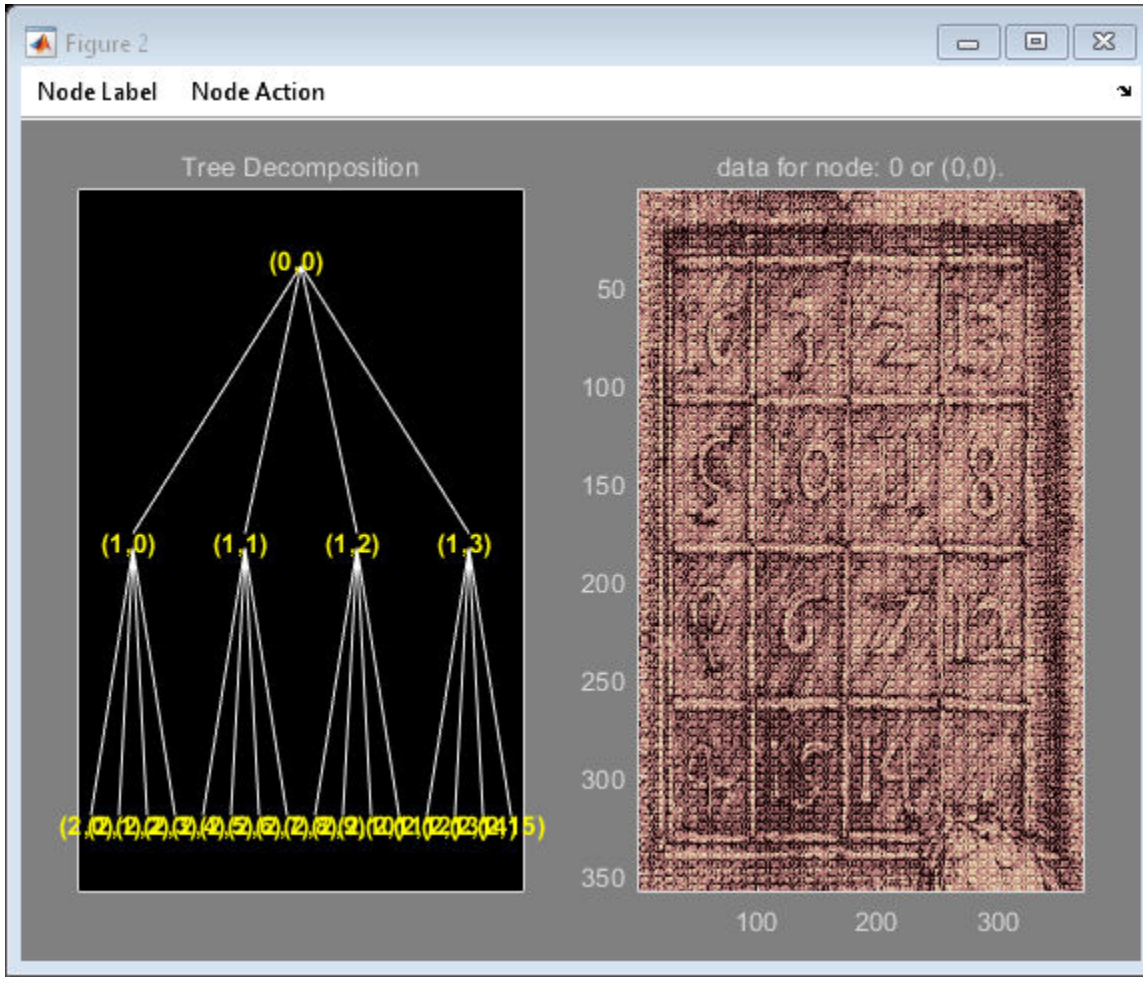
```
load detail;
imagesc(X); colormap gray; title('Original Image');
```




```
Tr = wpdec2(X,2,'sym4');
```

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.

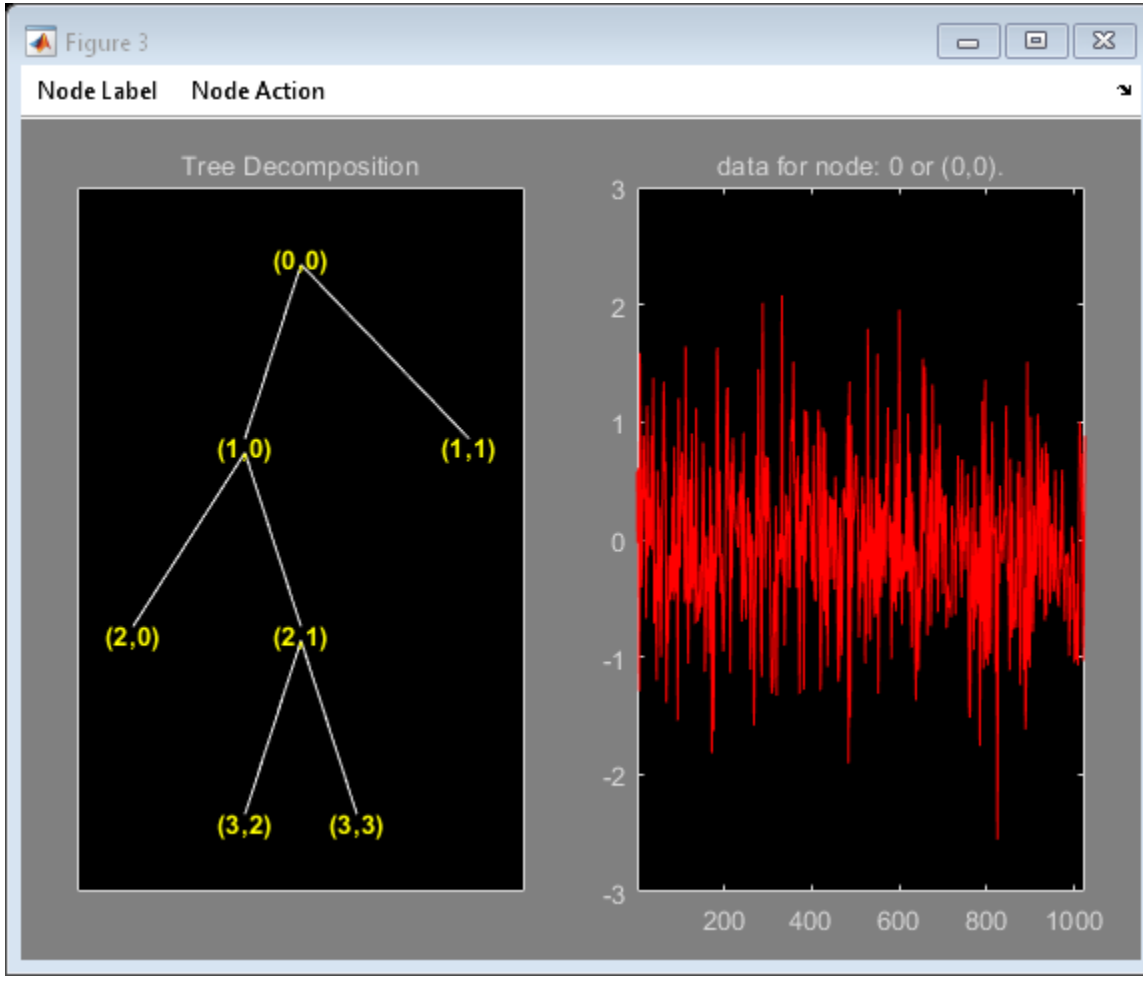
```
tr = cfs2wpt('sym4',[1 1024],[2 3 9 10]',2);
```

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)
```



Introduced before R2006a

cgauwavf

Complex Gaussian wavelet

Syntax

```
[PSI,X] = cgauwavf(LB,UB,N)
[PSI,X] = gauswavf(LB,UB,N,P)
[PSI,X] = gauswavf(LB,UB,N,WAVNAME)
```

Description

`[PSI,X] = cgauwavf(LB,UB,N)` returns the 1st order derivative of the complex-valued Gaussian wavelet, `PSI`, on an `N`-point regular grid, `X`, for the interval `[LB,UB]`. The effective support of the complex-valued Gaussian wavelets is `[-5 5]`.

`[PSI,X] = gauswavf(LB,UB,N,P)` returns the `P`th derivative. Valid values of `P` are integers from 1 to 8.

The complex Gaussian function is defined as $C_p e^{-ix} e^{-x^2}$. C_p is such that the 2-norm of the `P`th derivative of `PSI` is equal to 1.

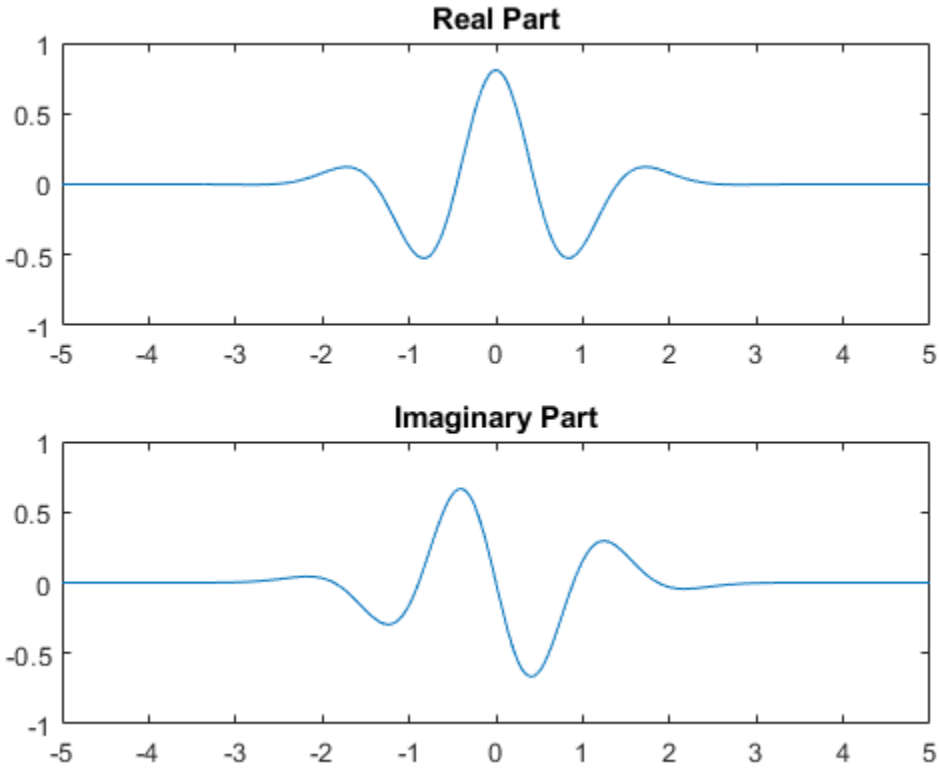
`[PSI,X] = gauswavf(LB,UB,N,WAVNAME)` uses the valid wavelet family short name `WAVNAME` plus the order of the derivative in a character vector, such as `'cgau4'`. To see valid character vectors for complex-valued Gaussian wavelets, use `waveinfo('cgau')` or use `wavemngr('read',1)` and refer to the Complex Gaussian section.

Examples

Create Complex Gaussian Wavelet

This example shows how to create a complex-valued Gaussian wavelet of order 4. The wavelet has an effective support of [-5,5] and is constructed using 1,000 samples.

```
lb = -5;
ub = 5;
n = 1000;
order = 4;
[psi,x] = cgauwavf(lb,ub,n,order);
subplot(211)
plot(x,real(psi))
title('Real Part');
subplot(212)
plot(x,imag(psi))
title('Imaginary Part');
```



See Also

`waveinfo`

Introduced before R2006a

chgwdeccfs

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 $1 \leq$ `LEV` $\leq$ `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`.

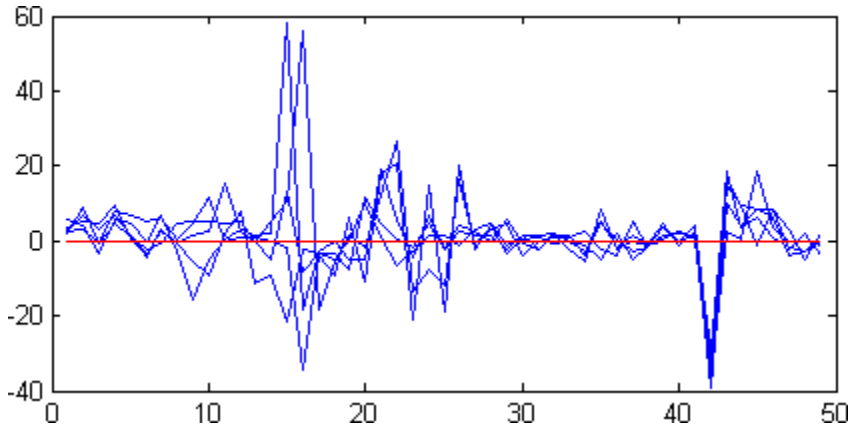
If `COEFS` (or *`CA`* or *`CD`*) is a single number, then it replaces all the related coefficients. Otherwise, `COEFS` (or *`CA`*, or *`CD`*) must be a matrix of appropriate size.

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

Introduced in R2007a

cmddenoise

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≤ nb_inter ≤6. For `nb_inter` ≥ 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.

Examples

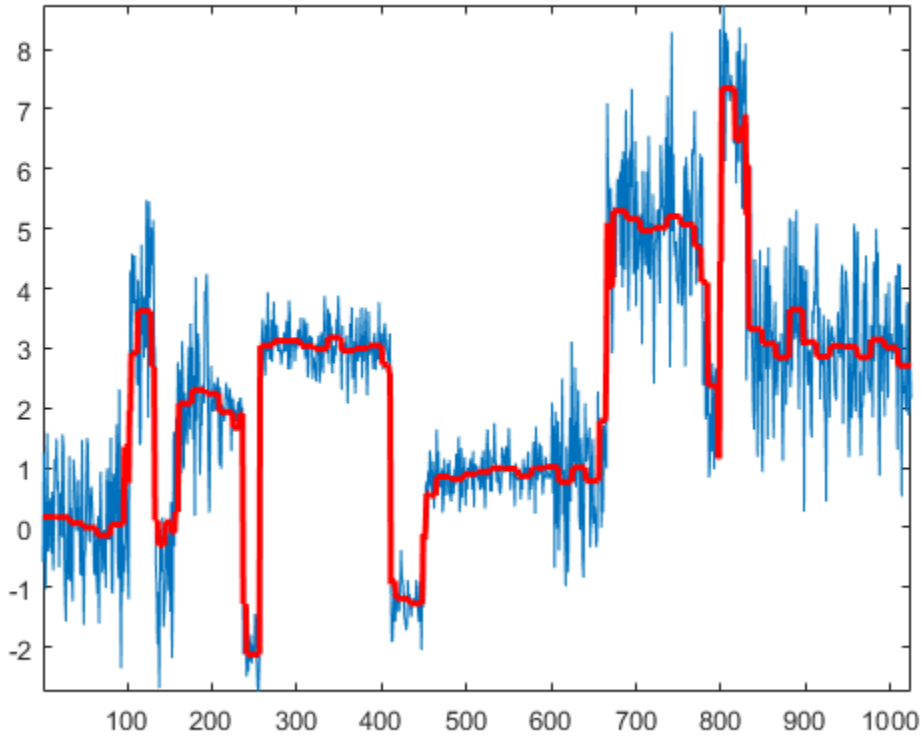
Denoising Blocks Signal with Haar Wavelet

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. |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;
```



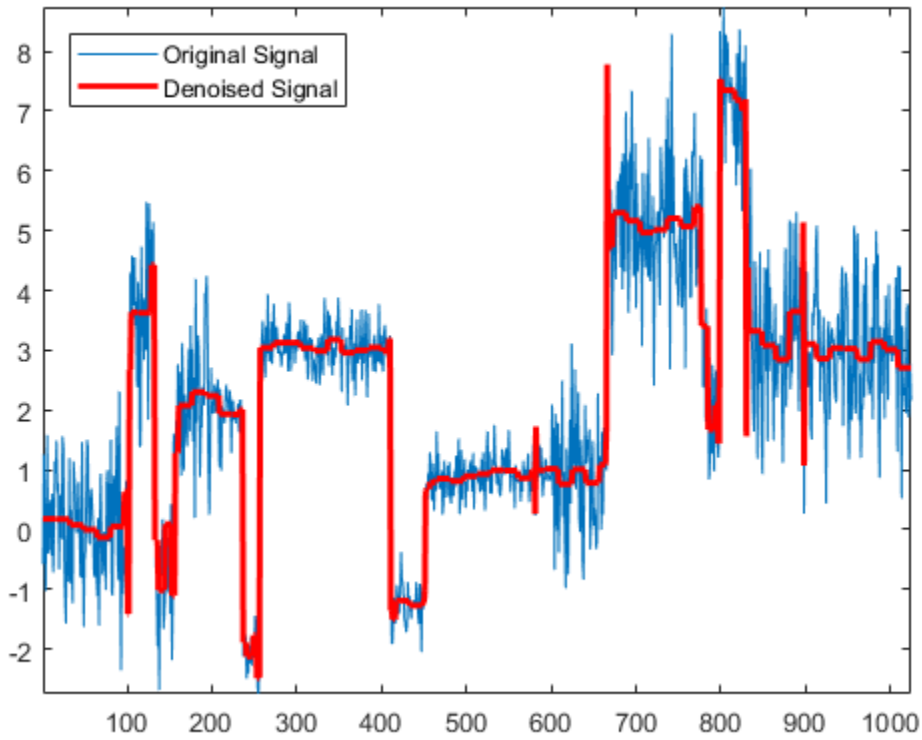
Denoising Blocks Signal with Hard Thresholding

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.

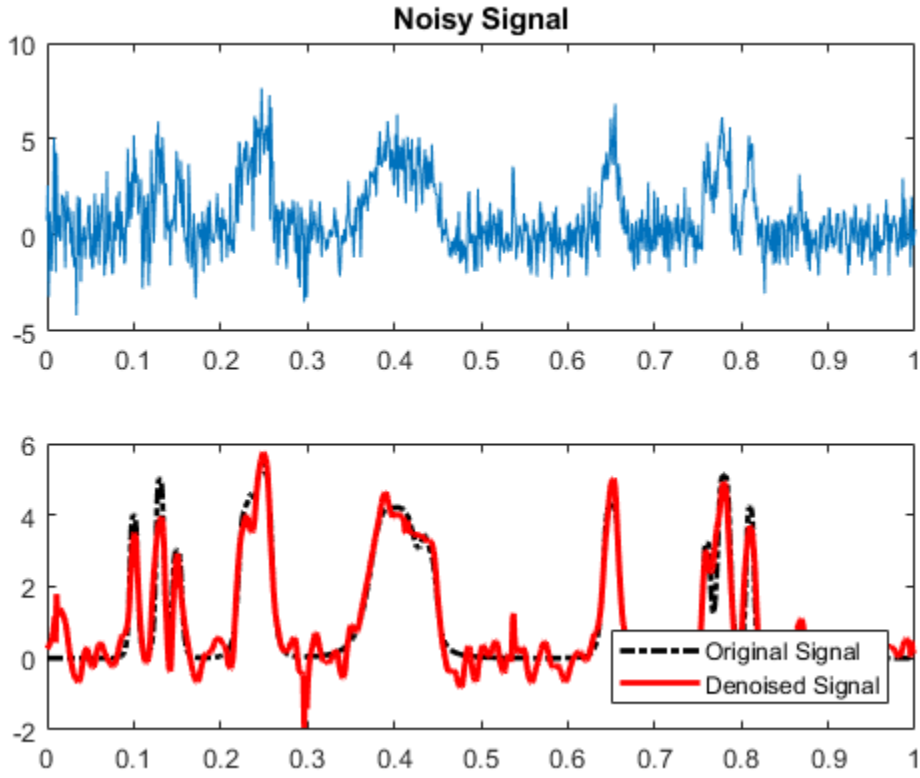
```
t = [0.1 0.13 0.15 0.23 0.25 0.40 0.44 0.65 0.76 0.78 0.81];
h = [4  -5 3 -4 5  -4.2   2.1   4.3  -3.1   5.1  -4.2];
h  = abs(h);
len = 1000;
w  = 0.01*[0.5 0.5 0.6 1 1 3 1 1 0.5 0.8 0.5];
tt = linspace(0,1,len);
x = zeros(1,len);
for j=1:11
  x = x + ( h(j) ./ (1+ ((tt-t(j))/w(j)).^4));
end
```

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;
sigden = cmddenoise(xx,'sym5',5,'s',2);
```

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.

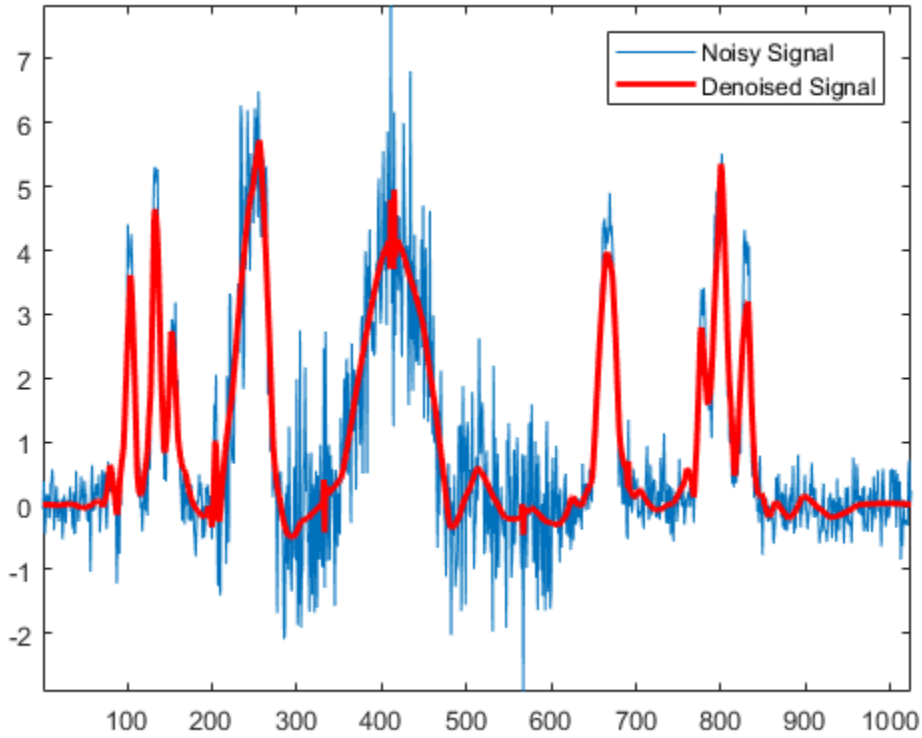
```
load nbumpr1.mat;
```

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.

```
wname = 'sym4';
level = 5;
sorh = 's';
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');
```



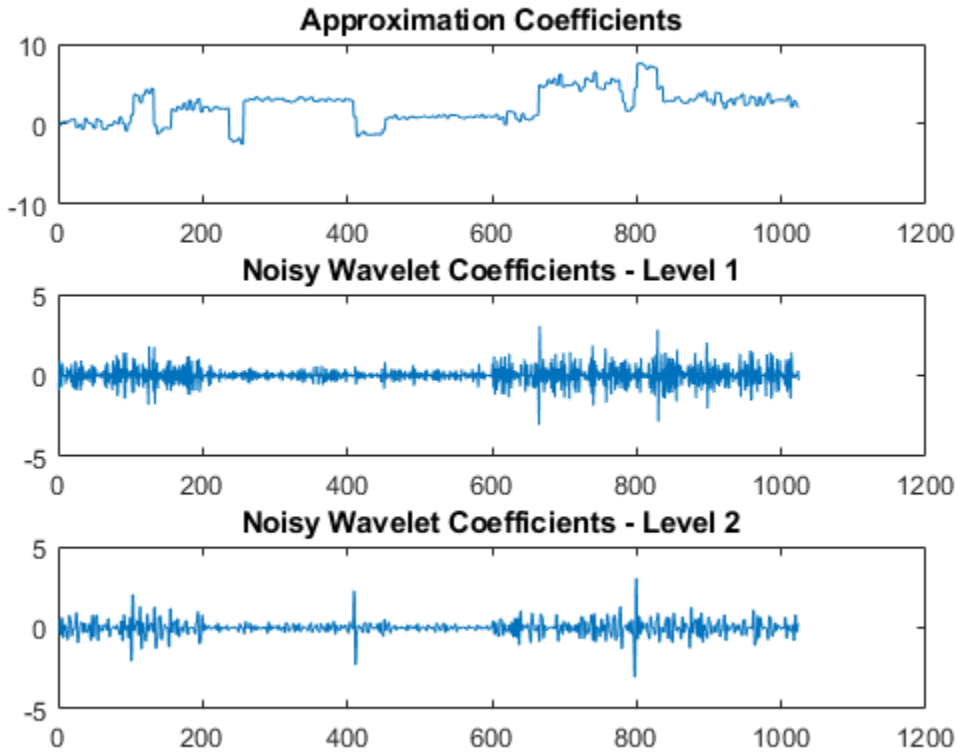
Return Denoised Wavelet Coefficients

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');
```

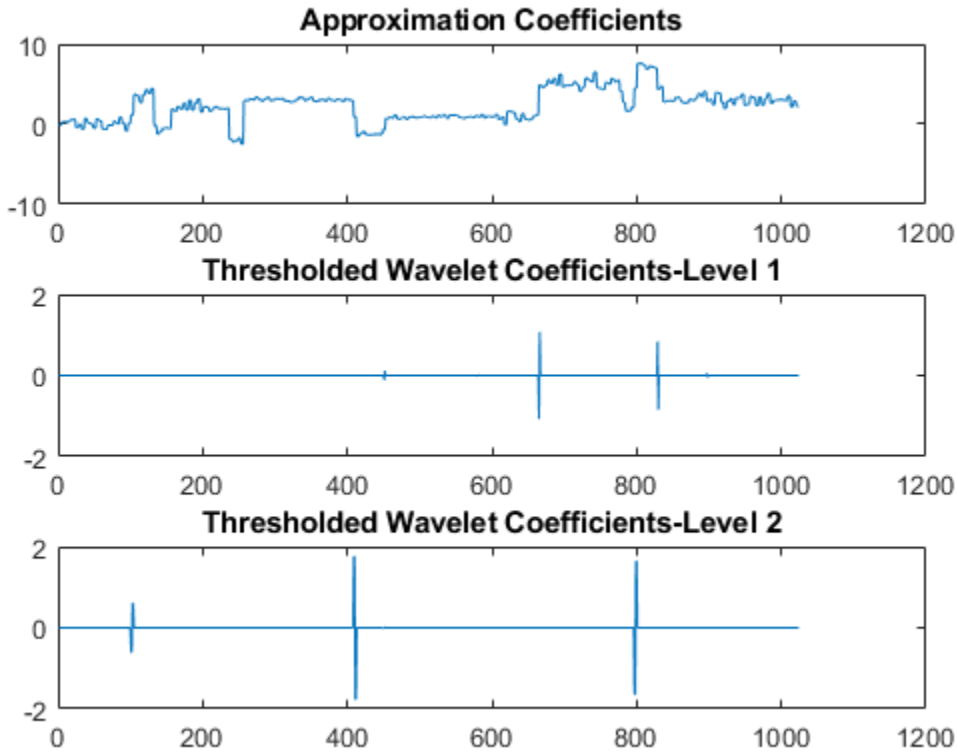
Plot reconstructions based on the level-2 approximation and level-2 and level-1 detail coefficients for the noisy signal.

```
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
```

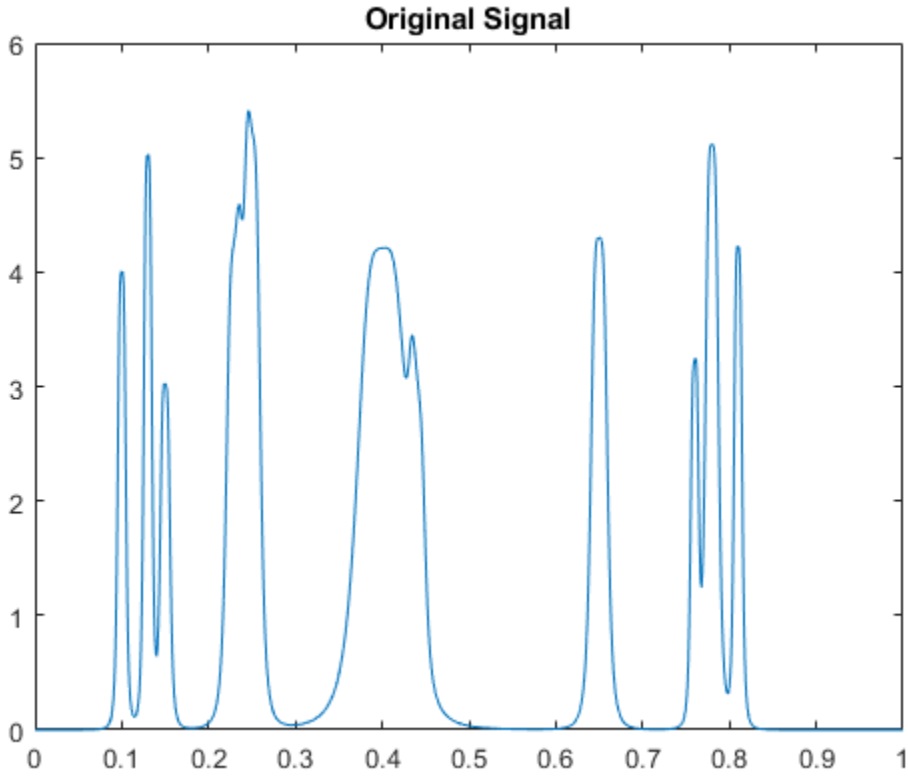


The approximation coefficients are identical in the noisy and denoised signal, but most of the detail coefficients in the denoised signal are close to zero.

Output Intervals and Thresholds

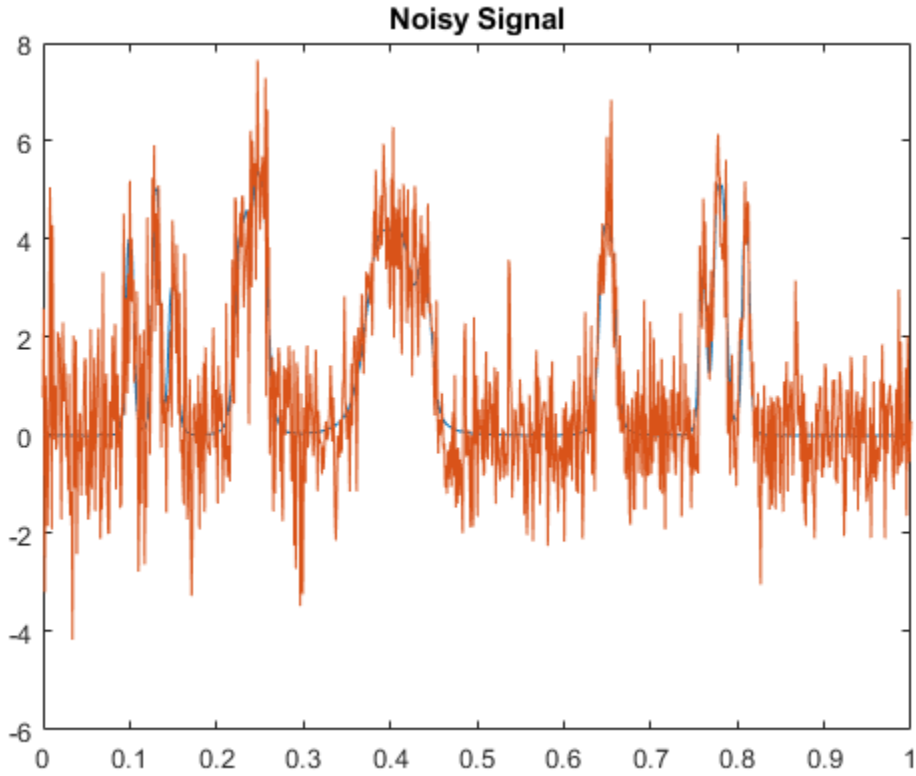
Create a signal sampled at 1 kHz. The signal consists of a series of bumps of various widths.

```
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');
hold on;
```



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;
plot(tt,xx);
title('Noisy Signal');
```



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}
```

```
ans =

   1.0e+03 *

    0.0010    0.3060    0.0038
    0.3060    0.4020    0.0018
```

```
    0.4020    1.0000    0.0032
```

`cmdnoise` 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.

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

```
          Begin        End          Threshold
```

```
cellfun(@disp,int_DepThr_Cell,'UniformOutput',false);
```

```
          1.00       1024.00          1.36

          1.00        613.00          1.73
        613.00       1024.00          1.00

          1.00        207.00          1.05
        207.00        613.00          2.51
        613.00       1024.00          1.00

          1.00        207.00          1.05
        207.00        597.00          2.52
        597.00        627.00          1.69
        627.00       1024.00          0.97

          1.00        207.00          1.05
        207.00        613.00          2.51
        613.00        695.00          1.20
        695.00        725.00          0.59
```

```
   725.00       1024.00          1.05

     1.00        207.00          1.05
   207.00        597.00          2.52
   597.00        627.00          1.69
   627.00        695.00          1.19
   695.00        725.00          0.59
   725.00       1024.00          1.05
```

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.

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

```
Found 2 change points.
```

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

character vector

Wavelet name, specified as a character vector. `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 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.

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`

Introduced in R2010a

cmorwavf

Complex Morlet wavelet

Syntax

```
[PSI,X] = cmorwavf(LB,UB,N)
[PSI,X] = cmorwavf(LB,UB,N,FB,FC)
```

Description

`[PSI,X] = cmorwavf(LB,UB,N)` returns the complex Morlet wavelet, `PSI`, with time-decay parameter, `FB`, and center frequency, `FC`, both equal to 1. The general expression for the complex Morlet wavelet is

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

`X` is evaluated on an *N*-point regular grid in the interval [`LB`,`UB`].

`[PSI,X] = cmorwavf(LB,UB,N,FB,FC)` returns values of the complex Morlet wavelet defined by a positive time-decay parameter, `FB`, and positive center frequency, `FC`.

`FB` controls the decay in the time domain and the corresponding energy spread (bandwidth) in the frequency domain. `FB` is the inverse of the variance in the frequency domain. Increasing `FB` makes the wavelet energy more concentrated around the center frequency and results in slower decay of the wavelet in the time domain. Decreasing `FB` results in faster decay of the wavelet in the time domain and less energy spread in the frequency domain. The value of `FB` does not affect the center frequency. When converting from scale to frequency, only the center frequency affects the frequency values. The energy spread or bandwidth parameter affects how localized the wavelet is in the frequency domain.

Examples

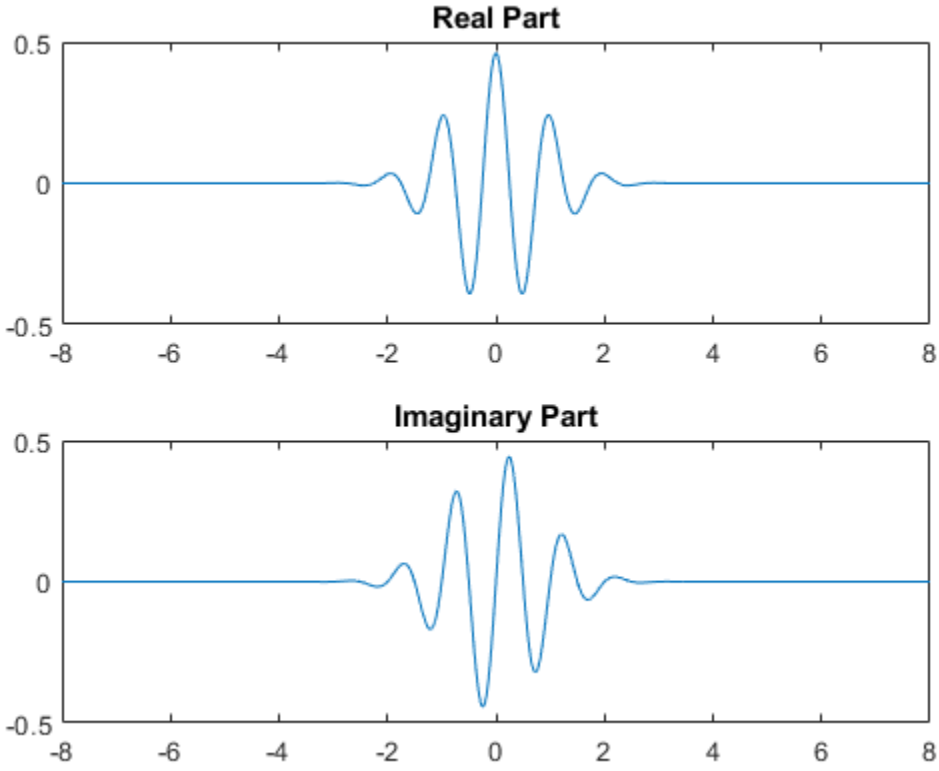
Complex Morlet Wavelet

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');
```

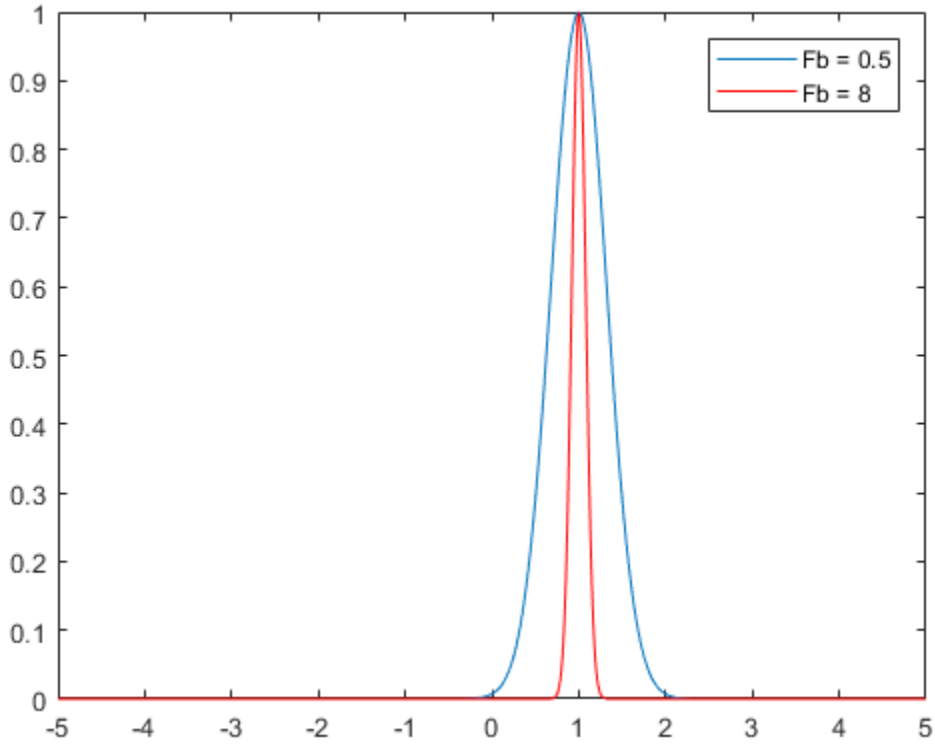


Effect of Bandwidth Parameter on Morlet Wavelet Shape

This example shows how the complex Morlet wavelet shape in the frequency domain is affected by the value of the bandwidth parameter (`Fb`). Both wavelets have a center frequency of 1. One wavelet has an `Fb` value of 0.5 and the other wavelet has a value of 8.

```
f = -5:.01:5;
Fc = 1;
Fb1 = 0.5;
Fb2 = 8;
psihat1 = exp(-pi^2*Fb1*(f-Fc).^2);
```

```
psihat2 = exp(-pi^2*Fb2*(f-Fc).^2);
plot(f,psihat1)
hold on;
plot(f,psihat2,'r')
legend('Fb = 0.5','Fb = 8')
```



The `Fb` bandwidth parameter for the complex Morlet wavelet is the inverse of the variance in frequency. Therefore, increasing Fb results in a narrower concentration of energy around the center frequency.

References

Teolis, A. (1998), *Computational signal processing with wavelets*, Birkhauser, p. 65.

See Also

`waveinfo`

Introduced before R2006a

coifwavf

Coiflet wavelet filter

Syntax

```
F = coifwavf(W)
```

Description

`F = coifwavf(W)` returns the scaling filter associated with the Coiflet wavelet specified by the character vector `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`

Introduced before R2006a

conofinf

Cone of influence

Syntax

```
cone = conofinf(wname,scales,LenSig,SigVal)
[cone,PL,PR] = conofinf(wname,scales,LenSig,SigVal)
[cone,PL,PR,PLmin,PRmax] = conofinf(wname,scales,LenSig,SigVal)
[PLmin,PRmax] = conofinf(wname,scales,LenSig)
[...] = 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 `SigVal` is a 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 at scale 1 for the points in `SigVal`. `PL` and `PR` are `length(SigVal)`-by-2 matrices. The left boundaries are `(1-PL(:,2))./PL(:,1)` and the right boundaries are `(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 character vector 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` is the 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.

Examples

Cone of Influence for Mexican Hat or Ricker Wavelet

Load the data.

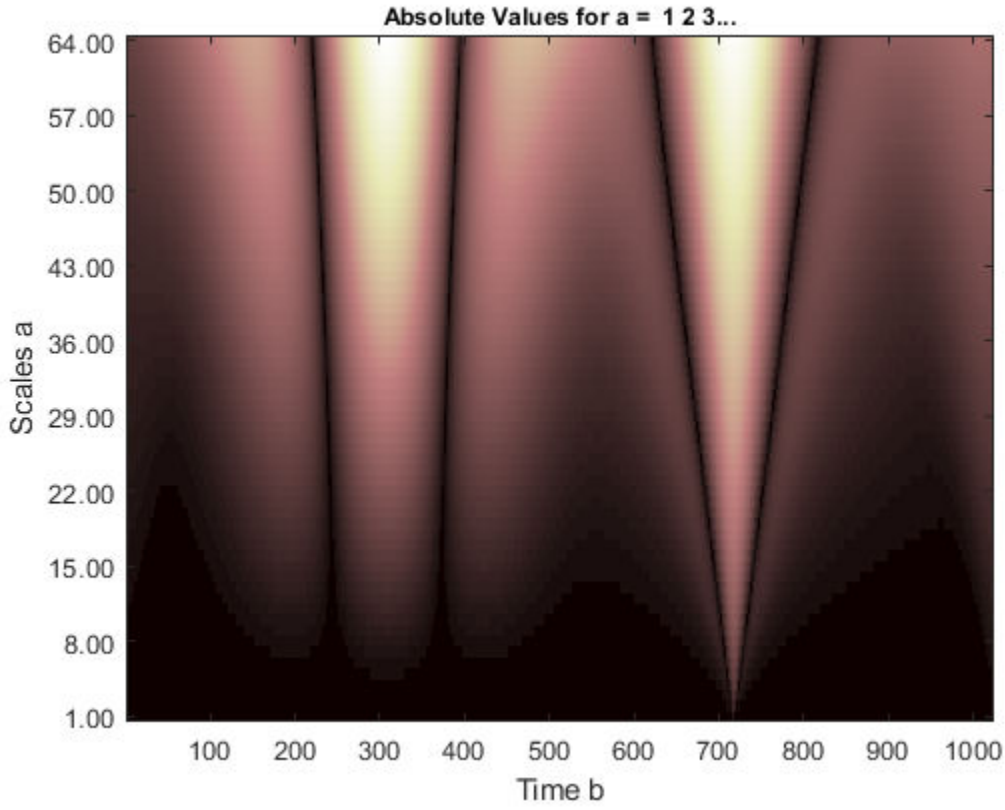
```
load cuspamax
signal = cuspamax;
```

Set up the wavelet.

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

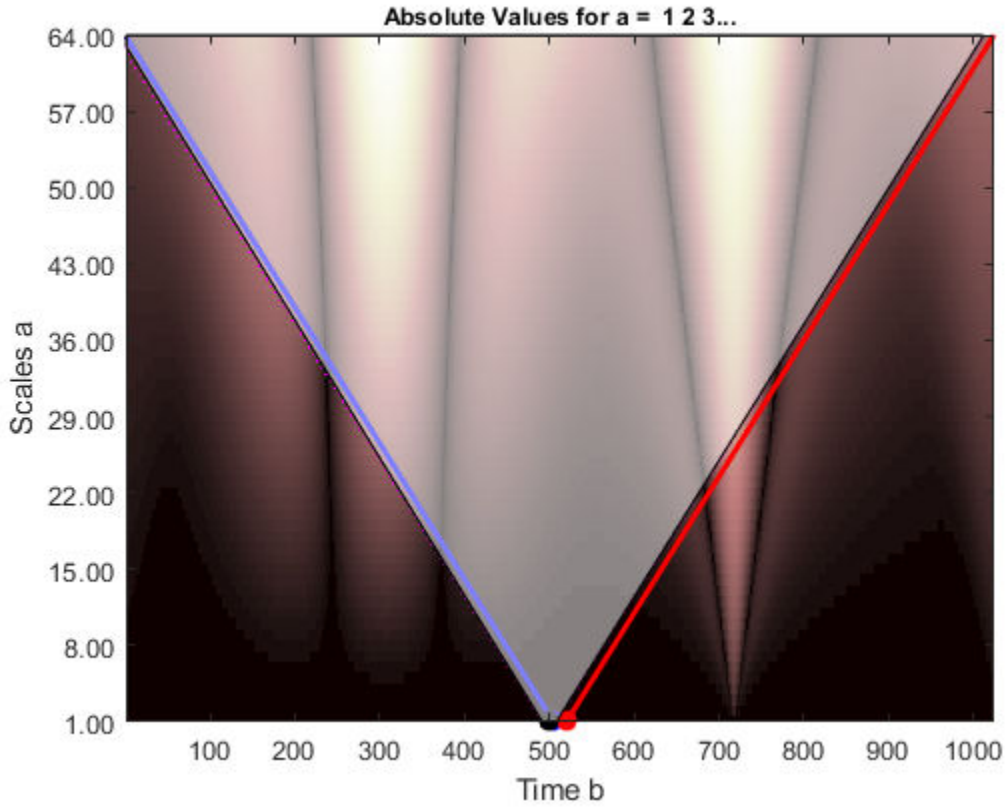
Plot the wavelet.

```
figure;
cwt(signal,scales,wname,'plot');
```



Plot the cone of influence.

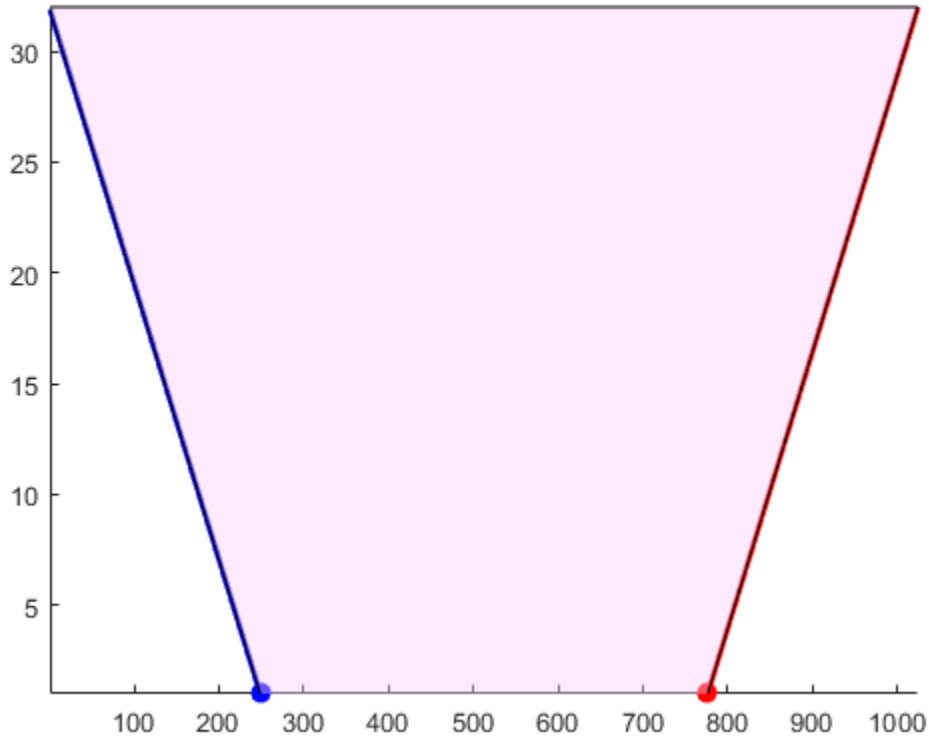
```
hold on
[cone,PL,PR,Pmin,Pmax] = conofinf(wname,scales,lenSIG,x,'plot');
set(gca,'Xlim',[1 lenSIG])
```



Cone of Influence Minimal and Maximal Vertices

Return the left minimal and right maximal vertices for the cone of influence (Morlet wavelet).

```
[PLmin,PRmax] = conofinf('morl',1:32,1024,[],'plot');
```




```
PLmin
```

```
PLmin =

   -0.1245   32.0000
```

```
PRmax
```

```
PRmax =

    0.1250  -96.0000
```

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:

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

and has 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.

References

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

See Also

`cwt` | `wavsupport`

Topics

"Continuous and Discrete Wavelet Transforms"

Introduced in R2010b

cwt

Continuous 1-D wavelet transform

Note See `cwt` for information on the older version of the `cwt`. The older version is no longer recommended.

Syntax

```
wt = cwt(x)
wt = cwt(x,wname)

[wt,f] = cwt(___,fs)
[wt,period] = cwt(___,ts)
[wt,f,coi] = cwt(___,fs)
[wt,period,coi] = cwt(___,ts)

[___] = cwt(___,Name,Value)

cwt(___)
```

Description

`wt = cwt(x)` returns the continuous wavelet transform (CWT) of `x`. The input, `x`, is a double-precision real- or complex-valued vector and must have at least four samples. The CWT is obtained using the analytic Morse wavelet with the symmetry parameter (gamma) equal to 3 and the time-bandwidth product equal to 60. `cwt` uses 10 voices per octave. The minimum and maximum scales are determined automatically based on the wavelet's energy spread in frequency and time. If `x` is real-valued, `wt` is a 2-D matrix where each row corresponds to one scale. The column size of `wt` is equal to the length of `x`. If `x` is complex-valued, `wt` is a 3-D matrix, where the first page is the CWT for the positive scales (analytic part or counterclockwise component) and the second page is the CWT for the negative scales (anti-analytic part or clockwise component).

`wt = cwt(x,wname)` uses the analytic wavelet specified by `wname` to compute the CWT. Valid options for `wname` are `'morse'`, `'amor'`, and `'bump'`, which specify the Morse,

Morlet, and bump wavelet, respectively. If you do not specify `wname`, `wname` defaults to `'morse'`.

`[wt,f] = cwt(___,fs)` specifies the sampling frequency, `fs`, in Hz as a positive scalar. `cwt` uses `fs` to determine the scale-to-frequency conversions and returns the frequencies, `f`, in Hz. If you do not specify a sampling frequency, `cwt` returns `f` in cycles per sample. If the input `x` is complex, the scale-to-frequency conversions apply to both pages of `wt`.

`[wt,period] = cwt(___,ts)` specifies the sampling interval, `ts`, as a positive `duration` scalar. The `duration` can be in years, days, hours, minutes, or seconds. `cwt` uses `ts` to compute the scale-to period conversion and returns the time periods in `period`. The array of durations in `period` have the same format property as `ts`. If the input `x` is complex, the scale-to-period conversions apply to both pages of `wt`.

`[wt,f,coi] = cwt(___,fs)` returns the cone of influence, `coi`, which shows where edge effects of the CWT become significant. The cone of influence for the CWT is in Hz. If the input `x` is complex, the cone of influence applies to both pages of `wt`.

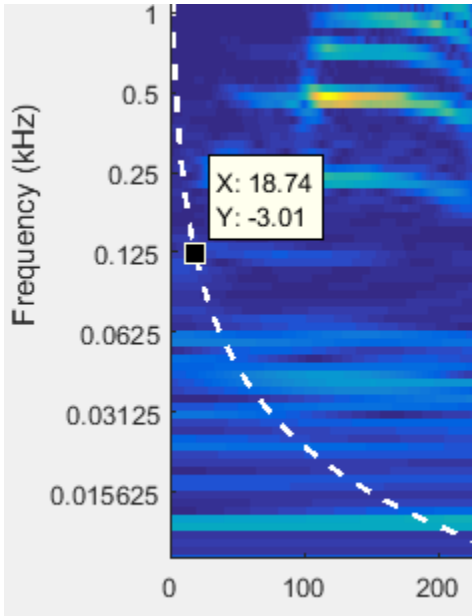
`[wt,period,coi] = cwt(___,ts)` returns the cone of influence, `coi`, which shows where edge effects of the CWT become significant. The cone of influence for the CWT is in cycles per sample. . If the input `x` is complex, the cone of influence applies to both pages of `wt`.

`[___] = cwt(___,Name,Value)` returns the CWT with additional options specified by one or more `Name,Value` pair arguments.

`cwt(___)` with no output arguments plots the CWT scalogram, which is the absolute value of the CWT as a function of time and frequency. The cone of influence showing where edge effects become significant is also plotted. Gray regions outside the dashed white line delineate regions where edge effects are significant. If the input signal is complex-valued, the positive (counterclockwise) and negative (clockwise) components are plotted in separate scalograms.

If you do not specify a sampling frequency, `fs`, or time interval, `ts`, the frequencies are plotted in cycles per sample. If you specify a sampling frequency, `fs`, the frequencies are in Hz. If you specify a sampling duration, the plot is a function of time and periods.

The *y*-axis of the scalogram uses a $\log_2$ scale. If you use a data cursor, the actual *y*-value is displayed. For example, if the axis value is approximately 0.125, the data cursor *y*-value is –3.01, which you can verify using `pow2(3.01)`.



Examples

Continuous Wavelet Transform Using Default Values

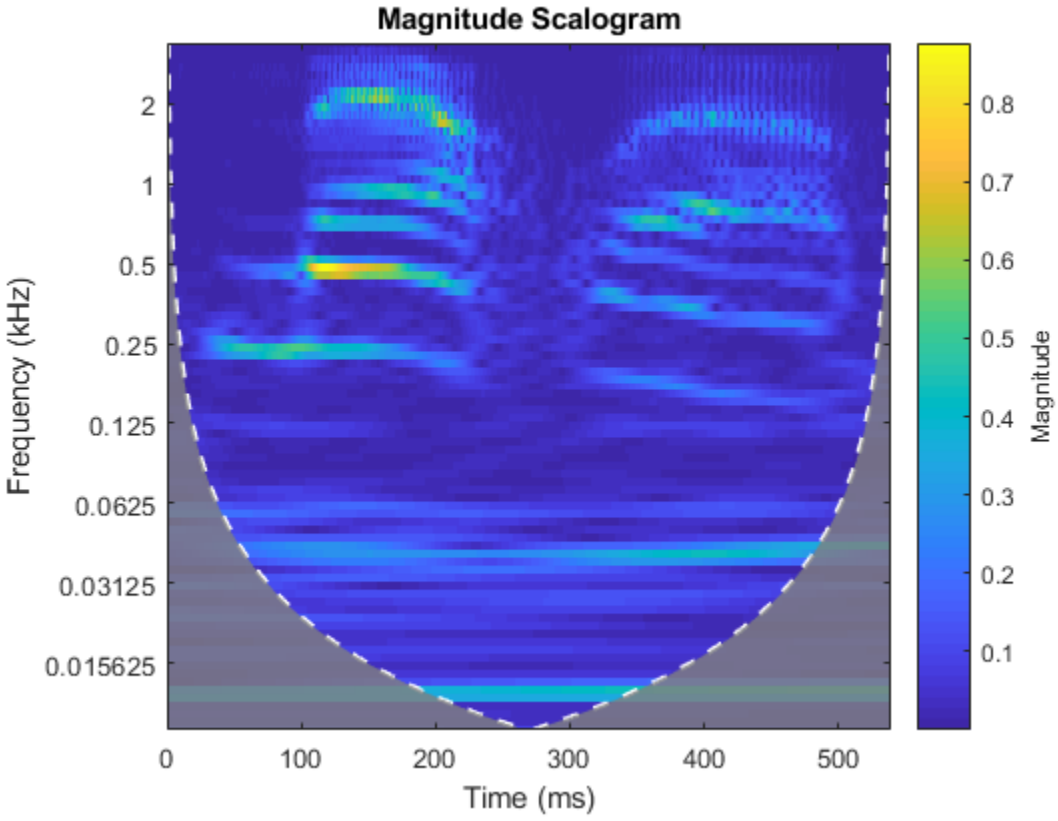
Obtain the continuous wavelet transform of a speech sample using default values.

```
load mtlb;
w = cwt(mtlb);
```

Continuous Wavelet Transform Using Specified Wavelet

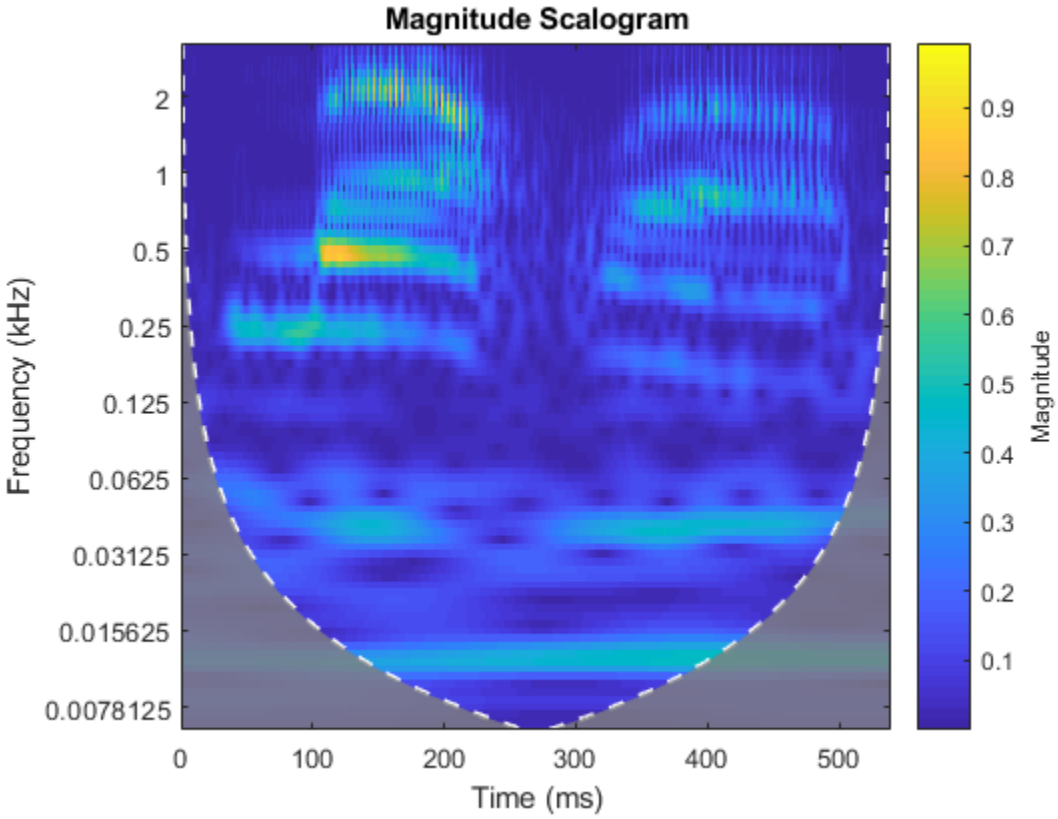
Obtain the continuous wavelet transform of a speech sample using the bump wavelet instead of the default Morse wavelet.

```
load mtlb;
cwt(mtlb,'bump',Fs);
```



Compare the result obtained from the CWT using the default Morse wavelet.

```
cwt(mtlb,Fs);
```



Continuous Wavelet Transform of Two Sine Waves

Create two sine waves with frequencies of 32 and 64 Hz. The data is sampled at 1000 Hz. The two sine waves have disjoint support in time.

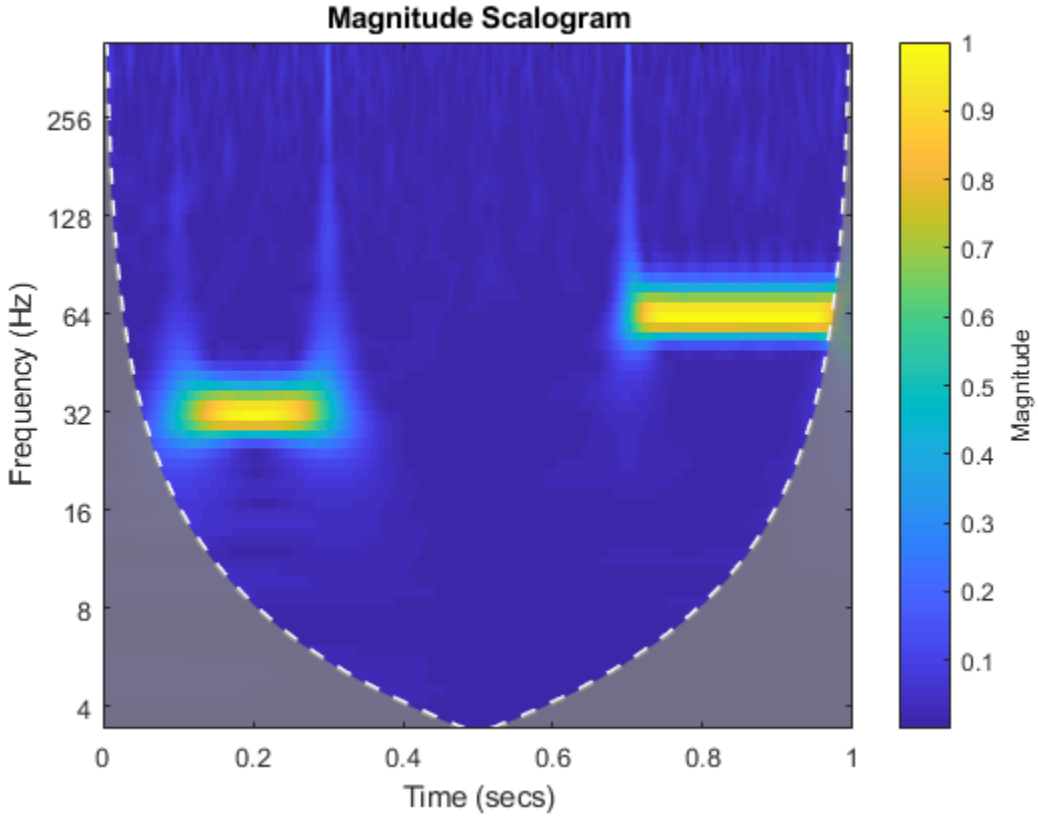
```
Fs = 1e3;
t = 0:1/Fs:1;
x = cos(2*pi*32*t).*(t>=0.1 & t<0.3) + sin(2*pi*64*t).*(t>0.7);
```

Add white Gaussian noise with a standard deviation of 0.05.

```
wgnNoise = 0.05*randn(size(t));
x = x + wgnNoise;
```

Obtain and plot the cwt using a Morse wavelet.

```
cwt(x,1000)
```



Continuous Wavelet Transform of Two Complex Exponentials

Create two complex exponentials, of different amplitudes, with frequencies of 32 and 64 Hz. The data is sampled at 1000 Hz. The two complex exponentials have disjoint support in time.

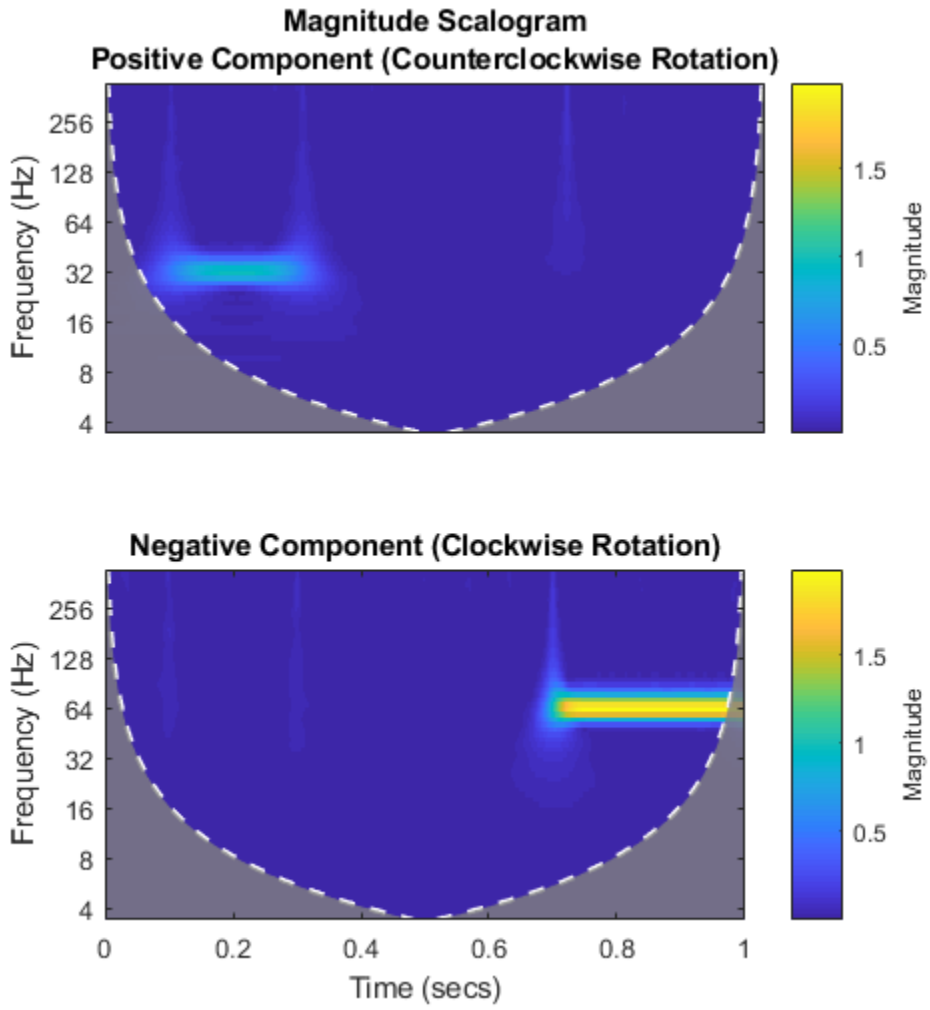
```
Fs = 1e3;
t = 0:1/Fs:1;
z = exp(1i*2*pi*32*t).*(t>=0.1 & t<0.3)+2*exp(-1i*2*pi*64*t).*(t>0.7);
```

Add complex white Gaussian noise with a standard deviation of 0.05.

```
wgnNoise = 0.05/sqrt(2)*randn(size(t))+1i*0.05/sqrt(2)*randn(size(t));
z = z+wgnNoise;
```

Obtain and plot the cwt using a Morse wavelet.

```
cwt(z,Fs)
```



Note the magnitudes of the complex exponential components in the colorbar are essentially their amplitudes even though they are at different scales. This is a direct result of the L1 normalization. You can verify this by executing this script and exploring each subplot with a datacursor.

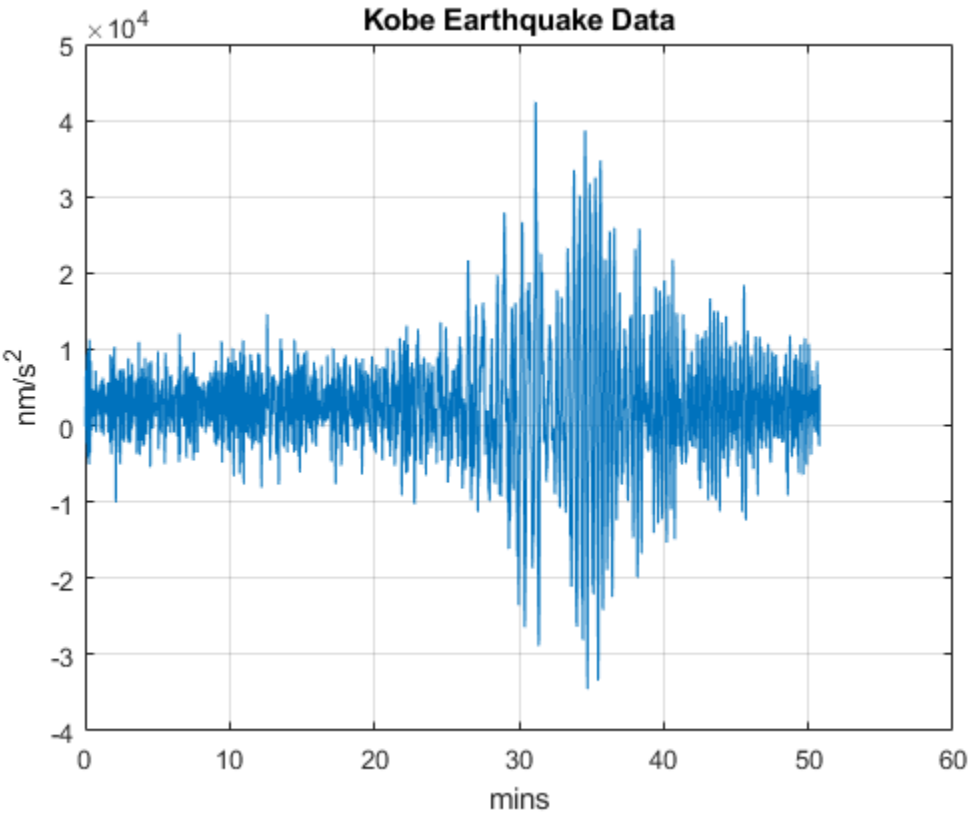
Continuous Wavelet Transform of Earthquake Data

Obtain the CWT of Kobe earthquake data. The sampling frequency is 1 Hz.

```
load kobe;
```

Plot the earthquake data.

```
plot((1:numel(kobe))./60,kobe);
xlabel('mins');
ylabel('nm/s^2');
grid on
title('Kobe Earthquake Data');
```

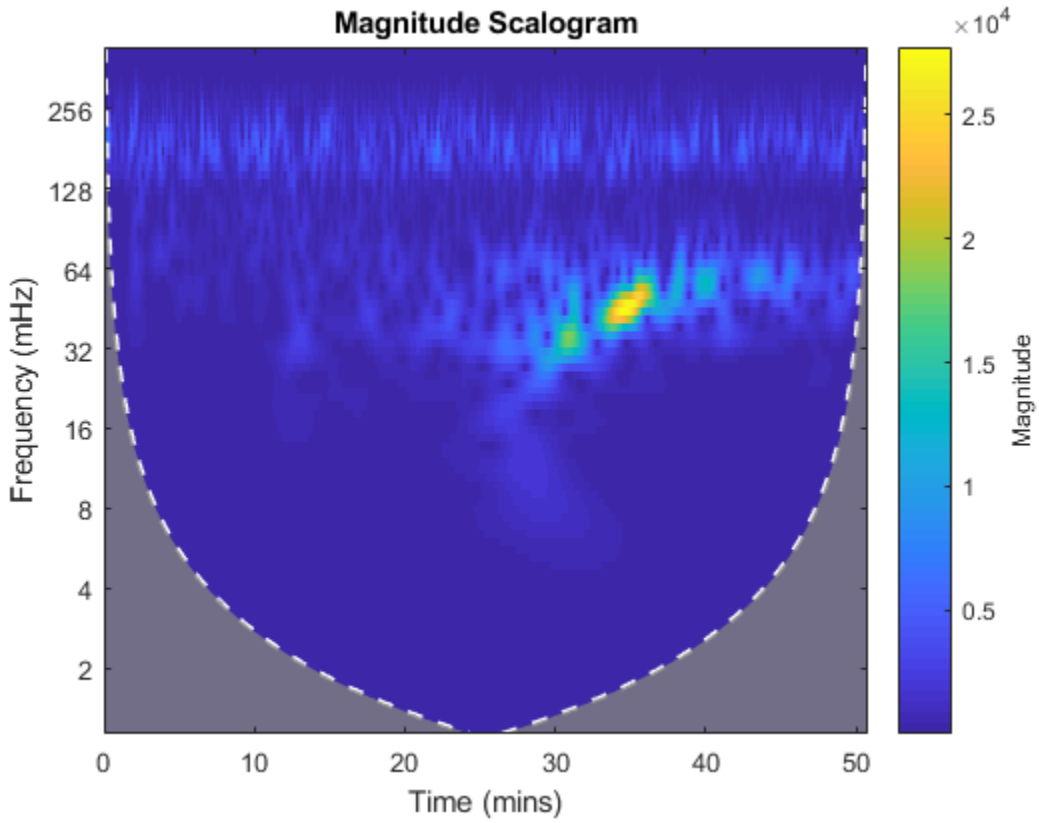


Obtain the CWT, frequencies, and cone of influence.

```
[wt,f,coi] = cwt(kobe,1);
```

Plot the data, including the cone of influence.

```
cwt(kobe,1);
```

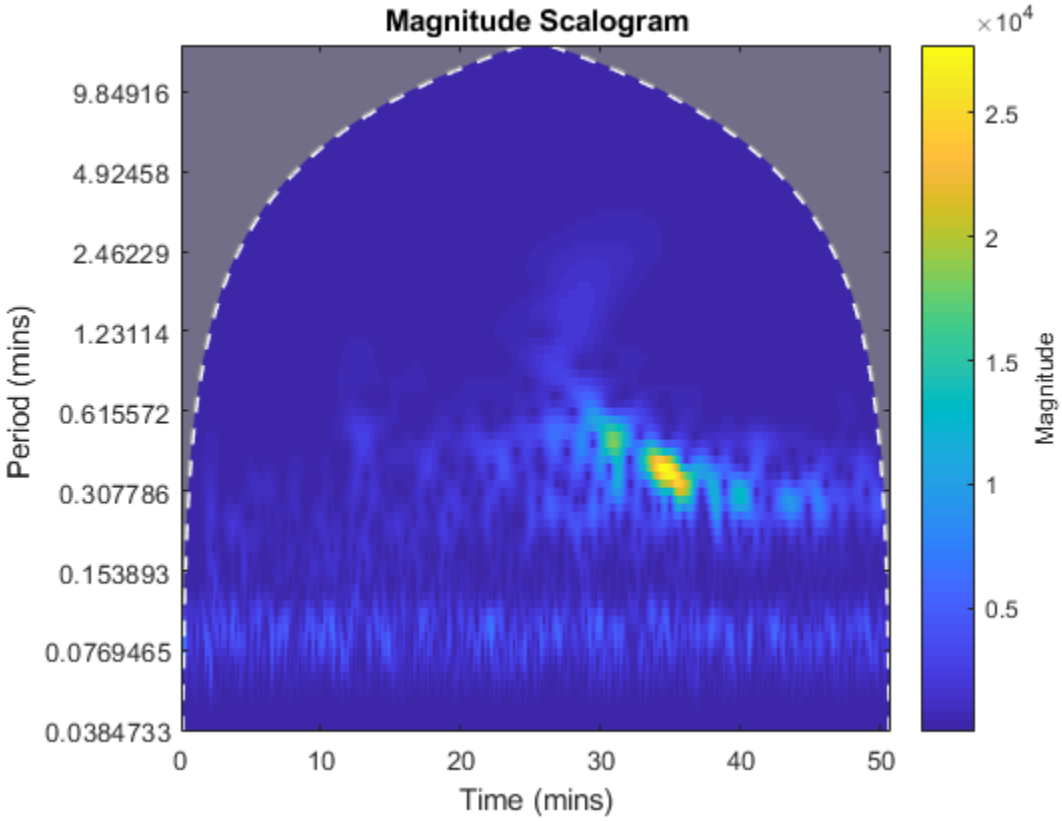


Obtain the CWT, time periods, and cone of influence by specifying a time duration instead of a sampling frequency.

```
[wt,periods,coi] = cwt(kobe,minutes(1/60));
```

View the same data by specifying a sampling period input instead of a frequency.

```
cwt(kobe,minutes(1/60));
```



Input Arguments

`x` — Input signal

vector of real or complex values

Input signal, specified as a row or column vector. `x` is a double-precision real- or complex-valued vector and must have at least four samples.

`wname` — Analytic wavelet

`'morse'` (default) | `'amor'` | `'bump'`

Analytic wavelet used to compute the CWT, specified as `'morse'`, `'amor'`, or `'bump'`. These character vectors specify the analytic Morse, Morlet, and bump wavelet, respectively.

The default Morse wavelet uses a symmetry parameter, γ, that is equal to 3 and has a time-bandwidth product that of 60.

`fs` — Sampling frequency

positive scalar

Sampling frequency, in Hz, specified as a positive scalar. If you specify `fs`, then you cannot specify `ts`.

`ts` — Time duration

positive `duration` scalar

Time duration, also known as the sampling interval, specified as a positive `duration` scalar. Valid durations are `years`, `days`, `hours`, `minutes`, and `seconds`. You cannot use calendar durations. If you specify `ts`, then you cannot specify `fs`.

Example: `wt = cwt(x,hours(12))`

Data Types: `duration`

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: `'ExtendSignal',false` indicates that the signal is not extended.

`ExtendSignal` — Extend input signal symmetrically

`true` (default) | `false`

Option to extend the input signal symmetrically by reflection, specified as the comma-separated pair consisting of `'ExtendSignal'` and either `true` or `false`. Extending the signal symmetrically can mitigate boundary effects. If you specify `true`, then the signal is extended. If you specify `false`, then the signal is not extended.

`VoicesPerOctave` — Number of voices per octave
`10` (default) | even integer from 4 to 48

Number of voices per octave to use for the CWT, specified as the comma-separated pair consisting of `'VoicesPerOctave'` and an even integer from 4 to 48. The CWT scales are discretized using the specified number of voices per octave. The energy spread of the wavelet in frequency and time automatically determines the minimum and maximum scales.

`NumOctaves` — Number of octaves
positive integer

Number of octaves, specified as the comma-separated pair consisting of `'NumOctaves'` and a positive integer. The number of octaves cannot exceed `floor(log2(numel(x)))-1`. Specifying a `NumOctaves` value overrides the automatic determination of the maximum scale. If you do not need to examine lower frequency values, use a smaller `NumOctaves` value.

`TimeBandwidth` — Time-bandwidth product of Morse wavelet
60 (default) | scalar greater than 3 and less than or equal to 120

Time-bandwidth product of the Morse wavelet, specified as the comma-separated pair consisting of `'TimeBandwidth'` and a scalar greater than 3 and less than or equal to 120. The symmetry parameter, gamma (γ), is fixed at 3. Wavelets with larger time-bandwidth products have larger spreads in time and narrower spreads in frequency. The standard deviation of the Morse wavelet in time is approximately `1/2*sqrt(TimeBandwidth/2)`.

If you specify `'TimeBandwidth'`, you cannot specify `'WaveletParameters'`. To specify both the symmetry and time-bandwidth product, use `'WaveletParameters'` instead.

`WaveletParameters` — Symmetry and time-bandwidth product of Morse wavelet
`(3,60)` (default) | two-element vector of scalars

Symmetry and time-bandwidth product of Morse wavelet, specified as the comma-separated pair consisting of `'WaveletParameters'` and a two-element vector of scalars.

The first element is the symmetry, γ, which must be greater than or equal to 1. The second element is the time-bandwidth product which must be greater than γ. The ratio of the time-bandwidth product to γ cannot exceed 40.

When γ is equal to 3, the Morse wavelet is perfectly symmetric in the frequency domain and the skewness is 0. When γ is greater than 3, the skewness is positive. When γ is less than 3, the skewness is negative.

If you specify `'WaveletParameters'`, you cannot specify `'TimeBandwidth'`.

Output Arguments

`wt` — Continuous wavelet transform

matrix

Continuous wavelet transform, returned as a matrix of complex values. By default, `cwt` uses the analytic Morse (3,60) wavelet, where 3 is the symmetry and 60 is the time-bandwidth product. `cwt` uses 10 voices per octave. If `x` is real-valued, `wt` is an *Na*-by-*N* matrix, where *Na* is the number of scales, and *N* is the number of samples in `x`. If `x` is complex-valued, `wt` is a 3-D matrix, where the first page is the CWT for the positive scales (analytic part or counterclockwise component) and the second page is the CWT for the negative scales (anti-analytic part or clockwise component). The minimum and maximum scales are determined automatically based on the energy spread of the wavelet in frequency and time. See “Algorithms” on page 1-93 for information on how the scales are determined.

Data Types: `double`

`f` — Frequencies

vector

Frequencies of the CWT, returned as a vector. If you specify a sampling frequency, `fs`, then `f` is in Hz. If you do not specify `fs`, `cwt` returns `f` in cycles per sample.

`period` — Time periods

array

Time periods, returned as an array of durations. The durations are in the same format as `ts`. Each row corresponds to a period.

`coi` — Cone of influence

array of doubles | array of durations

Cone of influence for the CWT, returned as either an array of doubles or array of durations. The cone of influence indicates where edge effects occur in the CWT. If you specify a sampling frequency, `fs`, the cone of influence is in Hz. If you specify a time duration, `ts`, the cone of influence is in periods. Due to the edge effects, give less credence to areas that are outside or overlap the cone of influence.

Algorithms

Minimum Scale

To determine the minimum scale, first obtain the Fourier transform, ω_x of the base wavelet. Then, find the peak frequency of that transformed data. For Morse wavelets, dilate the wavelet so that the Fourier transform of the wavelet at π radians is equal to 10% of the peak value. The smallest scale occurs at the largest frequency:

$$s_0 = \frac{\omega_x^{'}}{\pi}$$

As a result, the smallest scale is the minimum of (2, s_0). For Morse wavelets, the smallest scale is usually s_0. For the Morlet wavelet, the smallest scale is usually 2.

Maximum Scale

Both the minimum and maximum scales of the CWT are determined automatically based on the energy spread of the wavelet in frequency and time. To determine the maximum scale, CWT uses the following algorithm.

The standard deviation of the Morse wavelet in time, σ_t, is approximately $\sqrt{\frac{P}{2}}$, where P is the time-bandwidth product. The standard deviation in frequency, σ_f, is approximately $\frac{1}{2}\sqrt{\frac{2}{P}}$. If you scale the wavelet by some $s > 1$, the time duration changes to $2s\sigma_t = N$, which is the wavelet stretched to equal the full length (N samples) of the

input. You cannot translate this wavelet or stretch it further without causing it to wrap,

so the largest scale is $floor\left(\frac{N}{2\sigma_t}\right)$.

Wavelet transform scales are powers of 2 and are denoted by $s_0\left(2^{\frac{1}{NV}}\right)^j$. NV is the number of voices per octave, and j is from 0 to the largest scale. For a specific small scale, s_0:

$$s_0\left(2^{\frac{1}{NV}}\right)^j \le \frac{N}{2\sigma_t}$$

Converting to log2:

$$j\log_2\left(2^{\frac{1}{NV}}\right) \le \log_2\left(\frac{N}{2\sigma_t s_0}\right)$$

$$j \le NV\log_2\frac{N}{2\sigma_t s_0}$$

Therefore, the maximum scale is

$$2^{floor\left(NV\log_2\frac{N}{2\sigma_t s_0}\right)}$$

L1 Norm for CWT

In integral form, the CWT preserves energy. However, when you implement the CWT numerically, energy is not preserved. In this case, regardless of the normalization you use, the CWT is not an orthonormal transform. The `cwt` function uses L1 normalization

Wavelet transforms commonly use L2 normalization of the wavelet. For the L2 norm, dilating a signal by 1/s, where s is greater than 0, is defined as follows:

$$\left\|x\left(\frac{t}{s}\right)\right\|_2^2 = s\left\|x(t)\right\|_2^2$$

The energy is now s times the original energy. When included in the Fourier transform, multiplying by $1/\sqrt{s}$ produces different weights being applied to different scales, so that the peaks at higher frequencies are reduced more than the peaks at lower frequencies.

In many applications, L1 normalization is better. The L1 norm definition does not include squaring the value, so the preserving factor is $1/s$ instead of $1/\sqrt{s}$. Instead of high-frequency amplitudes being reduced as in the L2 norm, for L1 normalization, all frequency amplitudes are normalized to the same value. Therefore, using the L1 norm shows a more accurate representation of the signal. See example "Continuous Wavelet Transform of Two Complex Exponentials" on page 1-84.

References

[1] Lilly, J. M., and S. C. Olhede. "Generalized Morse Wavelets as a Superfamily of Analytic Wavelets." *IEEE Transactions on Signal Processing*. Vol. 60, No. 11, 2012, pp. 6036–6041.

[2] Lilly, J. M., and S. C. Olhede. "Higher-Order Properties of Analytic Wavelets." *IEEE Transactions on Signal Processing*. Vol. 57, No. 1, 2009, pp. 146–160.

[3] Lilly, J. M. *jLab: A data analysis package for Matlab*, version 1.6.2. 2016. http://www.jmlilly.net/jmlsoft.html.

See Also

`cwt` | `duration` | `dwt` | `icwt` | `wavedec` | `wavefun` | `waveinfo` | `wcodemat`

Topics

"Continuous and Discrete Wavelet Transforms"
"1-D Continuous Wavelet Analysis"
"Time-Frequency Analysis with the Continuous Wavelet Transform"
"Morse Wavelets"

Introduced in R2016b

cwt

Continuous 1-D wavelet transform

Note This version of `cwt` is no longer recommended. Use the updated `cwt` instead.

Syntax

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

Description

`coefs = cwt(x,scales,'wname')` returns the continuous wavelet transform (CWT) of the real-valued signal S. The wavelet transform is computed for the specified scales using the analyzing wavelet *wname*. `scales` is a 1-D vector with positive elements. The character vector *wname* denotes a wavelet recognized by `wavemngr`. `coefs` is a matrix with the number of rows equal to the length of `scales` and number of columns equal to the length of the input signal. The k-th row of `coefs` corresponds to the CWT coefficients for the k-th elemens in the scales vector.

`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. See “Definitions” on page 1-99 for coloration options.

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

`[coefs,sgram] = cwt(x,scales,'wname','scal')` returns and plots the scalogram. `'scal'` produces an image plot of the scalogram.

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

`[coefs,frequencies] = cwt(x,scales,wname, samplingperiod)` returns the frequencies in cycles per unit time corresponding to the scales and the analyzing wavelet *wname*. `samplingperiod` is a positive real-valued scalar. If the units of `samplingperiod` are seconds, the frequencies are in hertz.

`[coefs,sgram,frequencies] = cwt(x,scales,wname, samplingperiod,'scal')` returns the scalogram and the frequencies corresponding to the scales and the analyzing wavelet. If you have at least two elements in `scales`, you can also use the flag `'scalCNT'` to output the scalogram. The `samplingperiod` is only used in the conversion of scales to frequencies. Specifying `samplingperiod` does not affect the appearance of plots generated by `cwt`.

Examples

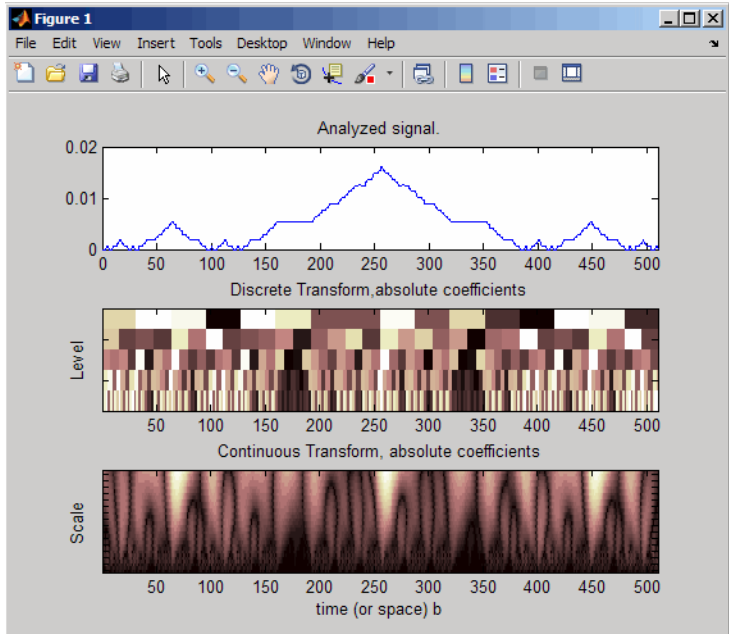
Plot the continuous wavelet transform and scalogram using sym2 wavelet at all integer scales from 1 to 32, using a fractal signal as input:

```
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',[0 510]);
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');
```



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.

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.

See Also

`cwt` | `dwt` | `wavedec` | `wavefun` | `waveinfo` | `wcodemat`

Topics

“New Wavelet for CWT”

Introduced before R2006a

cwtft

Continuous wavelet transform using FFT algorithm

Note This function is no longer recommended. Use `cwt` instead.

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 More About on page 1-107 for descriptions of valid analyzing wavelets.

For additional default values, see `scales` in "Name-Value Pair Arguments" on page 1-102.

`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 1-102 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 the More About on page 1-107 for descriptions of the default for each wavelet.
2. `ds` — Spacing between scales. The default `ds` depends on the wavelet. See the More About on page 1-107 for descriptions of the default for eacj wavelet. You can construct a linear or logarithmic scale vector using `ds`. See type on page 1-103 for a description of the type of spacing.

3. `nb` — Number of scales. The default `nb` depends on the wavelet. See the More About on page 1-107 for descriptions of the default for each wavelet.
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 1-105 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. To include a parameter for the wavelet, use a cell array. For example, to specify a fourth order derivative of a Gaussian wavelet, use `'wavelet',{'dog',4}` as in `cwtstruct = (sig,'wavelet',{'dog',4})`.

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, which is the Mexican hat or Ricker wavelet..
- `'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, which is also known as the Ricker 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

- `'bump'` — Bump wavelet

See the More About on page 1-107 for formal definitions of the supported analyzing wavelets and associated defaults.

Default: `'morl'`

`padmode`

Signal extension mode. See `dwtmode` for supported extension modes. By default, `cwtft` does not extend the signal prior to computing the CWT. In a Fourier-transform-based CWT algorithm, extending a signal can mitigate wrap-around effects. The number of CWT coefficients in each row of the output matrix `cwtstruct.cfs` is truncated to match the length of the input signal.

Output Arguments

`cwtstruct`

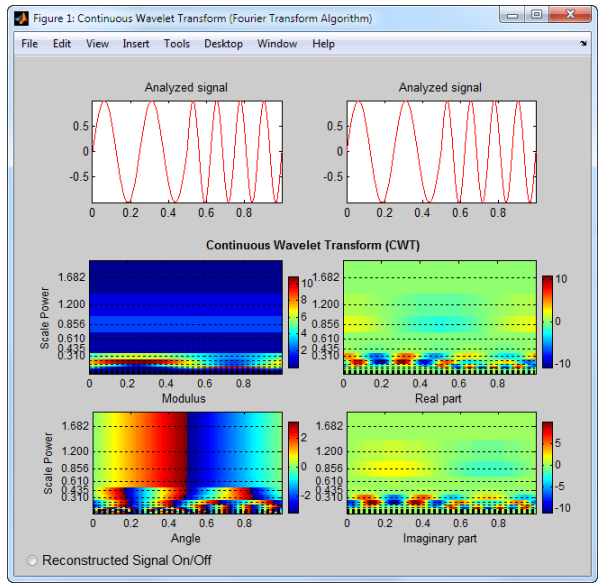
A structure array with six fields. The fields of the structure array are:

- `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.
- `scales` — Vector of scales at which the CWT is computed. The length of `cwtstruct.scales` is equal to the row dimension of `cwtstruct.cfs`.
- `frequencies` — Frequencies in cycles per unit time (or space) corresponding to the scales. If the sampling period units are seconds, the frequencies are in hertz. The elements of frequencies are in decreasing order to correspond to the elements in the scales vector. Use this field to examine the CWT in the time-frequency plane.
- `omega` — Vector of angular frequencies used in the Fourier transform of the wavelet. This field is used in `icwtft` and `icwtlin` for the inversion of the CWT for all wavelets except the bump wavelet.
- `meanSIG` — Mean of the analyzed signal
- `dt` — The sampling interval of the 1–D input signal
- `wav` — Analyzing wavelet

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');
```

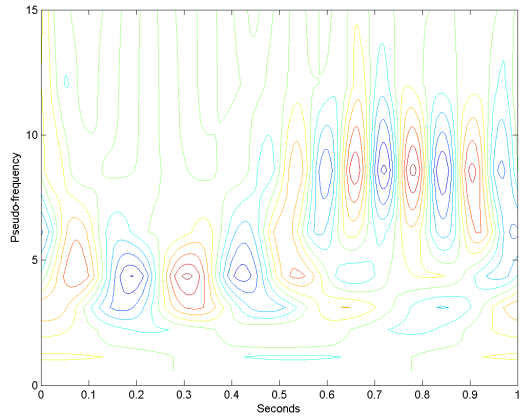


You can display or hide the reconstructed signal using the radio button at the bottom left of the figure. When you select the radio button, the maximum and quadratic relative errors are computed and displayed along with the reconstructed signal.

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.

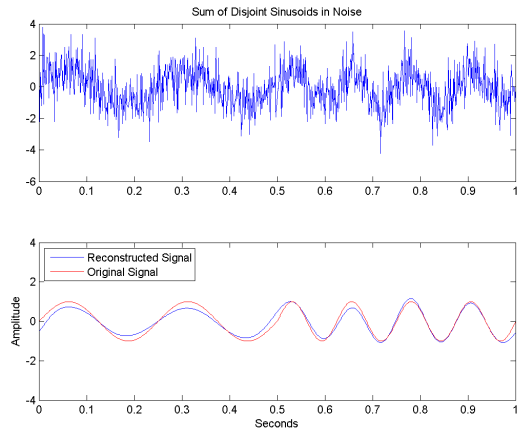
```
rng default % Reset random number generator for reproducible results
N = 1024;
% Sampling interval is 1/1023
t = linspace(0,1,N);
y = sin(2*pi*4*t).*(t<=0.5)+sin(2*pi*8*t).*(t>0.5);
ynoise = y+randn(size(t));
% Because the sampling interval differs from the default
% you must input it along with 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');
```



Definitions

Morlet Wavelet

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

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

where $U(\omega)$ is the Heaviside step function [5] on page 1-111.

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

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

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

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

The default value of ω_0 is 6.

The default smallest scale for the Morlet wavelets is `s0 = 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 `NbSc = fix(log2(length(sig)*dt/s0)/ds).`

The default scales for the Morlet wavelet are `s0*2.^((0:NbSc-1)*ds).`

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 Γ( ) denotes the gamma function [5] on page 1-111.

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 default smallest scale for the DOG wavelet is `s0 = 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 `NbSc = fix(log2(length(sig)*dt/s0)/ds).`

The default scales for the DOG wavelet are `s0*2.^((0:NbSc-1)*ds).`

Paul Wavelet

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

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

where *U(ω)* is the Heaviside step function [5] on page 1-111.

The default order of the Paul wavelet is 4.

The default smallest scale for the Paul wavelet is `s0 = 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 `NbSc = fix(log2(length(sig)*dt/s0)/ds)`.

The default scales for the Paul wavelet are `s0*2.^((0:NbSc-1)*ds)`.

Bump wavelet

The Fourier transform of the analytic bump wavelet, `'bump'`, with parameters μ and σ is

$$\breve{\psi}(s\omega) = e^{\left(1-\frac{1}{1-(s\omega-\mu)^2/\sigma^2}\right)} 1_{[(\mu-\sigma)/s,(\mu+\sigma)/s]}$$

where $1_{[(\mu-\sigma)/s,\,(\mu+\sigma)/s]}$ is the indicator function for the interval $(\mu-\sigma)/s \le \omega \le (\mu+\sigma)/s$.

Valid values for μ are [3,6]. Valid values for σ are [0.1, 1.2]. Smaller values of σ result in a wavelet with superior frequency localization but poorer time localization. Larger values of σ produce a wavelet with better time localization and poorer frequency localization.

The default values for μ and σ are 5 and 0.6 respectively.

The default smallest scale for the bump wavelet is `s0 = 2*dt` where `dt` is the sampling period.

The default spacing between scales for the bump wavelet is `ds=1/10`.

The default number of scales for the bump wavelet is `NbSc = fix(log2(length(sig)*dt/s0)/ds)`.

The default scales for the bump wavelet are `s0*2.^((0:NbSc-1)*ds)`.

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 "Continuous Wavelet Transform and Scale-Based Analysis".

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`.

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` | `icwtft` | `icwtlin`

Topics

“Continuous and Discrete Wavelet Transforms”
“Continuous Wavelet Transform and Scale-Based Analysis”
“Inverse Continuous Wavelet Transform”
“Time-Frequency Analysis with the Continuous Wavelet Transform”

Introduced in R2011a

cwtftinfo

Valid analyzing wavelets for FFT-based CWT

Note This function is no longer recommended. Use `cwt` instead.

Syntax

```
cwtftinfo
```

Description

`cwtftinfo` displays expressions for the Fourier transforms of valid analyzing wavelets for use with `cwtft`.

Examples

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

```
cwtftinfo
```

Definitions

Morlet Wavelet

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

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

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

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

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

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

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

The default value of ω_0 is 6.

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* or *Ricker* wavelet.

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

Paul Wavelet

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

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

where $U(\omega)$ is the Heaviside step function. The Paul wavelet is analytic.

The default order of the Paul wavelet is 4.

Bump wavelet

The Fourier transform of the analytic bump wavelet, `'bump'`, with parameters μ and σ is

$$\breve{\psi}(s\omega) = e^{\left(1-\frac{1}{1-(s\omega-\mu)^2/\sigma^2}\right)} \quad 1_{[(\mu-\sigma)/s,(\mu+\sigma)/s]}$$

where $1_{[(\mu-\sigma)/s,\,(\mu+\sigma)/s]}$ is the indicator function for the interval $(\mu-\sigma)/s \le \omega \le (\mu+\sigma)/s$.

Valid values for μ are [3,6]. Valid values for σ are [0.1, 1.2]. Smaller values of σ result in a wavelet with superior frequency localization but poorer time localization. Larger values of σ produce a wavelet with better time localization and poorer frequency localization.

The default values for μ and σ are 5 and 0.6 respectively.

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` | `icwtlin`

Topics

“Continuous and Discrete Wavelet Transforms”
“Continuous Wavelet Transform and Scale-Based Analysis”
“Inverse Continuous Wavelet Transform”

Introduced in R2011a

cwtftinfo2

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.

Examples

Available Wavelets with Parameters

```
cwtftinfo2
```

```
 CWTFTINFO2 Information on wavelets for CWTFT2
  CWTFTINFO2 provides information on the available wavelets
  for 2-D Continuous Wavelet Transform using FFT.
  The wavelets are defined by their Fourier transform.

  The formulae giving the Fourier transform of
  the wavelet which short name (see below) is SNAME
  will be displayed using CWTFTINFO2(SNAME).

  The table below gives the short name of each wavelet
  and the associated parameters: first, the name of parameter
```

```
and then the default value.

WAV_Param_Table = {...
    'morl'      ,
         defaults: omega0 = 6; sigma = 1; epsilon = 1;
    'mexh'      ,
         defaults: p = 2; sigmax = 1; sigmay = 1;
    'paul'      ,
         defaults: p = 4;
    'dog'       ,
         defaults: alpha = 2;
    'cauchy'    ,
         defaults: alpha = pi/6; sigma = 1; L = 4; M = 4;
    'escauchy'  ,
         defaults: alpha = pi/6; sigma = 1; L = 4; M = 4;
    'gaus'      ,
         defaults: p = 1; sigmax = 1; sigmay = 1;
    'wheel'     ,
         defaults:  sigma = 2;
    'fan'       ,
         defaults: omega0 = 5.336; sigma = 1; epsilon = 1; J = 6.5;
    'pethat'    ,
         defaults: No parameters.
    'dogpow'    ,
         defaults: alpha = 1.25; p = 2;
    'esmorl'    ,
         defaults: omega0 = 6; sigma = 1; epsilon = 1;
    'esmexh'
         defaults:   sigma = 1; epsilon = 0.5;
    'gaus2'     ,
         defaults:  p = 1; sigmax = 1; sigmay = 1;
    'gaus3'     ,
         defaults:  A = 1; B = 1; p = 1; sigmax = 1; sigmay = 1;
    'isodog'    ,
         defaults:  alpha = 1.25;
    'dog2'      ,
         defaults = alpha = 1.25;
    'isomorl'   ,
         defaults:  omega0 = 6; sigma = 1;
    'rmorl'     ,
         defaults:  omega0 = 6; sigma = 1; epsilon = 1;
    'endstop1'  ,
         defaults:  omega0 = 6;
    'endstop2'  ,
```

```
            defaults:  omega0 = 6; sigma = 1;
      'gabmexh'   ,
            defaults:  omega0 = 5.336; epsilon = 1;
      'sinc'      ,
            defaults:  Ax = 1; Ay = 1; p = 1; omega0X= 0; Omega0Y = 0;
      };

  The various wavelets may be grouped in families as follow:
    MORLET:  'morl'   , 'esmorl' , 'rmorl' , 'isomorl'
    DOG:     'dog'    , 'isodog' , 'dog2'  , 'dogpow'
    GAUSS:   'mexh'   , 'gaus'   , 'gaus2' , 'gaus3' , 'esmexh'
    PAUL:    'paul'
    CAUCHY:  'cauchy' , 'escauchy'
    WHEEL:   'wheel', 'pethat'
    MISCELLANEOUS : 'endstop1' , 'endstop2' , 'gabmexh' , 'sinc' , 'fan'

    REFERENCES
      Two-Dimensional Wavelets and their Relatives
      J.-P. Antoine, R. Murenzi, P. Vandergheynst andS. Twareque Ali
      Cambridge University Press - 2004

      Two-dimensional wavelet transform profilometry
      Fringe Pattern Analysis Using Wavelet
      Liverpool John Moores University
      http://www.ljmu.ac.uk

    See also CWTFT2
```

Display the Expression for the 2-D Fourier Transform

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

```
cwtftinfo2('cauchy')
```

cauchy

$$\widehat{\psi}\left(\omega_x,\omega_y\right)=\left[\sin\left(\alpha\right)\omega_x+\cos\left(\alpha\right)\omega_y\right]^L\ldots$$
$$\left[-\sin\left(\alpha\right)\omega_x+\cos\left(\alpha\right)\omega_y\right]^M\left[\tan\left(\alpha\right)\omega_x>\left|\omega_y\right|\right]e^{-\sigma\frac{\left[\left(\omega_x\right)^2+\left(\omega_y\right)^2\right]}{2}}$$
$$\sigma\in\left]0,+\infty\right[,\ \alpha\in\left]0,\frac{\pi}{2}\right[,\ L,M\left]0,+\infty\right[$$

cauchy

Input Arguments

`wname` — Wavelet name

character vector

Wavelet name, specified as a . 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';'sigma',1;'L',4;'M',4}
'escauchy'	{'alpha','pi/6';'sigma',1;'L',4;'M',4}
'gaus'	{'p',1;'sigmax',1;'sigmay',1}
'wheel'	{'sigma',2}
'fan'	{'Omega0X',5.336;'Sigma',1;'Epsilon',1;'J',6.5}
'pethat'	None
'dogpow'	{'alpha',1.25;'p',2}
'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}
'dog2'	{'alpha',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}

Wavelet name	Parameters
`'sinc'`	`{'Ax',1;'Ay',1;'p',1;'Omega0X', 0;'Omega0Y',0}`

Example: `cwtftinfo2('paul')`

Data Types: `char`

Introduced in R2013b

cwtft2

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.

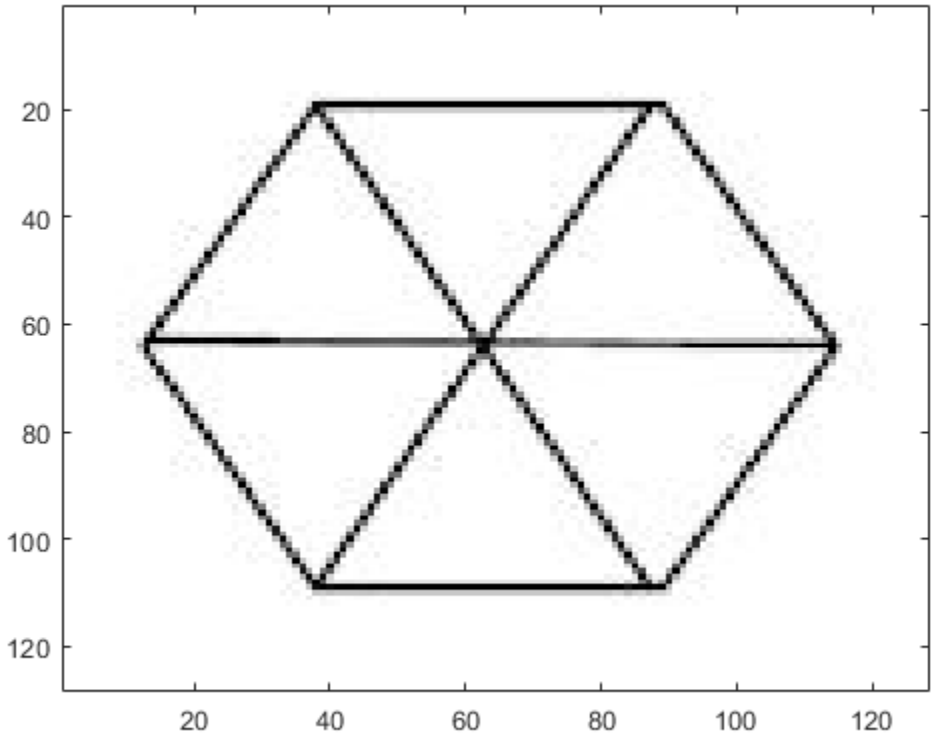
Examples

Compare Isotropic and Anisotropic Wavelets

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);
```



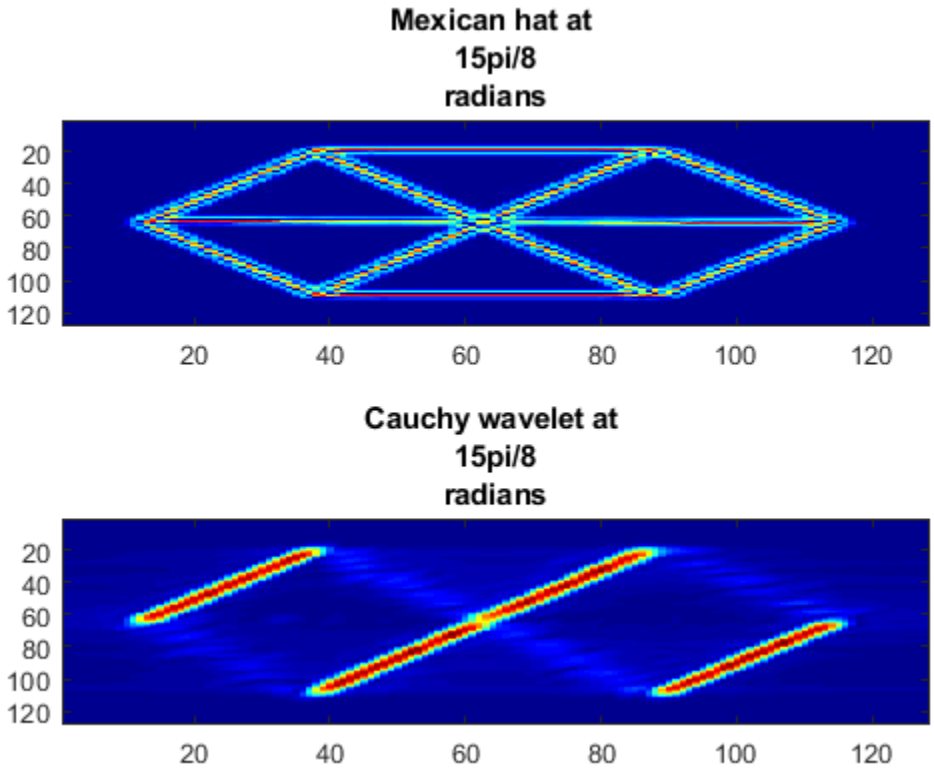
Obtain the scale-one 2-D CWT with both the Mexican hat and Cauchy wavelets. Specify a vector of angles going from 0 to 15π/8 in π/8 increments.

```
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)
```

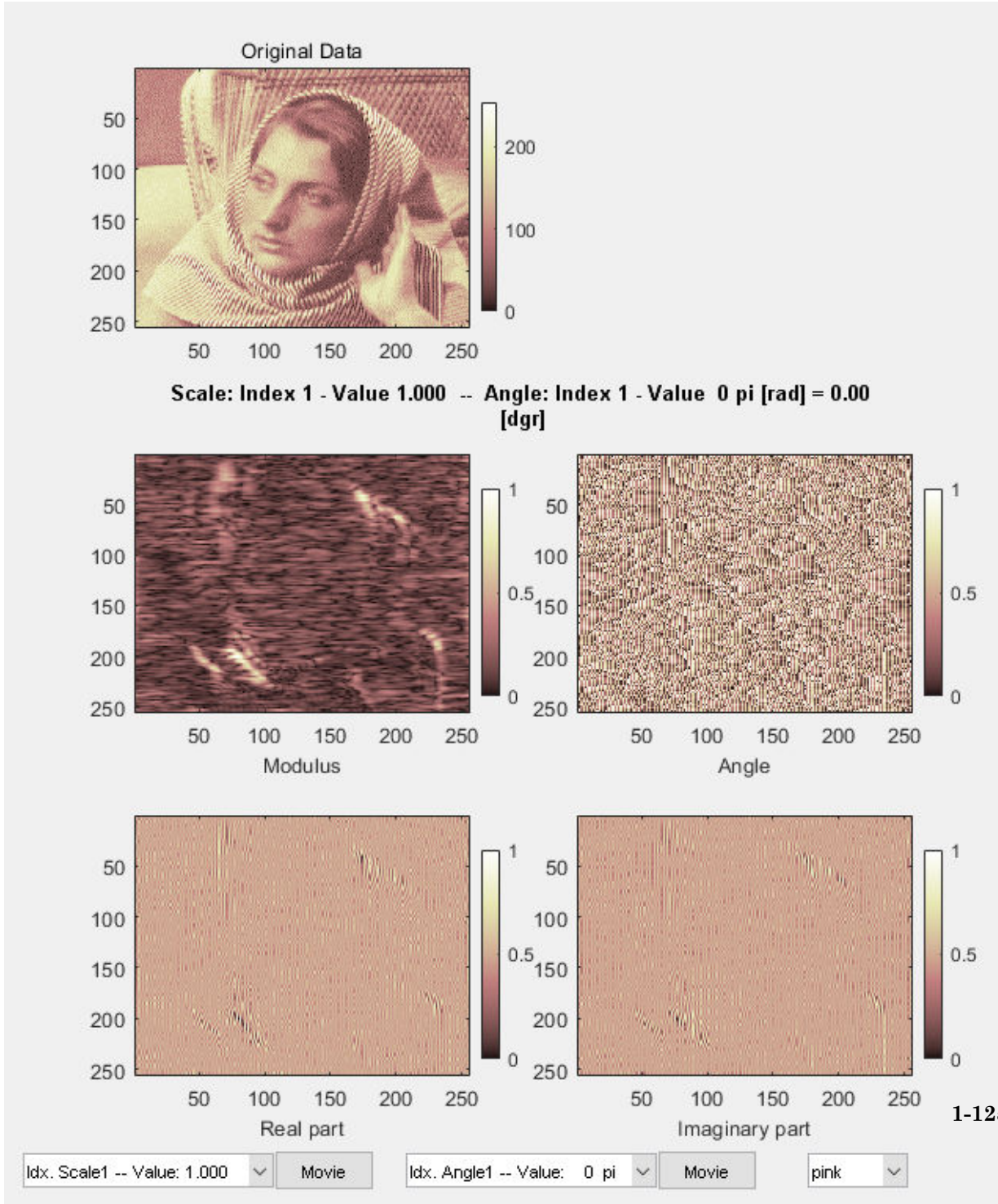
```
    subplot(211)
    imagesc(abs(cwtmexh.cfs(:,:,1,1,angn)));
    title(['Mexican hat at ' angz(angn) 'radians']);
    subplot(212)
    imagesc(abs(cwtcauchy.cfs(:,:,1,1,angn)));
    title(['Cauchy wavelet at ' angz(angn) 'radians']);
    pause(1);
end
```



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');
```


Original Data
Scale: Index 1 - Value 1.000 -- Angle: Index 1 - Value 0 pi [rad] = 0.00 [dgr]
Modulus
Angle
Real part
Imaginary part
Idx. Scale1 -- Value: 1.000
Movie
Idx. Angle1 -- Value: 0 pi
Movie
pink

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);
```

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

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 character vectors:

- `'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) | character vector | structure | cell array

Analyzing wavelet, specified as a comma-separated pair consisting of `'wavelet'` and a character vector, 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 character vector 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 character vector 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

See Also

`cwtftinfo2`

Topics

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

Introduced in R2013b

dbaux

Daubechies wavelet filter computation

The `dbaux` function generates the scaling filter coefficients for the "extremal phase" Daubechies wavelets.

Syntax

```
W = dbaux(N)
W = dbaux(N,SUMW)
```

Description

`W = dbaux(N)` is the order `N` Daubechies scaling filter such that `sum(W) = 1`.

Note

- Instability may occur when `N` is too large. Starting with values of `N` in the 30s range, function output will no longer accurately represent scaling filter coefficients.

`W = dbaux(N,SUMW)` is the order `N` Daubechies scaling filter such that `sum(W) = SUMW`.

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

Examples

Daubechies Extremal Phase Scaling Filter with Specified Sum

This example shows how to determine the Daubechies extremal phase scaling filter with a specified sum. The two most common values for the sum are $\sqrt{2}$ and 1.

Construct two versions of the `db4` scaling filter. One scaling filter sums to $\sqrt{2}$ and the other version sums to 1.

```
NumVanishingMoments = 4;
h = dbaux(NumVanishingMoments,sqrt(2));
m0 = 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 = 4.2589e-13
```

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

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

```
ans = 4.2589e-13
```

Extremal Phase

This example demonstrates that for a given support, the cumulative sum of the squared coefficients of a scaling filter increase more rapidly for an extremal phase wavelet than other wavelets.

First, set the order to 15 and generate the scaling filter coefficients for the Daubechies wavelet and Symlet. Both wavelets have support of length 29.

```
n = 15;
[~,~,LoR_db,~] = wfilters('db15');
[~,~,LoR_sym,~] = wfilters('sym15');
```

Next, generate the scaling filter coefficients for the order 5 Coiflet. This wavelet also has support of length 29.

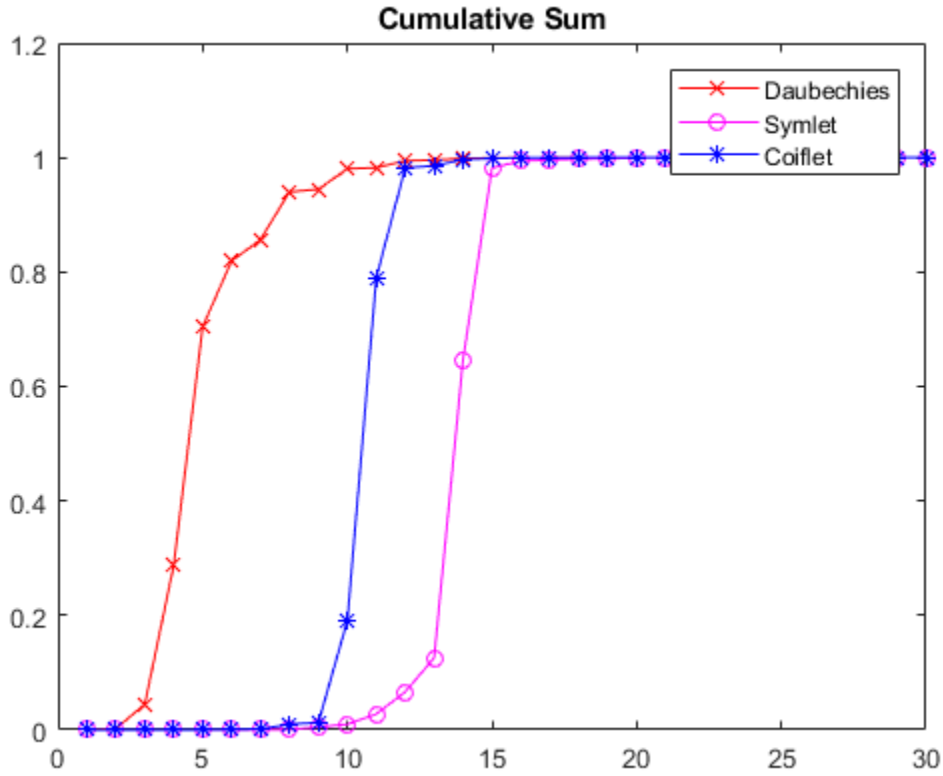
```
[~,~,LoR_coif,~] = wfilters('coif5');
```

Confirm the sum of the coefficients for all three wavelets equals $\sqrt{2}$.

```
sqrt(2)-sum(LoR_db)

ans = 2.2204e-16

sqrt(2)-sum(LoR_sym)

ans = 0

sqrt(2)-sum(LoR_coif)

ans = 2.2204e-16
```

Plot the cumulative sums of the squared coefficients. Note how rapidly the Daubechies sum increases. This is because its energy is concentrated at small abscissas. Since the Daubechies wavelet has extremal phase, the cumulative sum of its squared coefficients increases more rapidly than the other two wavelets.

```
plot(cumsum(LoR_db.^2),'rx-')
hold on
plot(cumsum(LoR_sym.^2),'mo-')
plot(cumsum(LoR_coif.^2),'b*-')
legend('Daubechies','Symlet','Coiflet')
title('Cumulative Sum')
```



Input Arguments

`N` — Order of Daubechies scaling filter

positive integer

Order of Daubechies scaling filter, specified as a positive integer.

Data Types: `single` | `double`

`SUMW` — Sum of coefficients

1 (default) | positive scalar

Sum of coefficients, specified as a positive scalar. Set to `sqrt(2)` to generate vector of coefficients whose norm is 1.

Data Types: `single` | `double`

Output Arguments

w — Scaling filter coefficients

vector

Scaling filter coefficients returned as a vector.

The scaling filter coefficients satisfy a number of properties. As the example "Daubechies Extremal Phase Scaling Filter with Specified Sum" on page 1-130 demonstrates, you can construct scaling filter coefficients with a specific sum. If $\{h_k\}$ denotes the set of order `N` Daubechies scaling filter coefficients, where $n = 1, ..., 2N$, then $\sum_{n=1}^{2N} h_n^2 = 1.$ The coefficients also satisfy the relation $\sum_n h(n)h(n-2k) = \delta(k).$ You can use these properties to check your results.

Limitations

- The computation of the `dbN` Daubechies scaling filter requires the extraction of the roots of a polynomial of order `4N`. Instability may occur beginning with values of `N` in the 30s.

Definitions

Extremal Phase

Constructing a compactly supported orthogonal wavelet basis involves choosing roots of a particular polynomial equation. Different choices of roots will result in wavelets whose phases are different. Choosing roots that lie within the unit circle in the complex plane results in a filter with highly nonlinear phase. Such a wavelet is said to have extremal

phase, and has energy concentrated at small abscissas. Let $\{h_k\}$ denote the set of scaling coefficients associated with an extremal phase wavelet, where $k = 1,...,N$. Then for any other set of scaling coefficients $\{g_k\}$ resulting from a different choice of roots, the following inequality will hold for all $J = 1,...,N$:

$$\sum_{k=1}^{J} g_k^2 \le \sum_{k=1}^{J} h_k^2$$

The inequality is illustrated in the example "Extremal Phase" on page 1-131. The $\{h_k\}$ are sometimes called a *minimal delay filter* [2].

Algorithms

The algorithm used is based on a result obtained by Shensa [3], 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

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

- 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 $2N$.

  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 [4] (p. 157).

References

[1] Daubechies, I. *Ten Lectures on Wavelets*, CBMS-NSF conference series in applied mathematics, SIAM Ed., 1992.

[2] Oppenheim, Alan V., and Ronald W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.

[3] 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.

[4] Strang, G., and T. Nguyen.*Wavelets and Filter Banks*. Wellesley, MA: Wellesley-Cambridge Press, 1996.

See Also

`dbwavf` | `symaux` | `wfilters`

Introduced before R2006a

dbwavf

Daubechies wavelet filter

Syntax

```
F = dbwavf(W)
```

Description

`F = dbwavf(W)` returns the scaling filter associated with Daubechies wavelet specified by the character vector *`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`

Introduced before R2006a

ddencmp

Default values for denoising or compression

Syntax

```
[THR,SORH,KEEPAPP,CRIT] = ddencmp(IN1,IN2,X)
[THR,SORH,KEEPAPP] = ddencmp(IN1,'wv',X)
[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.

You can use `ddencmp` for 1-D signals or 2-D images.

`[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).

`IN1` is `'den'` for denoising or `'cmp'` for compression.

`IN2` is `'wv'` for wavelet or `'wp'` for wavelet packet.

For wavelets (three output arguments):

`[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`.

For wavelet packets (four output arguments):

`[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 `wpdencmp`.

Examples

Default Global Threshold for Wavelet Denoising

Determine the default global denoising 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');


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


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');


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


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('cmp','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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Variable-size data support must be enabled.

See Also

`wdencmp` | `wenergy` | `wpdencmp`

Introduced before R2006a

dddtree

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'`.

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
```

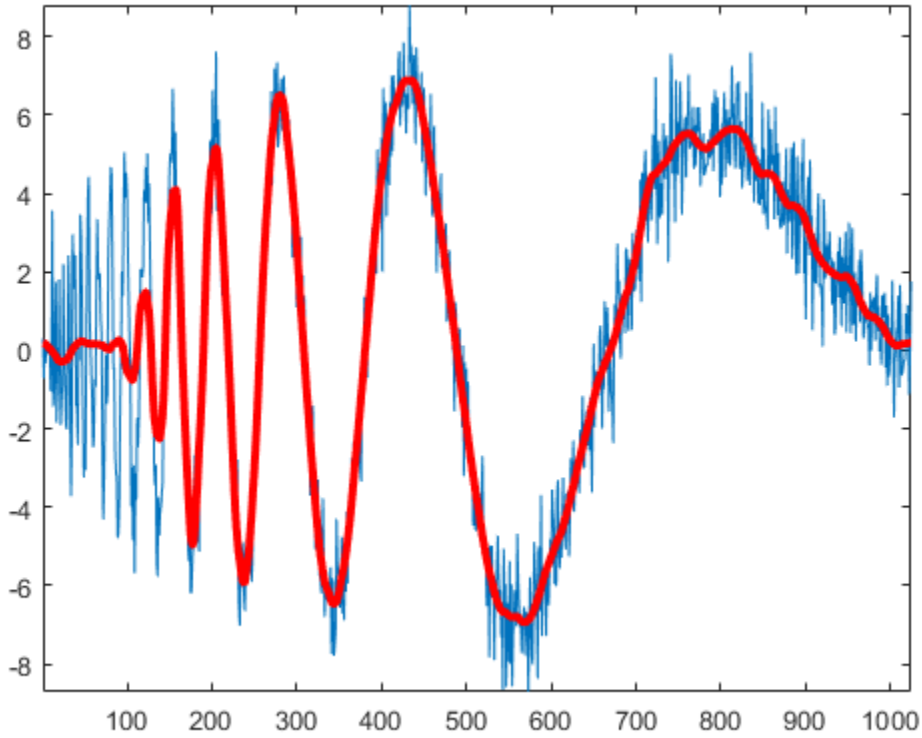
```
         0    0.2339
    0.5875   -0.7603
    0.7603    0.5875
    0.2339         0
   -0.0883   -0.1143
         0         0
    0.0352         0];
```

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;
```



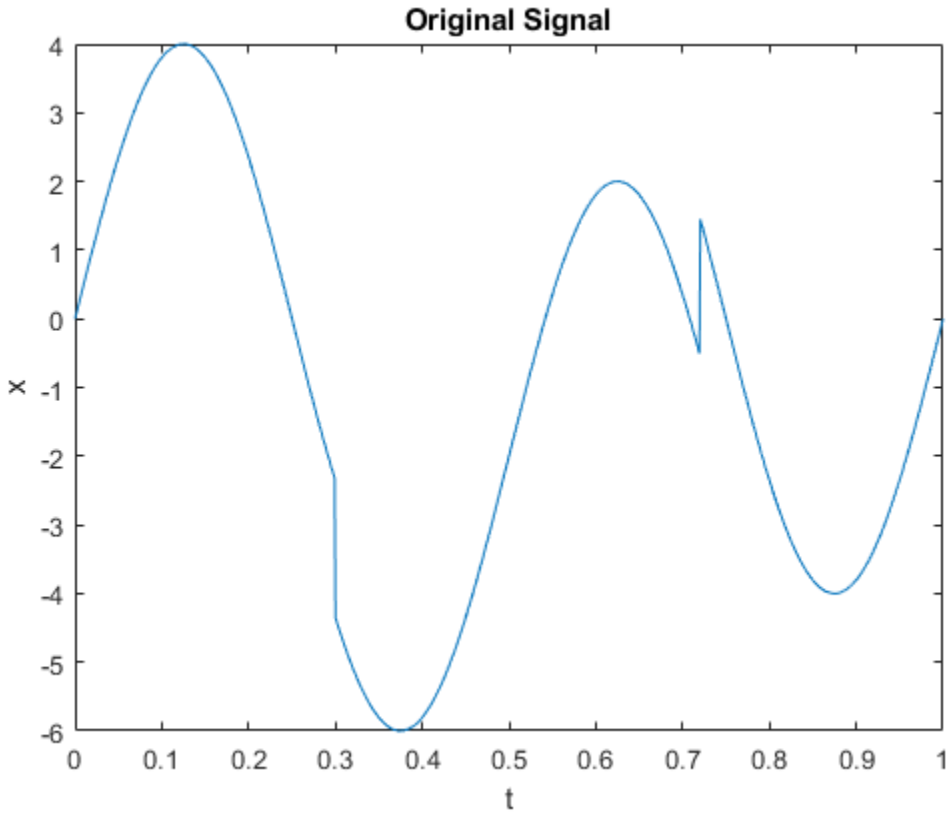
Double-Density Wavelet Transform

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.

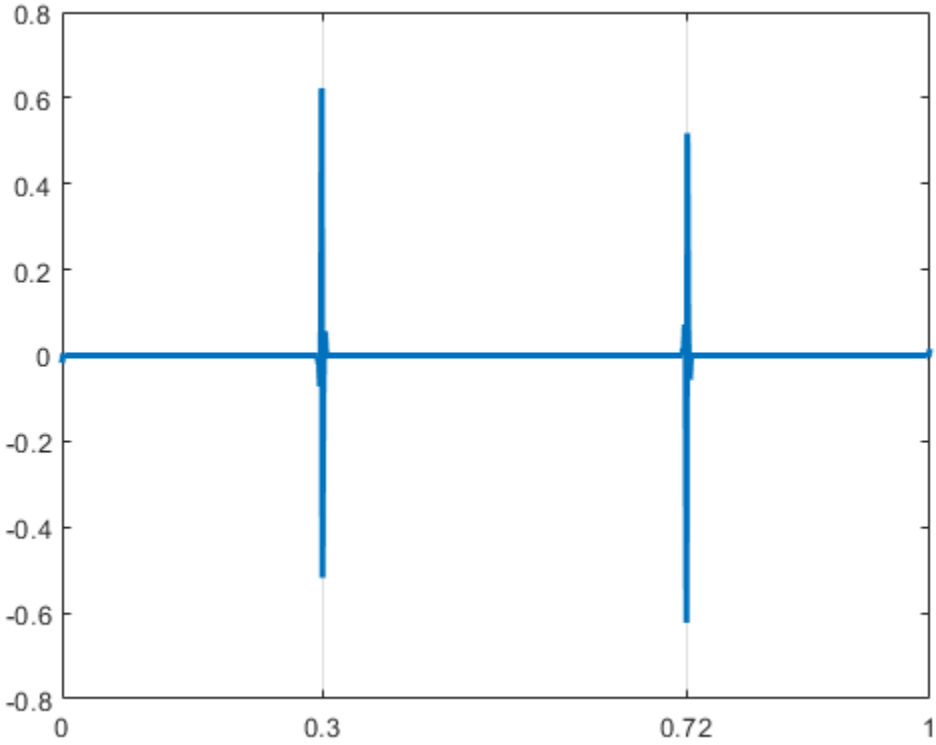
```
N = 1024;
t = linspace(0,1,1024);
x = 4*sin(4*pi*t);
```

```
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');
```



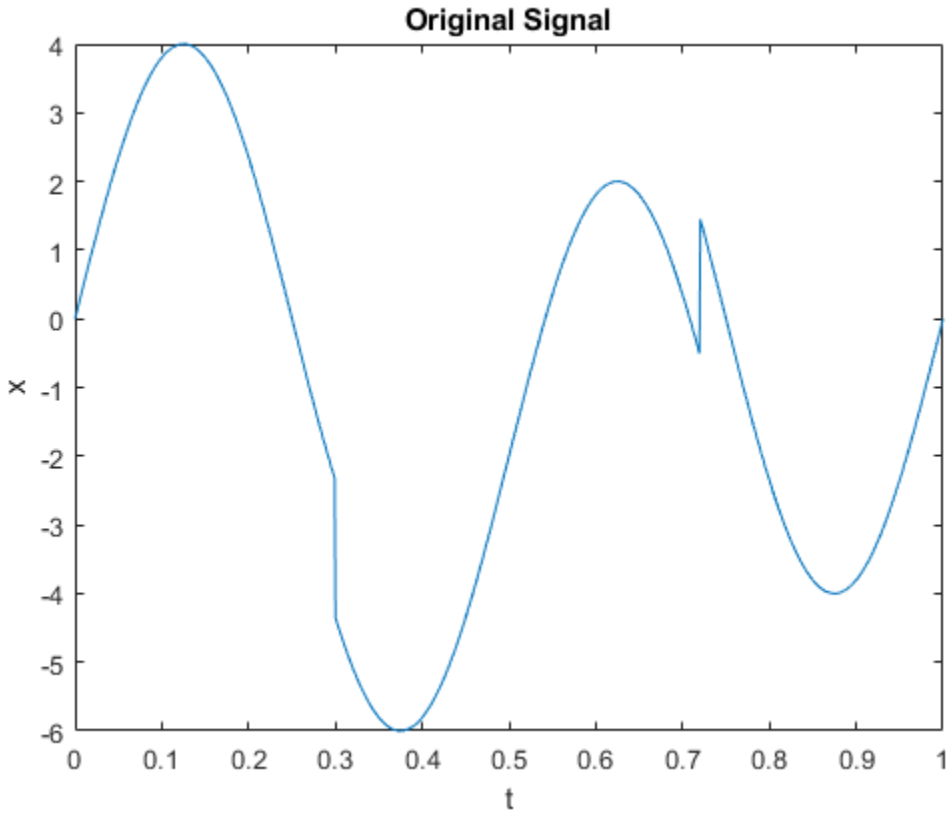
First-Level Detail Coefficients Approximation — Complex Dual-Tree

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.

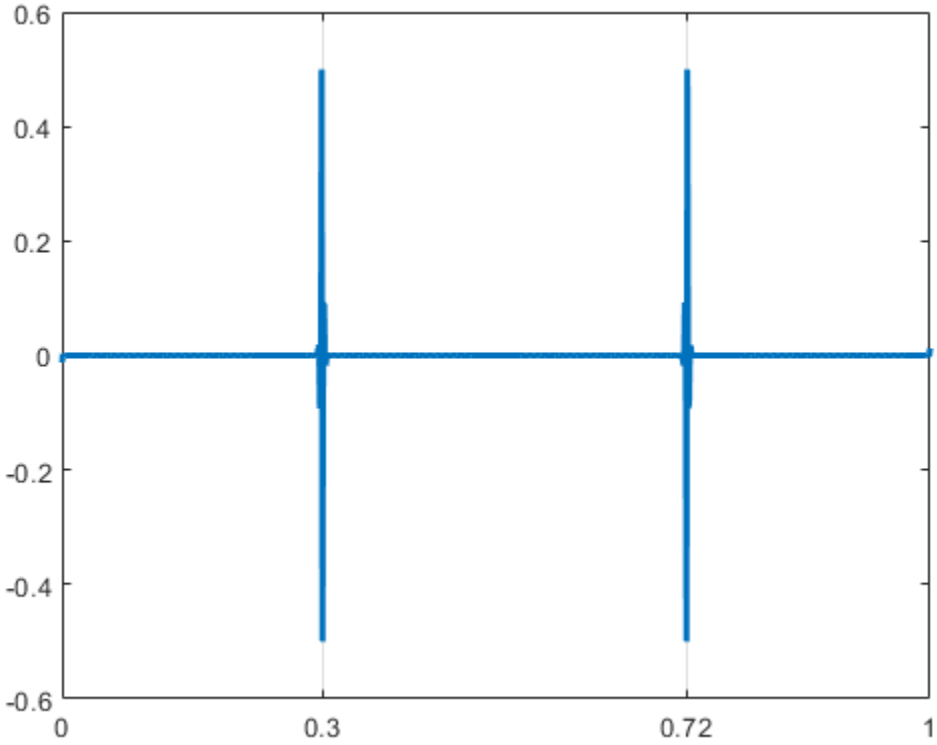
```
N = 1024;
t = linspace(0,1,1024);
x = 4*sin(4*pi*t);
```

```
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');
```



- “Analytic Wavelets Using the Dual-Tree Wavelet Transform”

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
character vector

Filter name, specified as a character vector. 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 names for the oversampled wavelet filter banks.

Data Types: `char`

`fname1` — First-stage filter name
character vector

First-stage filter name, specified as a character vector. 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
character vector

Filter name for stages > 1, specified as a character vector. 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.

See Also

`dddtree2` | `dddtreecfs` | `dtfilters` | `idddtree`

Topics

“Analytic Wavelets Using the Dual-Tree Wavelet Transform”
“Critically Sampled and Oversampled Wavelet Filter Banks”

Introduced in R2013b

dddtreecfs

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.

Examples

Reconstruction from 1-D Complex Dual-Tree Wavelet Transform

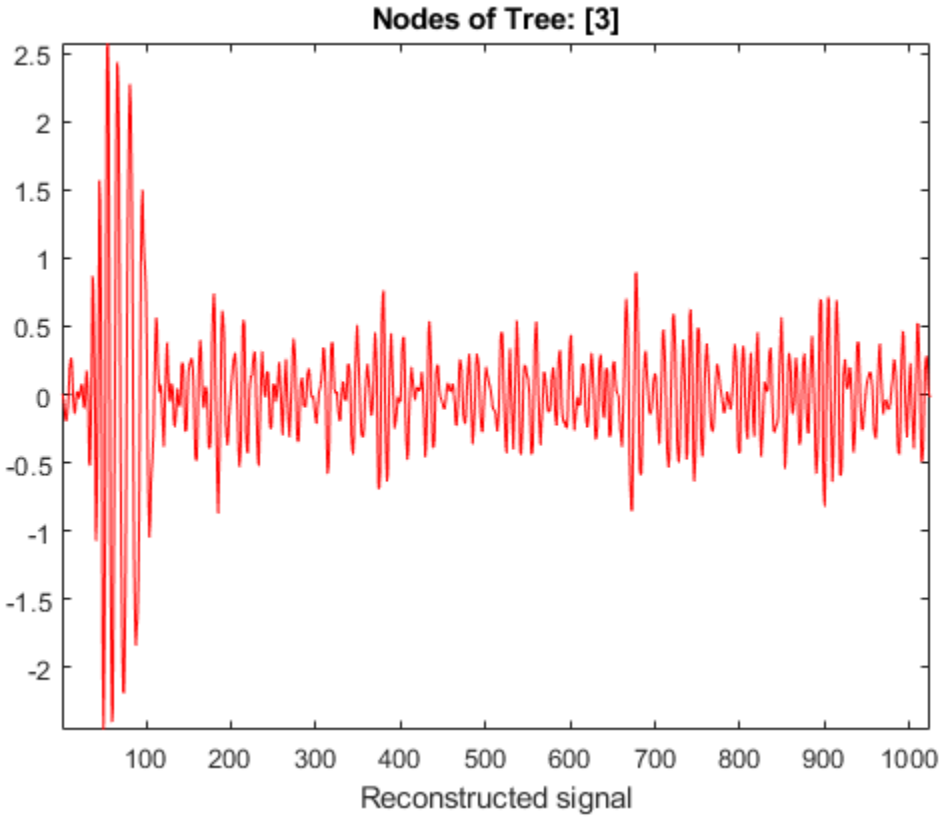
Obtain the complex dual-tree wavelet transform of the 1-D noisy Doppler signal. Reconstruct an approximation based on the level-three detail coefficients

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.

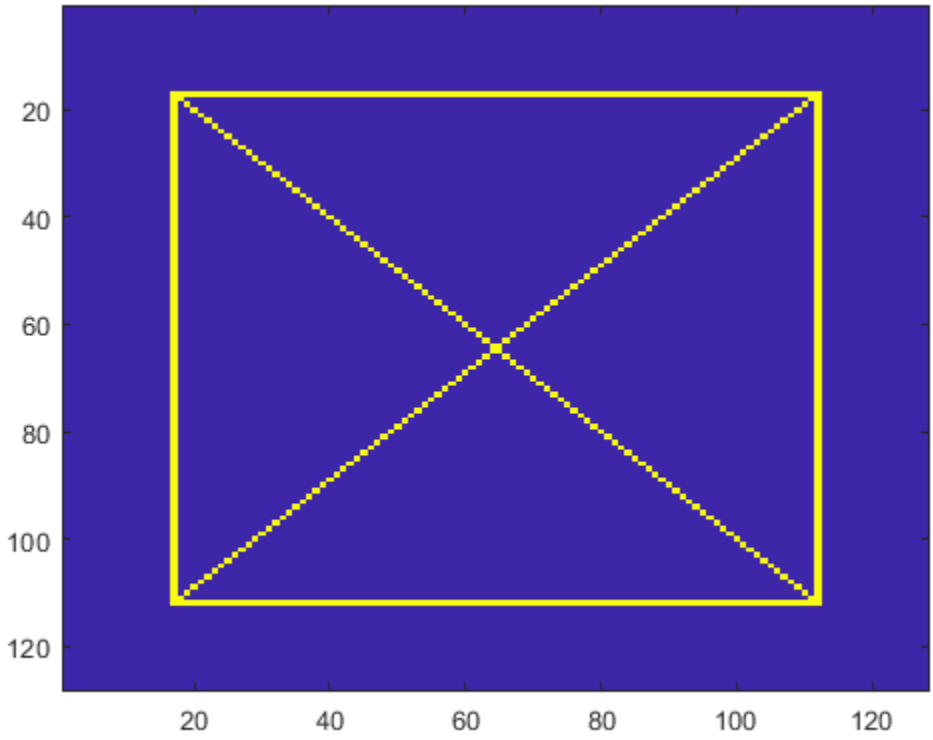
```
out = dddtreecfs('e',wt,'cumind',outputindices);
```

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)
```

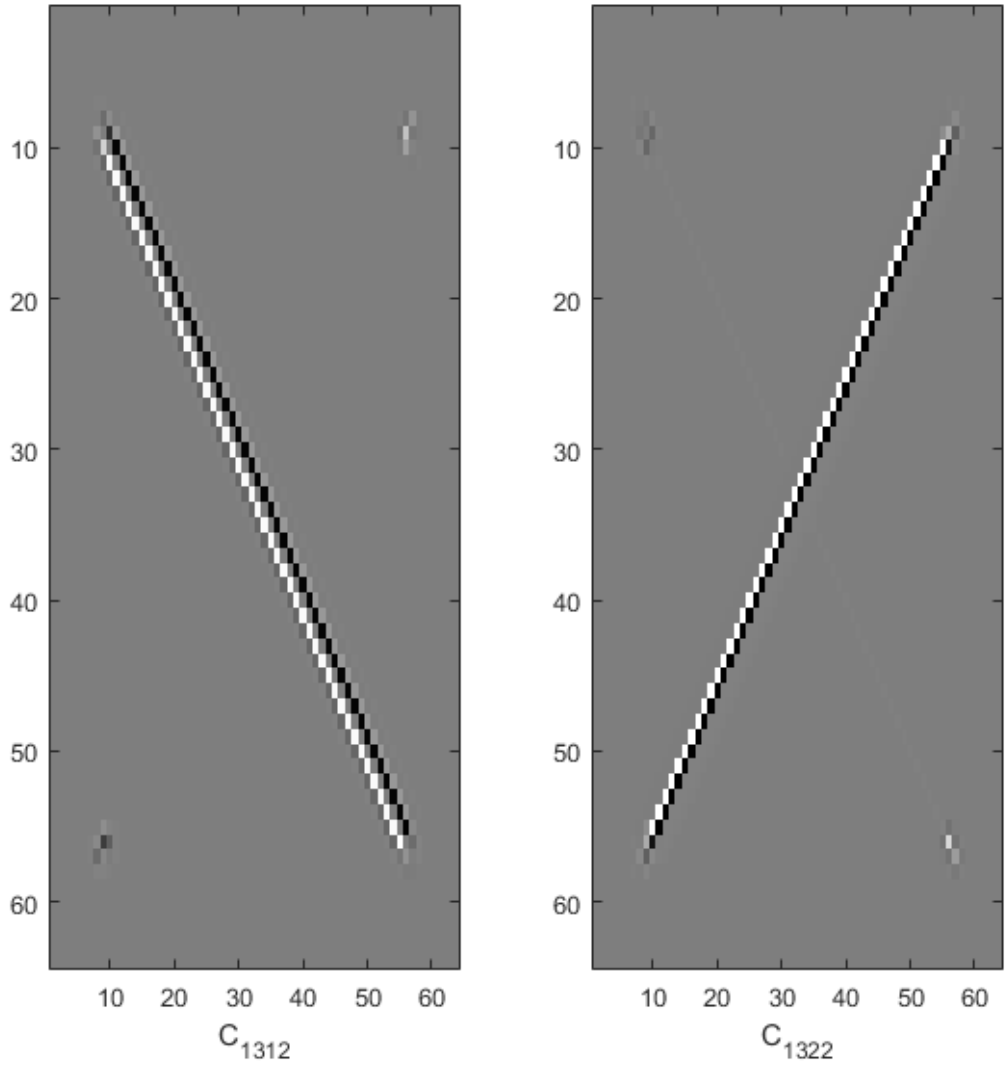


Obtain the complex dual-tree wavelet transform down to level 3.

```
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');
```


10
20
30
40
50
60
10 20 30 40 50 60
C1312
10
20
30
40
50
60
10 20 30 40 50 60
C1322

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`.

See Also

`dddtree` | `dddtree2` | `plotdt`

Introduced in R2013b

dddtree2

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'`.

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.

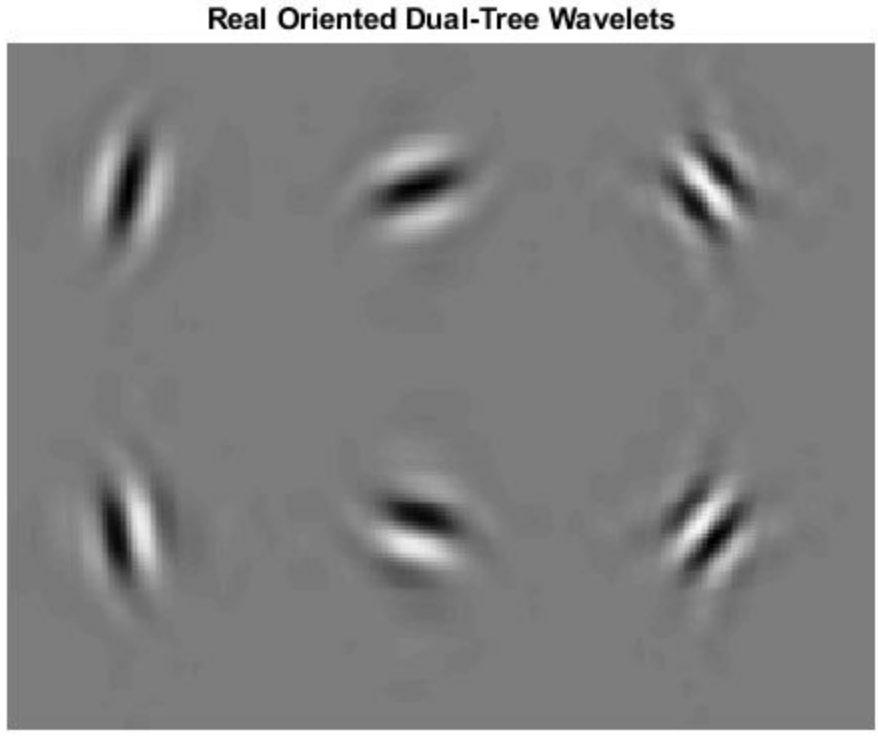
```
J = 4;
L = 3*2^(J+1);
N = L/2^J;
x = zeros(2*L,3*L);
wt = dddtree2('realdt',x,J,Faf,af);
```

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')
```

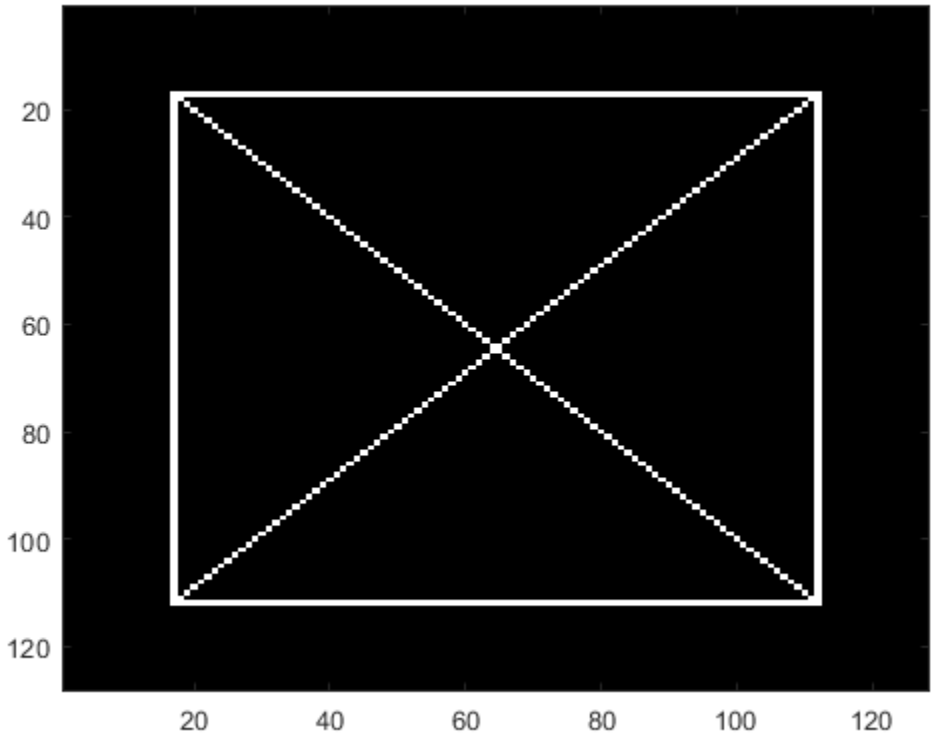


Double-Density Wavelet Transform

Obtain the double-density wavelet transform of an image.

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

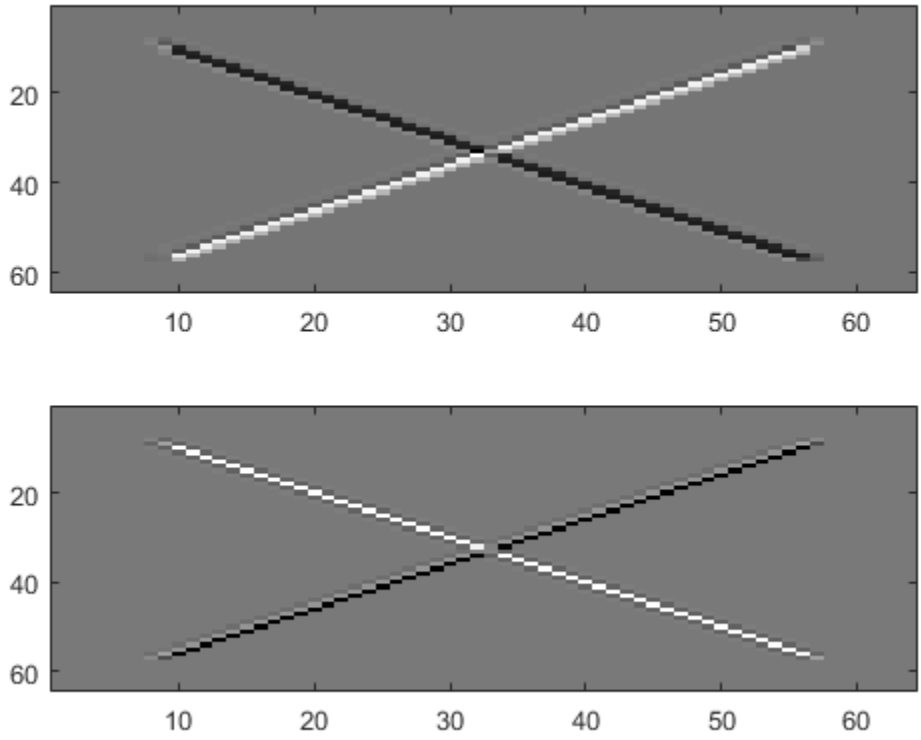
```
load xbox;
imagesc(xbox); colormap gray;
```




```
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);
```

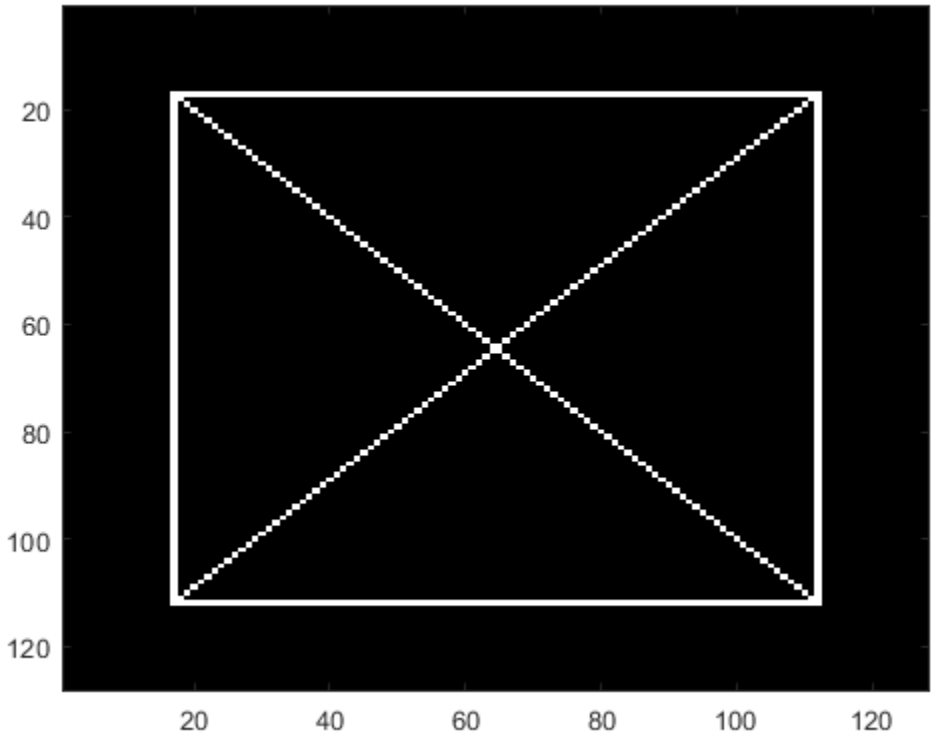


Complex Dual-Tree Wavelet Transform

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.

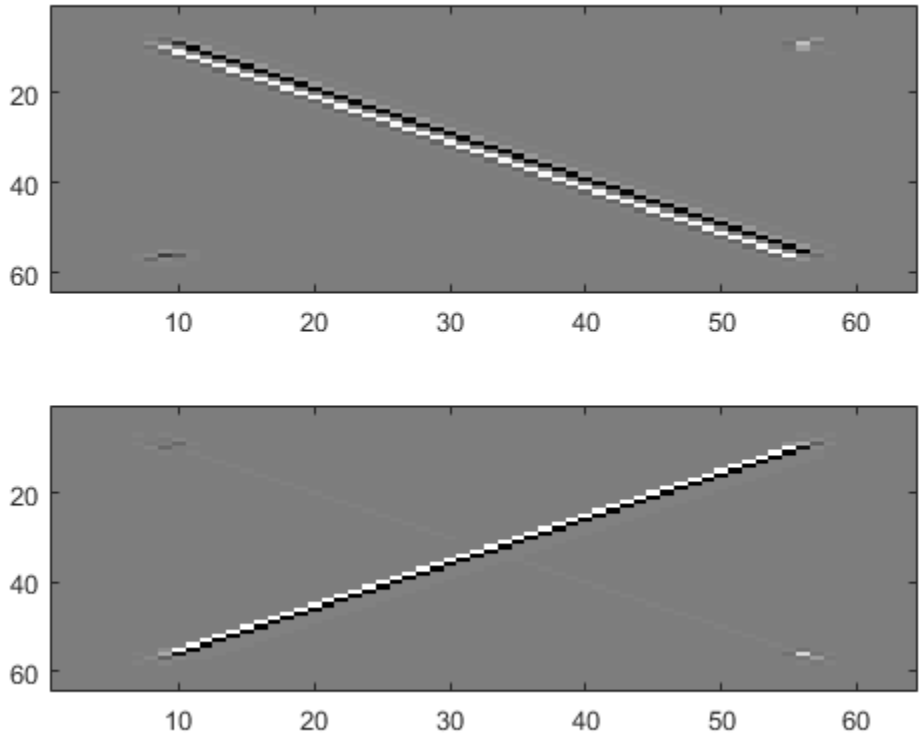
```
load xbox;
imagesc(xbox); colormap gray;
```




```
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));
```



- “Analytic Wavelets Using the Dual-Tree Wavelet Transform”

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
character vector

Filter name, specified as a character vector. 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
character vector

First-stage filter name, specified as a character vector. 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
character vector

Filter name for stages > 1, specified as a character vector. 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.

See Also

`dddtree` | `dddtreecfs` | `dtfilters` | `idddtree2`

Topics

"Analytic Wavelets Using the Dual-Tree Wavelet Transform"
"Critically Sampled and Oversampled Wavelet Filter Banks"

Introduced in R2013b

depo2ind

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.

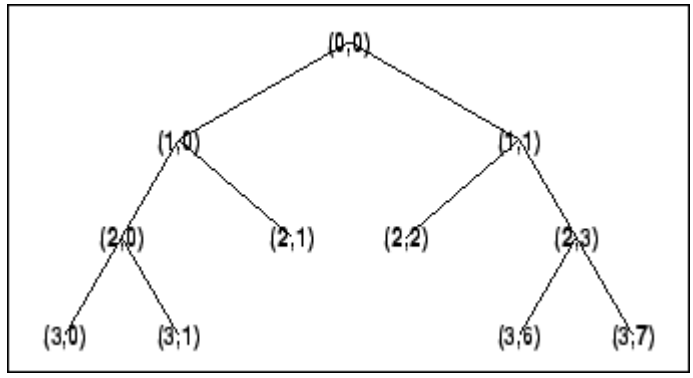
D and *P* are column vectors. The values of depths *D* and positions *P* must be such that $D \geq 0$ and $0 \leq P \leq ORD^{D-1}$.

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(O,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`

Introduced before R2006a

detcoef

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 ≤ N ≤ 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 ≤ N(j) ≤ 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

Detail Coefficients for 1-D Signal

This example shows how to obtain and plot the detail coefficients for an electrical current signal. This example uses zero-padding (see `dwtmode`).

Load the signal and select the first 3920 samples.

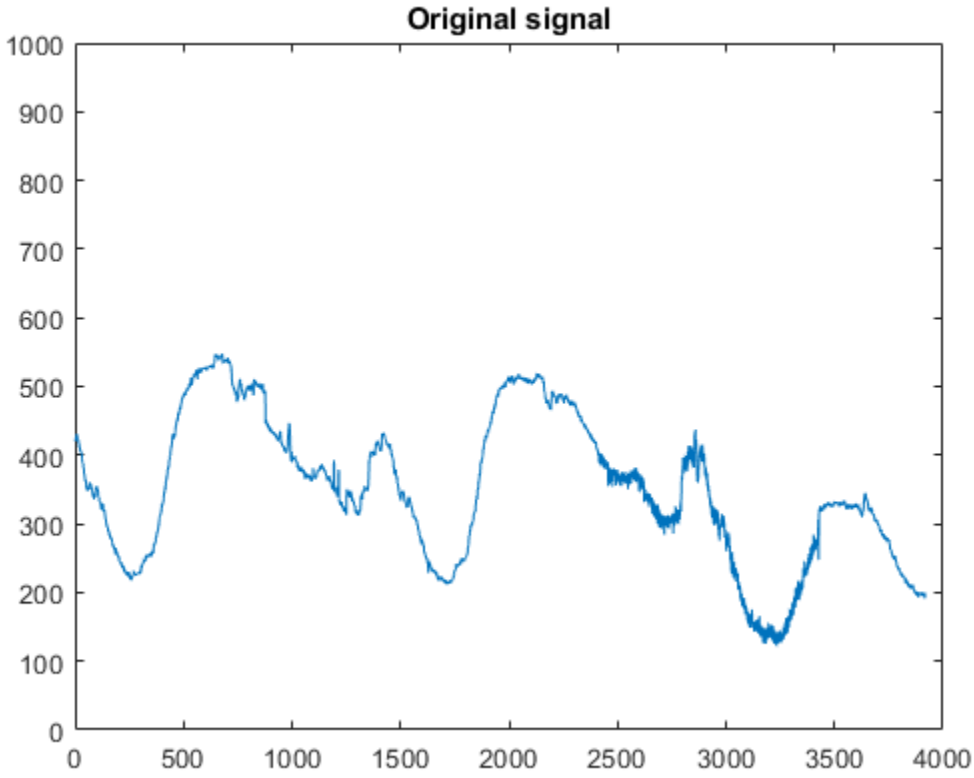
```
load leleccum;
s = leleccum(1:3920);
```

Perform the decomposition at level 3 using db1. Extract the detail coefficients at levels 1, 2, and 3 from the decomposition structure.

```
[c,l] = wavedec(s,3,'db1');
[cd1,cd2,cd3] = detcoef(c,l,[1 2 3]);
```

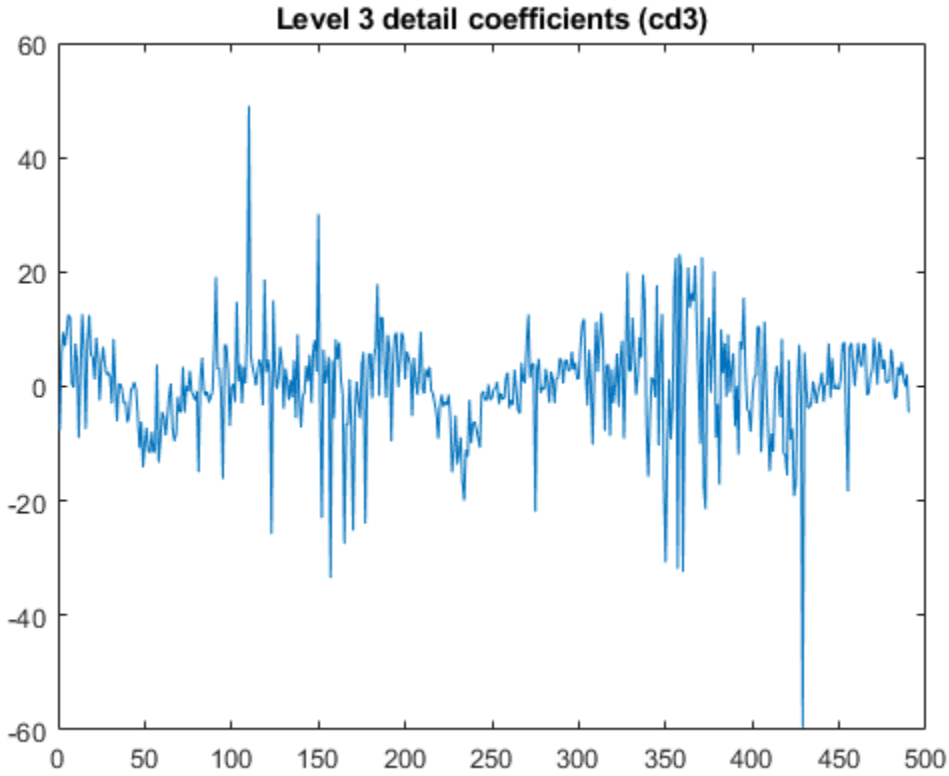
Plot the original signal.

```
plot(s)
title('Original signal')
ylim([0 1000])
```



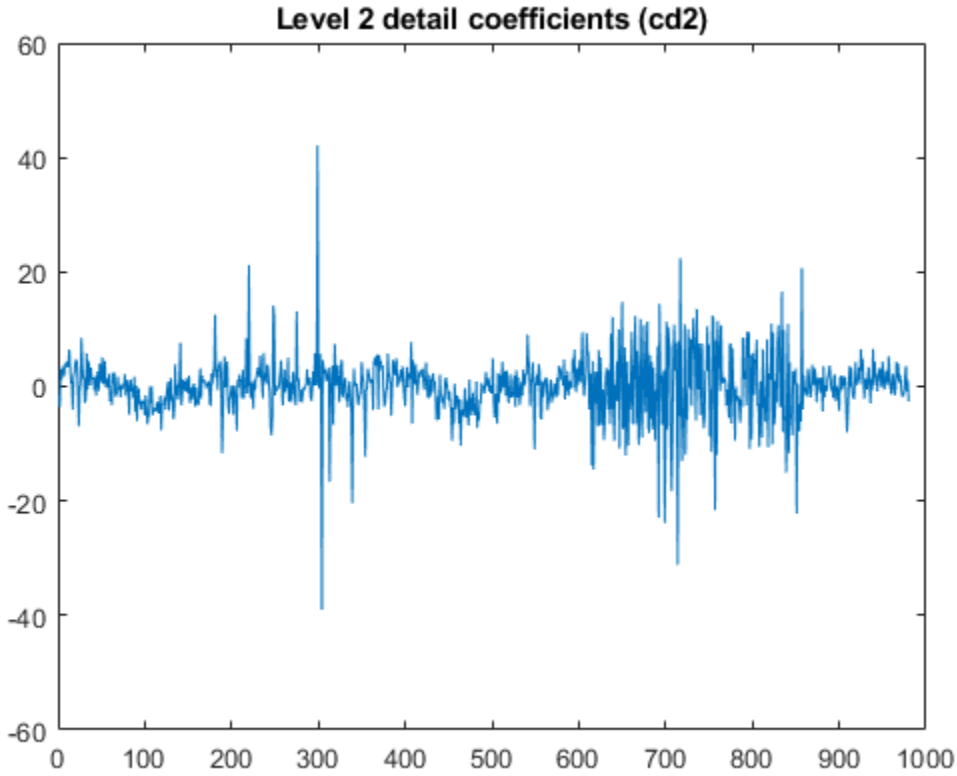
Plot the level 3 detail coefficients.

```
plot(cd3)
title('Level 3 detail coefficients (cd3)')
ylim([-60 60])
```



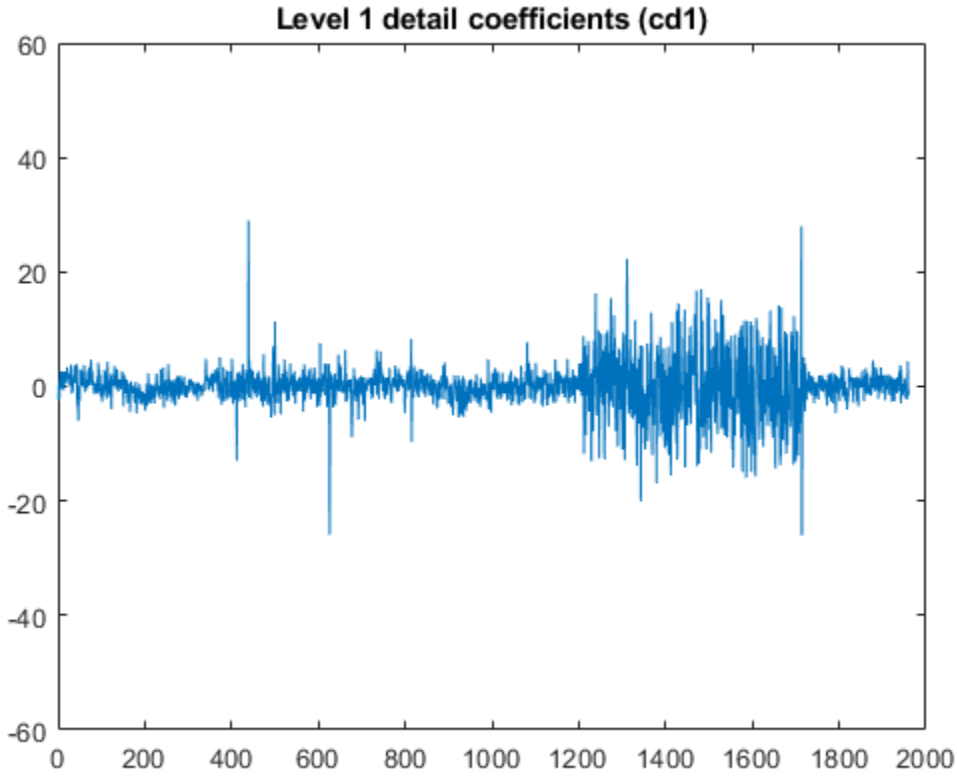
Plot the level 2 detail coefficients.

```
plot (cd2)
title('Level 2 detail coefficients (cd2)')
ylim([-60 60])
```



Plot the level 1 detail coefficients.

```
plot (cd1)
title('Level 1 detail coefficients (cd1)')
ylim([-60 60])
```



Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`appcoef` | `wavedec`

Introduced before R2006a

detcoef2

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 ≤ N ≤ 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)`.

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
```

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`appcoef2` | `wavedec2`

Introduced before R2006a

disp

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`

Introduced before R2006a

displs

Display lifting scheme

Syntax

```
S = displs(LS,FRM)
```

Description

`S = displs(LS,FRM)` returns a character vector describing the lifting scheme *`LS`*. The format character vector *`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`

Introduced before R2006a

drawtree

Draw wavelet packet decomposition tree (GUI)

Syntax

```
drawtree(T)
F = drawtree(T)
drawtree(T,F)
```

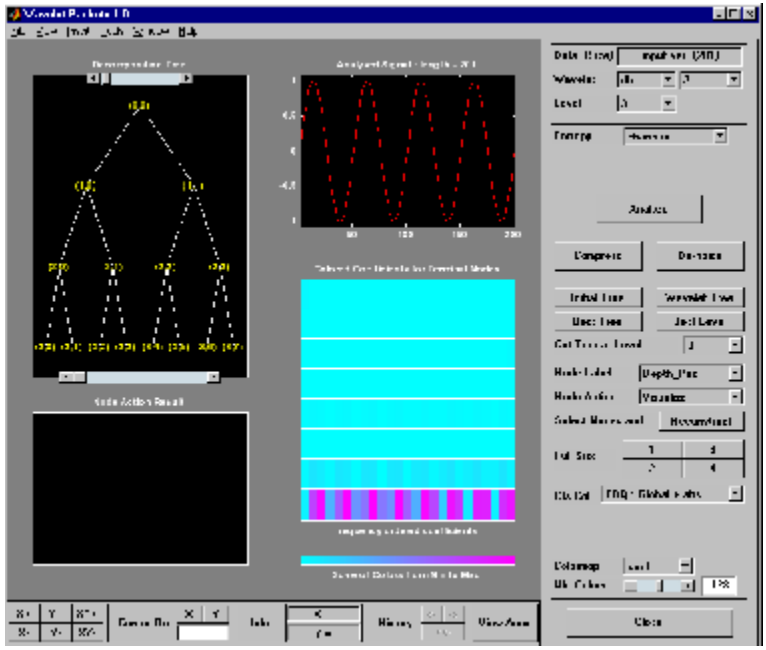
Description

`drawtree(T)` draws the wavelet packet tree *T*, and `F = drawtree(T)` also returns the figure's handle.

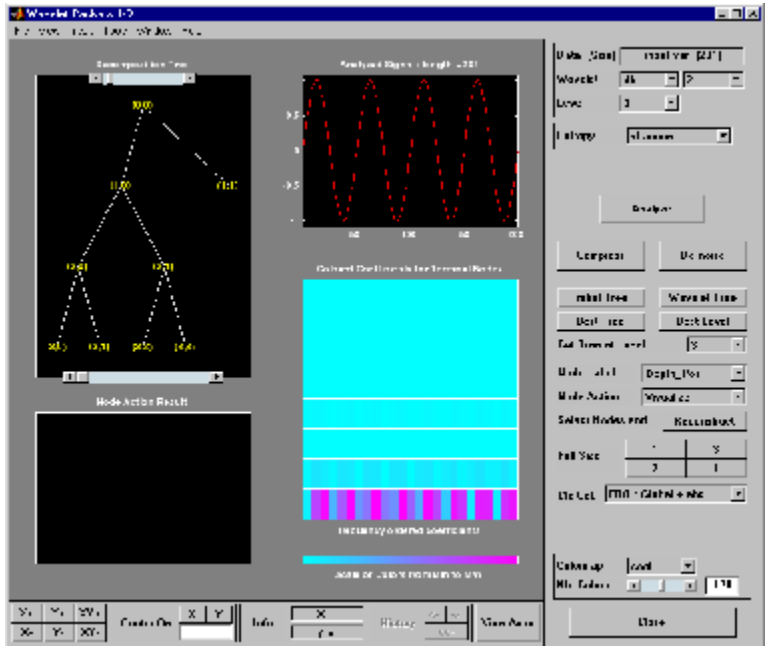
For an existing figure F produced by a previous call to the drawtree function, `drawtree(T,F)` draws the wavelet packet tree T in the figure whose handle is F.

Examples

```
x   = sin(8*pi*[0:0.005:1]);
t   = wpdec(x,3,'db2');
fig = drawtree(t);
```



```
%---------------------------------------
% Use command line function to modify t.
%---------------------------------------
t    = wpjoin(t,2);
drawtree(t,fig);
```



See Also

`readtree`

Introduced before R2006a

dtfilters

Analysis and synthesis filters for oversampled wavelet filter banks

Syntax

```
df = dtfilters(name)
[df,rf] = dtfilters(name)
```

Description

`df = dtfilters(name)` returns the decomposition (analysis) filters corresponding to the `name` character vector. These filters are used most often as input arguments to `dddtree` and `dddtree2`.

`[df,rf] = dtfilters(name)` returns the reconstruction (synthesis) filters corresponding to the `name` character vector.

Examples

Filters for Complex Dual-Tree Wavelet Transform

Obtain valid filters for the complex dual-tree wavelet transform. The transform uses Farras 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))
```

```
ans = 1.3678e-13
```

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))
```

```
ans = 2.3803e-13
```

Input Arguments

`name` — Filter name

`'dtf1'` | `'dddtf1'` | `'self1'` | `'self2'` | ...

Filter name, specified as a character vector. 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, 4, or 5 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, 4, or 5 specifies the Kingsbury Q-shift filters with N = 6, 10, 14, 16, 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'`.
- `'farras'` — Farras nearly symmetric filters for a two-channel perfect reconstruction filter bank. This option is meant to be used for one-tree transforms and is valid only for an orthogonal critically sampled wavelet transform, `'dwt'`. The output of `dtfilters` is a two-column matrix. The first column of the matrix is a scaling (lowpass) filter, and the second column is a wavelet (highpass) filter.
- `'FSfarras'` — Farras nearly symmetric first-stage filters intended for a dual-tree wavelet transform. With this option, the output of `dtfilters` is a cell array with two elements, one for each tree. Each element is a two-column matrix. The first column of the matrix is a scaling (lowpass) filter, and the second column is a wavelet (highpass) filter.
- `'qshiftN'` — Kingsbury Q-shift N-tap filters with N = 6, 10, 14, 16, or 18. The Kingsbury Q-shift filters are used most commonly 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.

See Also

`dddtree` | `dddtree2`

Introduced in R2013b

dtree

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`

Introduced before R2006a

dualtree3

3-D dual-tree complex wavelet transform

Syntax

```
[a,d] = dualtree3(x)
[a,d] = dualtree3(x,level)

[a,d] = dualtree3(___,Name,Value)

[a,d] = dualtree3(___,'excludeL1')
```

Description

`[a,d] = dualtree3(x)` returns the 3-D dual-tree complex wavelet transform of `x` at the maximum level, `floor(log2(min(size(x))))`.

`[a,d] = dualtree3(x,level)` returns the 3-D dual-tree wavelet transform down to `level`.

`[a,d] = dualtree3(___,Name,Value)` specifies options using name-value pair arguments in addition to any of the input arguments in previous syntaxes.

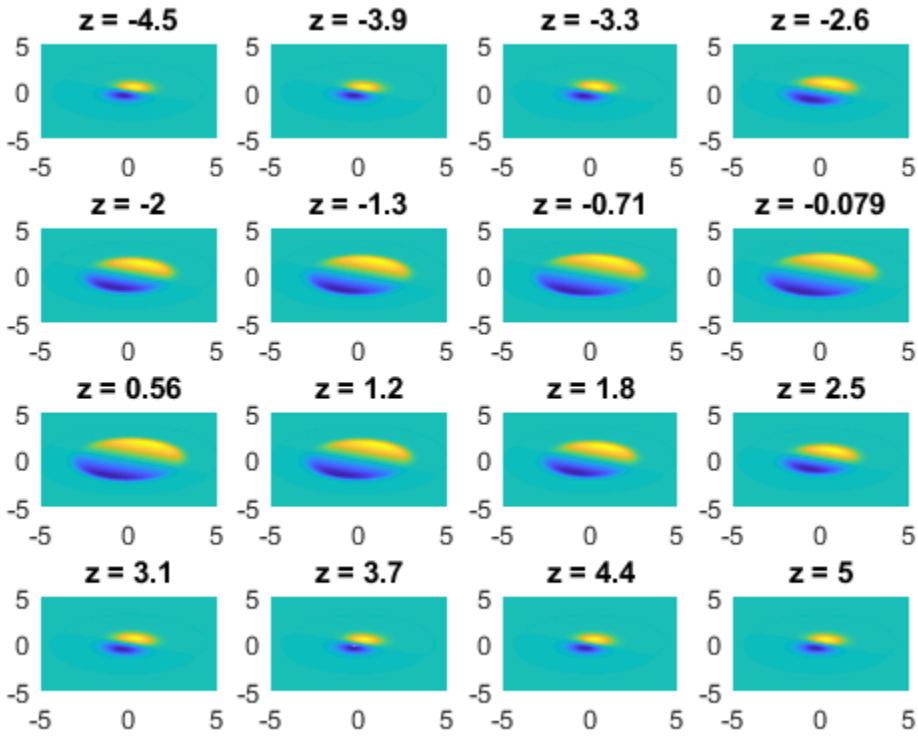
`[a,d] = dualtree3(___,'excludeL1')` excludes the first-level wavelet (detail) coefficients. Excluding the first-level wavelet coefficients can speed up the algorithm and saves memory. The first level does not exhibit the directional selectivity of levels 2 and higher. The perfect reconstruction property of the dual-tree wavelet transform holds only if the first-level wavelet coefficients are included. If you do not specify this option, or append `'includeL1'`, then the function includes the first-level coefficients.

Examples

Three-Dimensional Dual-Tree Transform of Volumetric Data

Generate a volumetric data set. Plot several cross-sections of the data seen from above. The data are not symmetric about the x-axis or the y-axis.

```
xl = 64;

xx = linspace(-5,5,xl);

[x,y,z] = meshgrid(xx);

G = (x+3*y)./(1+exp((x.^2+2*y.^2+z.^2)-10));

for k = 1:16
    subplot(4,4,k)
    surf(xx,xx,G(:,:,4*k))
    view(2)
    shading interp
    title(['z = ' num2str(xx(4*k),2)])
end
```



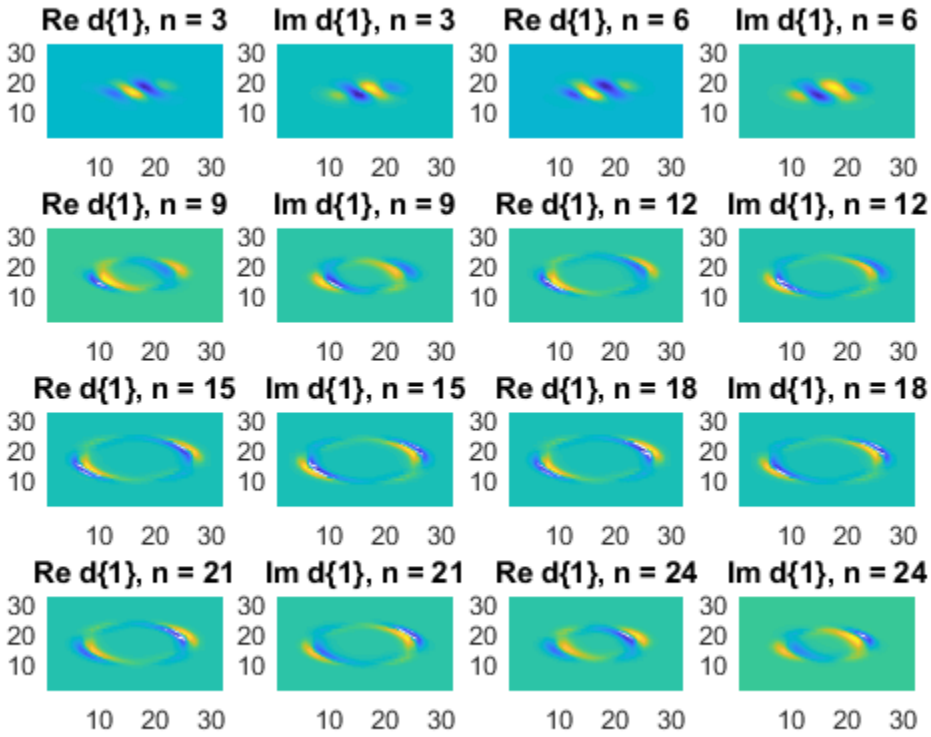
Compute the 3-D dual-tree transform of the data down to level 4. Specify a Hilbert Q-shift filter-pair length of 14.

```
[a,d] = dualtree3(G,4,'FilterLength',14);
```

Plot the real and imaginary parts of the first-level wavelet coefficients for selected subbands. The coefficients have the same directionality as the data. The imaginary parts are shifted versions of the real parts.

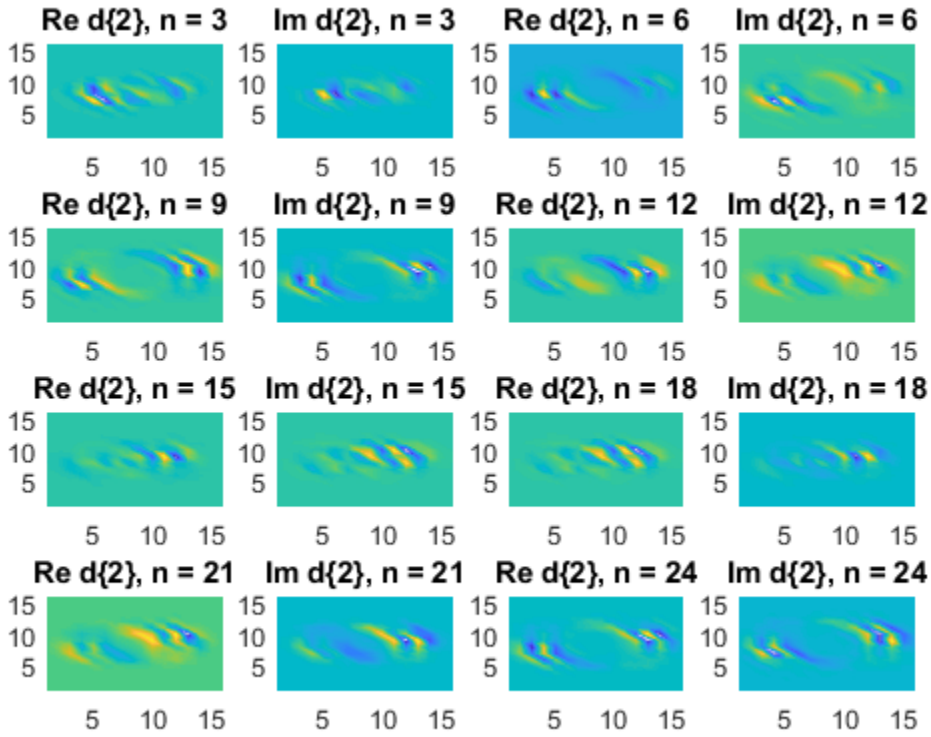
```
m = 1;

for k = 1:8
    subplot(4,4,2*k-1)
    surf(real(d{m}(:,:,3*k)))
```

```
    view(2)
    shading interp
    axis tight
    title(['Re d\{1\}, n = ' int2str(3*k)])

    subplot(4,4,2*k)
    surf(imag(d{m}(:,:,3*k)))

    view(2)
    shading interp
    axis tight
    title(['Im d\{1\}, n = ' int2str(3*k)])
end
```



Repeat the procedure for the second-level coefficients. When the level number increases by one, the array of wavelet coefficients decreases by half along the first two dimensions.

```
m = 2;

for k = 1:8
    subplot(4,4,2*k-1)
    surf(real(d{m}(:,:,3*k)))

    view(2)
    shading interp
    axis tight
    title(['Re d\{2\}, n = ' int2str(3*k)])

    subplot(4,4,2*k)
    surf(imag(d{m}(:,:,3*k)))

    view(2)
    shading interp
    axis tight
    title(['Im d\{2\}, n = ' int2str(3*k)])
end
```



Invert the transform, specifying the same filter-pair length. Check for perfect reconstruction.

```
xrec = idualtree3(a,d,'FilterLength',14);
max(abs(xrec(:)-G(:)))
```

```
ans = 7.1054e-15
```

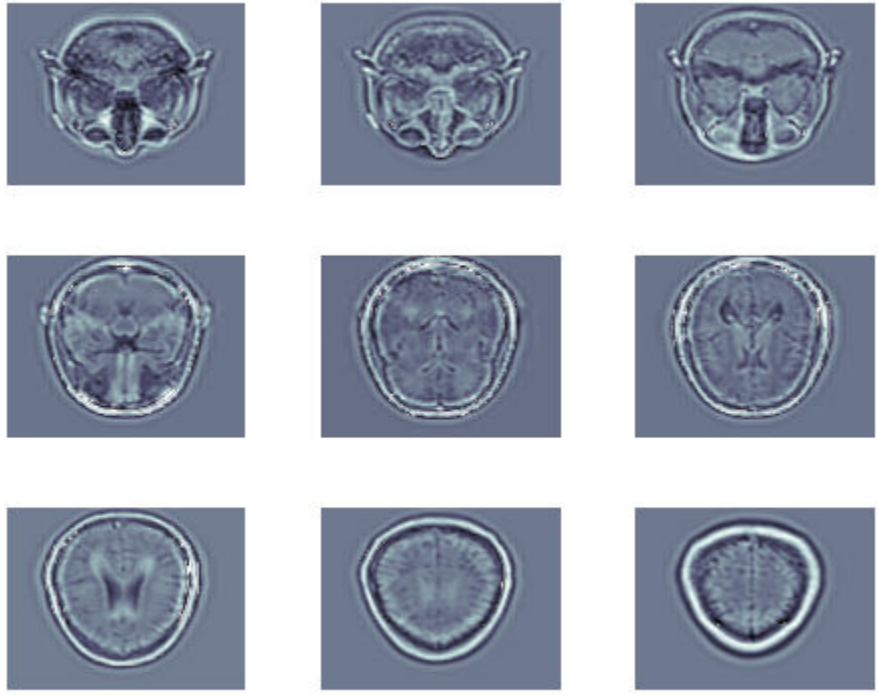
3-D Dual-Tree Transform of MRI Data

Load a set of MRI measurements of a human head. Truncate the data so that it is even along the third dimension. Compute the 3-D dual-tree transform, excluding the first-level wavelet coefficients.

```
load wmri

[A,D] = dualtree3(X(:,:,1:26),2,'excludeL1');
```

Reconstruct the data by inverting the transform. Set the final-level scaling coefficients explicitly to 0. Display an evenly spaced selection of reconstructed images.

```
imrec = idualtree3(A*0,D);

colormap bone
for kj = 1:9
    subplot(3,3,kj)
    surf(imrec(:,:,3*kj-2))
    shading interp
    view(2)
    axis tight off
end
```



- Dual-Tree Wavelet Transforms

Input Arguments

`x` — Input data

real 3-D array

Input data, specified as a real 3-D array. All three dimensions of `x` must be even and greater than or equal to 4.

Data Types: `double` | `single`

`level` — Transform level

`floor(log2(min(size(x))))` (default) | positive integer

Transform level, specified as a positive integer greater than or equal to 2 and less than or equal to `floor(log2(min(size(x))))`.

Data Types: `double` | `single`

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: `'LevelOneFilter','legall','FilterLength',6` computes a transform using LeGall analysis filters with scaling length 5 and wavelet length 3 at level 1, and length-6 Q-shift filters at levels 2 and greater.

`FilterLength` — Hilbert Q-shift filter-pair length

10 (default) | 6 | 14 | 16 | 18

Hilbert Q-shift filter-pair length, specified as the comma-separated pair consisting of `'FilterLength'` and one of 6, 10, 14, 16, or 18. `dualtree3` uses the orthogonal Hilbert Q-shift filter pair of length `'FilterLength'` for levels 2 and greater.

Data Types: `double` | `single`

`LevelOneFilter` — First-level biorthogonal analysis filter

`'nearsym5_7'` (default) | `'nearsym13_19'` | `'antonini'` | `'legall'`

First-level biorthogonal analysis filter, specified as the comma-separated pair consisting of `'LevelOneFilter'` and a character vector or string. By default, `dualtree3` uses for level 1 the near-symmetric biorthogonal wavelet filter with lengths 5 (scaling filter) and 7 (wavelet filter).

Data Types: `char` | `string`

Output Arguments

`a` — Final-level scaling coefficients

real-valued matrix

Final-level scaling (lowpass) coefficients, returned as a real-valued matrix.

Data Types: `double`

`d` — Wavelet coefficients

1-by-`level` cell array

Wavelet coefficients, returned as a 1-by-`level` cell array. There are 28 wavelet subbands in the 3-D dual-tree transform at each level.

Data Types: `double`

References

[1] Chen, H., and N. G. Kingsbury. "Efficient Registration of Nonrigid 3-D Bodies." *IEEE® Transactions on Image Processing*. Vol 21, January 2012, pp. 262–272.

[2] Kingsbury, N. G. "Complex Wavelets for Shift Invariant Analysis and Filtering of Signals." *Journal of Applied and Computational Harmonic Analysis*. Vol. 10, May 2001, pp. 234–253.

See Also

`dddtree2` | `idualtree3` | `wavedec3` | `waverec3`

Topics

Dual-Tree Wavelet Transforms

Introduced in R2017a

dwt

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. Compare this function to `wavedec`, which may be more useful for your application. The decomposition is done 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 character vector *`'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 character vector 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');
[Lo_D,Hi_D] = wfilters('bior3.5','d');
[A,D] = dwt(noisdopp,Lo_D,Hi_D);
```

DWT Using Wavelet and Scaling Filters

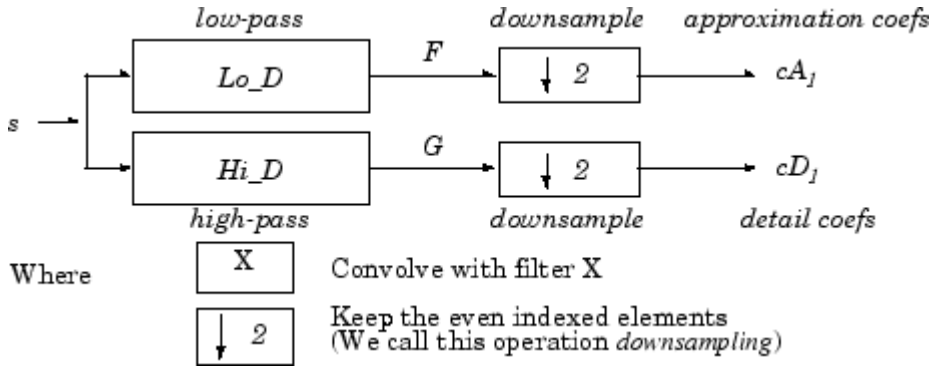
Obtain the level-1 DWT of the noisy Doppler signal using wavelet and scaling filters.

```
load noisdopp;
[Lo_D,Hi_D] = wfilters('bior3.5','d');
[A,D] = dwt(noisdopp,Lo_D,Hi_D);
```

Algorithms

Starting from a signal s of length N, 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 $2L$. For signal of length N, the signals F and G are of length $N + 2L - 1$, and then the coefficients CA_1 and CD_1 are of length $\left\lfloor \frac{N-1}{2} + L \right\rfloor$.

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.)

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`dwtmode` | `idwt` | `wavedec` | `waveinfo`

Introduced before R2006a

dwt2

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. Compare this function to `wavedec2`, which may be more useful for your application. The decomposition is done 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'`* character vector 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 character vector containing the desired extension mode.

An example of valid use is

```
[cA,cH,cV,cD] = dwt2(x,'db1','mode','sym');
```

Examples

2-D DWT of an Image

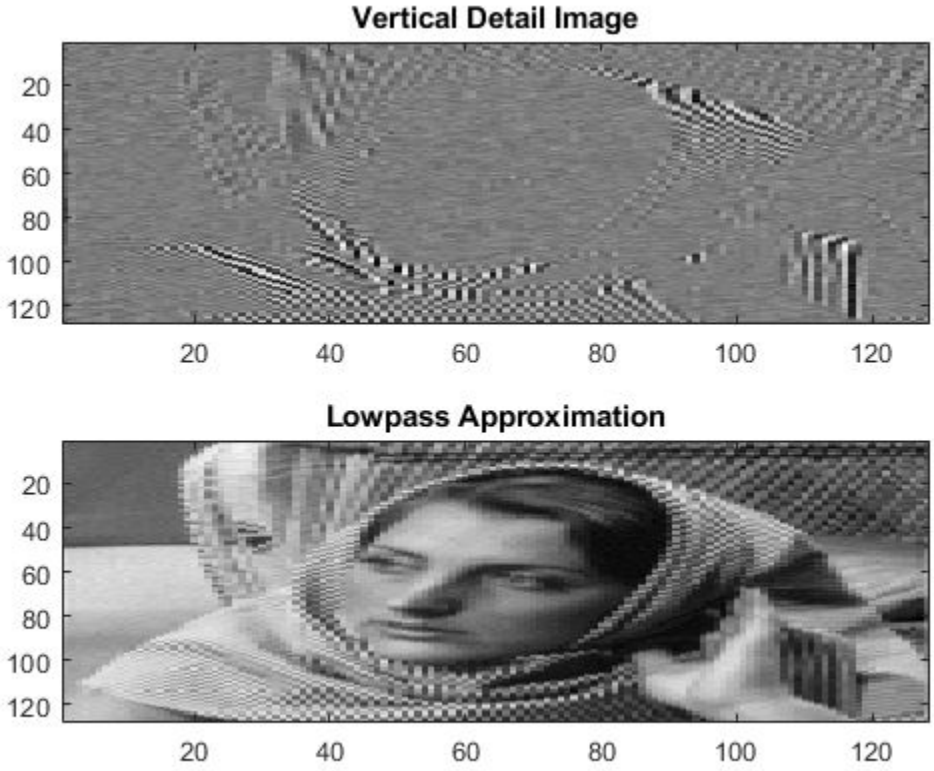
This example shows how to obtain the 2-D DWT of an image.

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')
```



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.

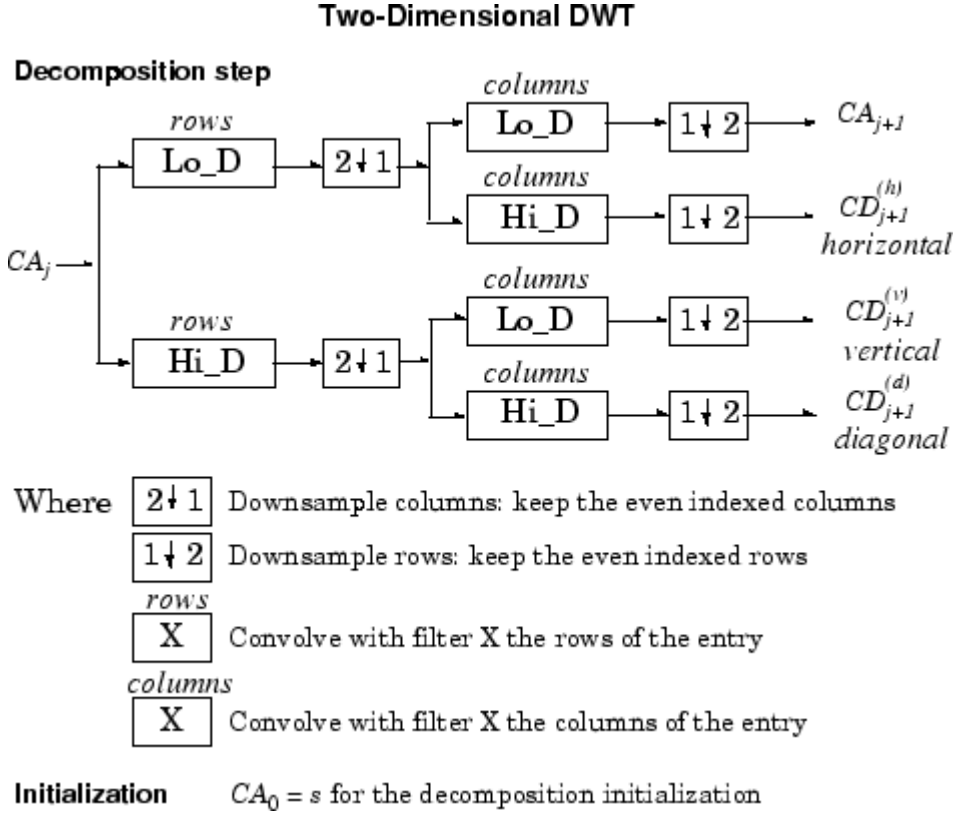
For more information on image formats, see the `image` and `imfinfo` reference pages.

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:



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.)

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`dwtmode` | `haart2` | `idwt2` | `ihaart2` | `wavedec2` | `waveinfo` | `waverec2`

Introduced before R2006a

dwt3

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 character vector 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 × 2 × 2 cell array containing the coefficients of the decomposition. `dec{i,j,k}`, `i,j,k = 1` or `2` contains the coefficients obtained by lowpass filtering (for `i` or `j` or `k = 1`) or highpass filtering (for `i` or `j` or `k = 2`). The `i` element filters along the rows of `X`, the `j` element filters along the columns, and the `k` element filters along the third dimension. For example, `dec{1,2,1}` is obtained by filtering `X` along the rows with the lowpass (scaling) filter, along the columns with the highpass (wavelet) filter, and along the third dimension with the lowpass (scaling) filter.

`WT = dwt3(X,W,...)` specifies three wavelets, one for each direction. `W = {'wname1','wname2','wname3'}` or `W` is a structure with 3 fields `'w1'`, `'w2'`, `'w3'` containing character vectors that are the names of wavelets.

`WT = dwt3(X,WF,...)` specifies four filters, two for decomposition, and two for reconstruction or 3 × 4 filters (one quadruplet by direction). `WF` is either a cell array (1 × 4) or (3 × 4) : `{LoD,HiD,LoR,HiR}` or a structure with the four fields `'LoD','HiD','LoR','HiR'`.

Examples

Single-Level Three-Dimensional Wavelet Decomposition

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 =

  struct with fields:

    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 =

  struct with fields:

    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 =

  struct with fields:

    sizeINI: [4 4 4]
    filters: [1x1 struct]
       mode: 'per'
        dec: {2x2x2 cell}
```

Decompose `X` using the filters given by `WF` and set the extension mode to symmetric.

```
WF = wt.filters;
wtBIS = dwt3(X,WF,'mode','sym')
```

```
wtBIS =

  struct with fields:

    sizeINI: [4 4 4]
    filters: [1x1 struct]
       mode: 'sym'
```

```
        dec: {2x2x2 cell}
```

See Also

dwtmode | idwt3 | wavedec3 | waveinfo | waverec3 | wfilters

Introduced in R2010a

dwtleader

Multifractal 1-D wavelet leader estimates

Syntax

```
[dh,h] = dwtleader(x)
[dh,h,cp] = dwtleader(x)
[dh,h,cp,tauq] = dwtleader(x)
[dh,h,cp,tauq,leaders] = dwtleader(___)
[dh,h,cp,tauq,leaders,structfunc] = dwtleader(___)

[___]= dwtleader(x,wname)
[___] = dwtleader(___,Name,Value)
```

Description

`[dh,h] = dwtleader(x)` returns the singularity spectrum, `dh`, and the Holder exponents, `h`, for the 1-D real-valued data, `x`. The singularity spectrum and Holder exponents are estimated for the linearly-spaced moments of the structure functions from –5 to +5.

`[dh,h,cp] = dwtleader(x)` also returns the first three log cumulants, `cp` of the scaling exponents.

`[dh,h,cp,tauq] = dwtleader(x)` also returns the scaling exponents for the linearly spaced moments from –5 to 5. Wavelet leaders are not defined for the finest scale.

`[dh,h,cp,tauq,leaders] = dwtleader(___)` also returns the wavelet leaders by scale.

`[dh,h,cp,tauq,leaders,structfunc] = dwtleader(___)` also returns the multiresolution structure functions.

`[___]= dwtleader(x,wname)` uses the orthogonal or biorthogonal wavelet specified by `wname` to compute the wavelet leaders and the fractal estimates.

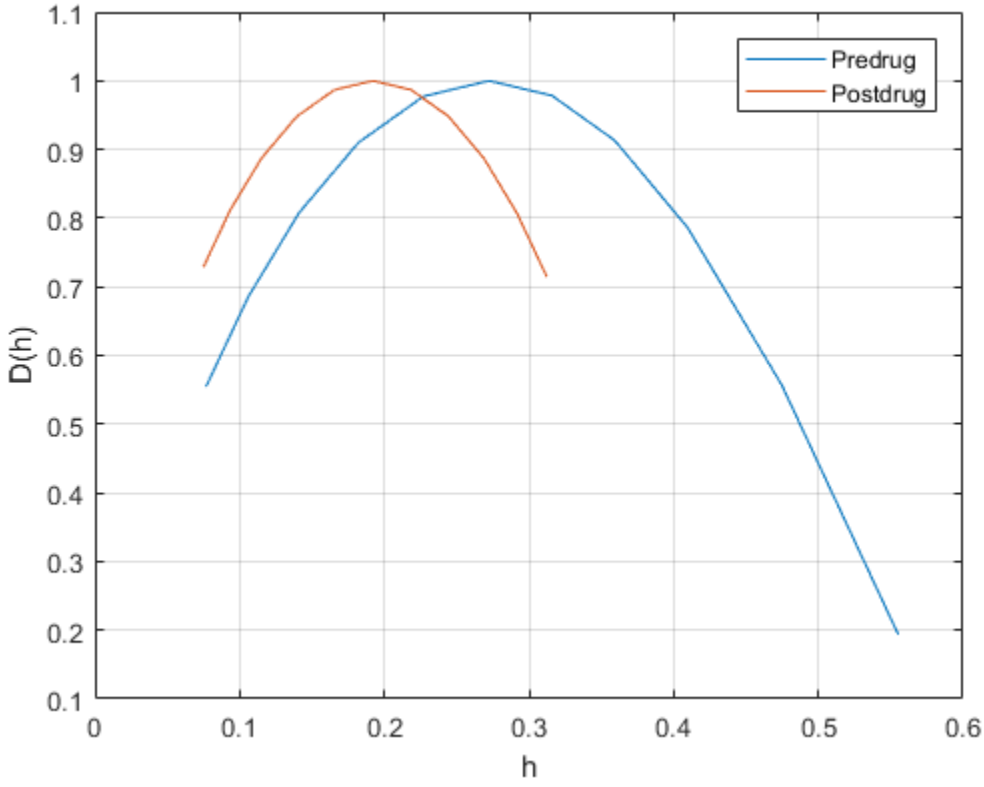
`[ ___ ] = dwtleader( ___ ,Name,Value)` returns the wavelet leaders and other specified outputs with additional options specified by one or more `Name,Value` pair arguments.

Examples

Multifractal Spectrum of Heart-Rate Variability

Compare the multifractal spectrum of heart-rate variability data before and after application of a drug that reduces heart dynamics.

```
load hrvDrug;
predrug = hrvDrug(1:4642);
postdrug = hrvDrug(4643:end);
[dhpre,hpre] = dwtleader(predrug);
[dhpost,hpost] = dwtleader(postdrug);
plot(hpre,dhpre,hpost,dhpost)
xlabel('h')
ylabel('D(h)')
grid on
legend('Predrug','Postdrug')
```



The spread of the Holder exponent values before drug administration (approximately 0.08 to 0.55) is much larger than the spread of the values afterward (approximately 0.08 to 0.31). This indicates that the heart rate has become more monofractal.

Brownian Noise Singularity Spectrum

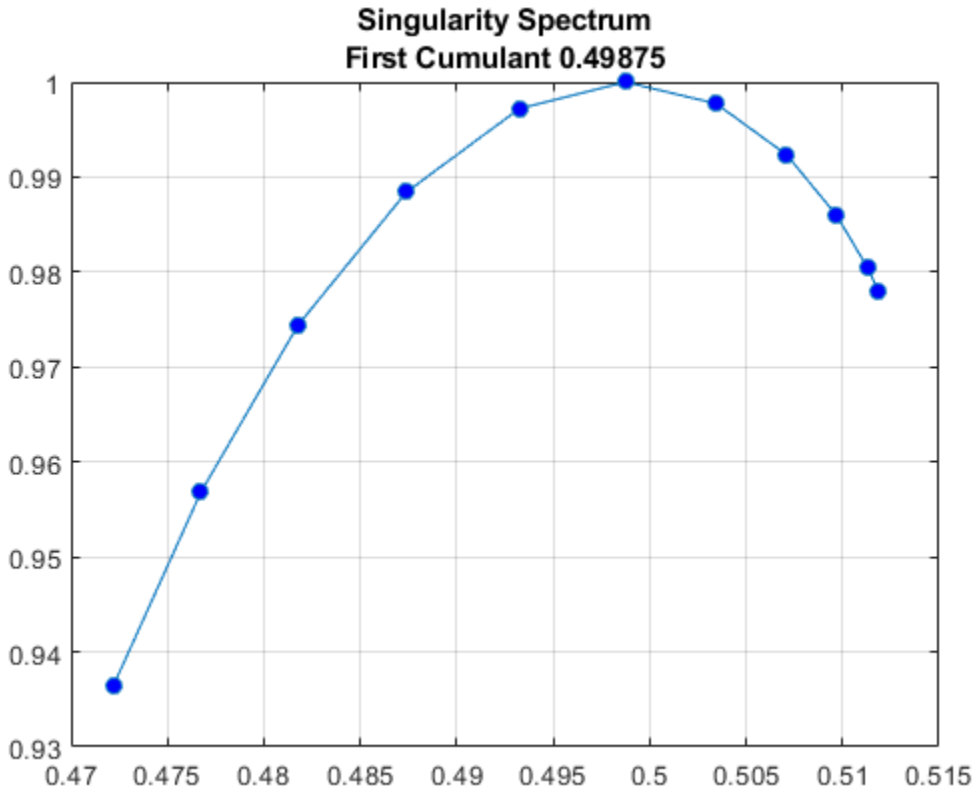
Compute the singularity spectrum and cumulants for a Brownian noise process.

Create the Brownian noise signal.

```
rng(100);
x = cumsum(randn(2^15,1));
```

Obtain and plot the singularity spectrum.

```
[dh,h,cp] = dwtleader(x);
plot(h,dh,'o-','MarkerFaceColor','b')
grid on
title({'Singularity Spectrum'; ['First Cumulant ' num2str(cp(1))]})
```



The small spread in the Holder exponents (approximately 0.472 to 0.512) indicates that this Brownian noise signal can be characterized by a global Holder exponent of 0.49875. The theoretical Holder exponent for Brownian motion is 0.5.

Obtain the cumulants.

```
cp
```

```
cp =

    0.4987   -0.0052   -0.0008
```

The first cumulant value is the slope of scaling exponents versus the moments. The second and third cumulants indicate the deviation from linearity. The first cumulant value and near-zero values of the second and third cumulants indicate that the scaling exponents are a linear function of the moments. Therefore, this Brownian motion signal is monofractal.

Multifractal Random Walk Cumulants

Compute the cumulants for a multifractal random walk. The multifractal random walk is a realization of a random process with a theoretical first cumulant of 0.75 and a second cumulant –0.05. The second cumulant value of –0.05 indicates that the scaling exponents deviate from a linear function with slope 0.75.

Load a random walk signal.

```
load mrw07505
```

Obtain and display the first and second cumulants.

```
[~,~,cp,tauq] = dwtleader(mrw07505);
cp([1 2])
```

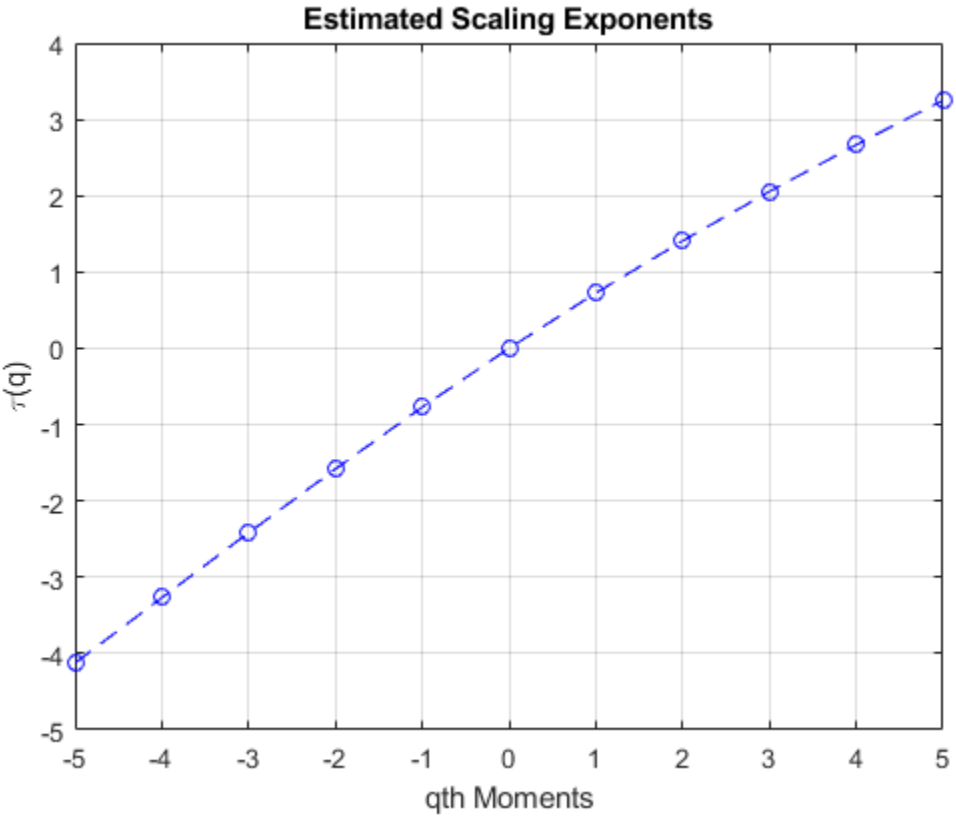
```
ans =

    0.7475   -0.0473
```

For monofractal processes, the scaling exponents are a linear function of the moments. Linearity is indicated by the second and third cumulants being close to zero. In this case, the nonzero second cumulant indicates that the process is multifractal.

Plot the scaling exponents for the q th moments.

```
plot(-5:5,tauq,'bo--')
title('Estimated Scaling Exponents')
grid on
```

```
xlabel('qth Moments')
ylabel('\tau(q)')
```



The scaling exponents are a nonlinear function of the moments.

- "Multifractal Analysis"

Input Arguments

x — Input signal

vector of real values

Input signal, specified as a 1-D vector of real values. For the default wavelet and minimum regression level, the time series must have at least 248 samples. For nondefault values, the minimum-required data length depends on the wavelet filter and the levels used in the regression model. The wavelet leaders technique works best for data with 8000 or more samples.

`wname` — Wavelet name
`'bior1.5'` (default) | character vector

Wavelet name, specified as a character vector. `wname` is a wavelet family short name and filter number recognized by the wavelet manager, `wavemngr`.

To query valid wavelet family short names, use `wavemngr('read')`. To determine whether a particular wavelet is orthogonal or biorthogonal, use `waveinfo` with the wavelet family short name, for example, `waveinfo('db')`. Alternatively, use `wavemngr` with the `'type'` option, for example, `wavemngr('type','fk4')`. A returned value of `1` indicates an orthogonal wavelet. A returned value of `2` indicates a biorthogonal wavelet.

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: `'MinRegressionLevel',5` sets the minimum regression level to 5.

`MinRegressionLevel` — Minimum regression level
`3` (default) | positive integer

Minimum regression level, *minlev*, specified as the comma-separated pair consisting of `'MinRegressionLevel'` and a positive integer greater than or equal to 2. Only levels greater than or equal to the specified minimum level are used in the multifractal estimates. `dwtleader` requires at least 6 wavelet leaders at the maximum level and two levels to be used in the multifractal estimates. The scale in the discrete wavelet transform corresponding to the minimum level is two^{minlev}. The smoother the data (that is, the closer the Holder exponents are to 1), the less likely that reducing the minimum regression level will degrade the results.

`MaxRegressionLevel` — Maximum regression level
positive integer

Maximum regression level, *maxlev*, specified as a positive integer greater than or equal to *minlev* + 1. The maximum level uses only levels less than or equal to *maxlev* in the multifractal estimates. The scale in the discrete wavelet transform corresponding to the maximum level is 2^{maxlev}. Specify a maximum regression level when you want to restrict the levels used in the regression to a value less than the default level. To determine the number of wavelet leaders by level, use the `leaders` output argument, or the `weights` field of the `structfunc` output argument. The default value is the largest level with at least six wavelet leaders

Output Arguments

`dh` — Singularity spectrum

vector

Singularity spectrum, returned as a vector. The singularity spectrum is estimated using structure functions determined for the linearly-spaced moments from –5 to 5. The structure functions are computed based on the wavelet leaders obtained using the biorthogonal spline wavelet filter. The biorthogonal spline wavelet filter that is used has one vanishing moment in the synthesis wavelet and five vanishing moments in the analysis wavelet (`'bior1.5'`). By default, multifractal estimates are derived from wavelet leaders at a minimum level of 3 and maximum level where there are at least six wavelet leaders.

`h` — Holder exponent estimates

1-by-11 vector of real scalars

Holder exponent estimates, returned as a 1-by-11 vector of scalars. Holder exponents characterize signal regularity. The closer a Holder exponent is to 1, the closer the function is to differentiable. Conversely, the closer the Holder exponent is to zero, the closer the function is to discontinuous.

Data Types: `double`

`cp` — Cumulants

vector

Cumulants, returned as a 1–by-3 vector of scalars. The vector contains the first three log cumulants of the scaling exponents. The first cumulant characterizes the linear behavior in the scaling exponents. The second and third cumulants characterize the departure from linearity.

Data Types: `double`

`tauq` — Scaling exponents

column vector

Scaling exponents, returned as a column vector. The exponents are for the linearly-spaced moments from –5 to +5.

`leaders` — Wavelet leaders

cell array

`leaders` is a cell array with the ith element containing the wavelet leaders at level i+1, or scale $2^{(i+1)}$. Wavelet leaders are not defined at level 1.

`structfunc` — Multiresolution structure functions

`struct`

Multiresolution structure functions for the global Holder exponent estimates, returned as a `struct`. The structure function for data `x` is defined as

$$S(q,a) = \frac{1}{n_a}\sum_{k=1}^{n_a} |T_x(a,k)|^q \simeq a^{\zeta(q)},$$

where a is the scale, q is the moment, T_x are the wavelet leaders by scale, n_a is the number of wavelet leaders at each scale, and $\zeta(q)$ is the scaling exponent. Expanding $\zeta(q)$ to a polynomial produces

$$\zeta(q) = c_1 q + c_2 q^2/2 + c3q^3/6 + ...$$

The scaling exponents can be estimated from the log-cumulants of the wavelet leader coefficients. When $\zeta(q)$ is a linear function, the signal is monofractal. When it deviates from linear, the signal is multifractal.

`structfunc` is a structure array containing the following fields:

- `Tq` — Measurements of the input, `x`, at various scales. `Tq` is a matrix of multiresolution quantities that depend jointly on time and scale. Scaling phenomena in `x` imply a power-law relationship between the moments of `Tq` and the scale. For `dwtleader`, the `Tq` field is an *Ns*-by-36 matrix, where *Ns* is the number of scales used in the multifractal estimates. The first 11 columns of `Tq` are the scaling exponent estimates by scale for each of the qth moments from –5 to 5. The next 11 columns

contain the singularity spectrum estimates, `dh`, for each of the *q*th moments. Columns 23–33 contain the Holder exponent estimates, `h`. The last three columns contain the estimates for the first-order, second-order, and third-order cumulants, respectively.

- `weights` — Weights used in the regression. The weights are the number of wavelet leaders by scale. `weights` is an *Ns*-by-1 vector.
- `logscales` — Scales used as predictors in the regression. `logscales` is an *Ns*-by-1 vector with the base-2 logarithm of the scales.

Algorithms

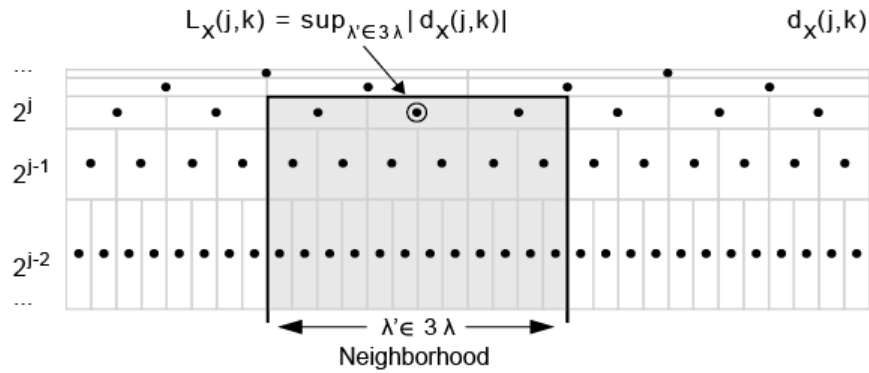
Wavelet leaders are derived from the critically sampled discrete wavelet transform (DWT) coefficients. Wavelet leaders offer significant theoretical advantages over wavelet coefficients in the multifractal formalism. Wavelet leaders are time- or space-localized suprema of the absolute value of the discrete wavelet coefficients. The time localization of the suprema requires that the wavelet coefficients are obtained using a compactly supported wavelet. The Holder exponents, which quantify the local regularity, are determined from these suprema. The singularity spectrum indicates the size of the set of Holder exponents in the data.

1-D wavelet leaders are defined as

$$L_x(j,k) = \sup_{\lambda' \subset 3\lambda_{j,k}} |d_x(j,k)|$$

where the scales are 2^j, translated to time positions $2^j k$. The time neighborhood is

$3\lambda_{j,k} = \lambda_{j,k-1} \cup \lambda_{j,k} \cup \lambda_{j,k+1}$, where $\lambda_{j,k} = \left[k2^j, (k+1)2^j\right)$. The time neighborhood is taken over the scale and all finer scales. $d_x(j,k)$ are the wavelet coefficients.



To calculate the wavelet leaders, $L_x(j,k)$:

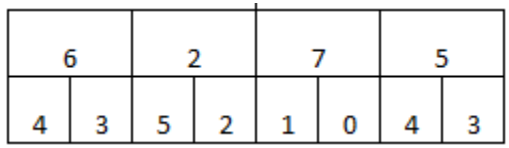
1. Compute the wavelet coefficients, $d_x(j,k)$, using the discrete wavelet transform and save the absolute value of each coefficient for each scale. Each finer scale has twice the number of coefficients than the next coarser scale. Each dyadic interval at scale 2^j can be written as a union of two intervals at a finer scale.

   $$[2^j k, 2^j (k+1)) = [2^{j-1}(2k), 2^{j-1}(2k+2))$$

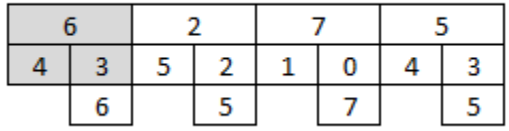
   $$[2^{j-1}(2k), 2^{j-1}(2k+2)) = [2^{j-1}(2k), 2^{j-1}(2k+1)) \cup [2^{j-1}(2k+1), 2^{j-1}(2k+2))$$

2. Start at the scale that is one level coarser than the finest obtained scale.
3. Compare the first value to all its finer dyadic intervals and obtain the maximum value.
4. Go to the next value and compare its value to all of its finer scale values.
5. Continue comparing the values with their nested values and obtaining the maxima.
6. From the maximum values obtained for that scale, examine the first three values and obtain the maximum of those neighbors. That maximum value is a leader for that scale.
7. Continue comparing the maximum values to obtain the other leaders for that scale.
8. Move to the next coarser scale and repeat the process.

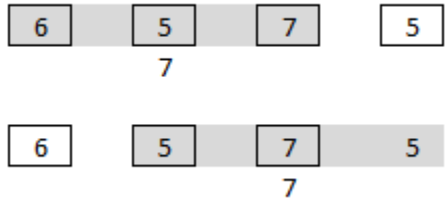
For example, assume that you have these absolute values of the coefficients at these scales:



Starting with the top row, which is the next coarsest level from the finest scale (bottom row), compare each value to its dyadic intervals and obtain the maxima.



Then, look at the three neighboring values and obtain the maximum. Repeat for the next three neighbors. These maxima, 7 and 7, are the wavelet leaders for this level.



References

[1] Wendt, H., and P. Abry. "Multifractality Tests Using Bootstrapped Wavelet Leaders." *IEEE Transactions on Signal Processing*. Vol. 55, No. 10, 2007, pp. 4811–4820.

[2] Jaffard, S., B. Lashermes, and P. Abry. “Wavelet Leaders in Multifractal Analysis.” *Wavelet Analysis and Applications*. T. Qian, M. I. Vai, and X. Yuesheng, Eds. 2006, pp. 219–264.

See Also

`wfbm` | `wtmm`

Topics

"Multifractal Analysis"

Introduced in R2016b

dwtmode

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".

`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

'mode'	DWT Extension Mode
'asymw'	Antisymmetric-padding (whole-point): boundary value antisymmetric replication
'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)

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 Wavelet Analyzer app 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` 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.

Note You should change the extension mode only by using `dwtmode`. Avoid changing the global variable directly.

Examples

Display and Change Signal Extension Mode

If the DWT extension mode global variable does not exist, the default is **Symmetrization**.

```
clear global
dwtmode
```

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

Display the current DWT signal extension mode.

```
dwtmode
```

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

Change the extension mode to **Periodization**.

```
dwtmode('per')
```

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

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

Display the current DWT signal extension mode.

```
dwtmode
```

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

References

[1] Strang, Gilbert, and Truong Nguyen. *Wavelets and Filter Banks*. Wellesley, MA: Wellesley-Cambridge Press, 1996.

See Also

Apps

Wavelet Analyzer

Functions

`dwt` | `dwt2` | `idwt` | `idwt2` | `wextend`

Introduced before R2006a

dyaddown

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`

Introduced before R2006a

dyadup

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.

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- If `X` is empty, generated code returns `X` and MATLAB returns `[]`.
- Suppose that all of the following conditions are true:
  - `X` is a variable-size array.
  - `X` is not a variable-length column vector (:-by-1).
  - `X` is a column vector at run time.
  - `'type'` is not supplied.

  In generated code, the output for `y = dyadup(X,k)`, where `k` is optional, matches the output for `y = dyadup(X,k,'c')`. In MATLAB, the output for `y = dyadup(X,k)` matches the output for `y = dyadup(X,k,'r')`.

  For code generation, when you do not specify `'type'`, if you want `dyadup` to treat `X` as a column vector, `X` must be a variable-length vector (:-by-1).

See Also

`dyaddown`

Introduced before R2006a

entrupd

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

Introduced before R2006a

fbspwavf

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 $1 \leq M$), 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]`.

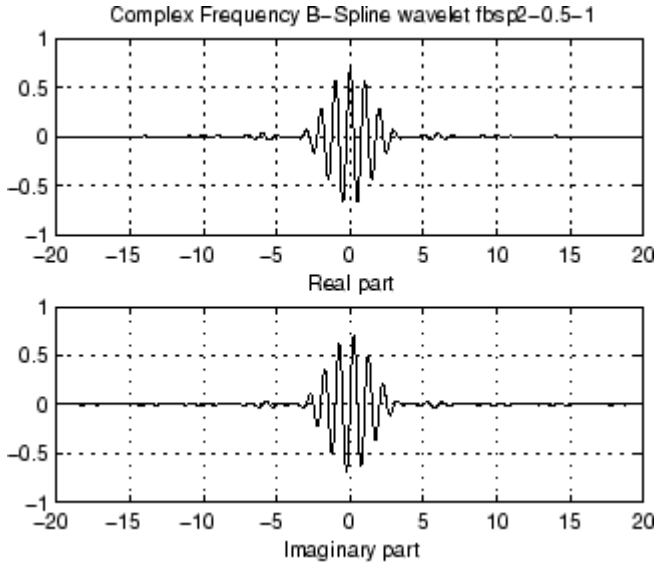
FB and *FC* must be such that `FC > 0` and > `FB > 0`.

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
```



References

Teolis, A. (1998), *Computational signal processing with wavelets*, Birkhauser, p. 63.

See Also

`waveinfo`

Introduced before R2006a

filt2ls

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`

Introduced before R2006a

fejerkorovkin

Fejer-Korovkin wavelet filters

Syntax

```
Lo = fejerkorovkin(wname)
```

Description

`Lo = fejerkorovkin(wname)` returns the Fejer-Korovkin scaling filter specified by `wname`. Valid entries for `wname` are `'fk4'`, `'fk6'`, `'fk8'`, `'fk14'`, `'fk18'`, and `'fk22'`. For information on the Fejer–Korovkin filters, see Nielson[1].

Examples

Fejer-Korovkin Filters

Construct and plot the Fejer-Korovkin (14) scaling function and wavelet.

Obtain the Fejer-Korovkin scaling filter and display its 14 coefficients.

```
Lo = fejerkorovkin('fk14')
```

```
Lo =

  Columns 1 through 7

    0.2604    0.6869    0.6116    0.0514   -0.2456   -0.0486    0.1243

  Columns 8 through 14

    0.0222   -0.0640   -0.0051    0.0298   -0.0033   -0.0093    0.0035
```

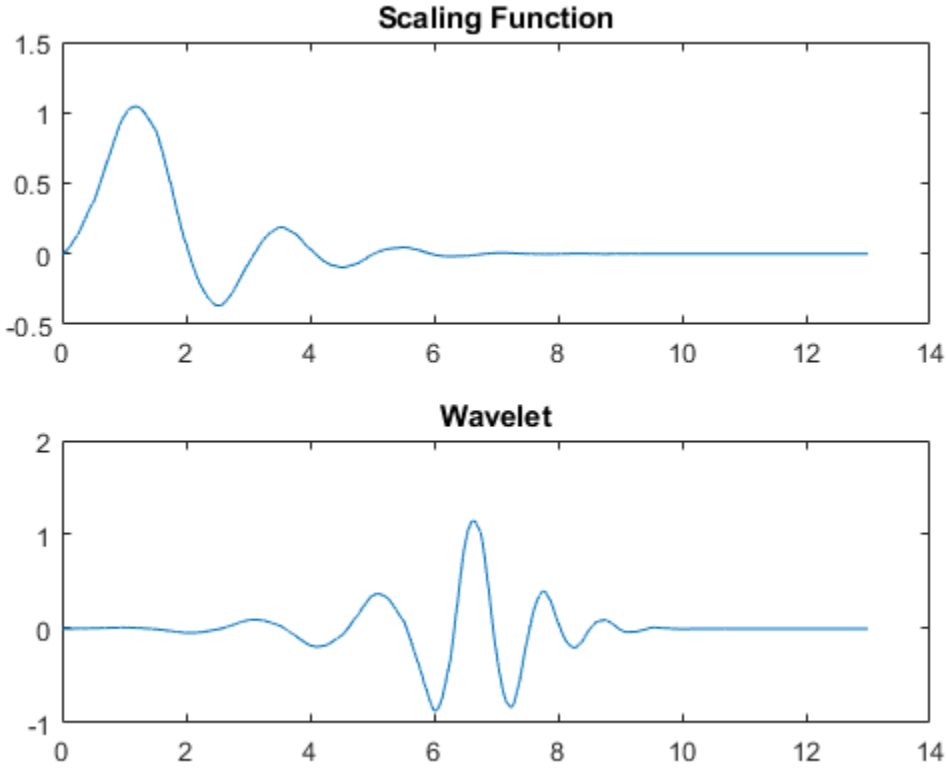
Use the scaling filter to obtain the wavelet filter and display its wavelet filter coefficients.

```
Hi = qmf(Lo)
```

```
Hi =

  Columns 1 through 7

    0.0035    0.0093   -0.0033   -0.0298   -0.0051    0.0640    0.0222

  Columns 8 through 14

   -0.1243   -0.0486    0.2456    0.0514   -0.6116    0.6869   -0.2604
```

`wavefun` provides an efficient way to construct and plot the scaling function and wavelet.

```
[phi,psi,xval] = wavefun('fk14');
subplot(2,1,1)
plot(xval,phi)
title('Scaling Function')
subplot(2,1,2)
plot(xval,psi)
title('Wavelet')
```



Input Arguments

`wname` — Filter name

`'fk4' | 'fk6' | 'fk8' | 'fk14' | 'fk18' | 'fk22'`

Filter name, specified as a character vector. The numeric value in each name is the number of Fejer-Korovkin filter coefficients.

Output Arguments

`Lo` — Scaling filter

vector

Scaling filter, returned as a vector.

References

[1] Nielsen, M. "On the construction and frequency localization of finite orthogonal quadrature filters." *Journal of Approximation Theory*. Vol. 108, pp. 36–52.

See Also

`coifwavf` | `dbwavf` | `symwavf`

Introduced in R2015b

gauswavf

Gaussian wavelet

Syntax

```
[PSI,X] = gauswavf(LB,UB,N)
[PSI,X] = gauswavf(LB,UB,N,P)
[PSI,X] = gauswavf(LB,UB,N,WAVNAME)
```

Description

`[PSI,X] = gauswavf(LB,UB,N)` returns the 1st order derivative of the Gaussian wavelet, `PSI`, on an `N`-point regular grid, `X`, for the interval `[LB,UB]`. The effective support of the Gaussian wavelets is `[-5 5]`.

`[PSI,X] = gauswavf(LB,UB,N,P)` returns the `P`th derivative. Valid values of `P` are integers from 1 to 8.

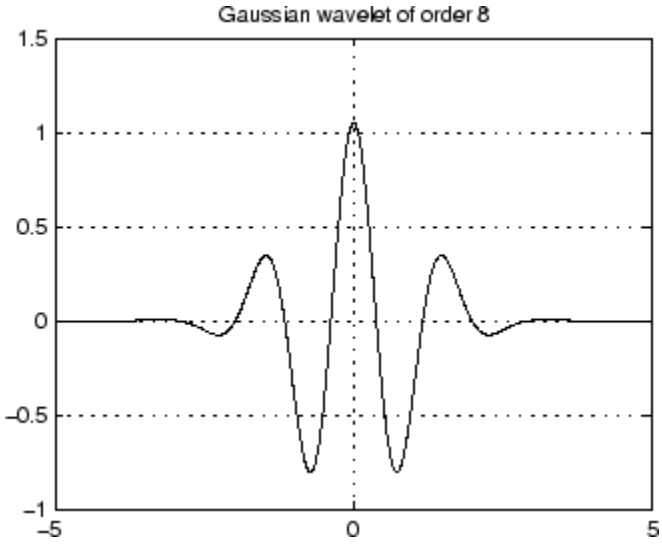
The Gaussian function is defined as $C_p e^{-x^2}$. C_p is such that the 2-norm of the `P`th derivative of `PSI` is equal to 1.

`[PSI,X] = gauswavf(LB,UB,N,WAVNAME)` uses the valid wavelet family short name `WAVNAME` plus the order of the derivative in a character vector, such as `'gaus4'`. To see valid character vectors for Gaussian wavelets, use `waveinfo('gaus')` or use `wavemngr('read',1)` and refer to the Gaussian section.

Note For visualizing the second or third order derivative of Gaussian wavelets, the convention is to use the negative of the normalized derivative. In the case of the second derivative, scaling by -1 produces a wavelet with its main lobe in the positive *y* direction. This scaling also makes the Gaussian wavelet resemble the Mexican hat, or Ricker, wavelet. The validity of the wavelet is not affected by the -1 scaling factor.

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
```



See Also

`waveinfo`

Introduced before R2006a

get

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`

Introduced before R2006a

haart

Haar 1-D wavelet transform

Syntax

```
[a,d] = haart(x)
[a,d] = haart(x,level)
[a,d] = haart(___,integerflag)
```

Description

`[a,d] = haart(x)` returns the approximation coefficients, `a`, and detail coefficients, `d`, of a 1-D Haar discrete wavelet transform. The input `x` can be univariate or multivariate data. The default `level` depends on the length of `x`.

`[a,d] = haart(x,level)` obtains the Haar transform down to the specified level.

`[a,d] = haart(___,integerflag)` specifies how the Haar transform handles integer-valued data, using any of the previous syntaxes.

Examples

Haar Transform of ECG Data

Obtain the Haar transform down to the default maximum level.

```
load wecg;
[a,d] = haart(wecg);
```

Haar Transform of Electricity Consumption Data Down to Specified Level

Obtain the Haar transform of a multivariate time series dataset of electricity consumption data down to level 4. The `signals` data is transposed so that each time series is in a column, rather than a row.

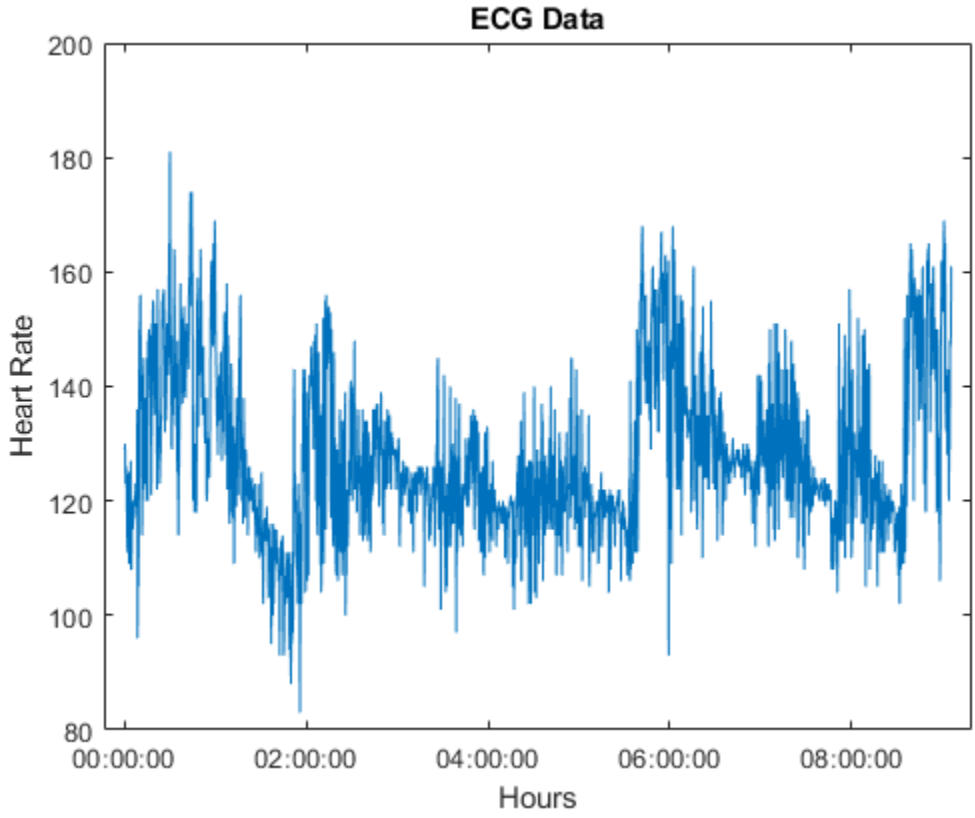
```
load elec35_nor;
signals = signals';
[a,d] = haart(signals,4);
```

Haar Transform of Integer Data Series

Obtain the Haar transform and inverse Haar transform of ECG heart rate data. The data is made up of integers only.

Load and plot the ECG data.

```
load BabyECGData;
plot(times,HR)
xlabel('Hours')
ylabel('Heart Rate')
title('ECG Data')
```

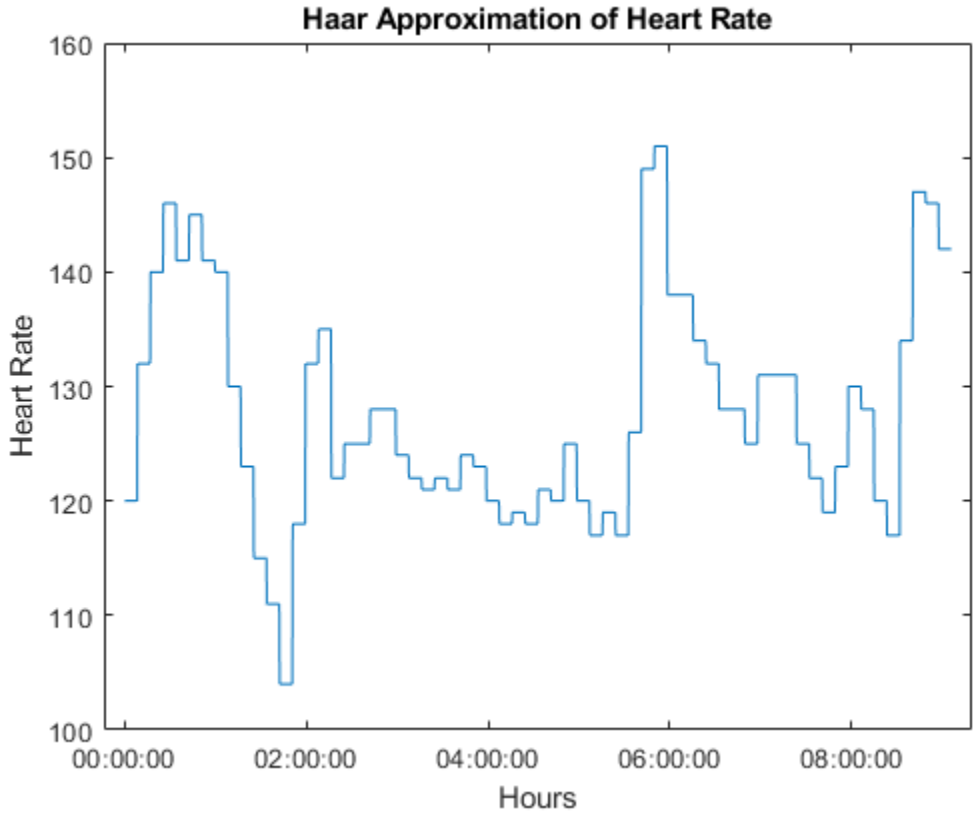


Obtain the Haar transform. Then, obtain the inverse Haar transform approximated at level 5. The scale for this level is 512 seconds, which is 2^5 times the sampling interval (16 seconds).

```
[a,d] = haart(HR,'integer');
HaarHR = ihaart(a,d,5,'integer');
```

Compare the reconstructed data to the original data.

```
figure;
plot(times,HaarHR)
xlabel('Hours')
ylabel('Heart Rate')
title('Haar Approximation of Heart Rate')
```



- “Haar Transforms for Time Series Data and Images”

Input Arguments

x — Input signal

vector of real values | matrix of real values

Input signal, specified as vector or matrix of real values. If x is a vector, it must be even length. If `x` is a matrix, each column must be even length, and `haart` operates on each column of `x`.

`level` — Maximum level

positive integer

Maximum level to which to perform the Haar transform, specified as a positive integer. The default value depends on the length of the input signal, `x`.

- If the length of `x` is a power of two, the Haar transform is obtained down to level `log2(length(x))`.
- If the length of `x` is even, but not a power of two, the Haar transform is obtained down to level `floor(log2(length(x)/2))`.

If `level` is 1, the detail coefficients, `d`, are returned as a vector or matrix, depending on whether the input is a vector or matrix, respectively.

`integerflag` — Integer-valued data handling

`'noninteger'` (default) | `'integer'`

Integer-valued data handling, specified as a character vector. `'noninteger'` does not preserve integer-valued data in the Haar transform, and `'integer'` preserves it. The `'integer'` option applies only if all elements of the input, `x`, are integers. For integer-valued input, `haart` returns integer-valued wavelet coefficients. For both `'noninteger'` and `'integer'`, however, the Haar transform algorithm uses floating-point arithmetic. The data type of outputs `a` and `d`, is always `double`.

Output Arguments

`a` — Approximation coefficients

scalar | vector | matrix

Approximation coefficients, returned as a scalar, vector, or matrix of coefficients, depending on the level to which the transform is calculated. Approximation, or scaling, coefficients are a lowpass representation of the input. At each level, the approximation coefficients are divided into coarser approximation and detail coefficients.

Data Types: `double`

`d` — Detail coefficients

scalar | vector | matrix | cell array

Detail coefficients, returned as a scalar, vector, matrix, or cell array. Detail coefficients are generally referred to as the wavelet coefficients and are a highpass representation of the input. The number of detail coefficients depends on the selected level and the length of the input. The order of the elements of d is from fine to coarse resolution levels. The coarsest resolution level element of the `d` cell array is a scalar value. If you specify only two levels, the detail coefficient is a scalar.

Data Types: `double`

See Also

`haart2` | `ihaart` | `ihaart2`

Topics

"Haar Transforms for Time Series Data and Images"

Introduced in R2016b

haart2

2-D Haar wavelet transform

Syntax

```
[a,h,v,d] = haart2(x)
[a,h,v,d] = haart2(x,level)
[a,h,v,d] = haart2(___,integerflag)
```

Description

`[a,h,v,d] = haart2(x)` performs the 2-D Haar discrete wavelet transform (DWT) of the matrix, `x`. `haart2` returns the approximation coefficients, `a`, at the coarsest level. `haart2` also returns cell arrays of matrices containing the horizontal, vertical, and diagonal detail coefficients by level. If the 2-D Haar transform is computed only at one level coarser in resolution, then `h`, `v`, and `d` are matrices. The default `level` depends on the number of rows of `x`.

`[a,h,v,d] = haart2(x,level)` performs the 2-D Haar transform down to the specified level.

`[a,h,v,d] = haart2(___,integerflag)` specifies how the 2-D Haar transform handles integer-valued data, using any of the previous syntaxes.

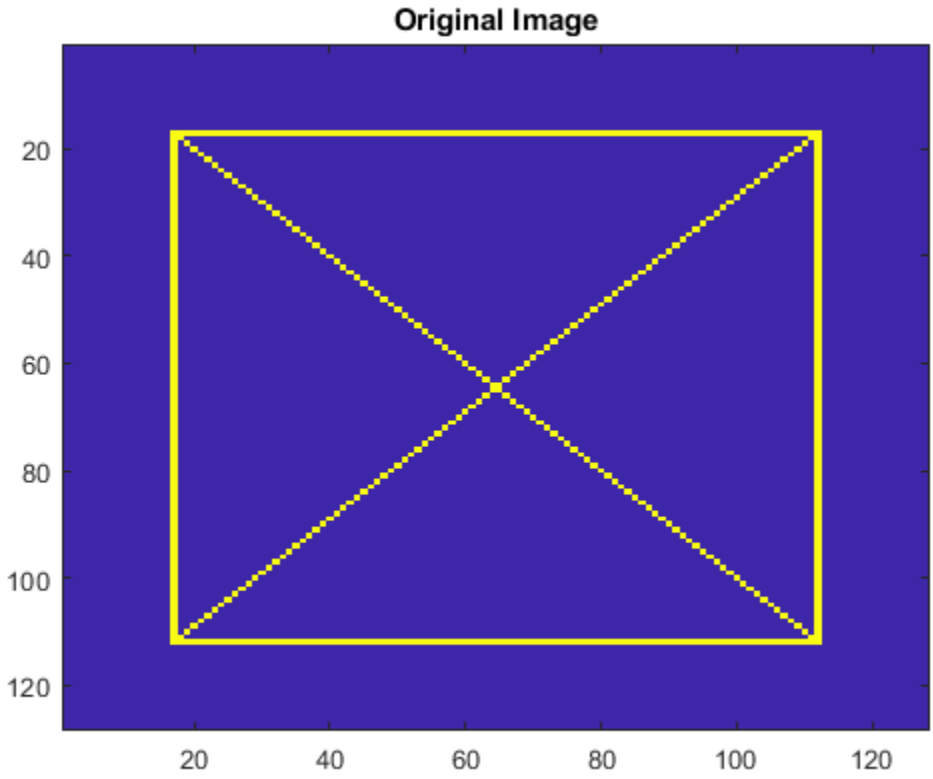
Examples

Haar Transform and First Level Details of 2-D Data

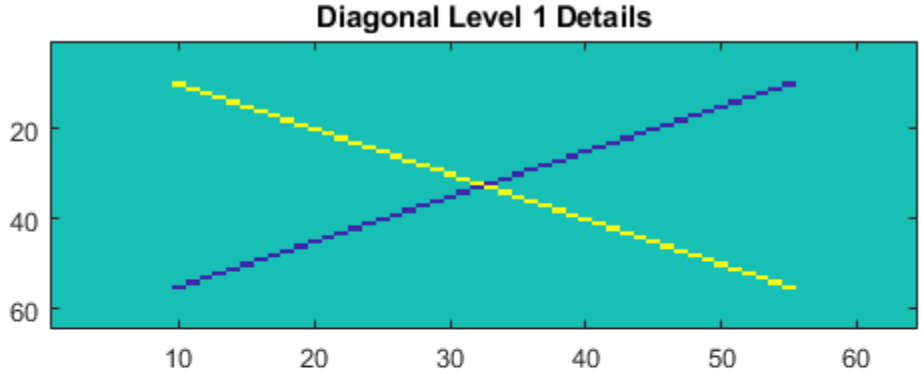
Obtain the 2-D Haar transform of 2-D data and plot its diagonal and horizontal level 1 details.

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

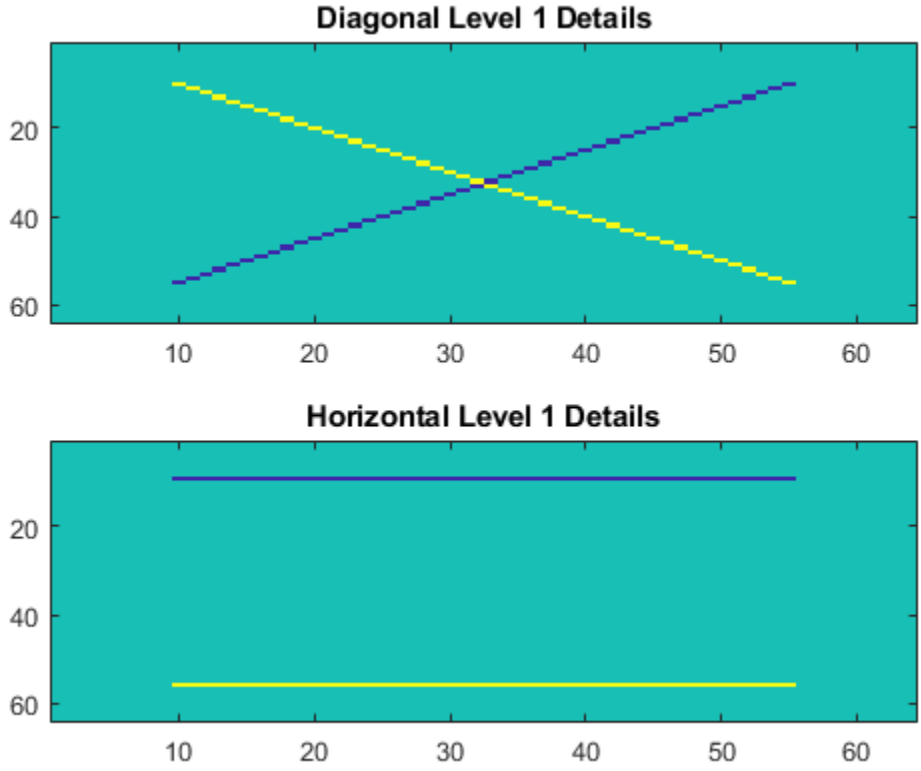
```
title('Original Image')
figure
[a,h,v,d] = haart2(xbox);
```



```
subplot(2,1,1)
imagesc(d{1})
title('Diagonal Level 1 Details');
```




```
subplot(2,1,2)
imagesc(h{1})
title('Horizontal Level 1 Details');
```

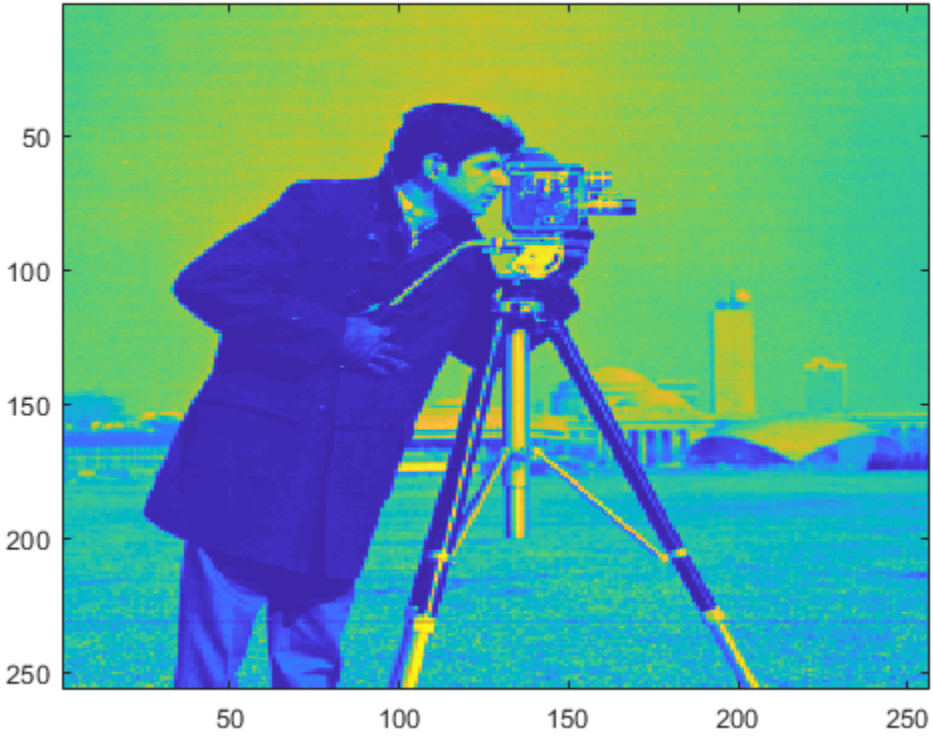


Haar Transform of Image Down to a Specified Level

Show the effect of limiting the maximum level of the 2-D Haar transform on an image.

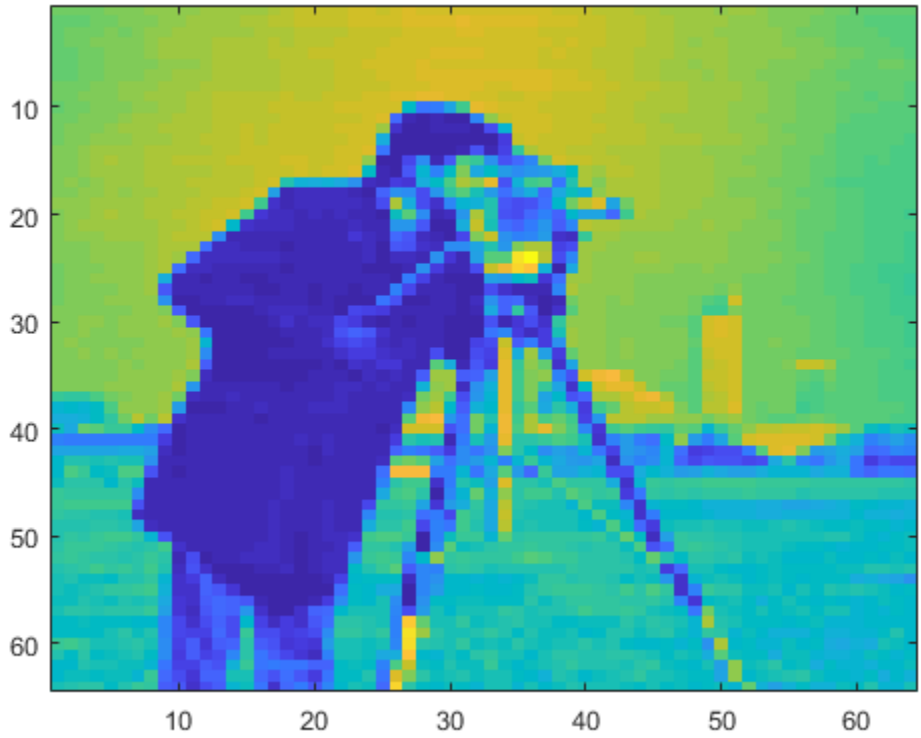
Load and display the image of a cameraman.

```
im = imread('cameraman.tif');
imagesc(im)
```



Obtain the 2-D Haar transform to level 2 and view the level 2 approximation.

```
[a2,h2,v2,d2] = haart2(im,2);
imagesc(a2)
```



Haar Transform Using Integer Image Data

Compare 2-D Haar transform results using the default `'noninteger'` flag and the `'integer'` flag. The cameraman image is `uint8` data, so its maximum value is 255.

Obtain the default Haar transform. The approximation detail coefficient is outside the range 0 to 255.

```
im = imread('cameraman.tif');
[a,h,v,d] = haart2(im);
a
```

```
a = 3.0393e+04
```

Obtain the Haar transform, limiting it to integer values. The approximation detail is an integer and is within the range of the original image data.

```
[a,h,v,d] = haart2(im,'integer');
a
```

```
a = 119
```

- “Haar Transforms for Time Series Data and Images”

Input Arguments

`x` — Input signal

matrix of real values

Input signal, specified as a 2-D or 3-D matrix of real values. If `x` is 3-D, the third dimension of `x` must equal 3. The row and column sizes of `x` must be even length.

`level` — Maximum level

positive integer

Maximum level to which to perform the 2-D Haar transform, specified as a positive integer. The default value depends on the length of the input signal, `x`.

- If both the row and column sizes of `x` are powers of two, the 2-D Haar transform is obtained down to `level` `log2(min(size(x)))`.
- If both the row and column sizes of `x` are even, but at least one is not a power of two, `level` is equal to `floor(log2(min(size(x)/2)))`.

If `level` is greater than 1, then `h`, `v`, and `d` are cell arrays. If `level` is equal to 1, then `h`, `v`, and `d` are matrices.

`integerflag` — Integer-valued data handling

`'noninteger'` (default) | `'integer'`

Integer-valued data handling, specified as a character vector. `'noninteger'` does not preserve integer-valued data in the 2-DHaar transform, and `'integer'` preserves it. The `'integer'` option applies only if all elements of the input, `x`, are integers. For

integer-valued input, `haart2` returns integer-valued wavelet coefficients. For both `'noninteger'` and `'integer'`, however, the 2-D Haar transform algorithm uses floating-point arithmetic. The data type of outputs `a`, `h`, `v`, and `d`, is always `double`.

Output Arguments

`a` — Approximation coefficients

scalar | matrix

Approximation coefficients, returned as a scalar or matrix of coefficients, depending on the level to which the transform is calculated. Approximation, or scaling, coefficients are a lowpass representation of the input. At each level, the approximation coefficients are divided into coarser approximation and detail coefficients.

Data Types: `double`

`h` — Horizontal detail coefficients

matrix | cell array

Horizontal detail coefficients by level, returned as a matrix or cell array of matrices. If `level` is greater than 1, `h` is a cell array. If `level` is equal to 1, the 2-D Haar transform is computed at only one level coarser in resolution and `h` is a matrix.

Data Types: `double`

`v` — Vertical detail coefficients

matrix | cell array

Vertical detail coefficients by level, returned as a matrix or cell array of matrices. If `level` is greater than 1, `v` is a cell array. If `level` is equal to 1, the 2-D Haar transform is computed at only one level coarser in resolution and `v` is a matrix.

Data Types: `double`

`d` — Diagonal detail coefficients

matrix | cell array

Diagonal detail coefficients by level, returned as a matrix or cell array of matrices. If `level` is greater than 1, `d` is a cell array. If `level` is equal to 1, the 2-D Haar transform is computed at only one level coarser in resolution and `d` is a matrix.

Data Types: `double`

See Also

`haart` | `ihaart` | `ihaart2`

Topics

“Haar Transforms for Time Series Data and Images”

Introduced in R2016b

icwt

Inverse continuous 1-D wavelet transform

Syntax

```
xrec = icwt(wt)
xrec = icwt(wt,wname)
xrec = icwt(wt,f,freqrange)
xrec = icwt(wt,period,periodrange)
xrec = icwt(___,Name,Value)
```

Description

`xrec = icwt(wt)` inverts the continuous wavelet transform (CWT) coefficient matrix `wt` using default values. `icwt` assumes that you obtained the CWT using `cwt` with the default Morse wavelet. This wavelet has a symmetry of 3 and a time bandwidth of 60. `icwt` also assumes that the CWT uses default scales. If `wt` is a 2-D matrix, `icwt` assumes that the CWT was obtained from a real-valued signal. If `wt` is a 3-D matrix, `icwt` assumes that the CWT was obtained from a complex-valued signal. For a 3-D matrix, the first page of the `wt` is the CWT of the positive (counterclockwise) component and the second page of `wt` is the negative (clockwise) component. The pages represent the analytic and anti-analytic parts of the CWT, respectively.

`xrec = icwt(wt,wname)` uses the analytic wavelet `wname` to invert the CWT. The specified wavelet must be the same as the wavelet used in `cwt`. Valid options for `wname` are `'morse'`, `'amor'`, and `'bump'`, which specify the Morse, Morlet, and bump wavelet, respectively.

`xrec = icwt(wt,f,freqrange)` inverts the CWT over the frequency range specified in `freqrange`. `f` is the scale-to-frequency conversion obtained from `cwt`.

`xrec = icwt(wt,period,periodrange)` inverts the CWT over the range of periods specified in `periodrange`.`period` is an array of durations obtained from `cwt` with a duration input. The `period` is the `cwt` output obtained using a `duration` input. The period range must be increasing and contained in `period`.

`xrec = icwt(___,Name,Value)` returns the inverse CWT with additional options specified by one or more `Name,Value` pair arguments.

Examples

Inverse Continuous Wavelet Transform of Speech Signal

Obtain the CWT of a speech sample and invert the CWT using the default analytic Morse wavelet.

```
load mtlb;
wt = cwt(mtlb);
xrec = icwt(wt);
```

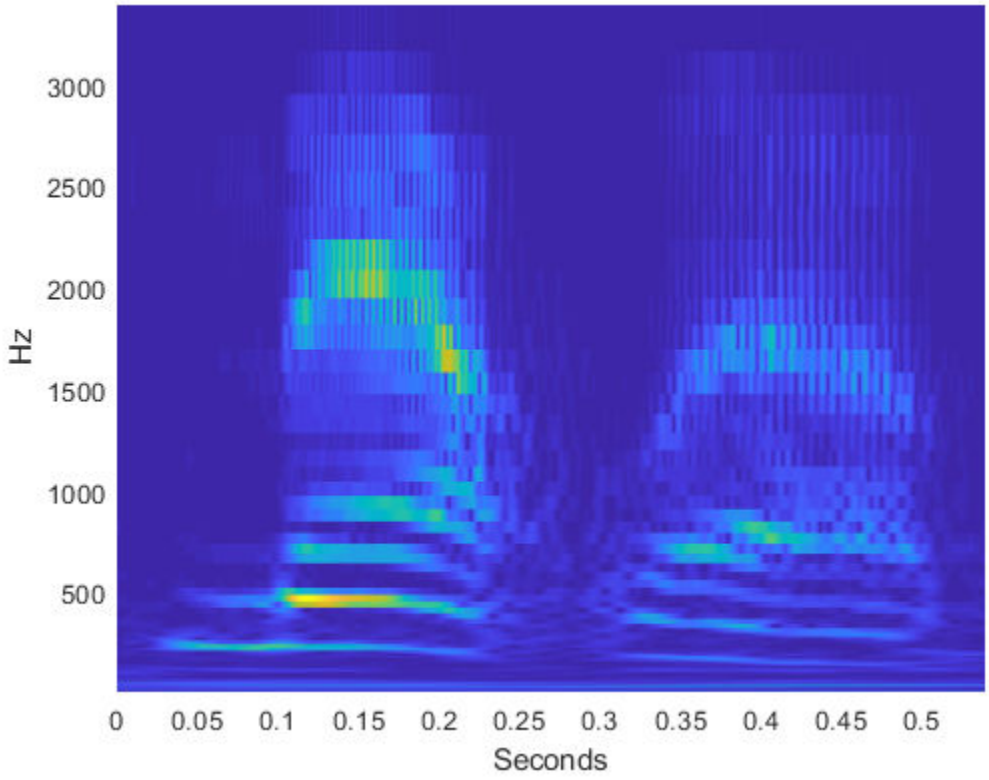
Inverse Continuous Wavelet Transform Using Specified Wavelet

Obtain the continuous wavelet transform of a speech sample and reconstruct the sample using the bump wavelet instead of the default Morse wavelet.

```
load mtlb;
dt = 1/Fs;
t = 0:dt:numel(mtlb)*dt-dt;
```

Obtain and plot the CWT.

```
[bumpmtlb,f] = cwt(mtlb,Fs,'bump');
p1 = pcolor(t,f,abs(bumpmtlb));
p1.EdgeColor = 'none';
xlabel('Seconds');
ylabel('Hz');
```

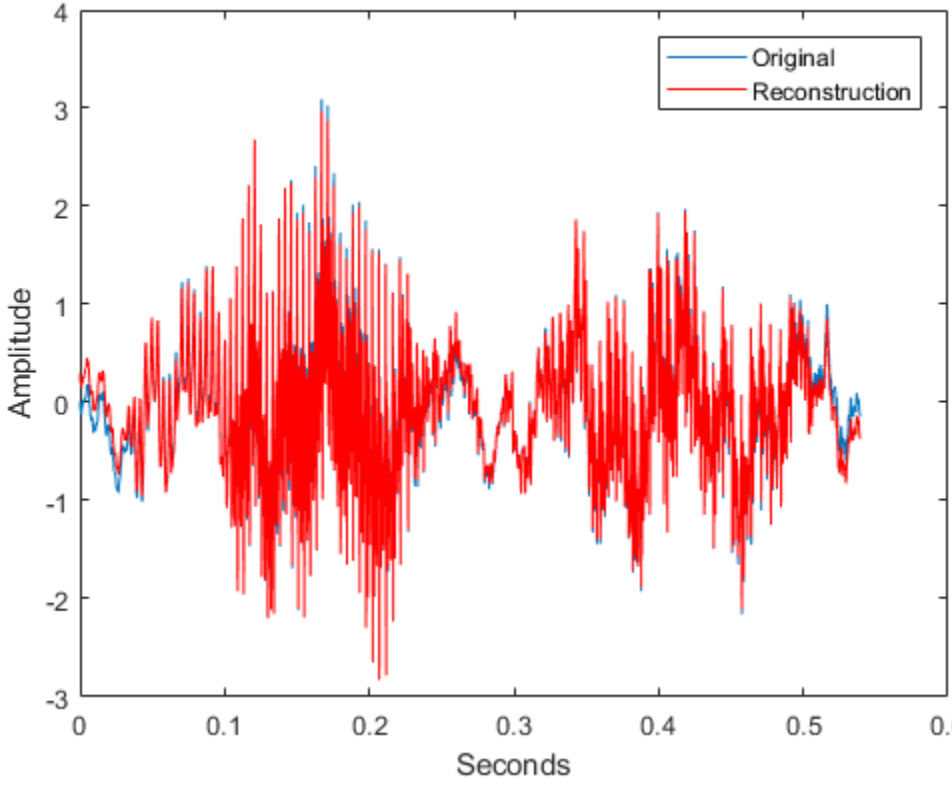


Obtain the inverse CWT.

```
xrec = icwt(bumpmtlb,'bump','SignalMean',mean(mtlb));
```

Plot the original and reconstructed signals.

```
plot(t,mtlb);
xlabel('Seconds');
ylabel('Amplitude');
hold on;
plot(t,xrec,'r');
legend('Original','Reconstruction');
```



Play and compare the original and reconstructed signals.

```
p = audioplayer(mtlb,Fs);
play(p);
pause(2);
px = audioplayer(xrec,Fs);
play(px);
```

Reconstruct Frequency-Localized Data

Reconstruct a frequency-localized approximation to the Kobe earthquake data by extracting information from the CWT. The extracted information corresponds to frequencies in the range [0.030 0.070] Hz.

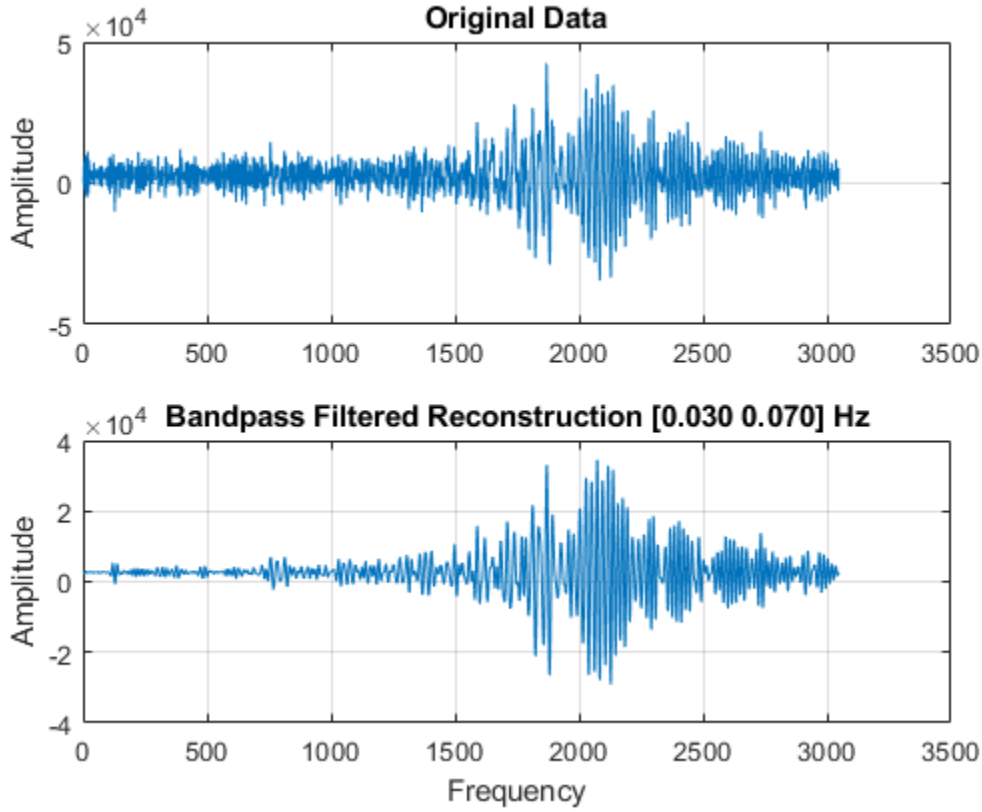
```
load kobe;
```

Obtain the CWT. Then, obtain the inverse CWT and add the signal mean back into the reconstructed data. The CWT does not preserve the signal mean.

```
[wt,f] = cwt(kobe,1);
xrec = icwt(wt,f,[0.030 0.070],'SignalMean',mean(kobe));
```

Plot the original and reconstructed data.

```
subplot(211)
plot(kobe);
grid on
title('Original Data');
ylabel('Amplitude')

subplot(212)
plot(xrec);
grid on
title('Bandpass Filtered Reconstruction [0.030 0.070] Hz');
xlabel('Frequency');
ylabel('Amplitude');
```



Reconstruct Data from Specific Time Period

Use the inverse continuous wavelet transform to reconstruct an approximation to El Nino data based on 2 to 8 year periods.

Load the El Nino data and obtain its CWT. The data is sampled monthly. To obtain the periods in years, specify the sampling interval as 1/12 of a year.

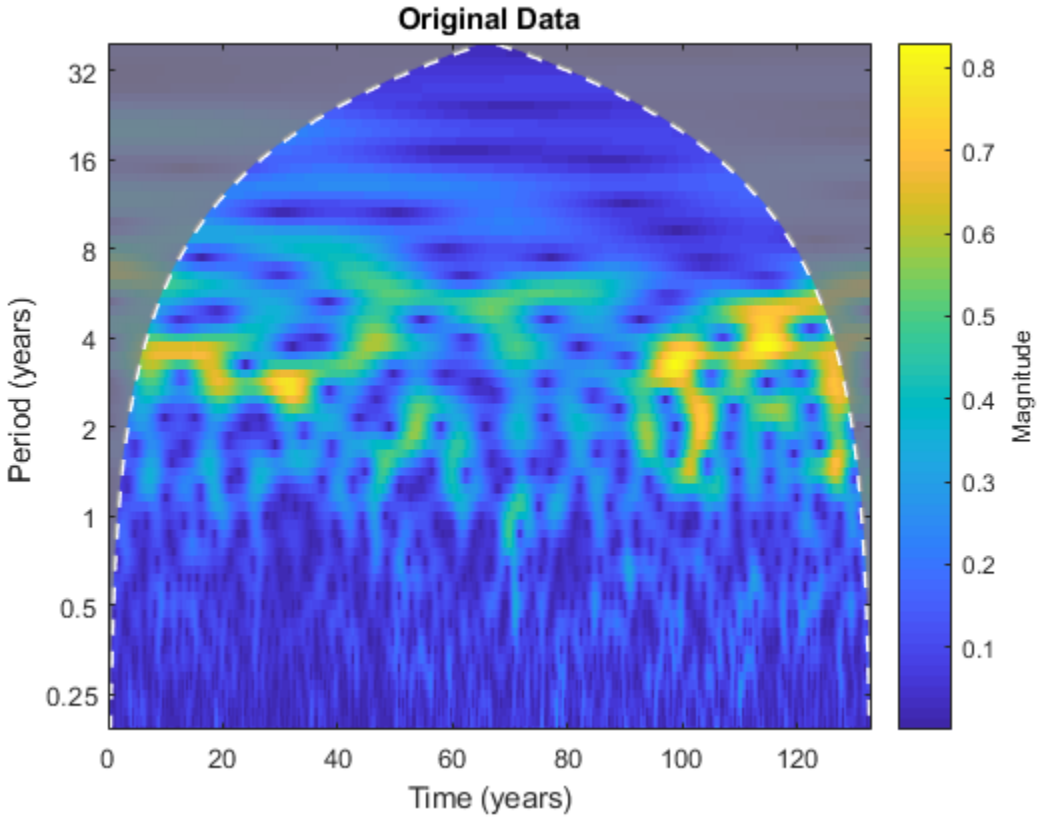
```
load ninoairdata;
[cfs,period] = cwt(nino,years(1/12));
```

Obtain the inverse CWT for periods of 2 to 8 years.

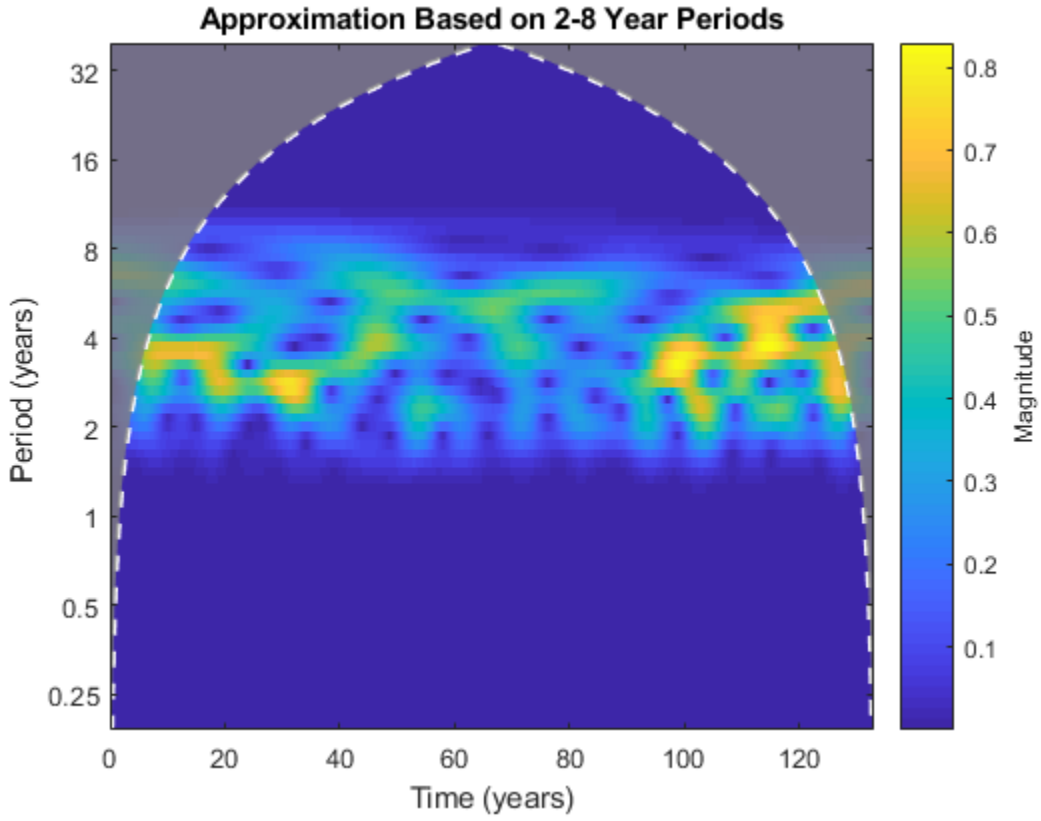
```
xrec = icwt(cfs,period,[years(2) years(8)]);
```

Plot the CWT of the reconstructed data and compare it to the CWT of the original data.

```
cwt(nino,years(1/12)); title('Original Data');
```



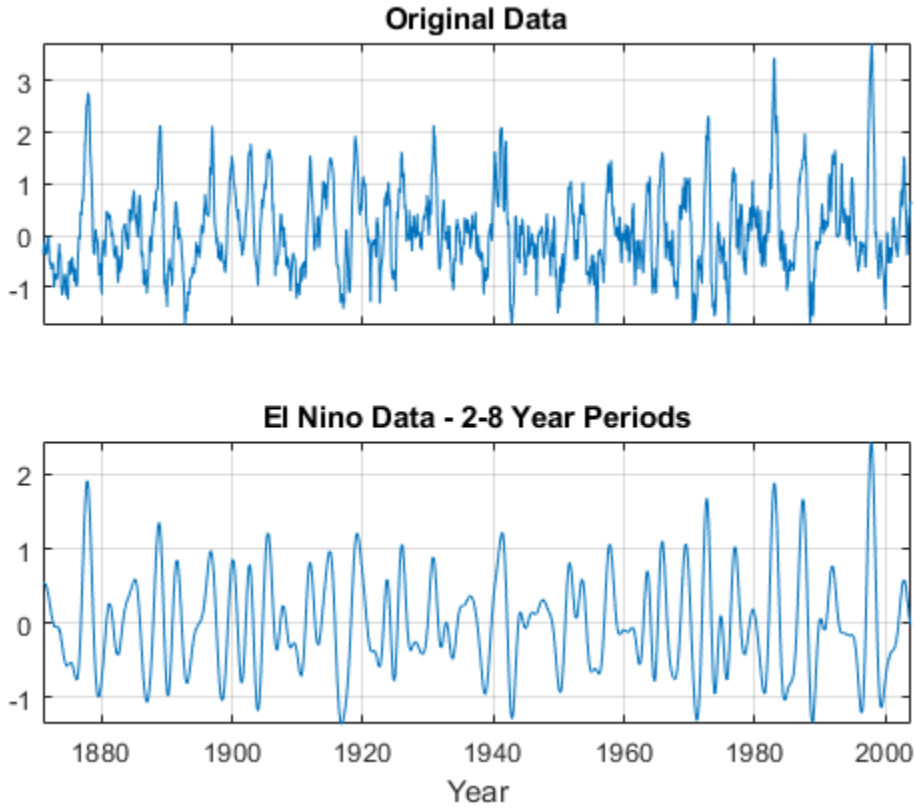

```
figure;
cwt(xrec,years(1/12)); title('Approximation Based on 2-8 Year Periods');
```



Compare the original data with the reconstructed data in time.

```
figure;
subplot(211)
plot(datayear,nino);
grid on;
ax = gca;
ax.XTickLabel = '';
axis tight;
title('Original Data');

subplot(212)
plot(datayear,xrec);
grid on;
```

```
axis tight;
xlabel('Year');
title('El Nino Data - 2-8 Year Periods');
```



Reconstruct Complex Data with Time-varying Trend

Add a trend to the continuous wavelet transform of a complex-valued dataset and reconstruct.

Obtain the CWT of the NPG2006 dataset.

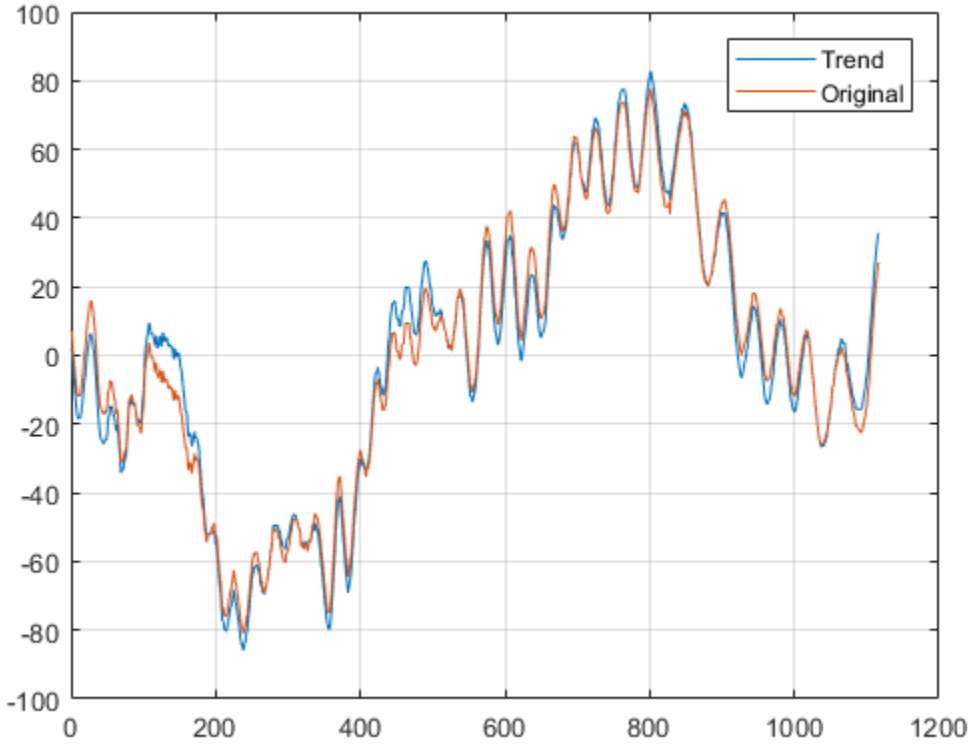
```
load npg2006.mat
wt = cwt(npg2006.cx);
```

Create a time-varying trend derived from the data.

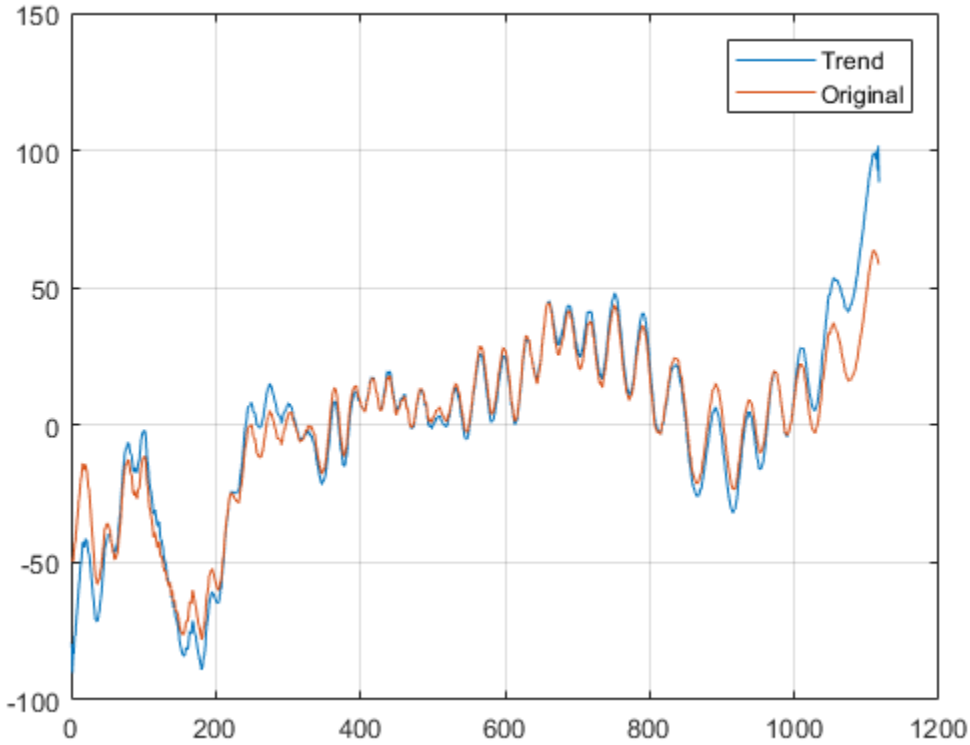
```
trend = smoothdata(npg2006.cx,'movmean',100);
```

Obtain the inverse CWT and add the trend. Plot the original data and the reconstructed data.

```
xrec = icwt(wt,'SignalMean',trend);
plot([real(xrec)' real(npg2006.cx)])
grid on
legend('Trend','Original')
```



```
figure
plot([imag(xrec)' imag(npg2006.cx)])
grid on
legend('Trend','Original')
```



Input Arguments

`wt` — Continuous wavelet transform coefficients

matrix

Continuous wavelet transform coefficients, specified as a matrix of complex values. `wt` is the output from the `cwt` function.

Data Types: `double`
Complex Number Support: Yes

`wname` — Analytic wavelet
`'morse'` (default) | `'amor'` | `'bump'`

Analytic wavelet used to invert the CWT, specified as `'morse'`, `'amor'`, or `'bump'`. These character vectors specify the analytic Morse, Morlet, and bump wavelet, respectively. The specified wavelet must be the same type of wavelet used to obtain the `cwt`.

The default Morse wavelet uses a default symmetry parameter, γ, that is 3 and has a default time bandwidth of 60.

`f` — CWT frequencies
vector

CWT frequencies, specified as a vector. The number of elements in the frequency vector must equal to the number of rows in the input CWT coefficient matrix, `wt`. If you specify `f`, you must also specify `freqrange`.

`freqrange` — Frequency range
two-element vector | 2-by-2 matrix

Frequency range for which to return inverse continuous wavelet transform values, specified as a two-element vector or 2-by-2 matrix. If `wt` is a 2-D matrix, `freqrange` must be a two-element vector. If `wt` is a 3-D matrix, `freqrange` can be a two-element vector or a 2-by-2 matrix. If `wt` is a 3-D matrix and `freqrange` is a vector, inversion is performed over the same frequency range in both the positive (analytic) and negative (anti-analytic) components of `wt`. If `freqrange` is a 2-by-2 matrix, the first row contains the frequency range for the positive part of `wt` (first page) and the second row contains the frequency range for the negative part of `wt` (second page). For a vector, the elements of `freqrange` must be strictly increasing and contained in the range of the frequency vector `f`. For a matrix, each row of `freqrange` must be strictly increasing and contained in the range of `f`. `f` is the scale-to-frequency conversion obtained in CWT. For the inversion of a complex-valued signal, you can specify one row of `freqrange` as a vector of zeros. If the first row of `freqrange` is a vector of zeros, only the negative (anti-analytic part) is used in the inversion. If you specify `freqrange`, you must also specify `f`.

For example `[0 0; 1/10 1/4]` inverts the negative (clockwise) component over the frequency range `[1/10 1/4]`. The positive (counterclockwise) component is first set to

all zeros before performing the inversion. Similarly, `[1/10 1/4; 0 0]` inverts the CWT by selecting the frequency range `[1/10 1/4]` from the positive (counterclockwise) component and setting the negative component to all zeros.

`period` — Time periods

vector

Time periods corresponding to the rows of CWT coefficient matrix `wt`, specified as a vector. `period` is the output of `cwt`, when the CWT is obtained using a `duration` input.

`periodrange` — Period range

two-element vector | 2-by-2 matrix

Period range for which to return inverse continuous wavelet transform values, specified as a two-element vector or 2-by-2 matrix. If `wt` is a 2-D matrix, `periodrange` must be a two-element vector of durations. If `wt` is a 3-D matrix, `periodrange` can be a two-element vector of durations or 2-by-2 matrix of durations. If `periodrange` is a vector of durations and `wt` is a 3-Dmatrix, inversion is performed over the same frequency range in both the positive (analytic) and negative (anti-analytic) components of `wt`. If `periodrange` is a 2-by-2 matrix of durations, the first row contains the period range for the positive part of `wt` (first page) and the second row contains the period range for the negative part of `wt` (second page). For a vector, the elements of `periodrange` must be strictly increasing and contained in the range of the period vector `period`. The elements of `periodrange` and `period` must have the same units. For a matrix, each row of `periodrange` must be strictly increasing and contained in the range of the period vector P. For the inversion of a complex-valued signal, you can specify one row of `periodrange` as a vector of zero durations. If the first row of `periodrange` is a vector of zero durations, only the negative (anti-analytic part) is used in the inversion. If you specify `periodrange`, you must also specify `period`.

For example `[seconds(0) seconds(0); seconds(1/10) seconds(1/4)]` inverts the negative(clockwise) component over the period range `[seconds(1/10) seconds(1/4)]`. The positive (counterclockwise) component is first set to all zeros before performing the inversion. Similarly, `[seconds(1/10) seconds(1/4); seconds(0) seconds(0)]` inverts the CWT by selecting the period range `[1/10 1/4]` from the positive (counterclockwise) component and setting the negative component to all zeros.

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: `'TimeBandwidth',45` sets the time bandwidth to 45.

`TimeBandwidth` — Time-bandwidth of Morse wavelet
`60` (default) | scalar greater than 3 and less than or equal to 120

Time bandwidth of the Morse wavelet, specified as a comma-separated pair consisting of `'TimeBandwidth'` and a scalar greater than 3 and less than or equal to 120. The specified time bandwidth must be the same as the time bandwidth used in the `cwt`. The symmetry of the Morse wavelet is assumed to be 3.

You cannot specify both the `'TimeBandwidth'` and `'WaveletParameters'`. If you specify `'TimeBandwidth'` , you cannot specify `'WaveletParameters'`. To specify both the symmetry and time bandwidth, use `'WaveletParameters'` instead.

`WaveletParameters` — Symmetry and time bandwidth of Morse wavelet
`(3,60)` (default) | two-element vector of scalars

Symmetry and time bandwidth of Morse wavelet, specified as the comma-separated pair consisting of `'WaveletParameters'` and a two-element vector of scalars. The first element of the vector is the symmetry, γ, and the second element is the time-bandwidth. The specified wavelet parameters must be the same as the parameters used in `cwt`.

You cannot specify both `'WaveletParameters'` and `'TimeBandwidth'`. If you specify `'WaveletParameters'`, you cannot specify `'TimeBandwidth'`. To specify the time bandwidth and use the default symmetry value of 3, use `'TimeBandwidth'` instead.

`SignalMean` — Signal mean
scalar | vector

Signal mean to add to the `icwt` output, specified as the comma-separated pair consisting of `'SignalMean'` and a scalar or vector. If signal mean is a vector, it must be the same length as the column size of the wavelet coefficient matrix. If `wt` is a 2-D matrix, the signal mean must be a real-valued scalar or vector. If `wt` is a 3-D matrix, the signal mean

must be a complex-valued scalar or vector. Because `cwt` does not preserve the signal mean, `icwt` does not contain the signal mean by default. Adding a non-zero signal mean to a frequency- or period-limited reconstruction adds a zero-frequency component to the reconstruction.

`VoicesPerOctave` — Number of voices per octave
`10` (default) | even integer from 4 to 48

Number of voices per octave, specified as the comma-separated pair consisting of `'VoicesPerOctave'` and an even integer from 4 to 48. The CWT scales are discretized using the specified number of voices per octave. The number of voices per octave must be the same as the number of voices per octave used to obtain the CWT. You cannot specify the number of voices per octave if you specify either the frequency, `f`, or duration, `period`.

Output Arguments

`xrec` — Inverse 1-D continuous wavelet transform
row vector

Inverse 1-D continuous wavelet transform, returned as a row vector.

Data Types: `double`

References

[1] Lilly, J. M., and S. C. Olhede. "Generalized Morse Wavelets as a Superfamily of Analytic Wavelets." *IEEE Transactions on Signal Processing*. Vol. 60, No. 11, 2012, pp. 6036–6041.

[2] Lilly, J. M., and S. C. Olhede. "Higher-Order Properties of Analytic Wavelets." *IEEE Transactions on Signal Processing*. Vol. 57, No. 1, 2009, pp. 146–160.

[3] Lilly, J. M. *jLab: A data analysis package for Matlab*, version 1.6.2. 2016. http://www.jmlilly.net/jmlsoft.html.

See Also

`cwt` | `duration` | `dwt` | `wavedec` | `wavefun` | `waveinfo` | `wcodemat` | `wcoherence` | `wsst`

Topics

"Morse Wavelets"
"Continuous and Discrete Wavelet Transforms"
"1-D Continuous Wavelet Analysis"
"Time-Frequency Analysis with the Continuous Wavelet Transform"

Introduced in R2016b

icwtft

Inverse CWT

Note This function is no longer recommended. Use `icwt` instead.

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.

- `cfs` — CWT coefficient matrix
- `scales` — Vector of scales

- `frequencies` — frequencies in cycles per unit time (or space) corresponding to the scales. If the sampling period units are seconds, the frequencies are in hertz. The elements of frequencies are in decreasing order to correspond to the elements in the scales vector.
- `omega` — Angular frequencies used in the Fourier transform
- `meanSig` — Mean of the analyzed signal
- `dt` — The sampling period
- `wav` — Analyzing wavelet used in the CWT with parameters specified

`cwtstruct` is the output of `cwtft`.

Output Arguments

`xrec`

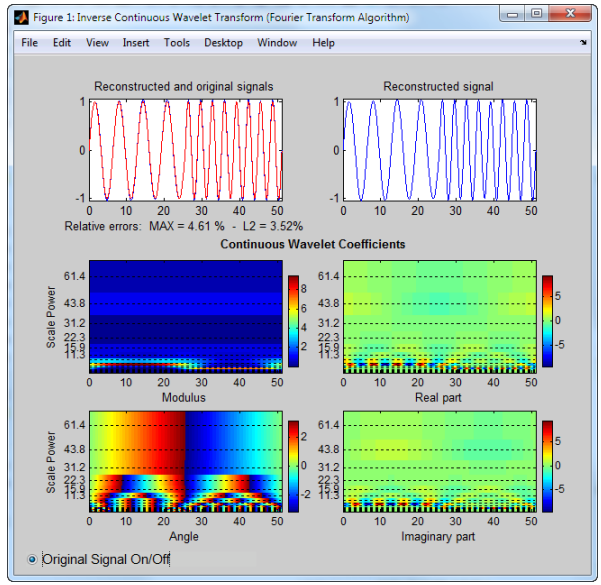
Reconstructed signal

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');
```

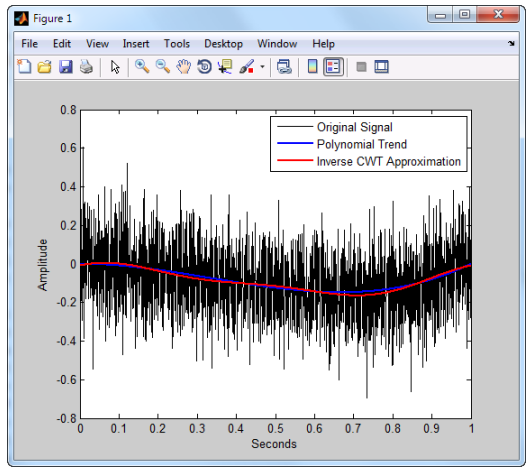
Select the radio button in the bottom left corner of the plot.

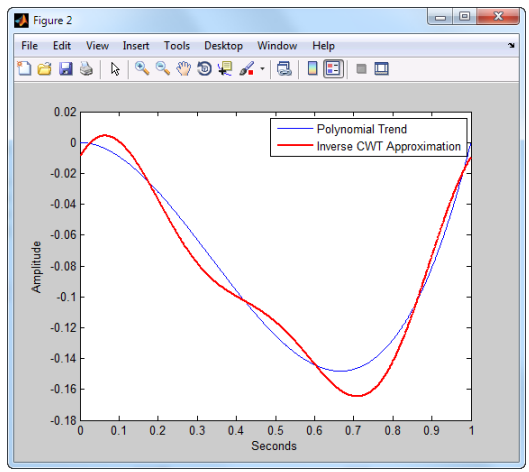


Use the inverse CWT to approximate a trend in a time series. Construct a time series consisting of a polynomial trend, a sinewave (oscillatory component), and additive white Gaussian noise. Obtain the CWT of the input signal and use the inverse CWT based on only the coarsest scales to reconstruct an 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');
```

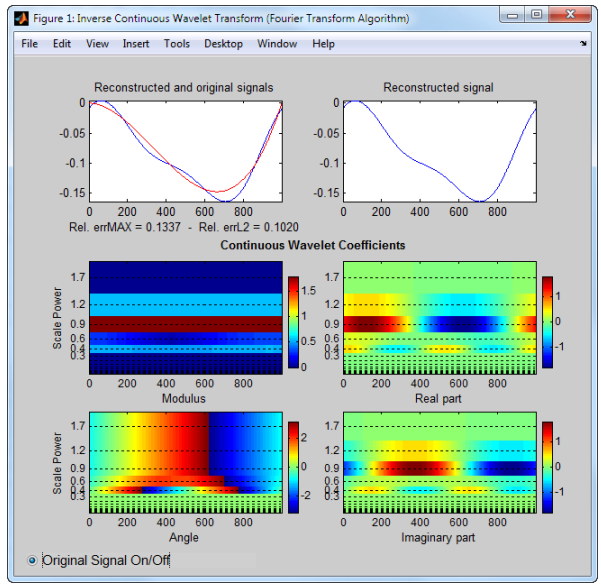
```
plot(t,x,'b','linewidth',2);
plot(t,xrec,'r','linewidth',2);
legend('Original Signal','Polynomial Trend',...
    'Inverse CWT Approximation');
figure
plot(t,x,'b'); hold on;
xlabel('Seconds'); ylabel('Amplitude');
plot(t,xrec,'r','linewidth',2);
legend('Polynomial Trend','Inverse CWT Approximation');
```





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');
```



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]

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` | `icwtlin`

Topics

“Continuous and Discrete Wavelet Transforms”
“Continuous Wavelet Transform and Scale-Based Analysis”
“Inverse Continuous Wavelet Transform”

Introduced in R2011a

icwtlin

Inverse continuous wavelet transform (CWT) for linearly spaced scales

Note This function is no longer recommended. Use `icwt` instead.

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 1-308 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 seven 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.
- `frequencies` — frequencies in cycles per unit time (or space) corresponding to the scales. If the sampling period units are seconds, the frequencies are in hertz. The elements of frequencies are in decreasing order to correspond to the elements in the scales vector.
- `omega` — Angular frequencies used in the Fourier transform in radians/sample
- `MeanSIG` — Signal mean
- `dt` — Sampling period in seconds
- `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. $m = 2$ is the Mexican-hat or Ricker wavelet.
  - `'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 $m = 2$. This wavelet is also known as the Ricker wavelet.
- `'paul'` — Paul wavelet
- `'bump'` — Bump wavelet

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.

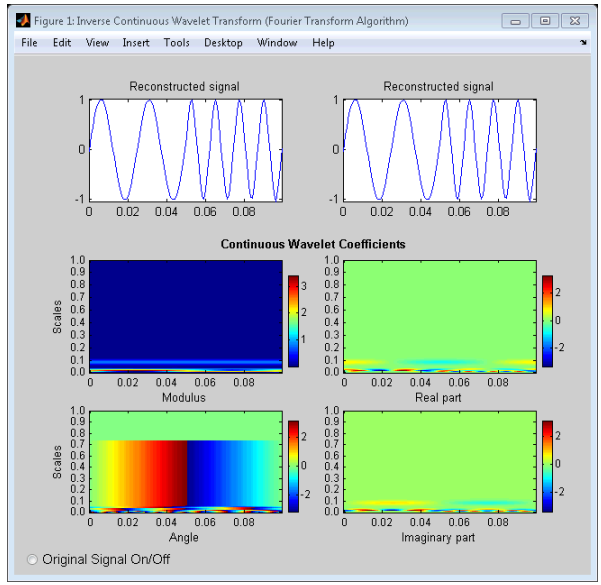
The purpose of the CWT inversion algorithm is not to produce a perfect reconstruction of the input signal. The inversion preserves time and scale-localized features in the reconstructed signal. The amplitude scaling in the reconstructed signal, however, can be

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');
```

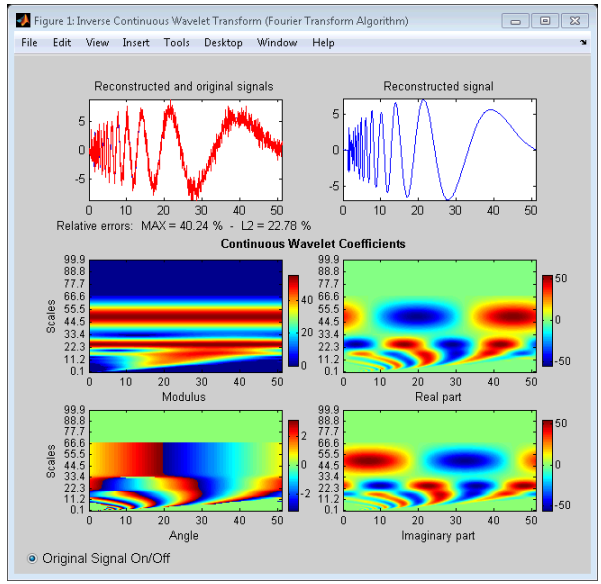


Reconstruct an approximation to a noisy Doppler signal based on thresholded coefficients. Use the universal threshold. Assume the sampling period is 0.05 seconds.

```
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.



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.

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.

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`

Topics

"Continuous and Discrete Wavelet Transforms"
"Continuous Wavelet Transform and Scale-Based Analysis"
"Inverse Continuous Wavelet Transform"

Introduced in R2011b

idddtree

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`.

Examples

Perfect Reconstruction Using 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))
```

```
ans = 1.9291e-12
```

- “Analytic Wavelets Using the Dual-Tree Wavelet Transform”

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`

See Also

`dddtree` | `dddtreecfs` | `plotdt`

Topics

“Analytic Wavelets Using the Dual-Tree Wavelet Transform”

“Critically Sampled and Oversampled Wavelet Filter Banks”

Introduced in R2013b

idddtree2

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`.

Examples

Perfect Reconstruction Using 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)))
```

```
ans = 7.3612e-12
```

- “Analytic Wavelets Using the Dual-Tree Wavelet Transform”

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`

See Also

`dddtree2` | `dddtreecfs`

Topics

“Analytic Wavelets Using the Dual-Tree Wavelet Transform”
“Critically Sampled and Oversampled Wavelet Filter Banks”

Introduced in R2013b

idualtree3

3-D dual-tree wavelet reconstruction

Syntax

```
xrec = idualtree3(a,d)
xrec = idualtree3(a,d,Name,Value)
```

Description

`xrec = idualtree3(a,d)` returns the inverse 3-D dual-tree complex wavelet transform of the final-level approximation coefficients, `a`, and cell array of wavelet coefficients, `d`.

`xrec = idualtree3(a,d,Name,Value)` specifies options using name-value pair arguments.

Examples

Wavelet Coefficients

Generate all-zero sets of scaling and wavelet coefficients by computing the 3-D dual-tree complex wavelet transform of an array of zeros.

```
zr = zeros(64,64,64);

[a,d] = dualtree3(zr,4);
```

Find the real (4,5) wavelet coefficient of the 19th subband of the third level by assigning 1 to the corresponding array element and inverting the transform.

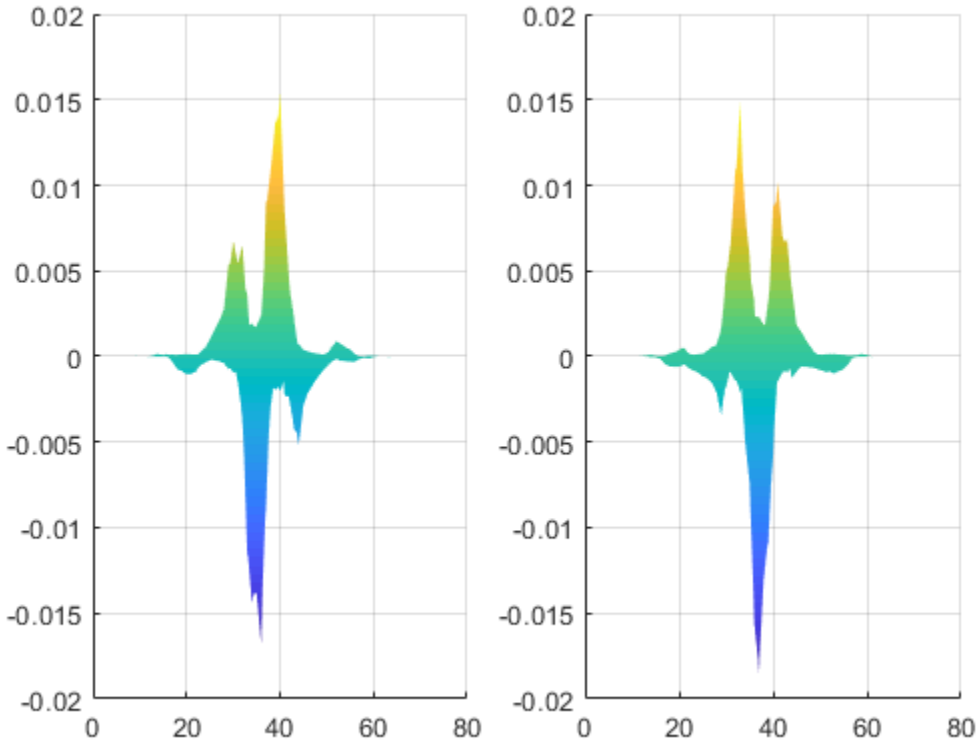
```
d{3}(4,5,19) = 1;

xr = idualtree3(a,d);
```

Find the corresponding imaginary coefficient assigning the imaginary unit to the array element and then inverting the transform.

```
[a,d] = dualtree3(zr,4);

d{3}(4,5,19) = 1j;

xi = idualtree3(a,d);
```

Display the 18th page of the real and imaginary reconstructions.

```
subplot(1,2,1)
surf(xr(:,:,18))
view(0,0)
zlim([-0.02 0.02])
shading interp

subplot(1,2,2)
surf(xi(:,:,18))
view(0,0)
zlim([-0.02 0.02])
shading interp
```



- Dual-Tree Wavelet Transforms

Input Arguments

`a` — Final-level scaling coefficients

real-valued matrix

Final-level scaling coefficients, specified as a real-valued matrix. `a` is an output of `dualtree3`.

Data Types: `single` | `double`

`d` — Wavelet coefficients

cell array

Wavelet coefficients, specified as a cell array. `d` is an output of `dualtree3`.

Data Types: `single` | `double`
Complex Number Support: Yes

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: `'LevelOneFilter','legall','FilterLength',6` inverts a transform using LeGall synthesis filters with scaling length 3 and wavelet length 5 at level 1, and length-6 Q-shift filters at levels 2 and greater.

`FilterLength` — Hilbert Q-shift filter-pair length

10 (default) | 6 | 14 | 16 | 18

Hilbert Q-shift filter-pair length, specified as the comma-separated pair consisting of `'FilterLength'` and one of 6, 10, 14, 16, or 18. The synthesis filters used by `idualtree3` must match the analysis filters used by `dualtree3`.

Data Types: `double` | `single`

`LevelOneFilter` — First-level biorthogonal analysis filter

`'nearsym5_7'` (default) | `'nearsym13_19'` | `'antonini'` | `'legall'`

First-level biorthogonal analysis filter, specified as the comma-separated pair consisting of `'LevelOneFilter'` and a character vector or string. By default, `idualtree3` uses the near-symmetric biorthogonal wavelet filter with lengths 7 (scaling synthesis filter) and 5 (wavelet synthesis filter) in the reconstruction.

Data Types: `char` | `string`

`OriginalDataSize` — Size of the original data

three-element vector of even integers

Size of the original data, specified as the comma-separated pair consisting of `'OriginalDataSize'` and a three-element vector of even integers. This vector must

match the size of the original input to the 3-D dual-tree wavelet transform. When the first-level wavelet coefficients are not available, the reconstructed data size can differ from the original input data size. If you call `dualtree3` with the `'excludeL1'` option, then `'OriginalDataSize'` adjusts the size of `xrec` to match the size of the original input data. If you do not use the `'excludeL1'` option, then this argument is ignored.

Data Types: `double` | `single`

Output Arguments

`xrec` — Inverse 3-D dual-tree complex wavelet transform

3-D array

Inverse 3-D dual-tree complex wavelet transform, returned as a 3-D array.

References

[1] Chen, H., and N. G. Kingsbury. “Efficient Registration of Nonrigid 3-D Bodies.” *IEEE Transactions on Image Processing*. Vol 21, January 2012, pp. 262–272.

[2] Kingsbury, N. G. “Complex Wavelets for Shift Invariant Analysis and Filtering of Signals.” *Journal of Applied and Computational Harmonic Analysis*. Vol. 10, May 2001, pp. 234–253.

See Also

`dddtree2` | `dualtree3` | `wavedec3` | `waverec3`

Topics

Dual-Tree Wavelet Transforms

Introduced in R2017a

idwt

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))
```

```
ans = 3.2156e-12
```

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))
```

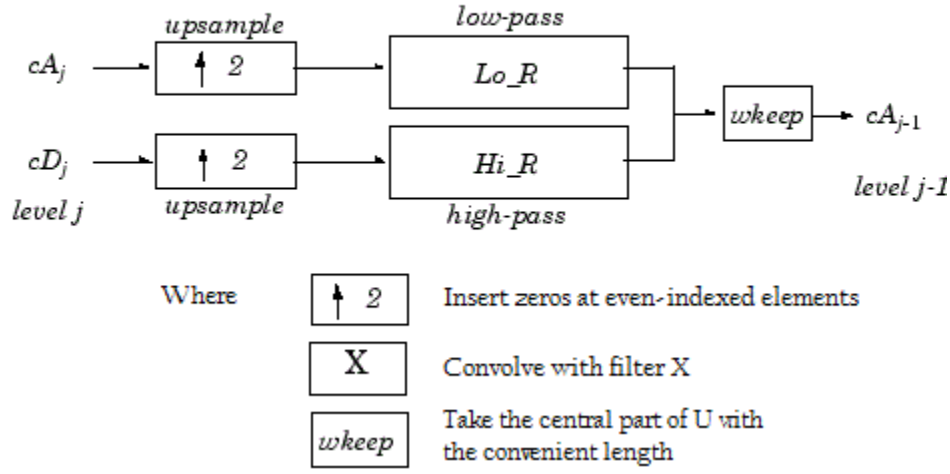
```
ans = 3.5527e-15
```

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.)

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`dwt` | `dwtmode` | `upwlev`

Introduced before R2006a

idwt2

Single-level inverse discrete 2-D wavelet transform

Syntax

```
X = idwt2(cA,cH,cV,cD,'wname')
X = idwt2(cA,cH,cV,cD,Lo_R,Hi_R)
X = idwt2(cA,cH,cV,cD,'wname',S)
X = idwt2(cA,cH,cV,cD,Lo_R,Hi_R,S)
idwt2(cA,cH,cV,cD,'wname')
X = idwt2(...,'mode',MODE)
X = idwt2(cA,[],[],[],...)
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`.

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
```

```
% 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
```

Tips

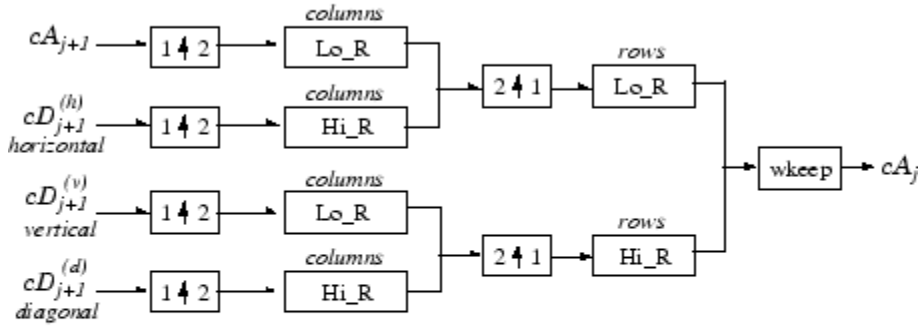
If cA,cH,cV,cD are obtained from an indexed image analysis or a truecolor image analysis, they are `m`-by-`n` matrices or `m`-by-`n`-by-3 arrays, respectively.

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

Algorithms

Two-Dimensional IDWT

Reconstruction step



Where

2 1	Upsample columns: insert zeros at odd-indexed columns.
1 2	Upsample rows: insert zeros at odd-indexed rows.
rows X	Convolve with filter X the rows of the entry.
columns X	Convolve with filter X the columns of the entry.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Variable-size data support must be enabled.

See Also

`dwt2` | `dwtmode` | `upwlev2`

Introduced before R2006a

idwt3

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 highpass filter.

- The char `'d'` (or `'h'` or `'D'` or `'H'`) which specifies the sum of all the components different from the lowpass component.

Examples

Single-Level Three-Dimensional Wavelet Reconstruction

Define the original 3-D data.

```
X  = reshape(1:64,4,4,4)
```

```
X =
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
```

Decompose `X` using `'db1'`.

```
wt = dwt3(X,'db1');
```

Reconstruct `X` from the coefficients. Verify that the reconstructed data agrees with the original data to machine precision.

```
XR = idwt3(wt);

dff = max(abs(X-XR))
```

```
dff =
dff(:,:,1) =

   1.0e-13 *

    0.0266    0.0355    0.0888    0.1066


dff(:,:,2) =

   1.0e-13 *

    0.1066    0.1066    0.2132    0.2132


dff(:,:,3) =

   1.0e-13 *

    0.1421    0.1421    0.2132    0.2132


dff(:,:,4) =

   1.0e-13 *

    0.3553    0.3553    0.2842    0.2842
```

Compute the reconstructed approximation, which consists of the lowpass component.

```
A  = idwt3(wt,'aaa');
```

Compute the sum of all the components different from the lowpass component.

```
D  = idwt3(wt,'d');
```

Reconstruct the component associated with lowpass in the *x* and *z* directions and highpass in the *y* direction.

```
ADA  = idwt3(wt,'ada');
```

See Also

dwt3 | wavedec3 | waverec3

Introduced in R2010a

ihaart

Inverse 1-D Haar wavelet transform

Syntax

```
xrec = ihaart(a,d)
xrec = ihaart(a,d,level)
xrec = ihaart(___,integerflag)
```

Description

`xrec = ihaart(a,d)` returns the inverse 1-D Haar transform, `xrec`, for the approximation coefficients, `a`, and the wavelet coefficients, `d`. Both `a` and `d` are obtained from `haart`.

`xrec = ihaart(a,d,level)` returns the inverse 1-D Haar transform at the specified level.

`xrec = ihaart(___,integerflag)` specifies how the inverse 1-D Haar transform handles integer-valued data, using any of the previous syntaxes.

Examples

Inverse Haar Transform of Noisy Data

Obtain the Haar and inverse Haar transforms of noisy data.

Load the noisy data signal

```
load noisdopp;
```

Obtain the Haar transform of the noisy signal.

```
[a,d] = haart(noisdopp);
```

Reconstruct the data by inverting the Haar transform.

```
xrec = ihaart(a,d);
```

Compare the original and reconstructed data by determining the maximum difference between them. The difference is essentially zero, which indicates a near-perfect reconstruction.

```
max(abs(xrec-noisdopp'))
```

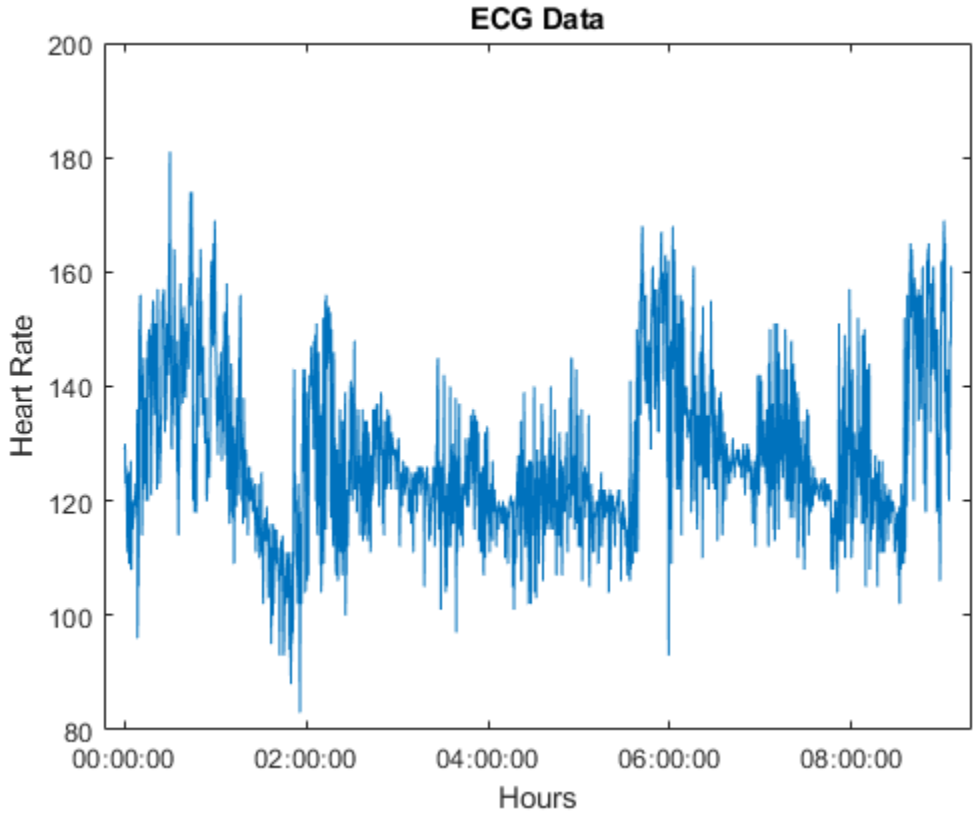
```
ans = 4.4409e-15
```

Inverse Haar Transform of ECG Data

Obtain the Haar transform and inverse Haar transform of ECG heart rate data.

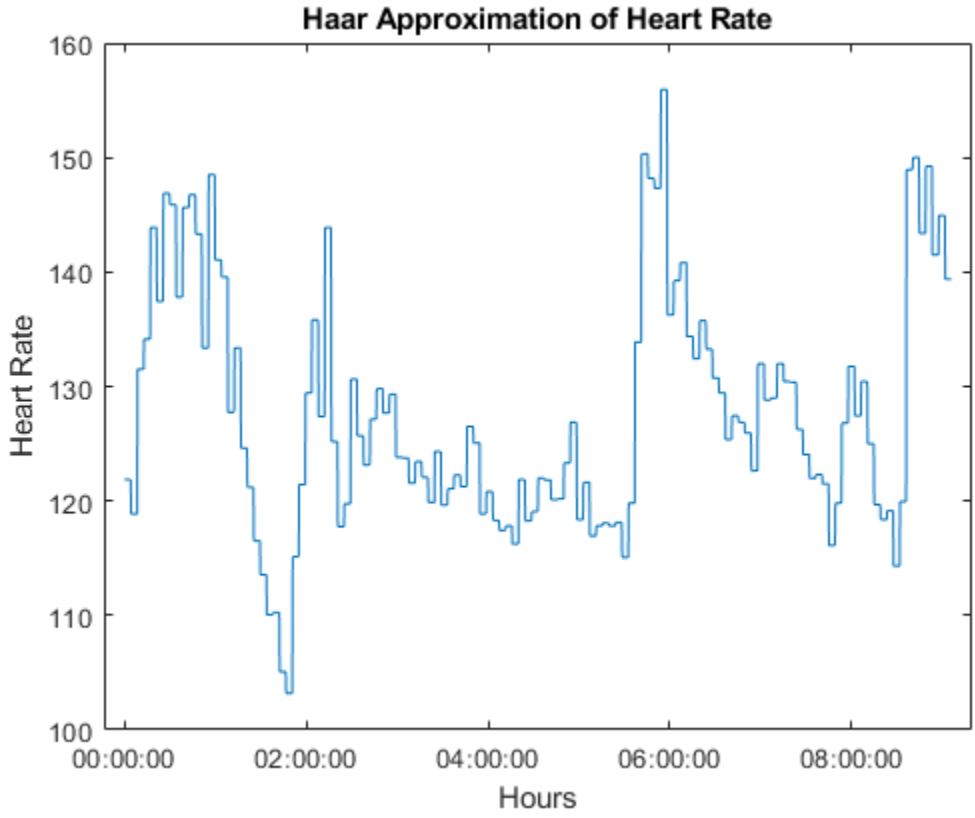
Load and plot the ECG data.

```
load BabyECGData;
plot(times,HR)
xlabel('Hours')
ylabel('Heart Rate')
title('ECG Data')
```



Obtain the Haar transform and inverse Haar transform. Compare the reconstructed data at level 4 to the original data.

```
[a,d] = haart(HR);
HaarHR = ihaart(a,d,4);
figure
plot(times,HaarHR)
xlabel('Hours')
ylabel('Heart Rate')
title('Haar Approximation of Heart Rate')
```



Inverse Haar Transform of Integer Data

Obtain the Haar and inverse Haar transforms for a series of random integers.

Create the series.

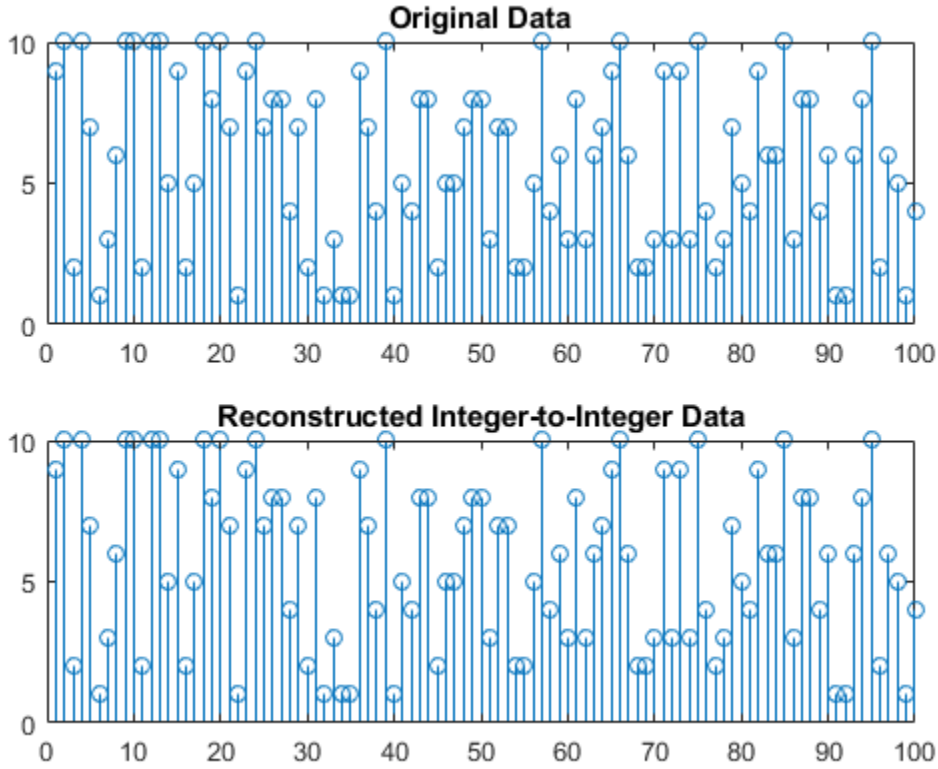
```
x = randi(10,100,1);
```

Obtain the Haar and inverse Haar transforms.

```
[a,d] = haart(x,'integer');
xrec = ihaart(a,d,'integer');
```

Plot and compare the original and reconstructed data.

```
subplot(2,1,1)
stem(x); title('Original Data')
subplot(2,1,2)
stem(xrec)
title('Reconstructed Integer-to-Integer Data')
```



Determine the maximum difference between the original data values and the reconstructed values. The difference is zero, which indicates perfect reconstruction.

```
max(abs(x(:)-xrec(:)))
```

```
ans = 0
```

- “Haar Transforms for Time Series Data and Images”

Input Arguments

`a` — Approximation coefficients

scalar | vector | matrix

Approximation coefficients, specified as a scalar, vector, or matrix of coefficients, depending on the level to which the Haar transform was calculated. `a` is an output from the `haart` function.

Approximation, or scaling, coefficients are a lowpass representation of the input. At each level the approximation coefficients are divided into coarser approximation and detail coefficients.

Data Types: `double`

`d` — Detail coefficients

scalar | vector | matrix | cell array

Detail coefficients, specified as a scalar, vector, matrix, or cell array of wavelet coefficients, which are a highpass representation of the input. `d` is an output from the `haart` function. The number of detail coefficients depends on the selected level and the length of the input. The order of the elements of `d` is from fine to coarse resolution levels.

If `d` is a cell array, it can contain scalars, vectors, or matrices. The level of the Haar transform equals the number of elements in `d`. The coarsest resolution level element of the `d` cell array is a scalar value.

If `d` is a vector or matrix, the Haar transform was computed only down to one level coarser in resolution. If you specify only two levels, the detail coefficient is a scalar.

If `a` and the elements of `d` are vectors, `xrec` is a vector. If `a` and the elements of `d` are matrices, `xrec` is a matrix, where each column is the inverse Haar transform of the corresponding columns in `a` and `d`.

Data Types: `double`

`level` — Maximum level

`0` (default) | nonnegative integer

Maximum level to which to invert the Haar transform, specified as a nonnegative integer. If `d` is a cell array, `level` is less than or equal to `length(d)-1`. If `d` is a vector or matrix, `level` must equal `0` or be unspecified. The level must be less than the level used to obtain `a` and `d` from `haart`.

`integerflag` — Integer-valued data handling
`'noninteger'` (default) | `'integer'`

Integer-valued data handling, specified as a character vector. `'noninteger'` does not preserve integer-valued data, and `'integer'` preserves it. The `'integer'` option applies only if all elements of `a` and `d` are integer-valued. You must have used `'integer'` with `haart` to obtain integer-valued `a` and `d` inputs. The inverse 1-D Haar transform algorithm, however, uses floating-point arithmetic.

Output Arguments

`xrec` — Inverse 1-D Haar wavelet transform
vector | matrix

Inverse 1-D Haar wavelet transform, returned as a vector or matrix. If `a` and the elements of `d` are vectors, `xrec` is a vector. If `a` and the elements of `d` are matrices, `xrec` is a matrix, where each column is the inverse 1-D Haar transform of the corresponding columns in `a` and `d`.

Data Types: `double`

See Also
`haart` | `haart2` | `ihaart2`

Topics
“Haar Transforms for Time Series Data and Images”

Introduced in R2016b

ihaart2

Inverse 2-D Haar wavelet transform

Syntax

```
xrec = ihaart2(a,h,v,d)
xrec = ihaart2(a,h,v,d,level)
xrec = ihaart2(___,integerflag)
```

Description

`xrec = ihaart2(a,h,v,d)` returns the inverse 2-D Haar transform, `xrec`, for the approximation coefficients, `a`, and the horizontal, vertical, and diagonal detail coefficients, `h`, `v`, and `d`. All the inputs, `a`, `h`, `v`, and `d`, are outputs of `haart2`.

`xrec = ihaart2(a,h,v,d,level)` returns the inverse 2-D Haar transform at the specified level.

`xrec = ihaart2(___,integerflag)` specifies how the inverse 2-D Haar transform handles integer-valued data, using any of the previous syntaxes.

Examples

Inverse 2-D Haar Transform of an Image

Obtain the inverse 2-D Haar transform of image and view the reconstructed image.

Load the image and obtain its 2-D Haar transform.

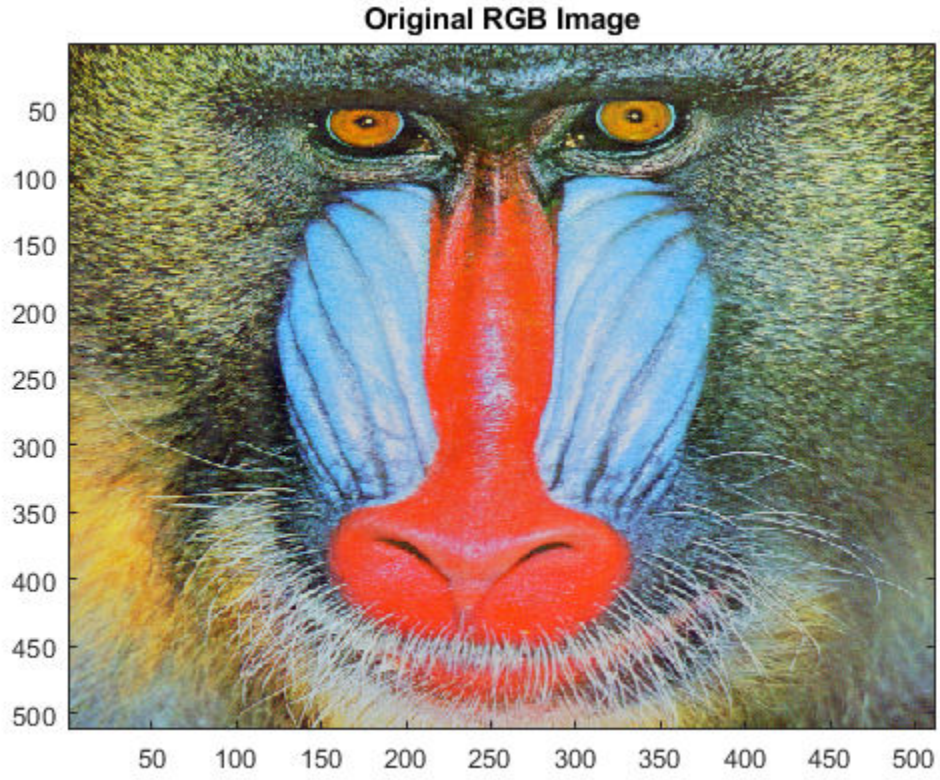
```
im = imread('mandrill.png');
[a,h,v,d] = haart2(im);
```

Use the inverse 2-D Haar transform to reconstruct the image.

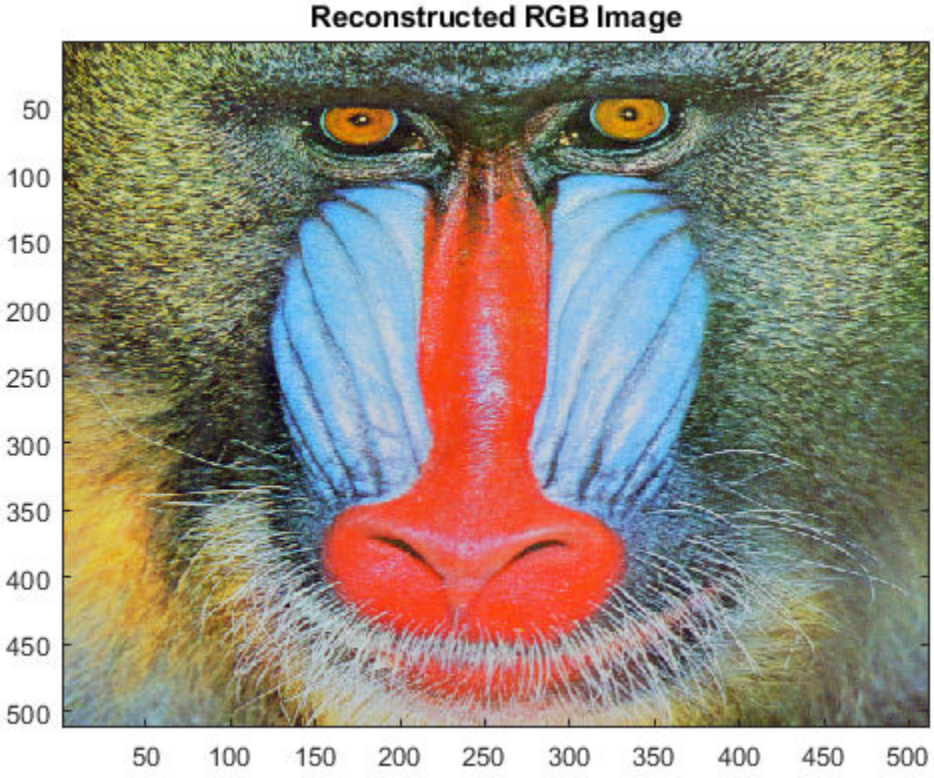
```
xrec = ihaart2(a,h,v,d);
```

Compare the original and reconstructed images.

```
imagesc(im)
title('Original RGB Image')
```




```
figure
imagesc(uint8(xrec))
title('Reconstructed RGB Image')
```

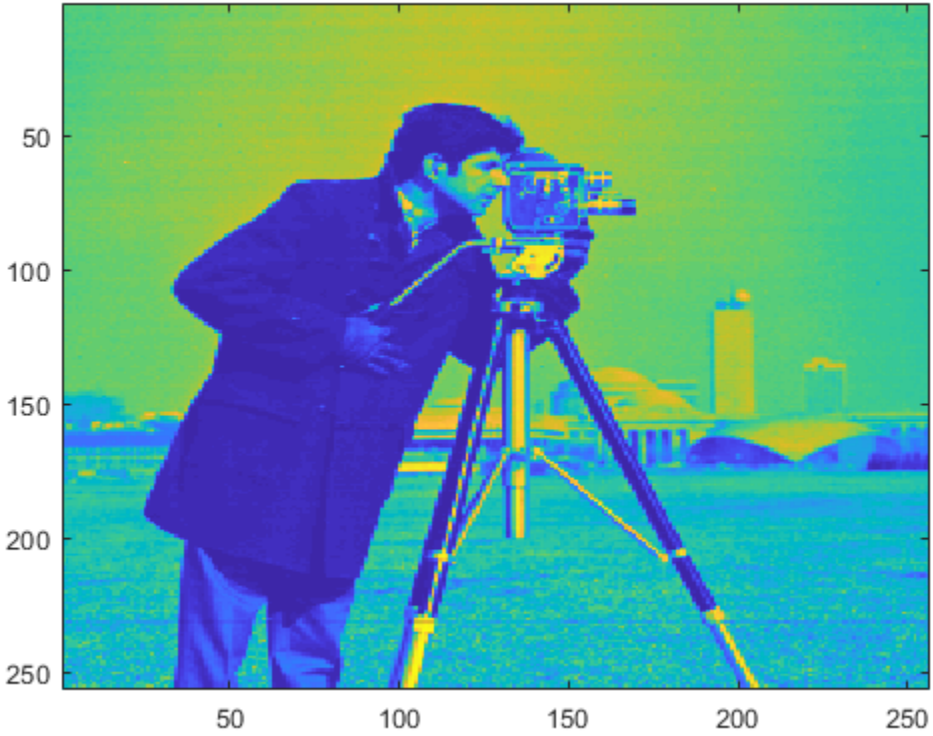


Inverse 2-D Haar Transform of Image Limited to Specified Level

Obtain the 2-D Haar transform of an image limiting the transform to 2 levels.

Load and view the image of a cameraman.

```
im = imread('cameraman.tif');
imagesc(im)
```

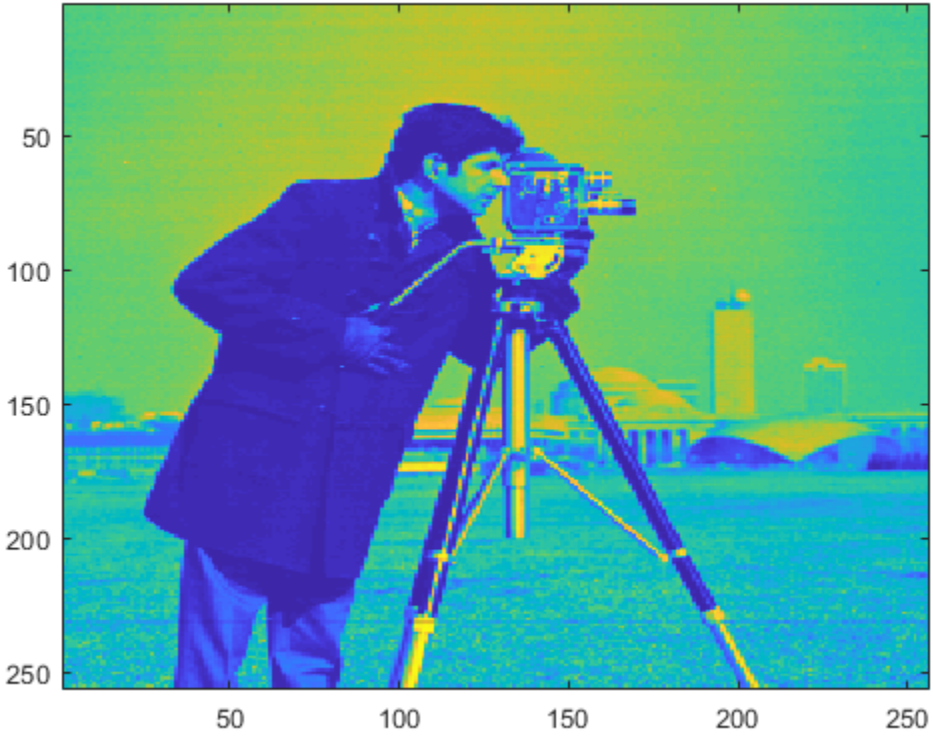


Obtain the 2-D Haar transform using the default maximum number of levels.

```
[a,h,v,d] = haart2(im);
```

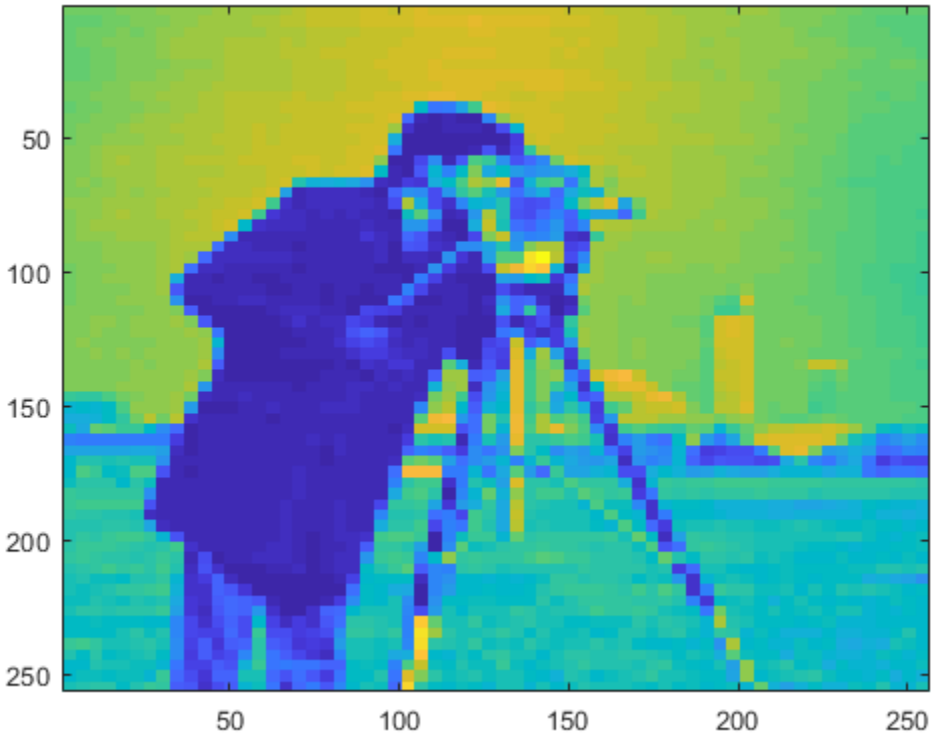
Reconstruct the image using the inverse 2-D Haar transform and view the image. Notice the near-perfect reconstruction.

```
xrec = ihaart2(a,h,v,d);
imagesc(xrec)
```



Reconstruct and view the image using the inverse 2-D Haar transform, limited to level 2. Level 2 corresponds to the fourth scale because scale is defined as 2^j, where j is the level.

```
xrec1 = ihaart2(a,h,v,d,2);
imagesc(xrec1)
```



Using fewer levels returns the average of the original image at level 2.

Inverse 2-D Haar Transform of Image Limited to Integer Data

Obtain the 2-D Haar transform of an image limiting the transform to integer data.

Load the image of a cameraman.

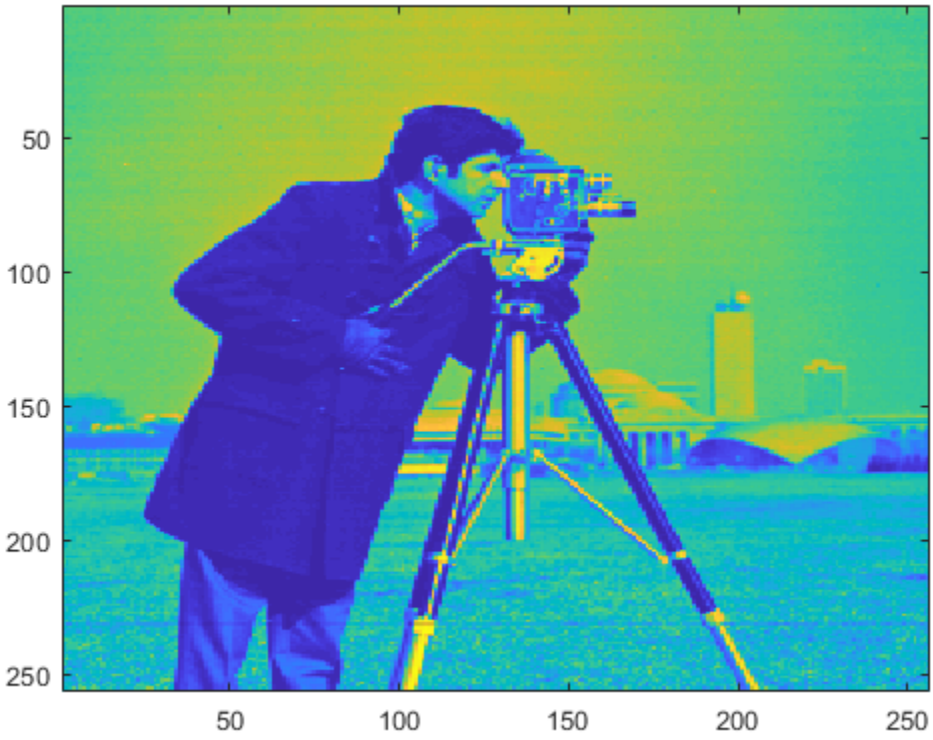
```
im = imread('cameraman.tif');
```

Obtain the 2-D Haar transform using the `'integer'` flag.

```
[a,h,v,d]=haart2(im,'integer');
```

Reconstruct the image using the inverse 2-D Haar transform and view the image.

```
xrec = ihaart2(a,h,v,d,'integer');
imagesc(xrec)
```



Use integer data when you need to reduce the amount of memory used compared to noninteger data.

- “Haar Transforms for Time Series Data and Images”

Input Arguments

`a` — Approximation coefficients

scalar | matrix

Approximation coefficients, specified as a scalar or matrix of coefficients, depending on the level to which the 2-D Haar transform was calculated. `a` is an output from the `haart2` function. Approximation, or scaling, coefficients are a lowpass representation of the input. If `a` and the elements of `h`, `v`, and `d`, are vectors, `xrec` is a vector. If `a` and the elements of `h`, `v`, and `d` are matrices, `xrec` is a matrix, where each column is the inverse 2-D Haar transform of the corresponding columns in `a` and `h`, `v`, or `d`.

Data Types: `double`

`h` — Horizontal detail coefficients

matrix | cell array

Horizontal detail coefficients by level, specified as a matrix or cell array of matrices. `h` is an output from the `haart2` function. If `h` is a matrix, the 2-D Haar transform was computed only down to one level coarser in resolution.

Data Types: `double`

`v` — Vertical detail coefficients

matrix or | cell array

Vertical detail coefficients by level, specified as a matrix or cell array of matrices. `v` is an output from the `haart2` function. If `v` is a matrix, the 2-D Haar transform was computed only down to one level coarser in resolution.

Data Types: `double`

`d` — Diagonal detail coefficients

matrix or | cell array

Diagonal detail coefficients by level, specified as a matrix or cell array of matrices. `d` is an output from the `haart2` function. If `d` is a matrix, the 2-D Haar transform was computed only down to one level coarser in resolution.

Data Types: `double`

`level` — Maximum level

`0` (default) | nonnegative integer

Maximum level to which to invert the Haar transform, specified as a nonnegative integer. If `h` is a cell array, `level` is less than or equal to `length(h)-1`. If `h` is a vector or matrix, `level` must equal `0` or be unspecified.

`integerflag` — Integer-valued data handling

`'noninteger'` (default) | `'integer'`

Integer-valued data handling, specified as a character vector. `'noninteger'` does not preserve integer-valued data in the 2-D Haar transform, and `'integer'` preserves it. The `'integer'` option applies only if all elements of inputs, `a`, `h`, `v`, and `d`, are integer-valued. The inverse 2-D Haar transform algorithm, however, uses floating-point arithmetic.

Output Arguments

`xrec` — Inverse 2-D Haar wavelet transform

matrix

2-D Haar wavelet transform, returned as a matrix.

Data Types: `double`

See Also

`haart` | `haart2` | `ihaart`

Topics

“Haar Transforms for Time Series Data and Images”

Introduced in R2016b

ilwt

Inverse 1-D lifting wavelet transform

Syntax

```
X = ilwt(AD_In_Place,W)
X = ilwt(CA,CD,W)
X = ilwt(AD_In_Place,W,LEVEL)
X = ILWT(CA,CD,W,LEVEL)
X = ilwt(AD_In_Place,W,LEVEL,'typeDEC',typeDEC)
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.

`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`).

`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.

`X = ilwt(AD_In_Place,W,LEVEL)` or `X = ILWT(CA,CD,W,LEVEL)` computes the lifting wavelet reconstruction, at level `LEVEL`.

`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`.

Instead of a lifted wavelet name, you may use the associated lifting scheme `LS`: `X = ilwt(...,LS,...)` instead of `X = ILWT(...,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);

% 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

Introduced before R2006a

ilwt2

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`.

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
```

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.

See Also

`lwt2`

Introduced before R2006a

imlpt

Inverse multiscale local 1-D polynomial transform

Syntax

```
y = imlpt(coefs,T,coefsPerLevel,scalingMoments)
y = imlpt(___,Name,Value)
```

Description

`y = imlpt(coefs,T,coefsPerLevel,scalingMoments)` returns the inverse multiscale local polynomial 1-D transform (MLPT) of `coefs`. The inputs to `imlpt` must be the outputs of `mlpt`.

`y = imlpt(___,Name,Value)` specifies `mlpt` properties using one or more `Name,Value` pair arguments and the input arguments from the previous syntax.

Examples

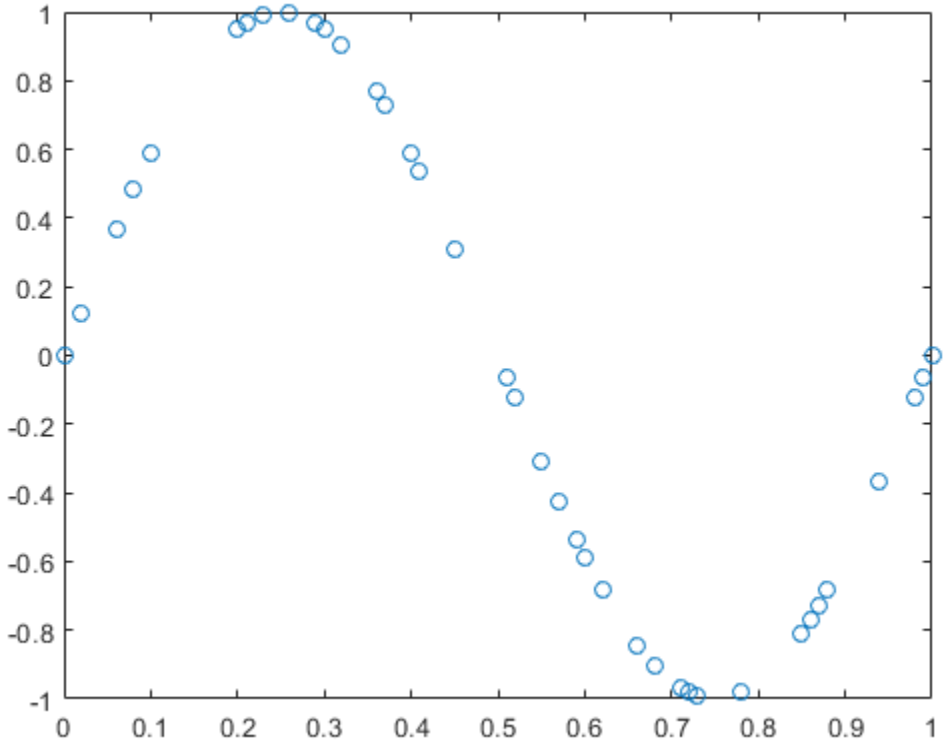
Multiscale Local 1-D Polynomial Transform and Inverse Trasform

Create a signal with nonuniform sampling and verify good reconstruction when performing the `mlpt` and `imlpt`.

Create and plot a sine wave with non-uniform sampling.

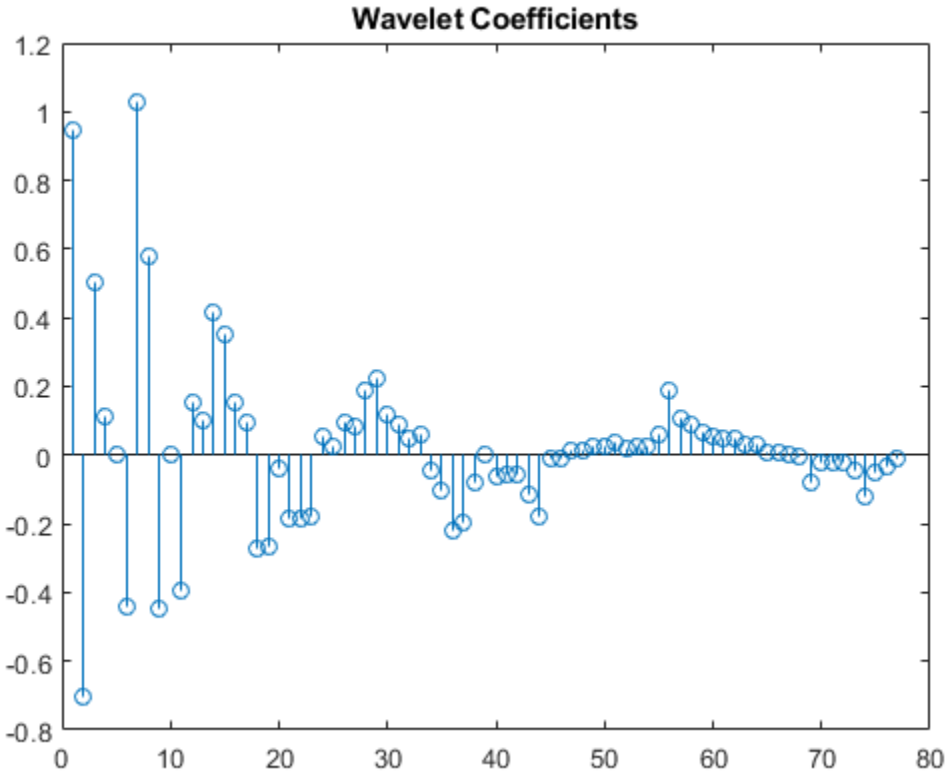
```
timeVector = 0:0.01:1;
sineWave = sin(2*pi*timeVector)';

samplesToErase = randi(100,100,1);
sineWave(samplesToErase) = [];
timeVector(samplesToErase) = [];

figure(1)
```

```
plot(timeVector,sineWave,'o')
hold on
```



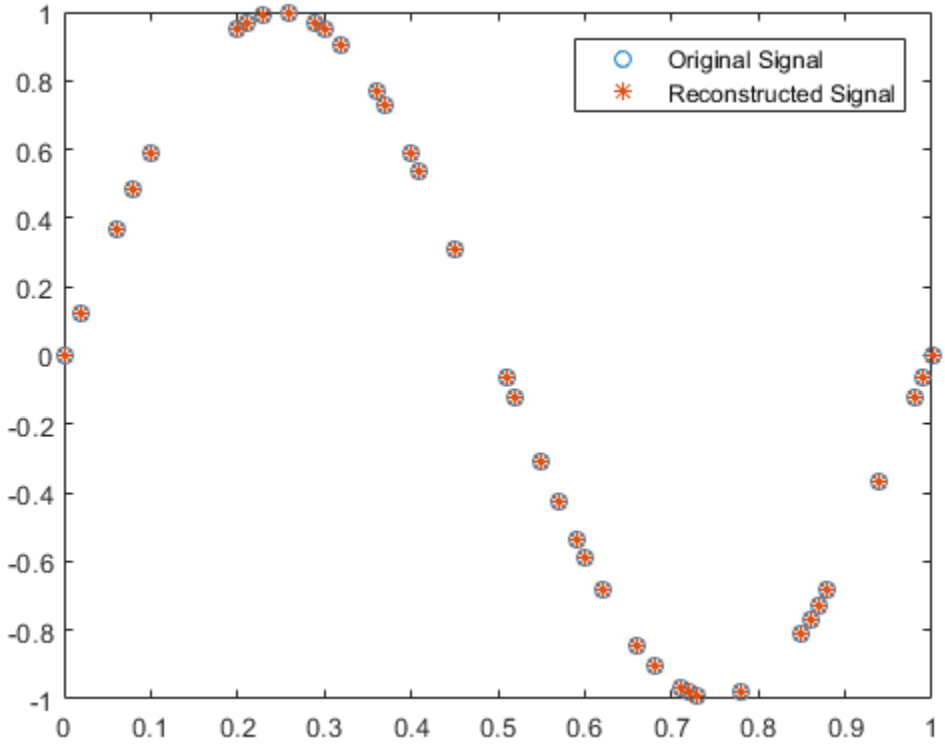
Perform the multiscale local 1-D polynomial trasform (`mlpt`) on the signal. Visualize the coefficients.

```
[coefs,T,coefsPerLevel,scalingMoments] = mlpt(sineWave,timeVector);

figure(2)
stem(coefs)
title('Wavelet Coefficients')
```



Perform the inverse multiscale local 1-D polynomial trasform (`imlpt`) on the coefficients. Visualize the reconstructed signal.

```
y = imlpt(coefs,T,coefsPerLevel,scalingMoments);

figure(1)
plot(T,y,'*')
legend('Original Signal','Reconstructed Signal')
hold off
```



Look at the total error to verify good reconstruction.

```
reconstructionError = sum(abs(y-sineWave))
```

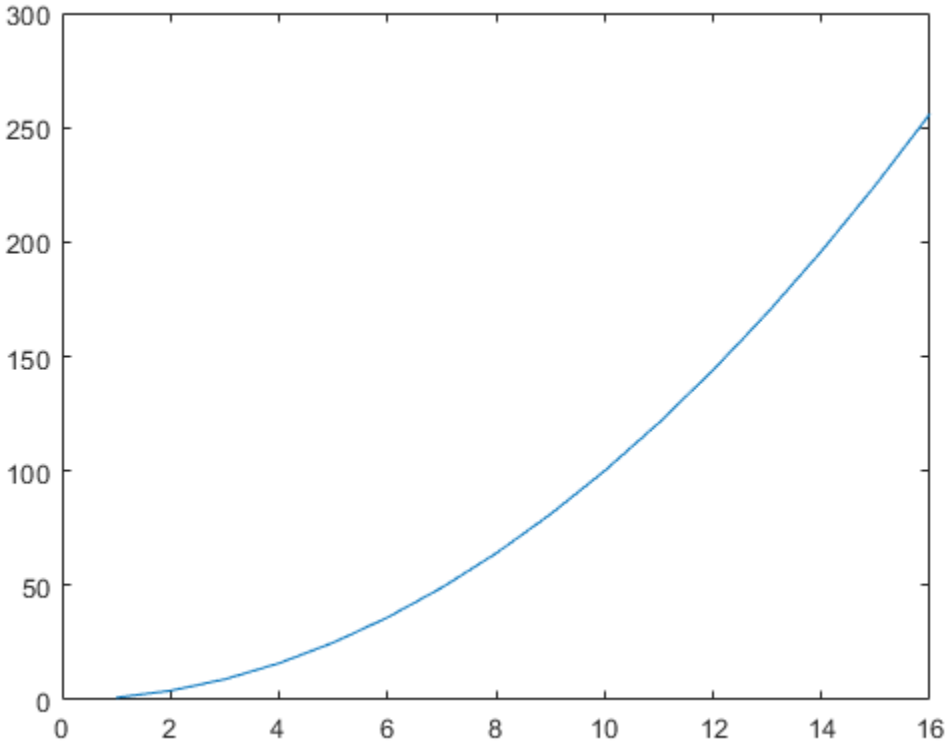
```
reconstructionError = 1.7552e-15
```

Specify Nondefault Dual Moments

Specify nondefault dual moments by using the `mlpt` function. Compare the results of analysis and synthesis using the default and nondefault dual moments.

Create an input signal and visualize it.

```
T = (1:16)';
x = T.^2;
plot(x)
hold on
```



Perform the forward and inverse transform for the input signal using the default and nondefault dual moments.

```
[w2,t2,nj2,scalingmoments2] = mlpt(x,T);
y2 = imlpt(w2,t2,nj2,scalingmoments2);

[w3,t3,nj3,scalingmoments3] = mlpt(x,T,'dualmoments',3);
y3 = imlpt(w3,t3,nj3,scalingmoments3,'dualmoments',3);
```

Plot the reconstructed signal and verify perfect reconstruction using both the default and nondefault dual moments.

```
plot(y2,'o')
plot(y3,'*')
legend('Original Signal', ...
       'DualMoments = 3', ...
       'DualMoments = 2 (Default)');

fprintf('\nMean Reconstruction Error:\n');
```

```
Mean Reconstruction Error:
```

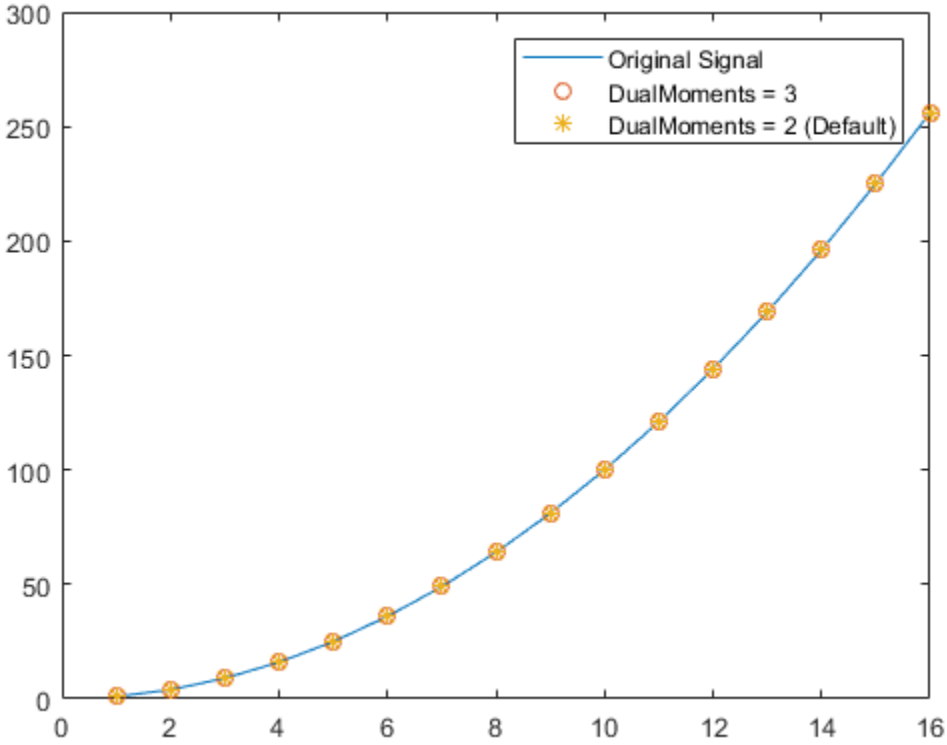
```
fprintf('  - Nondefault dual moments: %0.2f\n',mean(abs(y3-x)));
```

```
  - Nondefault dual moments: 0.00
```

```
fprintf('  - Default dual moments: %0.2f\n\n',mean(abs(y2-x)));
```

```
  - Default dual moments: 0.00
```

```
hold off
```



- Smoothing Nonuniformly Sampled Data

Input Arguments

`coefs` — MLPT coefficients

vector | matrix

MLPT coefficients, specified as a vector or matrix of MLPT coefficients returned by the `mlpt` function.

Data Types: `double`

`T` — Sampling instants corresponding to output
vector | `duration` array

Sampling instants corresponding to `y`, specified as a vector or `duration` array of increasing values returned by the `mlpt` function.

Data Types: `double` | `duration`

`coefsPerLevel` — Coefficients per resolution level
vector

Coefficients per resolution level, specified as a vector containing the number of coefficients at each resolution level in `coefs`. `coefsPerLevel` is an output argument of the `mlpt` function.

The elements of `coefsPerLevel` are organized as follows:

- `coefsPerLevel(1)` — Number of approximation coefficients at the coarsest resolution level.
- `coefsPerLevel(i)` — Number of detail coefficients at resolution level `i`, where `i = numLevel - i + 2` for `i = 2, ..., numLevel + 1`. `numLevel` is the number of resolution levels used to calculate the MLPT. `numLevel` is inferred from `coefsPerLevel`: `numLevel = length(coefsPerLevel-1)`.

The smaller the index `i`, the lower the resolution. The MLPT is two times redundant in the number of detail coefficients, but not in the number of approximation coefficients.

Data Types: `double`

`scalingMoments` — Scaling function moments
matrix

Scaling function moments, specified as a `length(coefs)`-by-`P` matrix, where `P` is the number of primal moments specified by the MLPT.

Data Types: `double`

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: `'DualMoments',3` computes a transform using three dual vanishing moments.

`DualMoments` — Number of dual vanishing moments
`2` (default) | `3` | `4`

Number of dual vanishing moments in the lifting scheme, specified as the comma-separated pair consisting of `'DualMoments'` and `2`, `3` or `4`. The number of dual moments must match the number used by `mlpt`.

Data Types: `double`

Output Arguments

`y` — Reconstructed signal
vector | matrix

Reconstructed signal, returned as a vector or matrix, depending on the inputs to the `mlpt` function.

Data Types: `double`

Algorithms

Maarten Jansen developed the theoretical foundation of the multiscale local polynomial transform (MLPT) and algorithms for its efficient computation [1][2][3]. The MLPT uses a lifting scheme, wherein a kernel function smooths fine-scale coefficients with a given bandwidth to obtain the coarser resolution coefficients. The `mlpt` function uses only local polynomial interpolation, but the technique developed by Jansen is more general and admits many other kernel types with adjustable bandwidths [2].

References

[1] Jansen, M. "Multiscale Local Polynomial Smoothing in a Lifted Pyramid for Non-Equispaced Data". *IEEE Transactions on Signal Processing*. Vol. 61, Number 3, 2013, pp.545-555.

[2] Jansen, M., and M. Amghar. "Multiscale local polynomial decompositions using bandwidths as scales". *Statistics and Computing* (forthcoming). 2016.

[3] Jansen, M., and Patrick Oonincx. *Second Generation Wavelets and Applications*. London: Springer, 2005.

See Also

`mlpt` | `mlptdenoise` | `mlptrecon`

Topics

Smoothing Nonuniformly Sampled Data

Introduced in R2017a

imodwpt

Inverse maximal overlap discrete wavelet packet transform

Syntax

```
xrec = imodwpt(coefs)
xrec = imodwpt(coefs,wname)
xrec = imodwpt(coefs,lo,hi)
```

Description

`xrec = imodwpt(coefs)` returns the inverse maximal overlap discrete wavelet packet transform (inverse MODWPT), in `xrec`. The inverse transform is for the terminal node coefficient matrix (`coefs`) obtained using `modwpt` with the default length 18 Fejer-Korovkin (`'fk18'`) wavelet.

`xrec = imodwpt(coefs,wname)` returns the inverse MODWPT using the orthogonal filter specified the character vector `wname`. This filter must be the same filter used in `modwpt`.

`xrec = imodwpt(coefs,lo,hi)` returns the inverse MODWPT using the orthogonal scaling filter, `lo`, and wavelet filter, `hi`.

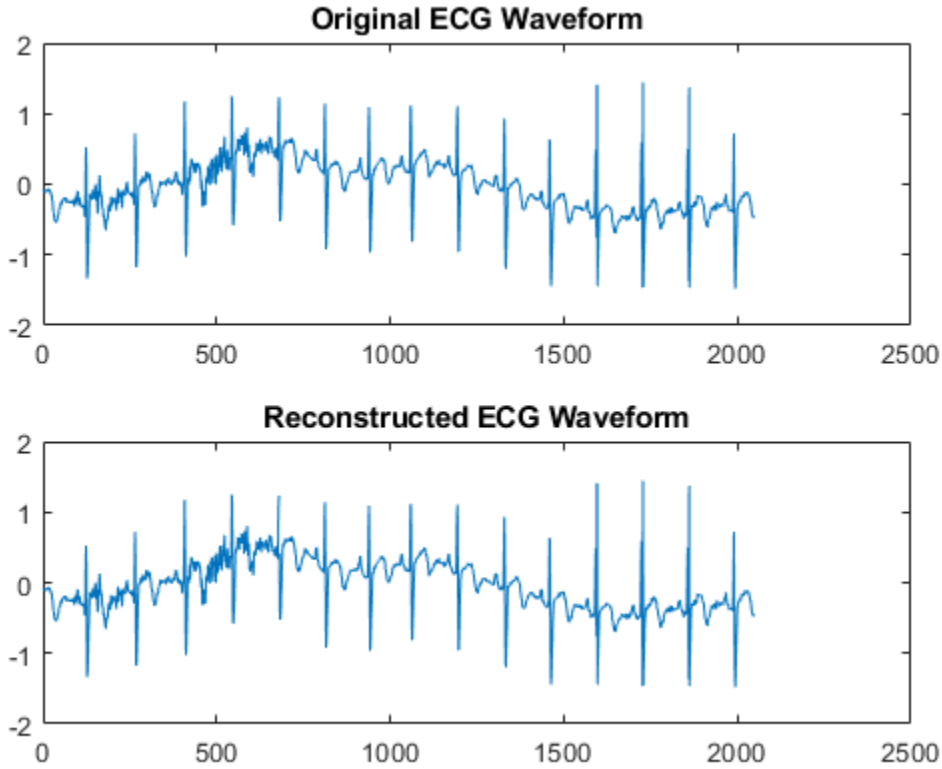
Examples

Perfect Reconstruction with the Inverse MODWPT

Obtain the MODWPT of an ECG waveform and demonstrate perfect reconstruction using the inverse MODWPT.

```
load wecg;
wpt = modwpt(wecg);
```

```
xrec = imodwpt(wpt);
subplot(2,1,1)
plot(wecg);
title('Original ECG Waveform');
subplot(2,1,2)
plot(xrec);
title('Reconstructed ECG Waveform');
```



Find the largest absolute difference between the original signal and the reconstruction. The difference is on the order of 10^{-11}, which demonstrates perfect reconstruction.

```
max(abs(wecg-xrec'))
```

```
ans = 1.7902e-11
```

Inverse MODWPT Using Daubechies Extremal Phase Wavelet

Obtain the MODWPT of Southern Oscillation Index data using the Daubechies extremal phase wavelet with two vanishing moments (`'db2'`). Reconstruct the signal using the inverse MODWPT.

```
load soi;
wsoi = modwpt(soi,'db2');
xrec = imodwpt(wsoi,'db2');
```

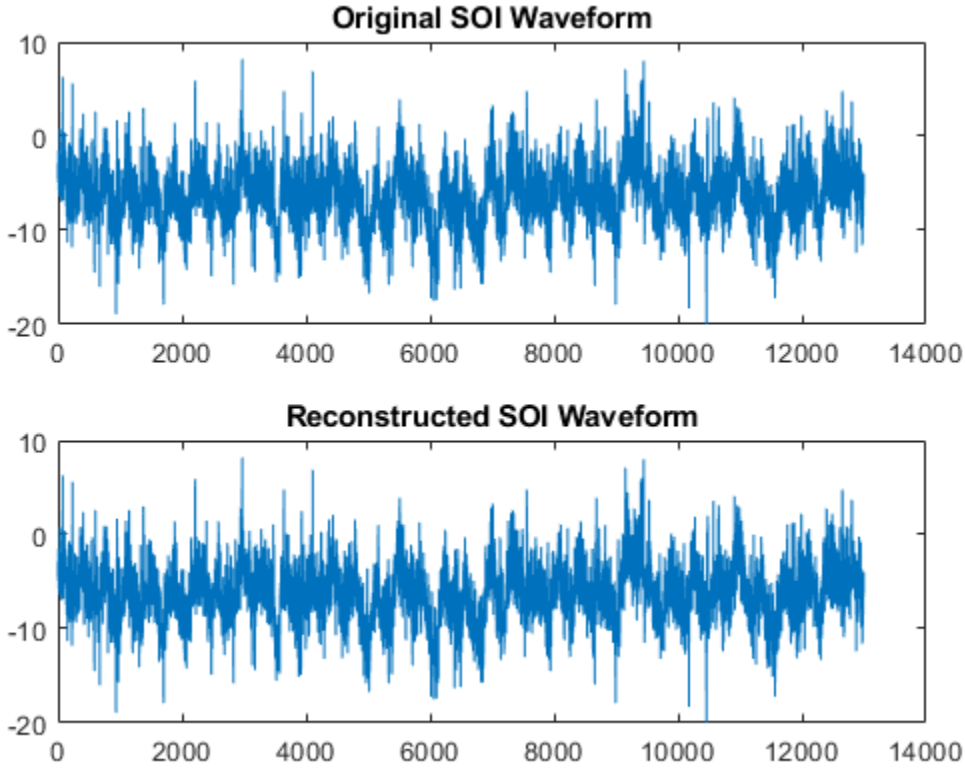
Inverse MODWPT Using Scaling and Wavelet Filters

Obtain the MODWPT of Southern Oscillation Index data using specified scaling and wavelets filters with the Daubechies extremal phase wavelet with two vanishing moments (`'db2'`).

```
load soi;
[lo,hi] = wfilters('db2');
wpt = modwpt(soi,lo,hi);
xrec = imodwpt(wpt,lo,hi);
```

Plot the original SOI waveform and the reconstructed waveform.

```
subplot(2,1,1)
plot(soi)
title('Original SOI Waveform');
subplot(2,1,2)
plot(xrec)
title('Reconstructed SOI Waveform')
```



Input Arguments

`coefs` — Terminal node coefficients

matrix

Terminal node coefficients of a wavelet packet tree, specified as a matrix. You must obtain the coefficient matrix from `modwpt` using the `'FullTree',false` option. `'FullTree',false` is the default value of `modwpt`.

Data Types: `double`

`wname` — Synthesizing wavelet filter

`fk18` (default) | character vector

Synthesizing wavelet filter used to invert the MODWPT, specified as a character vector. The specified wavelet must be the same wavelet as used in the analysis with `modwpt`.

`lo` — Scaling filter

even-length real-valued vector

Scaling filter, specified as an even-length real-valued vector. `lo` must be the same scaling filter as used in the analysis with `modwpt`. You cannot specify both a scaling-wavelet filter pair and a `wname` filter.

`hi` — Wavelet filter

even-length real-valued vector

Wavelet filter, specified as an even-length real-valued vector. `hi` must be the same wavelet filter used in the analysis with `modwpt`. You cannot specify both a scaling-wavelet filter pair and a `wname` filter.

Output Arguments

`xrec` — Inverse maximal overlap discrete wavelet packet transform

row vector

Inverse maximal overlap discrete wavelet packet transform, returned as a row vector. The inverse transform is the reconstructed version of the original signal based on the MODWPT terminal node coefficients. `xrec` has the same number of columns as the input `coefs` matrix.

References

[1] Percival, D. B., and A. T. Walden. *Wavelet Methods for Time Series Analysis*. Cambridge, UK: Cambridge University Press, 2000.

[2] Walden, A.T., and A. Contreras Cristan. "The phase-corrected undecimated discrete wavelet packet transform and its application to interpreting the timing of events." *Proceedings of the Royal Society of London A*. Vol. 454, Issue 1976, 1998, pp. 2243-2266.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`modwpt` | `modwptdetails`

Introduced in R2016a

imodwt

Inverse maximal overlap discrete wavelet transform

Syntax

```
xrec = imodwt(w)
xrec = imodwt(w,wname)
xrec = imodwt(w,Lo,Hi)
xrec = imodwt(___,lev)
xrec = imodwt(___,'reflection')
```

Description

`xrec = imodwt(w)` returns in `xrec` a reconstructed version of the signal. The reconstructed signal is based on `w`, the maximal overlap discrete wavelet transform (MODWT) coefficients and on the level of reconstruction, which defaults to zero.

`xrec = imodwt(w,wname)` reconstructs the signal using `wname`, the orthogonal wavelet. `wname` must be the same wavelet used to analyze the signal input to `modwt`. The reconstruction is up to level 0, which is a perfect reconstruction of the original signal.

`xrec = imodwt(w,Lo,Hi)` reconstructs the signal using the orthogonal scaling filter `Lo` and the wavelet filter `Hi`. The `Lo` and `Hi` filters must be the same filters used to analyze the signal input to `modwt`. The reconstruction is up to level 0, which is a perfect reconstruction of the original signal.

`xrec = imodwt(___,lev)` reconstructs the signal up to level `lev`. `xrec` is a projection onto the scaling space at level `lev`.

`xrec = imodwt(___,'reflection')` uses the reflection boundary condition in the reconstruction. If you specify `'reflection'`, `imodwt` assumes that the length of the original signal length is one half the number of columns in the input coefficient matrix. By default, `imodwt` assumes periodic signal extension at the boundary.

Examples

Perfect Reconstruction with the Inverse MODWT

Obtain the MODWT of an ECG signal and demonstrate perfect reconstruction.

Load the ECG signal data and obtain the MODWT.

```
load wecg;
```

Obtain the MODWT and the Inverse MODWT.

```
w = modwt(wecg);
xrec = imodwt(w);
```

Use the L-infinity norm to show that the difference between the original signal and the reconstruction is extremely small. The largest absolute difference between the original signal and the reconstruction is on the order of 10^{-12}, which demonstrates perfect reconstruction.

```
norm(abs(xrec'-wecg),Inf)
```

```
ans = 2.3253e-12
```

Inverse MODWT with Specified Wavelet

Obtain the MODWT of Deutsche Mark-U.S. Dollar exchange rate data and demonstrate perfect reconstruction.

Load the Deutsche Mark-U.S. Dollar exchange rate data.

```
load DM_USD;
```

Obtain the MODWT and the Inverse MODWT using the `'db2'` wavelet.

```
wdm = modwt(DM_USD,'db2');
xrec = imodwt(wdm,'db2');
```

Use the L-infinity norm to show that the difference between the original signal and the reconstruction is extremely small. The largest absolute difference between the original

signal and the reconstruction is on the order of 10^{-13}, which demonstrates perfect reconstruction.

```
norm(abs(xrec'-DM_USD),Inf)
```

```
ans = 1.6362e-13
```

Inverse MODWT with Specified Filters

Obtain the MODWT of an ECG signal using the Fejer-Korovkin filters.

Load the ECG data.

```
load wecg;
```

Create the 8-coefficient Fejer-Korovkin filters.

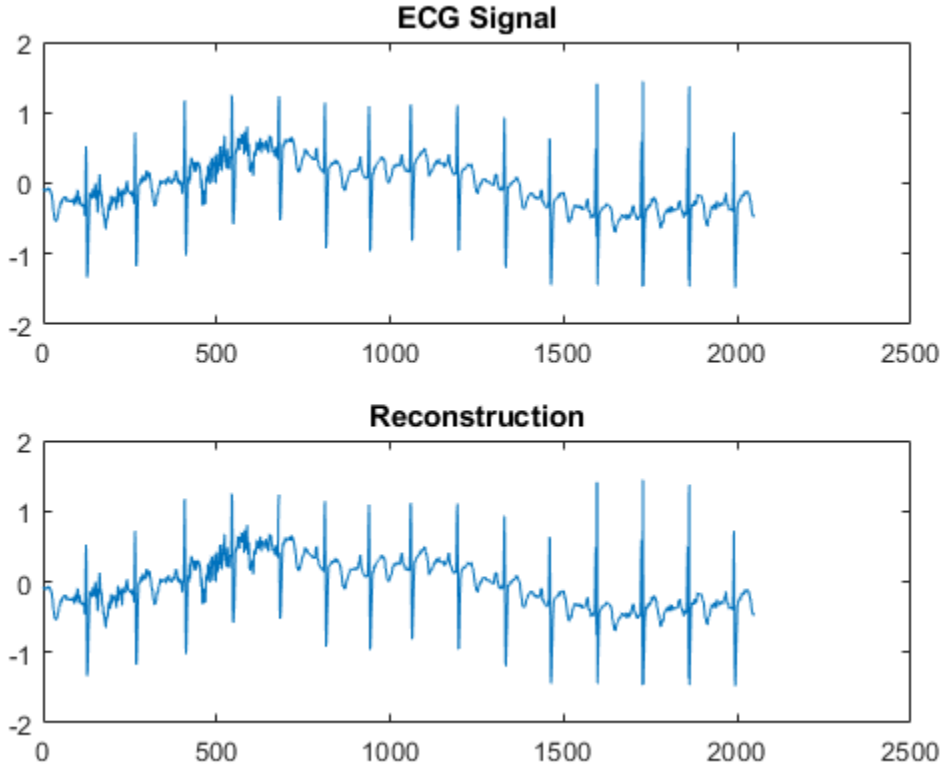
```
[Lo,Hi] = wfilters('fk8');
```

Obtain the MODWT and Inverse MODWT.

```
wtecg = modwt(wecg,Lo,Hi);
xrec = imodwt(wtecg,Lo,Hi);
```

Plot the original data and the reconstruction.

```
subplot(2,1,1)
plot(wecg)
title('ECG Signal');
subplot(2,1,2)
plot(xrec)
title('Reconstruction')
```



Obtain Projection onto Scaling Space

Obtain the MODWT of an ECG signal down to the maximum level and obtain the projection of the ECG signal onto the scaling space at level 3.

Load the ECG data.

```
load wecg;
```

Obtain the MODWT.

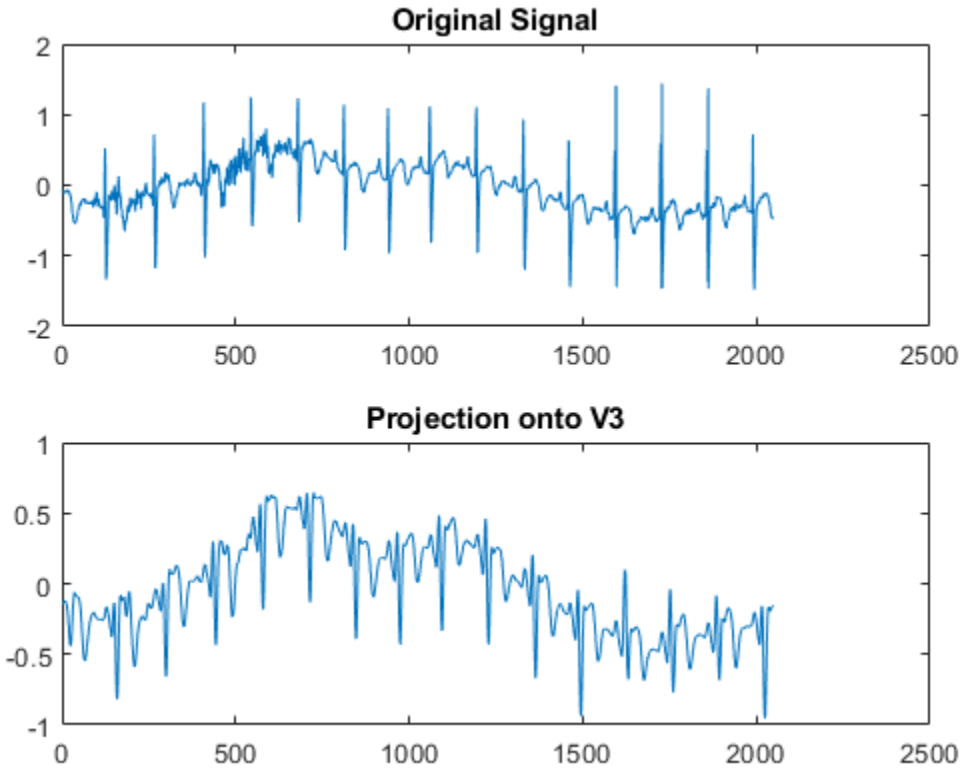
```
wtecg = modwt(wecg);
```

Obtain the projection of the ECG signal onto V_3, the scaling space at level three by using the `imodwt` function.

```
v3proj = imodwt(wtecg,3);
```

Plot the original signal and the projection.

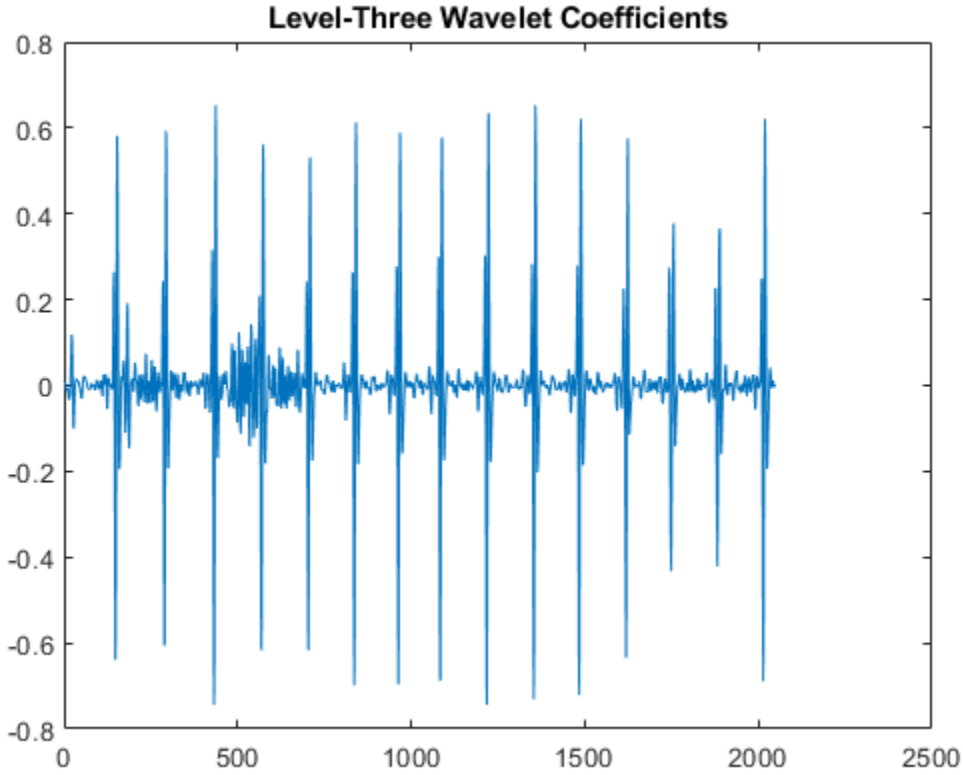
```
subplot(2,1,1)
plot(wecg)
title('Original Signal')
subplot(2,1,2)
plot(v3proj)
title('Projection onto V3')
```



Note that the spikes characteristic of the R waves in the ECG are missing in the V_3 approximation. You can see the missing details by examining the wavelete coefficients at level three.

Plot the level-three wavelet coefficients.

```
figure
plot(wtecg(3,:))
title('Level-Three Wavelet Coefficients')
```



Inverse MODWT with Reflection Boundary

Obtain the inverse MODWT using reflection boundary handling for Southern Oscillation Index data. The sampling period is one day. `imodwt` with the `'reflection'` option assumes that the input matrix, which is the `modwt` output, is twice the length of the original signal length. `imodwt` reflection boundary handling reduces the number of wavelet and scaling coefficients at each scale by half.

```
load soi;
wsoi = modwt(soi,4,'reflection');
xrecsoi = imodwt(wsoi,'reflection');
```

Use the L-infinity norm to show that the difference between the original signal and the reconstruction is extremely small. The largest absolute difference between the original signal and the reconstruction is on the order of 10^{-11}, which demonstrates perfect reconstruction.

```
norm(abs(xrecsoi'-soi),Inf)
```

```
ans = 1.6433e-11
```

Input Arguments

`w` — MODWT transform

matrix

MODWT transform, specified as a matrix of size L+1-by-N. `w` is the output of `modwt`, which is the MODWT of an N-point input signal down to level L. By default, `imodwt` assumes that you obtained the MODWT using the `'sym4'` wavelet with periodic boundary handling.

Data Types: `double`

`wname` — Synthesis wavelet

`'sym4'` (default) | `'db`L`'` | `'coif`L`'` | `'haar`L`'` | `'fk`L`'` | `'sym`L`'` | character vector

Synthesis wavelet, specified as a character vector. The synthesis wavelet must be the same wavelet used in the analysis with `modwt`.

`Lo` — Scaling filter

even-length real-valued vector

Scaling filter, specified as an even-length real-valued vector. You can specify `Lo` only if you do not specify `wname`. `Lo` must be the same scaling filter used in the analysis with `modwt`.

`Hi` — Wavelet filter

even-length real-valued vector

Wavelet filter, specified as an even-length real-valued vector. You can specify `Hi` only if you do not specify `wname`. `Hi` must be the same wavelet filter used in the analysis with `modwt`.

`lev` — Reconstruction level

0 (default) | nonnegative integer

Reconstruction level, specified as a nonnegative integer between 0 and `size(w,1)-2`. The level must be less than the level used to obtain `w` from `modwt`. If `lev` is 0 and you do not modify the coefficients, `imodwt` produces a perfect reconstruction of the signal.

Output Arguments

`xrec` — Reconstructed signal

row vector

Reconstructed version of the original signal based on the MODWT and the level of reconstruction, returned as a row vector.

References

[1] Percival, D. B., and A. T. Walden. *Wavelet Methods for Time Series Analysis*. Cambridge, UK: Cambridge University Press, 2000.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`modwt` | `modwtmra`

Introduced in R2015b

ind2depo

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])`.

Examples

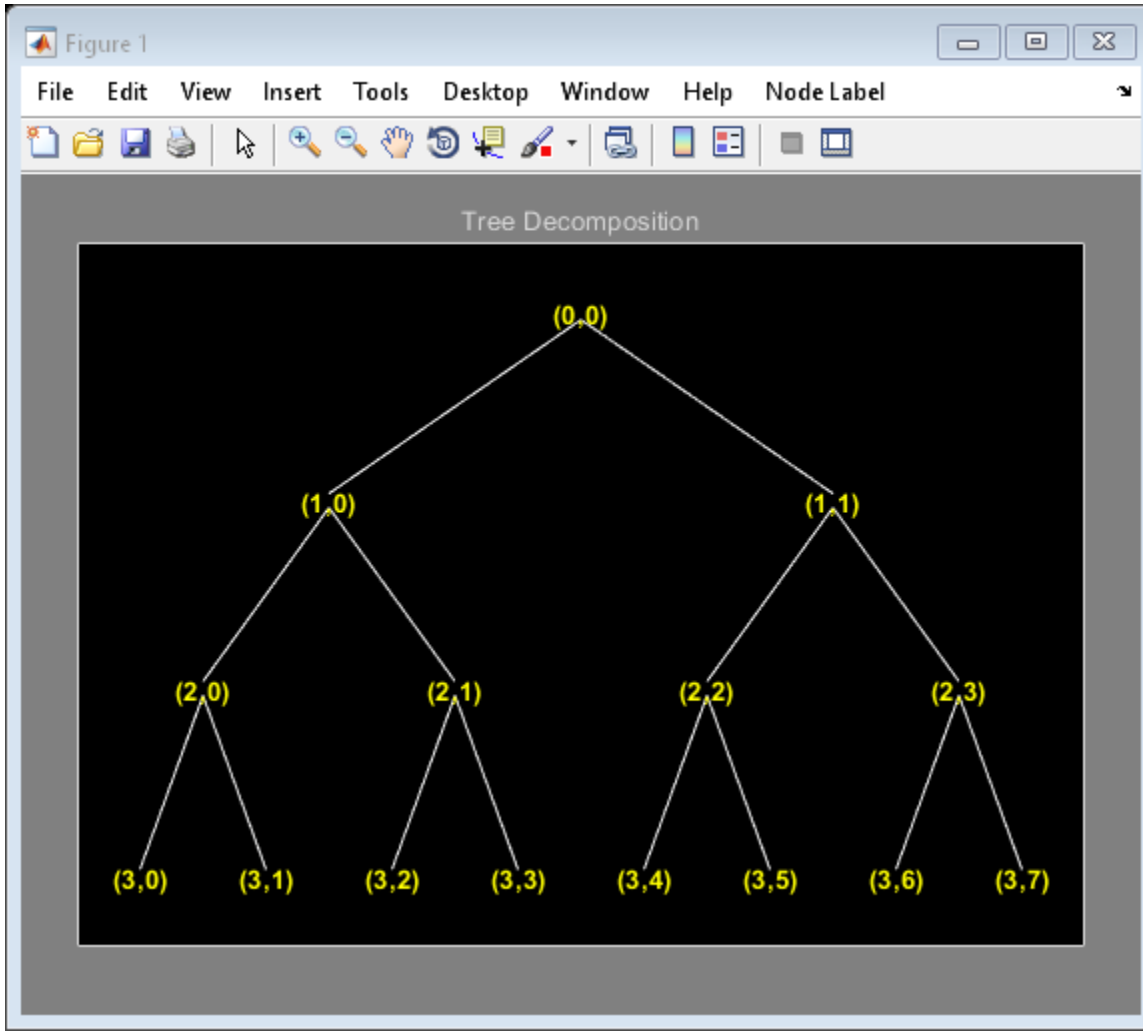
Depth and Position in Wavelet Packet Tree

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 =

  15x2 table

    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`

Introduced before R2006a

intwave

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 $-\infty$ 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 character vector 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 ψ_{dec} and the wavelet reconstruction function ψ_{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.

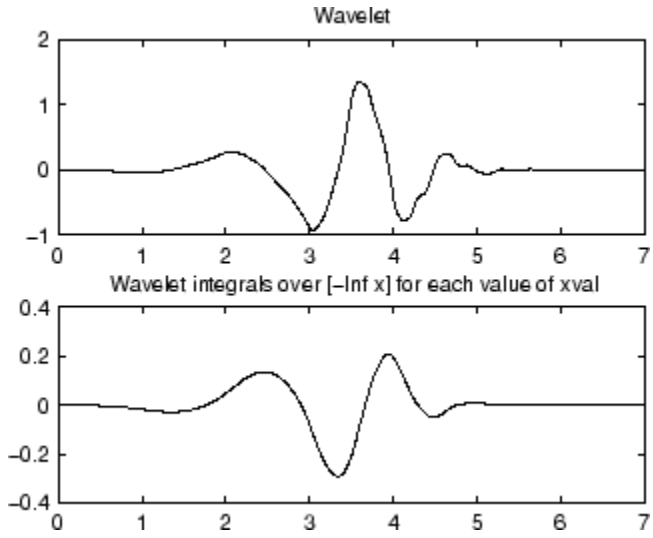
When `IN2` is equal to the special value 0, `intwave('wname',0)` is equivalent to `intwave('wname',8,IN3)`.

`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']);
```



Algorithms

First, the wavelet function is approximated on a grid of 2^{PREC} points using `wavefun`. A piecewise constant interpolation is used to compute the integrals using `cumsum`.

See Also

`wavefun`

Introduced before R2006a

isnode

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.

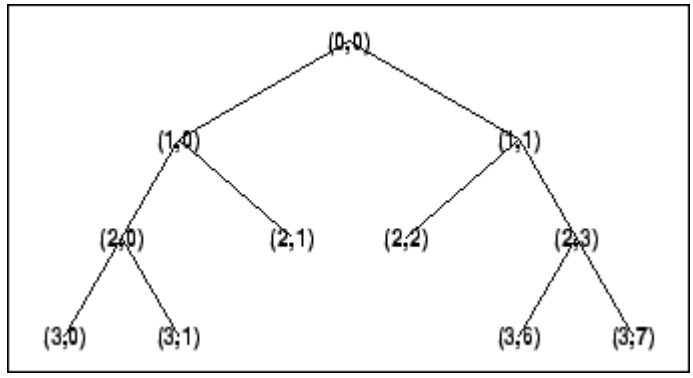
N can be a column vector containing the indices of nodes or a matrix, that contains the depths and positions of nodes.

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.

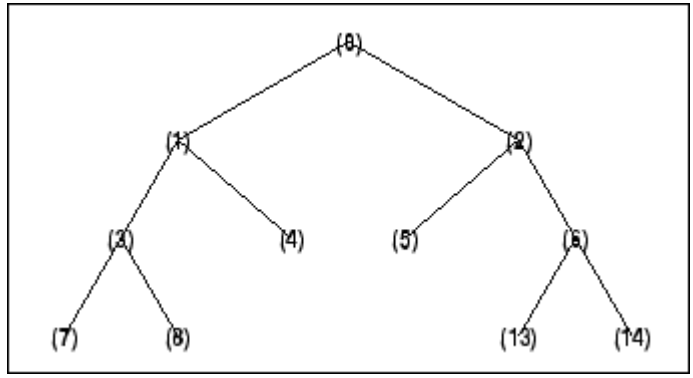
The nodes are numbered from left to right and from top to bottom. The root index is 0.

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`

Introduced before R2006a

istnode

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.

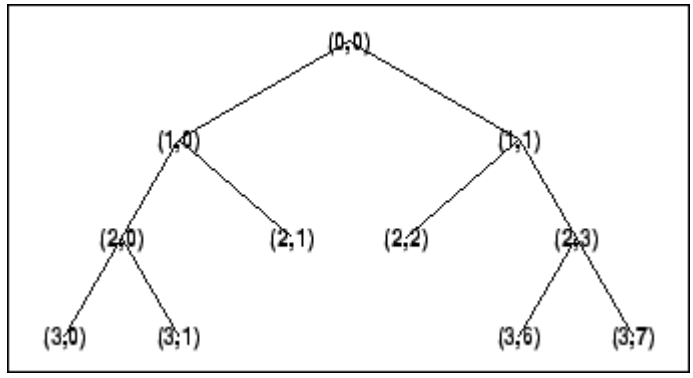
`N` can be a column vector containing the indices of nodes or a matrix that contains the depths and positions of nodes.

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.

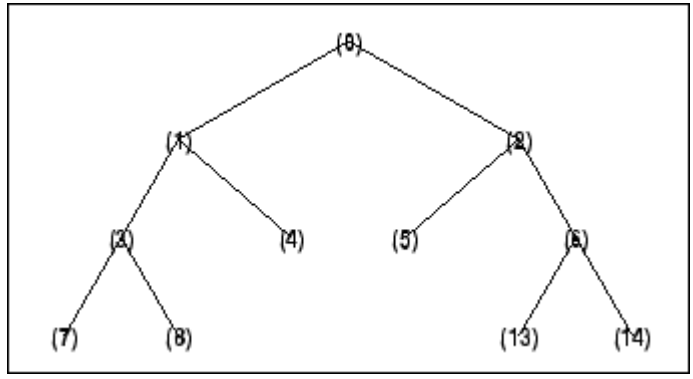
The nodes are numbered from left to right and from top to bottom. The root index is 0.

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

Introduced before R2006a

iswt

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 an orthogonal or a biorthogonal wavelet. Specify the wavelet using its name (*'wname'*, see `wfilters` for more information) or its 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

Multilevel Stationary Wavelet Reconstruction

Demonstrate perfect reconstruction using `swt` and `iswt` with a biorthogonal wavelet.

```
load noisbloc
[Lo_D,Hi_D,Lo_R,Hi_R] = wfilters('bior3.5');
[swa,swd] = swt(noisbloc,3,Lo_D,Hi_D);
recon = iswt(swa,swd,Lo_R,Hi_R);
norm(noisbloc-recon)

ans = 1.1386e-13
```

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`

Introduced before R2006a

iswt2

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(A(:,:,1,:),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)
X = iswt2(A(:,:,1,:),H,V,D,'wname')
```

Description

`iswt2` performs a multilevel 2-D stationary wavelet reconstruction using either an orthogonal or a biorthogonal wavelet. Specify the wavelet using its name (`'wname'`, see `wfilters` for more information) or its reconstruction filters (`Lo_R` and `Hi_R`).

`X = iswt2(SWC,'wname')` or `X = iswt2(A,H,V,D,wname)` reconstructs the signal `X`, based on the multilevel stationary wavelet decomposition structure `SWC` or `[A,H,V,D]` (see `swt2`).

If multilevel stationary wavelet decomposition structure `SWC` or `[A,H,V,D]` was generated from a 2-D matrix, the syntax `X = iswt2(A(:,:,end),H,V,D,'wname')` reconstructs the signal `X`.

If the stationary wavelet decomposition structure `SWC` or `[A,H,V,D]` was generated from a single level stationary wavelet decomposition of a 3-D matrix, `X = iswt2(A(:,:,1,:),H,V,D,'wname')` reconstructs the signal `X`.

`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)` or `X = iswt2(A(:,:,`

`1,:),H,V,D,'wname')` 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.

Note

- `iswt2` synthesizes `X` from the coefficient arrays generated by `swt2`. `swt2` uses double-precision arithmetic internally and returns double-precision coefficient matrices. `swt2` warns if there is a loss of precision when converting to double.
- To distinguish a single-level decomposition of a truecolor image from a multilevel decomposition of an indexed image, the approximation and detail coefficient arrays of truecolor images are 4-D arrays. See the Release Notes for details. Also see examples “Stationary Wavelet Transform of an Image” on page 1-404 and “Inverse Stationary Wavelet Transform of an Image” on page 1-409.

  If an `K`-level decomposition is performed, the dimensions of the `A`, `H`, `V`, and `D` coefficient arrays are `m`-by-`n`-by-3-by-`K`.

  If a single-level decomposition is performed, the dimensions of the `A`, `H`, `V`, and `D` coefficient arrays are `m`-by-`n`-by-1-by-3. Since MATLABremoves singleton last dimensions by default, the third dimension of the coefficient arrays is singleton.

Examples

Multilevel Two-Dimensional Stationary Wavelet Reconstruction

Demonstrate perfect reconstruction using `swt2` and `iswt2` with an orthogonal wavelet.

```
load woman
[Lo_D,Hi_D,Lo_R,Hi_R] = wfilters('db6');
[ca,chd,cvd,cdd] = swt2(X,3,Lo_D,Hi_D);
recon = iswt2(ca,chd,cvd,cdd,Lo_R,Hi_R);
norm(X-recon)
```

```
ans = 1.0126e-08
```

Stationary Wavelet Transform of an Image

In this example you obtain single-level and multilevel stationary wavelet decompositions of a truecolor image. You view results of each decomposition.

Load in a truecolor image. The image is a 3-D array of type `uint8`. Since `swt2` requires the first and second dimensions both be divisible by a power of 2, extract a portion of the image and view it.

```
imdata = imread('ngc6543a.jpg');
x = imdata(1:512,1:512,:);
imagesc(x)
```

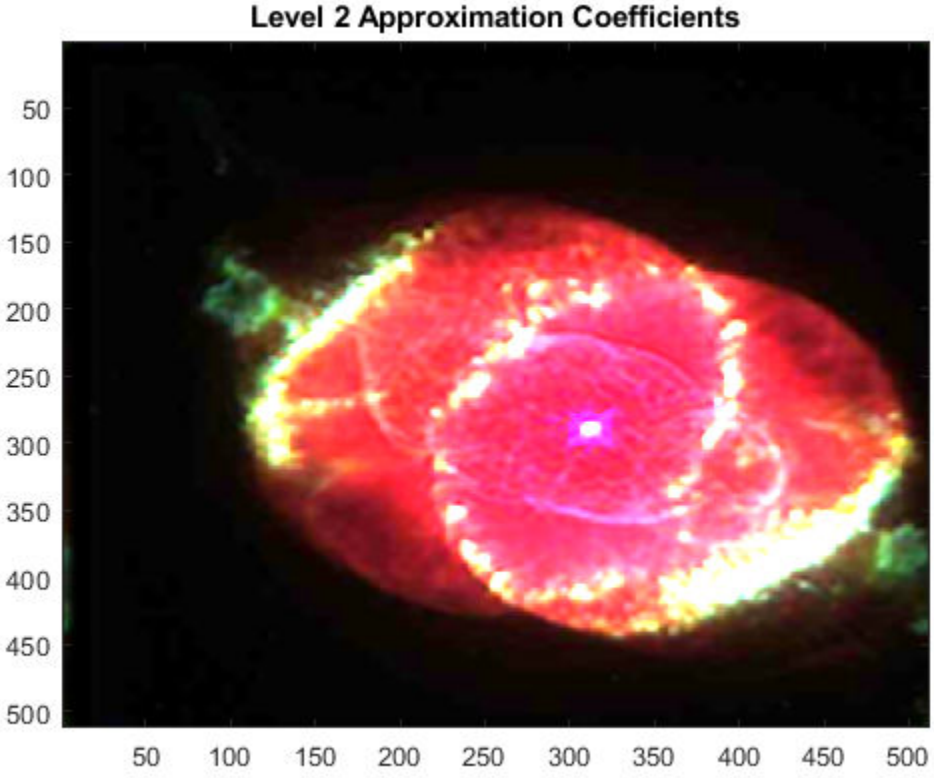
Obtain the 4-level stationary wavelet decomposition of the image using the `db4` wavelet. Return the approximation coefficients and the horizontal, vertical, and detail coefficients as separate arrays. Note the dimensions of the output arrays.

```
[a,h,v,d] = swt2(x,4,'db4');
size(a)

ans =

   512   512     3     4


size(h)
```

```
ans =

   512   512     3     4


size(v)

ans =

   512   512     3     4


size(d)

ans =

   512   512     3     4
```

The output arrays are all of type `double`. View the level 2 approximation coefficients. Since the approximation coefficients are of type `double`, cast them as `uint8`, which was the original datatype of the image.

```
figure
imagesc(uint8(a(:,:,:,2)))
title('Level 2 Approximation Coefficients')
```



Reconstruct the image by performing the inverse transform. Compute the difference between the original image and reconstruction. Since the reconstruction is of type `double`, cast the reconstruction as type `uint8` before computing the difference.

```
rec = iswt2(a,h,v,d,'db4');
maxdiff = max(abs(uint8(rec(:))-x(:)));
disp(['maximum difference = ' num2str(maxdiff)])
```

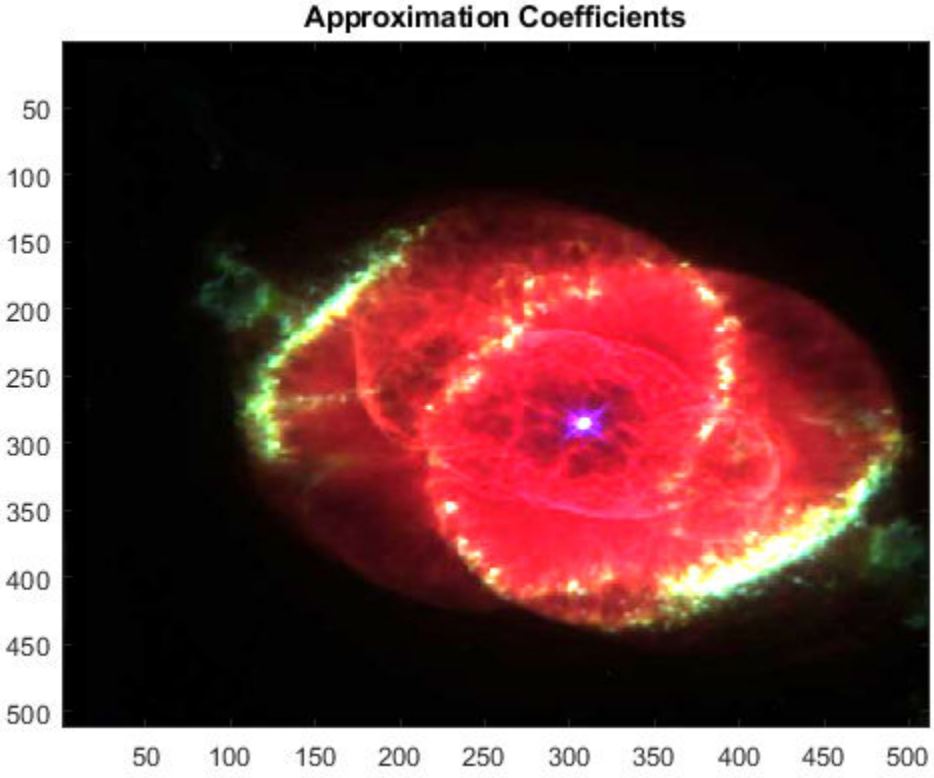
```
maximum difference = 0
```

Obtain the single-level stationary wavelet decomposition of the image using the `db4` wavelet. Return the approximation coefficients and horizontal, vertical, and detail coefficients in separate arrays. Note the dimensions of the output arrays.

```
[a,h,v,d] = swt2(x,1,'db4');
size(a)

ans =

   512   512     1     3


size(h)

ans =

   512   512     1     3


size(v)

ans =

   512   512     1     3


size(d)

ans =

   512   512     1     3
```

View the approximation coefficients. To prevent an error when using `imagesc`, first squeeze the approximation coefficients array to remove the singleton dimension.

```
asqueeze = squeeze(a);
size(asqueeze)

ans =

   512   512     3


figure
imagesc(uint8(asqueeze))
title('Approximation Coefficients')
```



Reconstruct the image by performing the inverse transform. Compute the difference between the original image and reconstruction. Since the reconstruction is of type `double`, cast the reconstruction as type `uint8` before computing the difference.

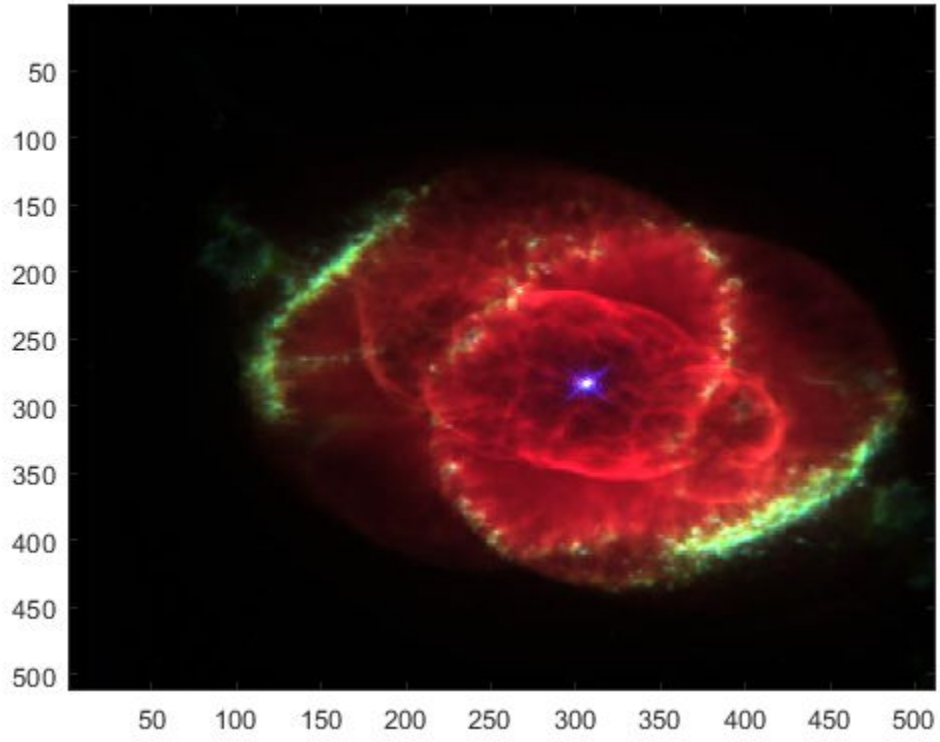
```
rec = iswt2(a,h,v,d,'db4');
maxdiff = max(abs(uint8(rec(:))-x(:)));
disp(['maximum difference = ' num2str(maxdiff)])

maximum difference = 0
```

Inverse Stationary Wavelet Transform of an Image

This example shows how to reconstruct a truecolor image from a single-level stationary wavelet decomposition using 3-D approximation and detail coefficient arrays.

Load in a truecolor image. The image is a 3-D array of type `uint8`. Since `swt2` requires the first and second dimensions both be divisible by a power of 2, extract a portion of the image and view it.

```
imdata = imread('ngc6543a.jpg');
x = imdata(1:512,1:512,:);
imagesc(x)
```



Obtain the single-level stationary wavelet decomposition of the image using the `db4` wavelet. Return the approximation coefficients and horizontal, vertical, and detail coefficients in separate arrays. Note the dimensions of the output arrays.

```
[a,h,v,d] = swt2(x,1,'db4');
size(a)

ans =

   512   512     1     3


size(h)

ans =

   512   512     1     3


size(v)

ans =

   512   512     1     3


size(d)

ans =

   512   512     1     3
```

Squeeze the coefficient arrays to remove their singleton dimensions. Note the dimensions of the squeezed arrays.

```
asq = squeeze(a);
hsq = squeeze(h);
vsq = squeeze(v);
dsq = squeeze(d);
size(asq)

ans =

   512   512     3
```

```
size(hsq)

ans =

   512   512     3


size(vsq)

ans =

   512   512     3


size(dsq)

ans =

   512   512     3
```

So that `iswt2` can properly reconstruct the trueimage from the new coefficient arrays, insert a singleton dimension by reshaping the squeezed arrays. Reconstruct the image with the reshaped coefficient arrays.

```
a2 = reshape(asq,[512,512,1,3]);
h2 = reshape(hsq,[512,512,1,3]);
v2 = reshape(vsq,[512,512,1,3]);
d2 = reshape(dsq,[512,512,1,3]);
rec = iswt2(a2,h2,v2,d2,'db4');
```

Compute the difference between the original image and reconstruction. Since the reconstruction is of type `double`, cast the reconstruction as type `uint8` before computing the difference.

```
maxdiff = max(abs(uint8(rec(:))-x(:)));
disp(['maximum difference = ' num2str(maxdiff)])

maximum difference = 0
```

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.

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`

Introduced before R2006a

iwsst

Inverse wavelet synchrosqueezed transform

Syntax

```
xrec = iwsst(sst)
xrec = iwsst(sst,f,freqrange)
xrec = iwsst(sst,iridge)
xrec = iwsst(___,wav)
xrec = iwsst(___,iridge,Name,Value)
```

Description

`xrec = iwsst(sst)` inverts the input synchrosqueezed transform, `sst`, and returns the inverse in vector `xrec`. To obtain the `sst` input, use the `wsst` function. The `iwsst` function assumes that you obtain `sst` using the analytic Morlet wavelet.

Note The wavelet transform does not preserve a nonzero mean. After inverting the synchrosqueezed transform, you must add back the original signal mean.

`xrec = iwsst(sst,f,freqrange)` inverts the synschrosqueezed transform for a specified range of frequencies, `freqrange`, contained in the frequency vector, `f`. The frequency vector, `f`, is the output of `wsst`.

`xrec = iwsst(sst,iridge)` inverts the synchrosqueezed transform along the time-frequency ridges specified by `iridge`, the index column vector. `iridge` is the output of `wsstridge`. The `xrec` output is the same size as `iridge`.

`xrec = iwsst(___,wav)` uses the analytic wavelet specified by `wav` to invert the synchrosqueezed transform. This wavelet must be the same wavelet as used in `wsst`. You can include any of the input arguments from previous syntaxes.

`xrec = iwsst(___,iridge,Name,Value)` returns the inverse synchrosqueezed transform with additional options specified by one or more `Name,Value` pair arguments.

Examples

Inverse Synchrosqueezed Transform of Chirp

Obtain the wavelet synchrosqueezed transform of a quadratic chirp using default values. Then reconstruct the signal using the inverse wavelet synchrosqueezed transform.

```
load quadchirp;
sst = wsst(quadchirp);
xrec = iwsst(sst);
```

Synchrosqueezed and Inverse Synchrosqueezed Transform of Chirp

Obtain the wavelet synchrosqueezed transform of a quadratic chirp sampled at 1000 Hz. Then reconstruct the chirp.

Load the chirp and obtain the synchrosqueezed transform.

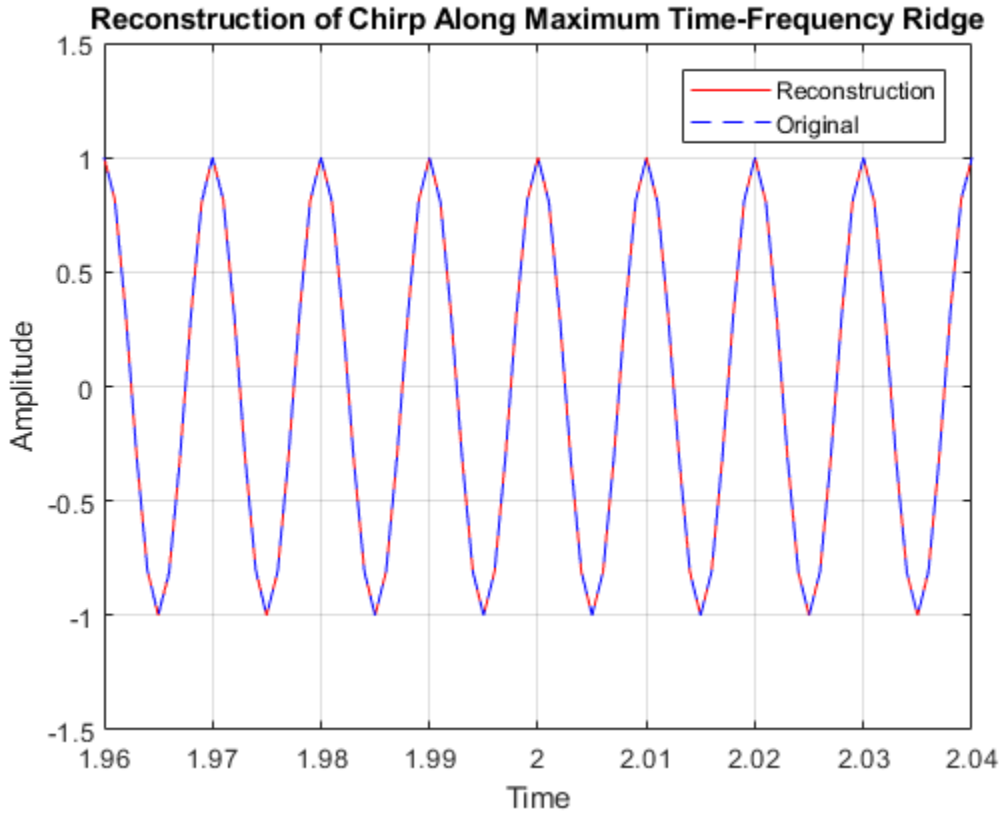
```
load quadchirp;
sstchirp = wsst(quadchirp,'ExtendSignal',true);
```

Extract the maximum energy time-frequency ridge and reconstruct the signal mode along the ridge.

```
[~,iridge] = wsstridge(sstchirp);
xrec = iwsst(sstchirp,iridge);
```

Plot and zoom in on the original and reconstructed signal.

```
plot(tquad,xrec,'r');
hold on;
plot(tquad,quadchirp,'b--');
xlabel('Time'); ylabel('Amplitude');
set(gca,'ylim',[-1.5 1.5]);
legend('Reconstruction','Original');
grid on;
title('Reconstruction of Chirp Along Maximum Time-Frequency Ridge');
zoom xon
zoom(50)
```



Inverse Synchrosqueezed Transform of Range of Frequencies

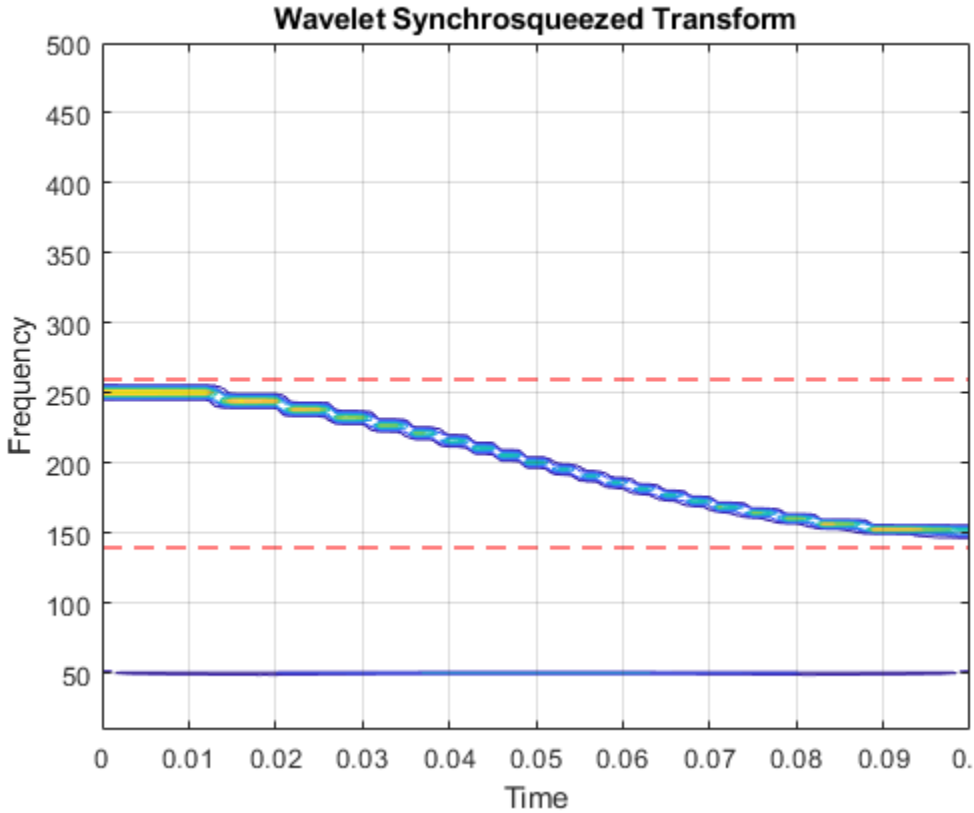
Obtain the inverse synchrosqueezed transform for a specified frequency range of a two-component signal. The input is a combination of an amplitude-modulated and a frequency-modulated signal.

Create the signal.

```
t = 0:0.001:0.1;
x1 = (2+0.5*cos(2*pi*10*t)).*cos(2*pi*200*t+10*sin(2*pi*5*t));
x2 = cos(2*pi*50*t);
sig = x1+x2;
```

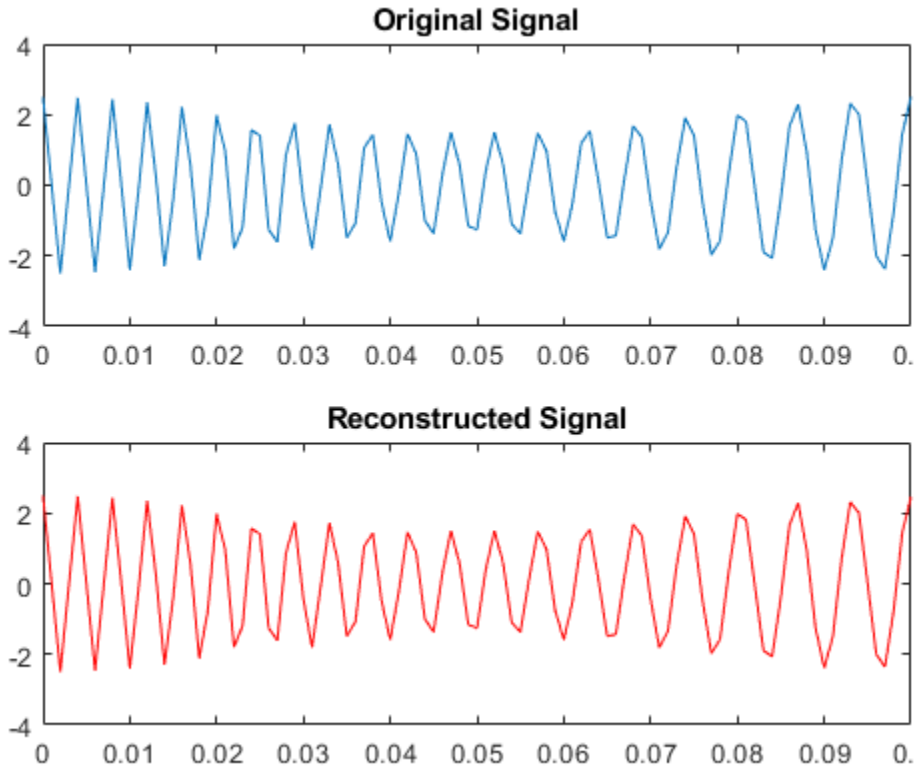
Obtain the wavelet synchrosqueezed transform and plot the resulting two frequency components.

```
[sst,f] = wsst(sig,1000,'ExtendSignal',true);
contour(t,f,abs(sst));
grid on;
title('Wavelet Synchrosqueezed Transform');
ylabel('Frequency');
xlabel('Time');
hold on
plot(t,140*ones(size(t)),'r--');
plot(t,260*ones(size(t)),'r--');
```



Obtain the inverse synchrosqueezed transform for frequencies from 140 Hz to 260 Hz. Plot the result.

```
xrec = iwsst(sst,f,[140,260]);
subplot(2,1,1);
plot(t,x1);
title('Original Signal');
subplot(2,1,2);
plot(t,xrec,'r');
title('Reconstructed Signal');
```



Synchrosqueezed and Inverse Synchrosqueezed Transform of Speech Signal

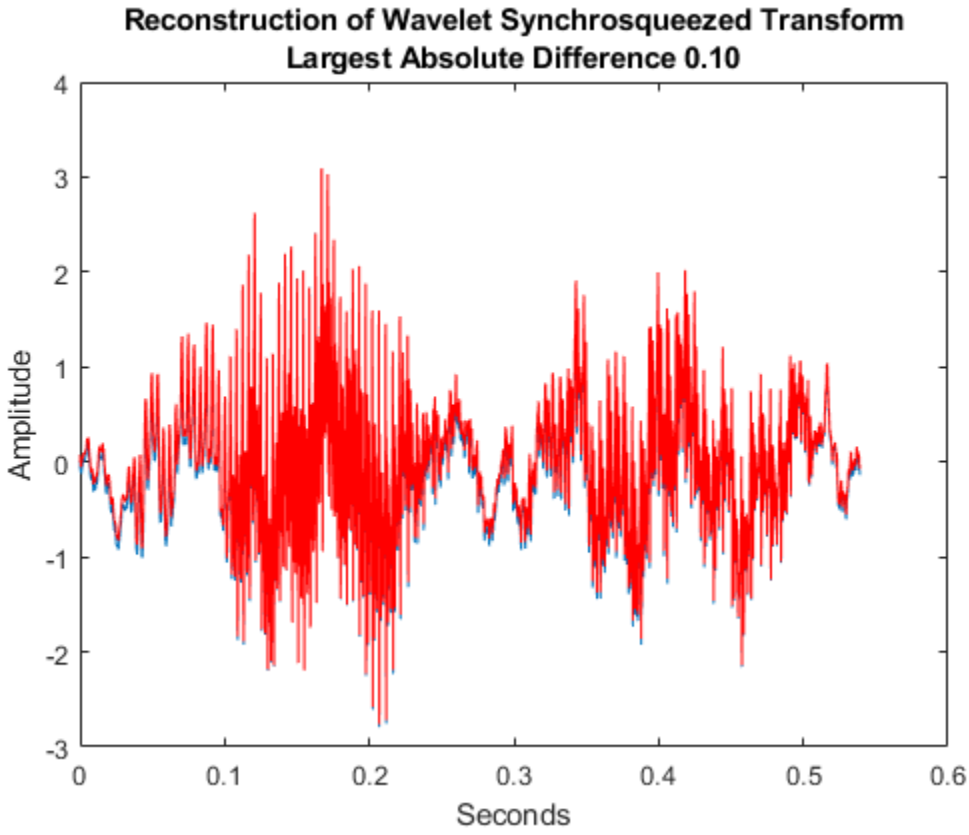
Obtain the wavelet synchrosqueezed transform and inverse synchrosqueezed transform of a speech sample using the bump wavelet.

Load the speech signal and obtain the synchrosqueezed transform and inverse synchrosqueezed transform.

```
load mtlb;
dt = 1/Fs;
t = 0:dt:numel(mtlb)*dt-dt;
Txmtlb = wsst(mtlb,'bump');
xrec = iwsst(Txmtlb,'bump');
```

Obtain the L-infinity norm of the difference between the original waveform and the reconstruction. Plot the results.

```
Linf = norm(abs(mtlb-xrec),Inf);
plot(t,mtlb);
hold on;
xlabel('Seconds'); ylabel('Amplitude');
plot(t,xrec,'r');
title({'Reconstruction of Wavelet Synchrosqueezed Transform'; ...
['Largest Absolute Difference ' num2str(Linf,'%1.2f')]});
```



Synchrosqueezed Transform Using Specified Number of Bins for Chirp

This example shows how to invert the wavelet synchrosqueezed transform using a specified number of frequency bins for a quadratic chirp. The chirp is sampled at 1000 Hz.

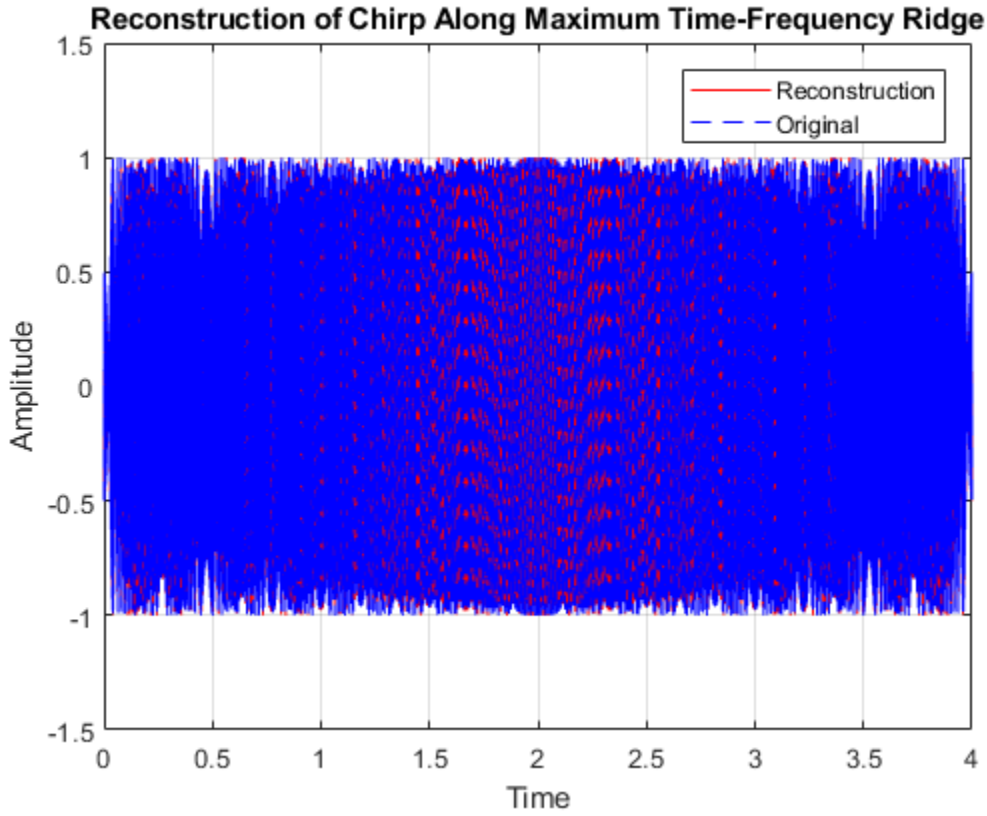
```
load quadchirp;
sstchirp = wsst(quadchirp,'ExtendSignal',true);
```

Extract the maximum energy time-frequency ridge using 10 bins on each side of the iridge index and reconstruct the signal mode along the ridge.

```
[~,iridge] = wsstridge(sstchirp);
xrec = iwsst(sstchirp,iridge,'NumFrequencyBins',10);
```

Plot the original and reconstructed signal.

```
plot(tquad,xrec,'r');
hold on;
plot(tquad,quadchirp,'b--');
xlabel('Time'); ylabel('Amplitude');
set(gca,'ylim',[-1.5 1.5]);
legend('Reconstruction','Original');
grid on;
title('Reconstruction of Chirp Along Maximum Time-Frequency Ridge');
```



- “Time-Frequency Reassignment and Mode Extraction with Synchrosqueezing”

Input Arguments

`sst` — Synschrosqueezed transform

matrix

Synschrosqueezed transform, specified as a matrix. `sst` is the output from the `wsst` function.

`f` — Synschrosqueezed transform frequencies

vector

Synchrosqueezed transform frequencies corresponding to the rows of the synchrosqueezed transform, specified as a vector. The number of elements in the frequency vector is equal to the number of rows in the `sst` input. If you specify `f`, you must also specify `freqrange`.

`freqrange` — Frequency range

two-element vector

Frequency range for which to return inverse synchrosqueezed transform values, specified as a two-element vector. The values of `freqrange` must be in the range of the values of the frequencies, `f`. The first and second elements of `freqrange` define the start and end of the frequency range, where the frequency values in that range must be positive and strictly increasing. If you specify `freqrange`, you must also specify `f`.

`iridge` — Time-frequency ridge row indices

vector or matrix

Time-frequency ridge row indices of the synchrosqueezed transform specified as a vector or matrix. `iridge` is the output of the `wsstridge` function. If `iridge` is a matrix, `iwsst` inverts the synchrosqueezed transform along the first column of `iridge`. Then, it iteratively reconstructs along subsequent columns of `iridge`. The sizes of `iridge` and the `xrec` output are the same.

`wav` — Analytic wavelet

`'analmorl'` (default) | `'bump'`

Analytic wavelet used to compute the inverse synchrosqueezed transform, specified as `'analmorl'` or `'bump'`. These character vectors specify the analytic Morlet and bump wavelet, respectively. You must use the same wavelet in the reconstruction that you used to compute the synchrosqueezed transform, `sst`.

Example:

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: `'NumFrequencyBins',12`

`NumFrequencyBins` — Number of frequency bins
16 (default) | positive integer

Number of additional frequency bins to include on either side of each `iridge` index bin, specified as the comma-separated pair consisting of `'NumFrequencyBins'` and a positive integer. If the number of additional bins exceeds the number of frequency bins available at a particular time step, `iwsst` truncates the reconstruction at the first or last frequency bin. The default, `16`, is one half the default number of voices per octave.

You can include this `Name,Value` argument in any position in a syntax. To specify this argument, you also specify `iridge`, which is the output of `wsstridge`. You cannot include a frequency, `f` and frequency range, `freqrange`, if you include the number of frequency bins.

Output Arguments

`xrec` — Inverse synchrosqueezed transform
vector or matrix

Inverse synchrosqueezed transform, returned as a vector or matrix. If you do not specify an `iridge` input, `xrec` is a column vector with the same number of rows as `sst`. If you specify an `iridge` input, `xrec` is the same size as `iridge`.

References

[1] Daubechies, I., I., J. Lu, and H. T. Wu. "Synchrosqueezed Wavelet Transforms: an Empricial Mode Decomposition-like Tool", *Applied and Computational Harmonic Analysis*. Vol. 30(2), pp. 243–261.

[2] Thakur, G., E. Brevdo, N. S. Fučkar, and H. T. Wu. "The Synchrosqueezing algorithm for time-varying spectral analysis: robustness properties and new paleoclimate applications." *Signal Processing*. Vol. 93, pp. 1079–1094.

See Also

`wsst` | `wsstridge`

Topics

“Time-Frequency Reassignment and Mode Extraction with Synchrosqueezing”
“Wavelet Synchrosqueezing”

Introduced in R2016a

laurmat

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`

Introduced before R2006a

laurpoly

Laurent polynomials constructor

Syntax

```
P = laurpoly(C,d)
P = laurpoly(C,'dmin',d)
P = laurpoly(C,'dmax',d)
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`

Introduced before R2006a

leaves

Determine terminal nodes

Syntax

```
N = leaves(T)
[N,K] = leaves(T,'sort')
N = leaves(T,'dp')
[N,K] = leaves(T,'sortdp')
[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.

The nodes are ordered from left to right as in tree *T*.

`[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.

`N = leaves(T,'dp')` returns a matrix `N`, which contains the depths and positions of terminal nodes.

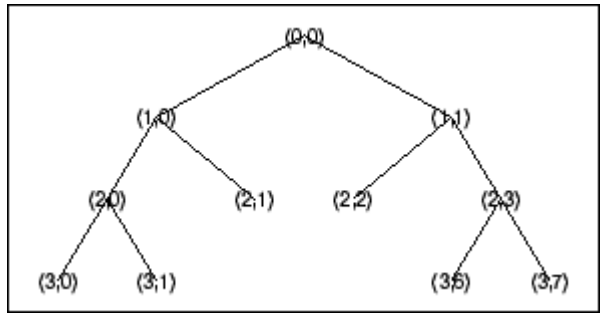
`N(i,1)` is the depth of the `i`-th terminal node, and `N(i,2)` is the position of the `i`-th terminal node.

`[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

noleaves | tnodes

Introduced before R2006a

liftfilt

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`

Introduced before R2006a

liftwave

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
`'lazy'`
`'haar'`
`'db1'`, `'db2'`, `'db3'`, `'db4'`, `'db5'`, `'db6'`, `'db7'`, `'db8'`
`'sym2'`, `'sym3'`, `'sym4'`, `'sym5'`, `'sym6'`, `'sym7'`, `'sym8'`

Cohen-Daubechies-Feauveau wavelets
`'cdf1.1'`,`'cdf1.3'`,`'cdf1.5'`
`'cdf3.1'`,`'cdf3.3'`,`'cdf3.5'`
`'cdf5.1'`,`'cdf5.3'`,`'cdf5.5'`
`'cdf2.2'`,`'cdf2.4'`,`'cdf2.6'`
`'cdf4.2'`,`'cdf4.4'`,'cdf4.6'
`'cdf6.2'`,`'cdf6.4'`,`'cdf6.6'`

WNAME Values
'biorX.Y'
'rbioX.Y'
'bs3'
'rbs3'
'9.7'
'r9.7'

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`

Introduced before R2006a

localmax

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 beginning 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.

Default: `true`

Output Arguments

lmaxima

Matrix with local maxima chains. `lmaxima` only has nonzero entries at the locations of local maxima in the absolute values of `inputmatrix`. Denote the row index of `lmaxima` by `R`. You can determine the value of `lmaxima` at a local maximum in row `R` as follows:

- If `R>initRow`, the value of `lmaxima` at a local maximum is 1.
- If `R=initRow`, the value of `lmaxima` at a local maximum is the column index in row `R`.
- If `R<initRow`, the value of `lmaxima` at a local maximum in row `R` is the column index of the nearest local maximum in row `R+1`.

  To illustrate this, if `inputmatrix` is:

```
 3     2     5     3
 4     6     3     2
 4     4     7     4
 4     6     2     2
```

  `lmaxima` with `initRow = 4` and `regflag = false` is:

```
     0     0     2     0
     0     3     0     0
```

```
     0     0     2     0
     0     2     0     0
```

`lmaxima` with `initRow = 3` and `regflag = false` is:

```
     0     0     2     0
     0     3     0     0
     0     0     3     0
     0     1     0     0
```

- 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

Local Maxima of a Matrix

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)
```

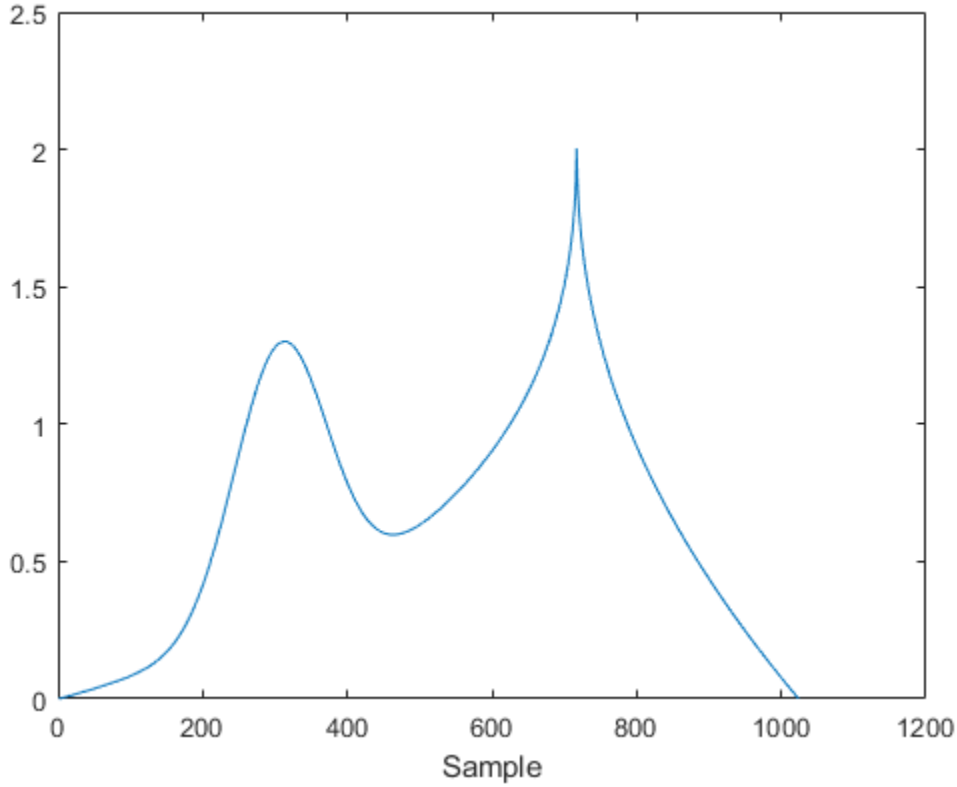
CWT Coefficient Moduli and Maxima Lines

Determine the local maxima from the CWT of the `cuspamax` signal using the default Morse wavelet. Plot the CWT coefficient moduli and maxima lines.

```
load cuspamax;
```

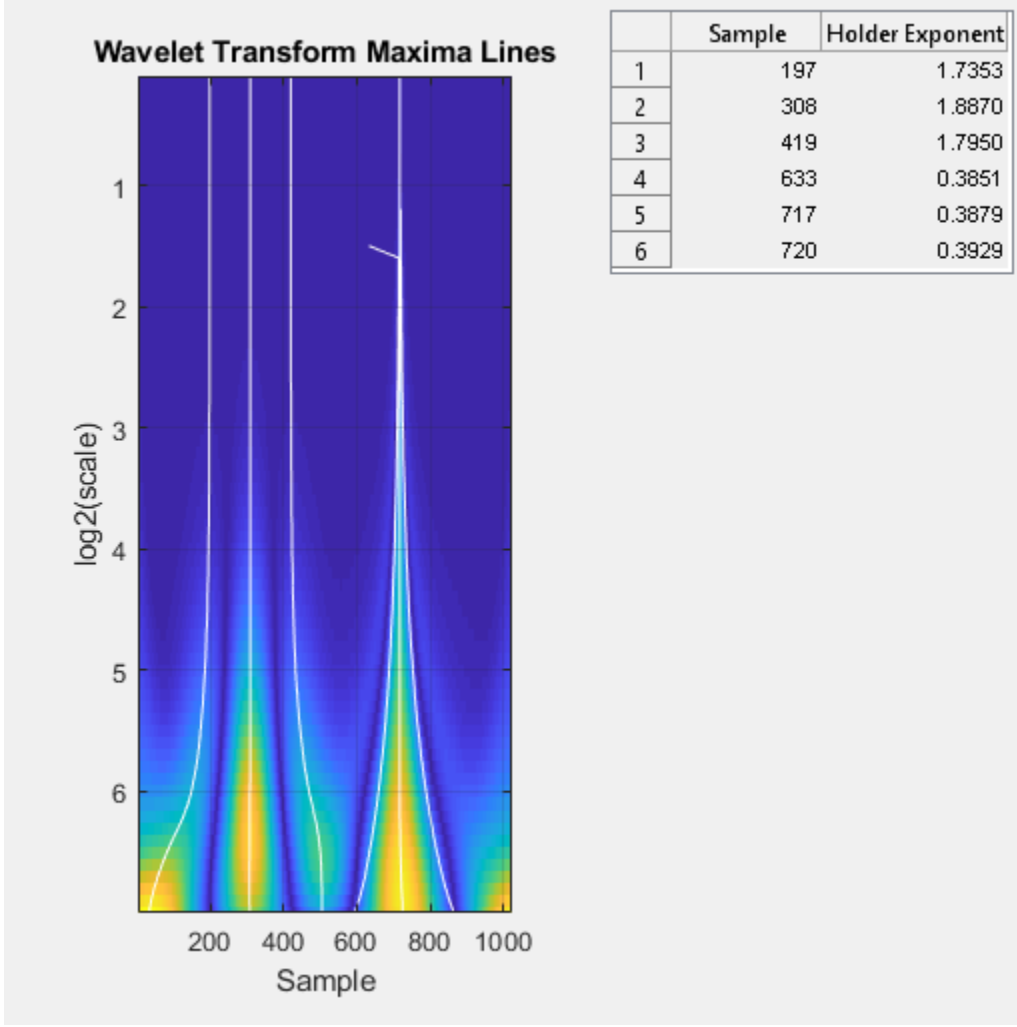
Plot the `cuspamax` signal and notice the shape of the signal near samples 300 and 700. The signal shows a cusp near sample 700.

```
plot(cuspamax);
xlabel('Sample');
```



Plot the wavelet transform modulus maxima and note the local Holder exponent values at samples 308 and 717.

```
wtmm(cuspamax,'ScalingExponent','local');
```



	Sample	Holder Exponent
1	197	1.7353
2	308	1.8870
3	419	1.7950
4	633	0.3851
5	717	0.3879
6	720	0.3929

Holder exponent values indicate the strength of the singularities in a signal. Signal locations where the local Holder exponent is 0 are discontinuous at that location. Locations with Holder exponenets greater than or equal to 1 are differentiable. Holder exponent values less than but close to 1 indicate that the signal at the location is almost differentiable. The closer the Holder exponent value is to 0, the stronger the singularity.

The Holder exponent at sample 308 is 1.9 and at sample 717 is 0.39. The low Holder value at sample 717 confirms that the signal is not differentiable and has a fairly strong singularity at that point.

Introduced in R2008a

ls2filt

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`

Introduced before R2006a

lsinfo

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

$P(z) = C(1)*z^d + C(2)*z^(d-1) + ... + C(m)*z^(d-m+1)$

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`

Introduced before R2006a

lwt

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`

Introduced before R2006a

lwt2

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`.

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
```

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 .

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`

Introduced before R2006a

lwtcoef

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

Introduced before R2006a

lwtcoef2

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.

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
```

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.

See Also

`ilwt2` | `lwt2`

Introduced before R2006a

mdwtcluster

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 and Machine Learning Toolbox™

`'dirDec'`	`'r'` (row) or `'c'` (column). Default value is `'r'`.
`'level'`	Level of the DWT decomposition. Default value is: `level=fix(log2(size(X,d)))` 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 and Machine Learning Toolbox `pdist` function. Default value is `'euclidean'`.
`'linkage'`	See Statistics and Machine Learning 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`:

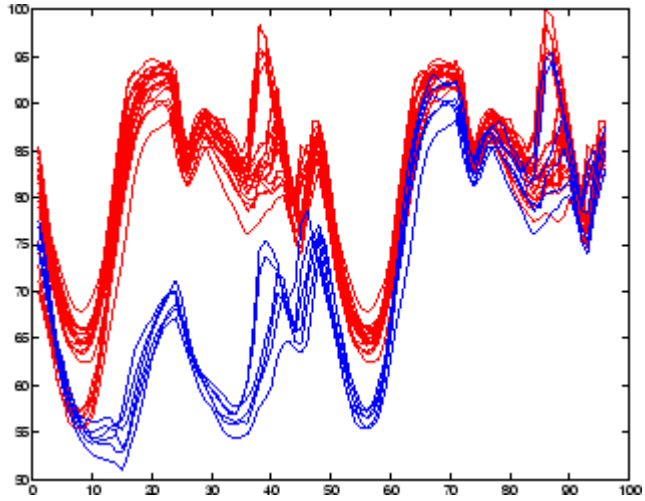
`S.Idx(:,j)`	Contains the cluster numbers obtained from the hierarchical cluster tree (see `cluster` in the Statistics and Machine Learning Toolbox software).
`S.Incons(:,j)`	Contains the inconsistent values of each non-leaf node in the hierarchical cluster tree (see Statistics and Machine Learning Toolbox software function `inconsistent`).
`S.Corr(j)`	Contains the cophenetic correlation coefficients of the partition (see Statistics and Machine Learning 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`

Introduced in R2008a

mdwtdec

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).

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

`mdwtrec` | `wavedec`

Introduced in R2007a

mdwtrec

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 $0 \le LEV \le LEVDEC$.

- 'd' or 'cd', LEV must be such that 1 ≤ LEV ≤ 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`

Introduced in R2007a

measerr

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 Arguments

`X`

`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 Arguments

PSNR

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.

Examples

Measure Approximation Quality in an RGB image

Approximate an RGB image and calculate approximation quality metrics.

Load an image.

```
X = imread('africasculpt.jpg');
```

Define the image approximation.

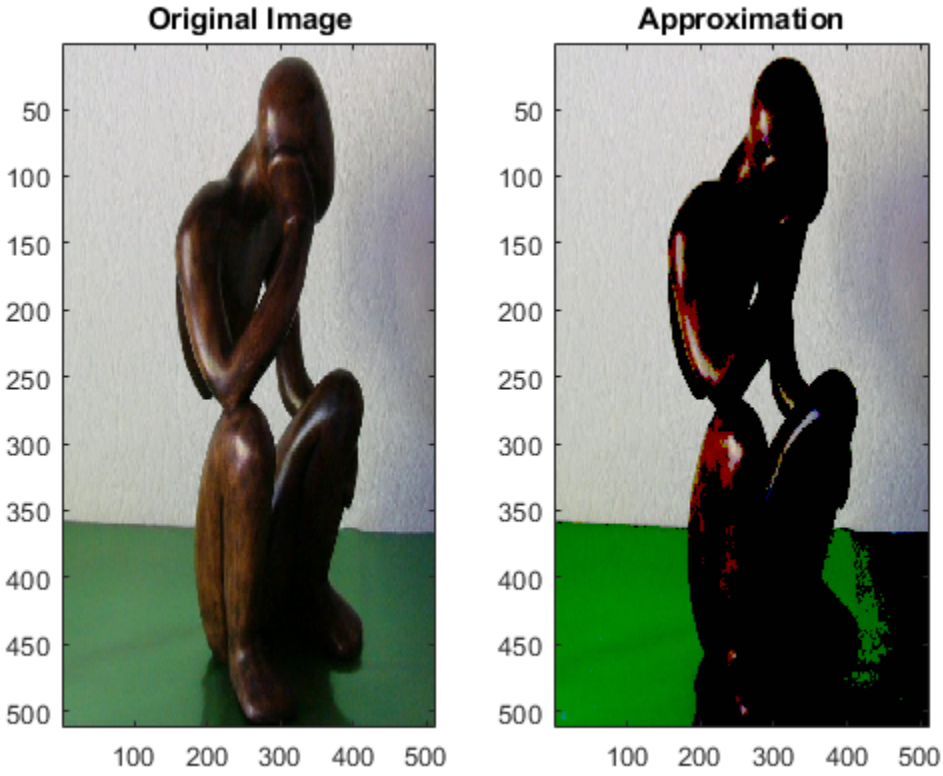
```
Xapp = X;
Xapp(X<=100) = 1;
```

Calculate the approximation quality metrics.

```
[psnr,mse,maxerr,L2rat] = measerr(X,Xapp)

psnr =

   17.5287


mse =

   1.1487e+03


maxerr =

    99


L2rat =

    0.9398
```

Display the image and its approximation.

```
figure
subplot(1,2,1)
image(X)
title('Original Image')
subplot(1,2,2)
image(Xapp)
title('Approximation')
```



Approximation Quality Metrics

Approximate an image and calculate approximation quality metrics.

Load an image.

```
load woman;
```

Define the image approximation.

```
Xapp = X;
Xapp(X<=50) = 1;
```

Calculate approximation quality metrics.

```
[psnr,mse,maxerr,L2rat] = measerr(X,Xapp)
```

```
psnr = 26.5884
```

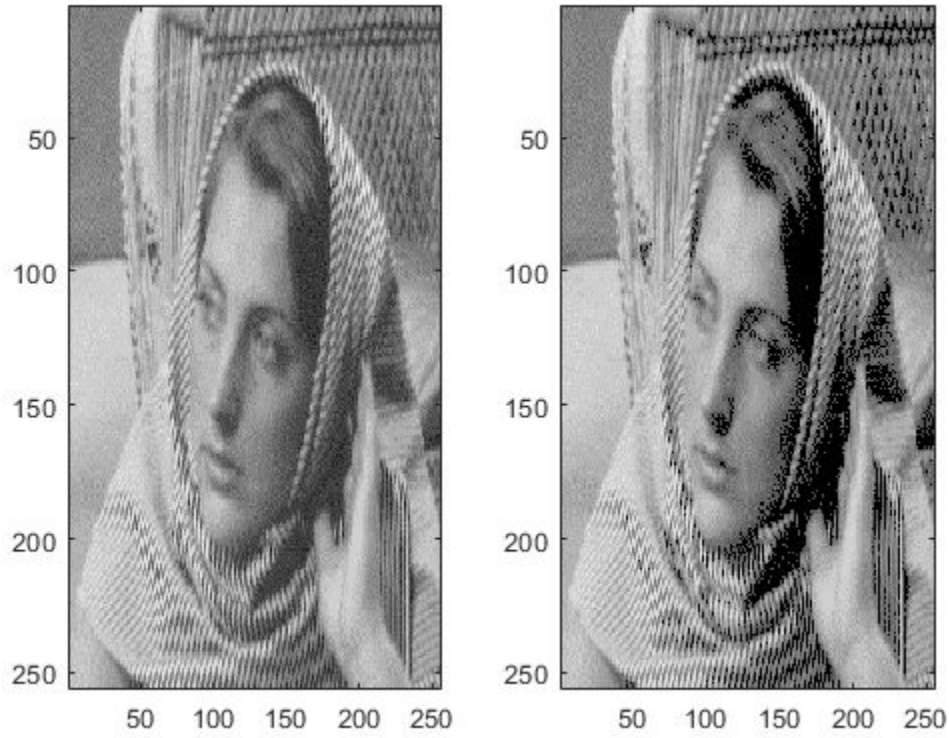
```
mse = 142.6411
```

```
maxerr = 47
```

```
L2rat = 0.9910
```

Create a plot.

```
figure; colormap(map);
subplot(1,2,1); image(X);
subplot(1,2,2); image(Xapp);
```



- “Wavelet Data Compression”
- “Wavelet Denoising and Nonparametric Function Estimation”

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:

$$\frac{||X-Y||^2}{N}$$

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`

Topics

"Wavelet Data Compression"
"Wavelet Denoising and Nonparametric Function Estimation"

Introduced in R2010b

mexihat

Mexican hat (Ricker) 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]`. The Mexican hat wavelet is also known as the Ricker wavelet.

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.

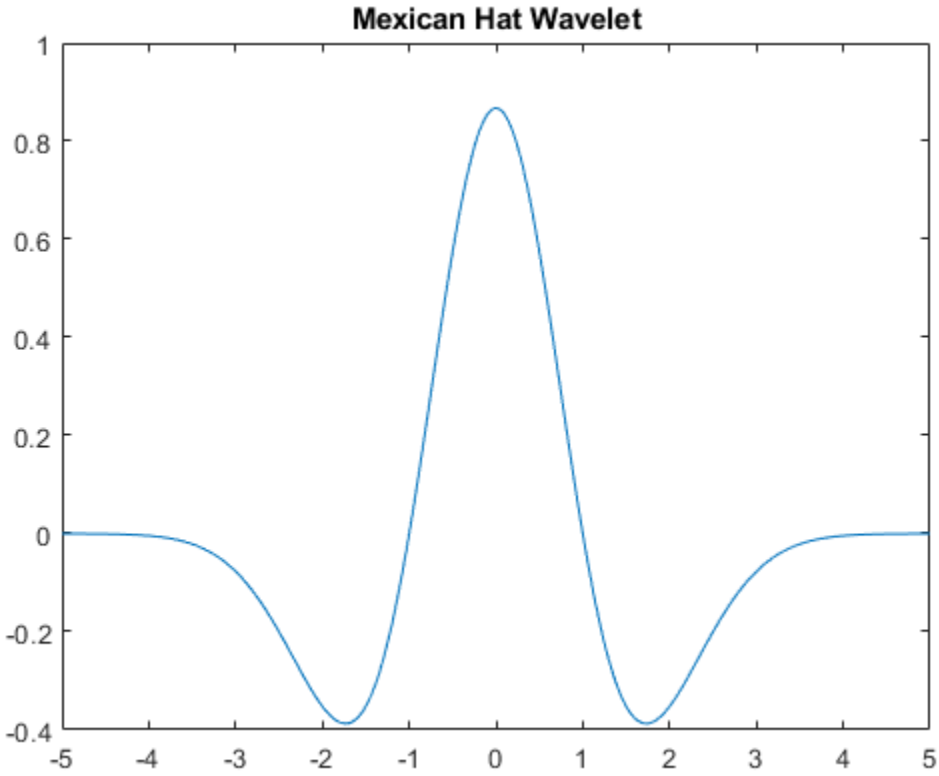
Note You can use `gauswavf` to obtain a second order derivative of a Gaussian wavelet. If you use the negative of this normalized derivative, the resulting wavelet resembles the Mexican hat wavelet.

Examples

Mexican Hat Wavelet

Create a Mexican hat wavelet with support on [-5,5]. Use 1,000 sample points. Plot the result.

```
lb = -5;
ub = 5;
N = 1000;
[psi,xval] = mexihat(lb,ub,N);
plot(xval,psi)
title('Mexican Hat Wavelet');
```



See Also

`waveinfo`

Introduced before R2006a

meyer

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.

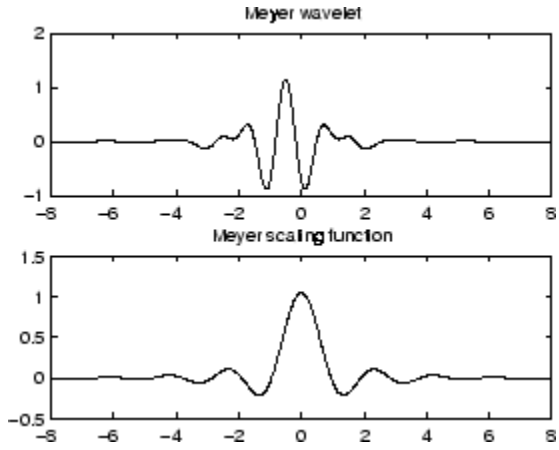
The Meyer wavelet and scaling function are defined in the frequency domain.

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')
```



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`

Introduced before R2006a

meyeraux

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`

Introduced before R2006a

mlpt

Multiscale local 1-D polynomial transform

Syntax

```
[coefs,T,coefsPerLevel,scalingMoments] = mlpt(x,t)
[coefs,T,coefsPerLevel,scalingMoments] = mlpt(x,t,numLevel)
[coefs,T,coefsPerLevel,scalingMoments] = mlpt(x)
[coefs,T,coefsPerLevel,scalingMoments] = mlpt(___,Name,Value)
```

Description

`[coefs,T,coefsPerLevel,scalingMoments] = mlpt(x,t)` returns the multiscale local polynomial 1-D transform (MLPT) of input signal `x` sampled at the sampling instants, `t`. If `x` or `t` contain `NaN`s, the union of the `NaN`s in `x` and `t` is removed before obtaining the `mlpt`.

`[coefs,T,coefsPerLevel,scalingMoments] = mlpt(x,t,numLevel)` returns the transform for `numLevel` resolution levels.

`[coefs,T,coefsPerLevel,scalingMoments] = mlpt(x)` uses uniform sampling instants for `x` as the time instants if `x` does not contain `NaN`s. If `x` contains `NaN`s, the `NaN`s are removed from `x` and the nonuniform sampling instants are obtained from the numeric elements of `x`.

`[coefs,T,coefsPerLevel,scalingMoments] = mlpt(___,Name,Value)` specifies `mlpt` properties using one or more `Name,Value` pair arguments and any of the previous input arguments.

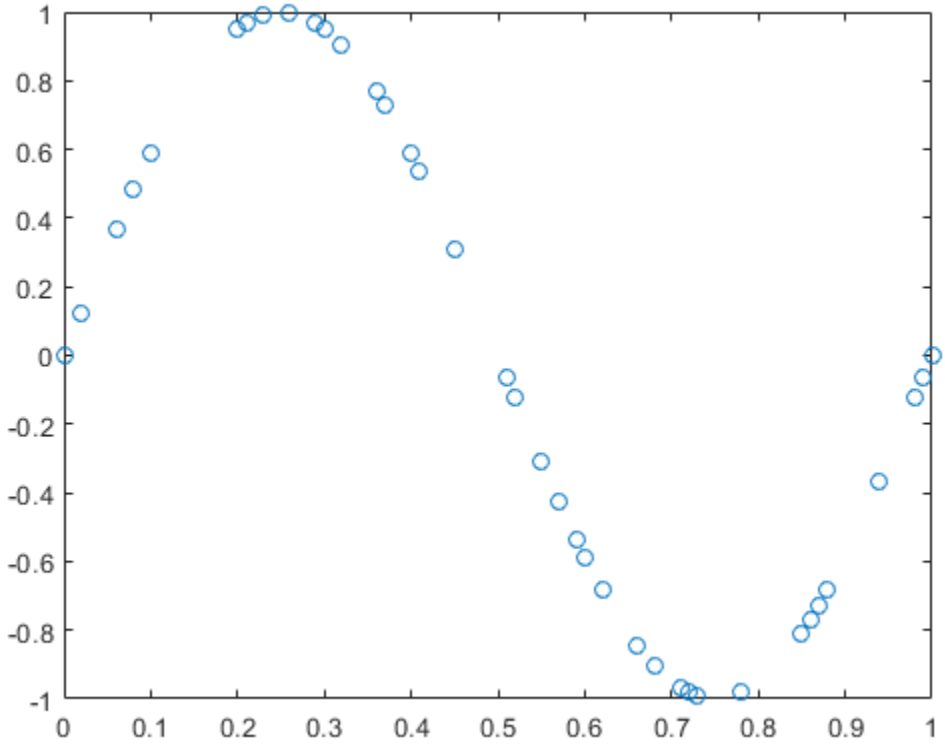
Examples

Multiscale Local 1-D Polynomial Transform and Inverse Trasform

Create a signal with nonuniform sampling and verify good reconstruction when performing the `mlpt` and `imlpt`.

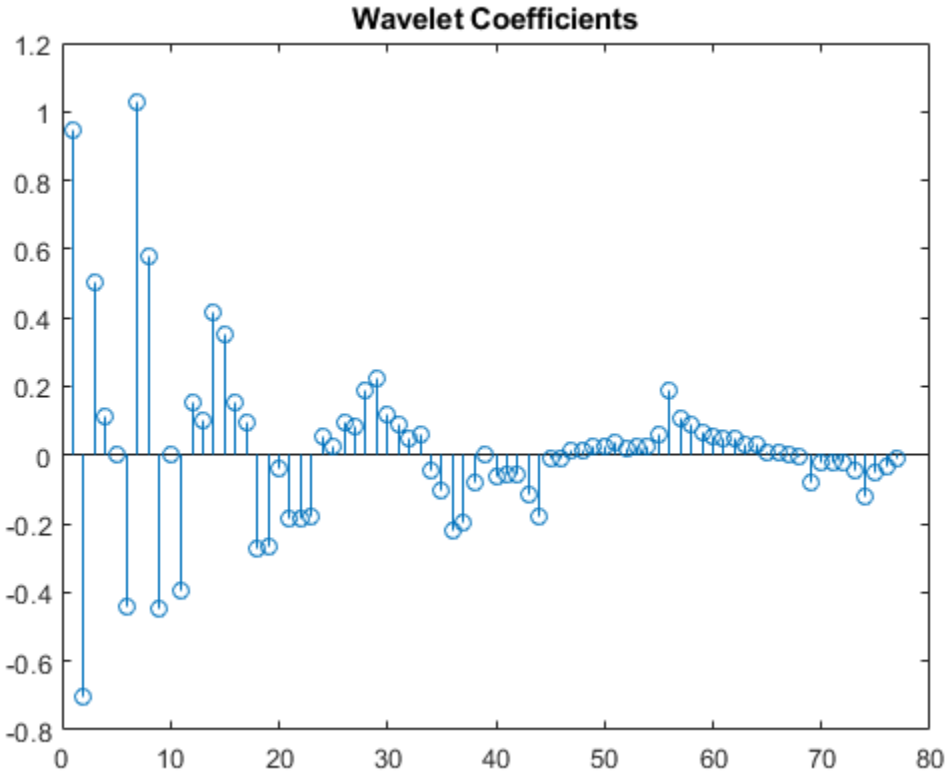
Create and plot a sine wave with non-uniform sampling.

```
timeVector = 0:0.01:1;
sineWave = sin(2*pi*timeVector)';

samplesToErase = randi(100,100,1);
sineWave(samplesToErase) = [];
timeVector(samplesToErase) = [];

figure(1)
plot(timeVector,sineWave,'o')
hold on
```



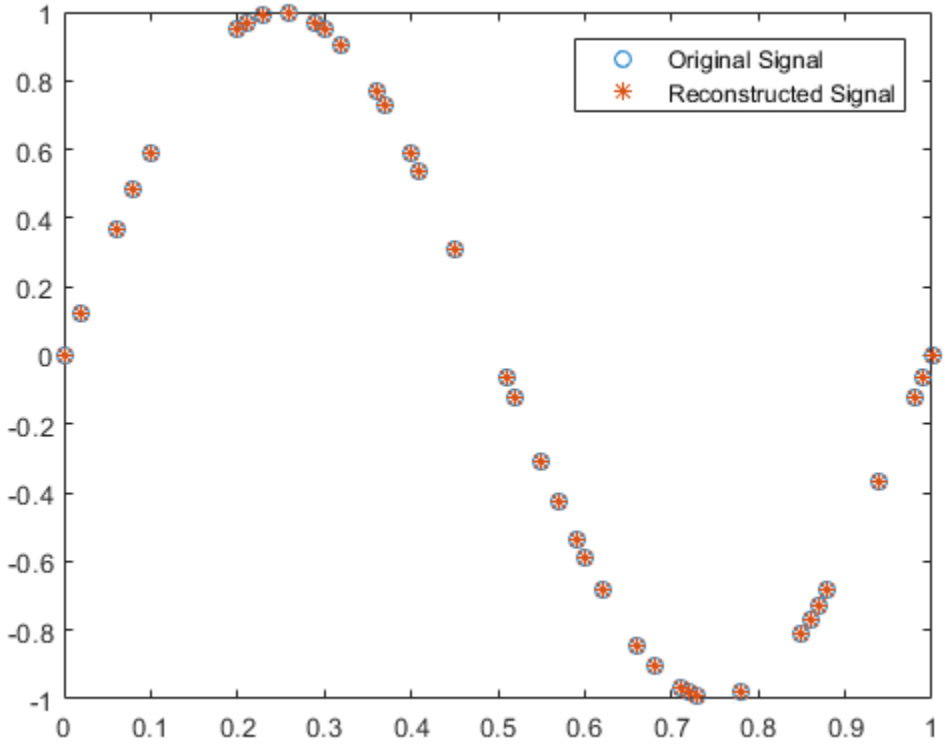
Perform the multiscale local 1-D polynomial trasform (`mlpt`) on the signal. Visualize the coefficients.

```
[coefs,T,coefsPerLevel,scalingMoments] = mlpt(sineWave,timeVector);

figure(2)
stem(coefs)
title('Wavelet Coefficients')
```



Perform the inverse multiscale local 1-D polynomial trasform (`imlpt`) on the coefficients. Visualize the reconstructed signal.

```
y = imlpt(coefs,T,coefsPerLevel,scalingMoments);

figure(1)
plot(T,y,'*')
legend('Original Signal','Reconstructed Signal')
hold off
```



Look at the total error to verify good reconstruction.

```
reconstructionError = sum(abs(y-sineWave))
```

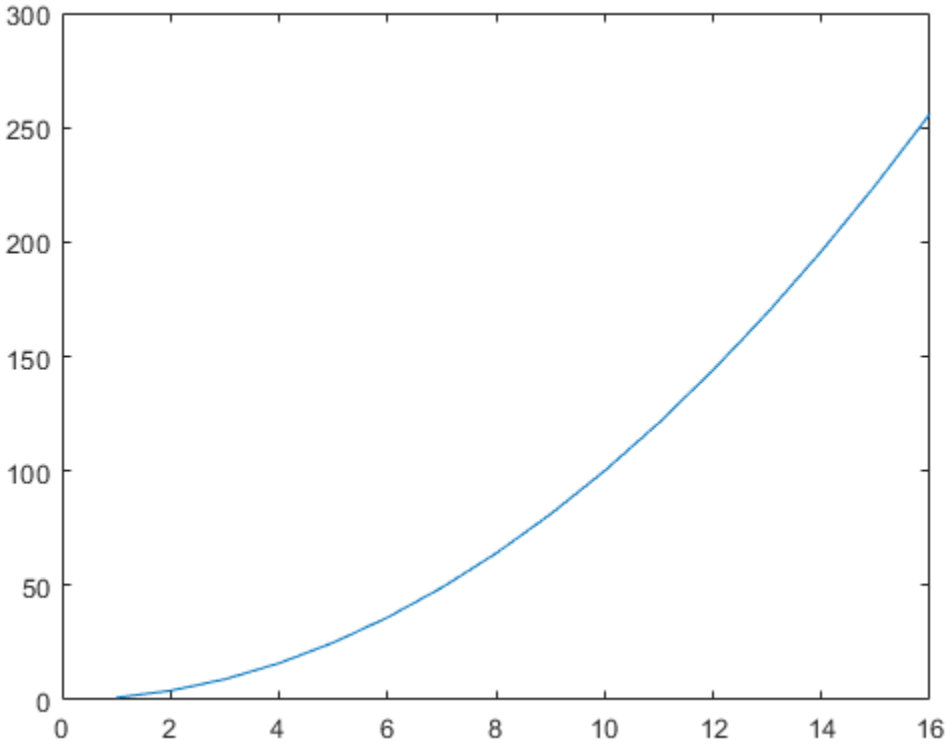
```
reconstructionError = 1.7552e-15
```

Specify Nondefault Dual Moments

Specify nondefault dual moments by using the `mlpt` function. Compare the results of analysis and synthesis using the default and nondefault dual moments.

Create an input signal and visualize it.

```
T = (1:16)';
x = T.^2;
plot(x)
hold on
```



Perform the forward and inverse transform for the input signal using the default and nondefault dual moments.

```
[w2,t2,nj2,scalingmoments2] = mlpt(x,T);
y2 = imlpt(w2,t2,nj2,scalingmoments2);

[w3,t3,nj3,scalingmoments3] = mlpt(x,T,'dualmoments',3);
y3 = imlpt(w3,t3,nj3,scalingmoments3,'dualmoments',3);
```

Plot the reconstructed signal and verify perfect reconstruction using both the default and nondefault dual moments.

```
plot(y2,'o')
plot(y3,'*')
legend('Original Signal', ...
       'DualMoments = 3', ...
       'DualMoments = 2 (Default)');

fprintf('\nMean Reconstruction Error:\n');
```

```
Mean Reconstruction Error:
```

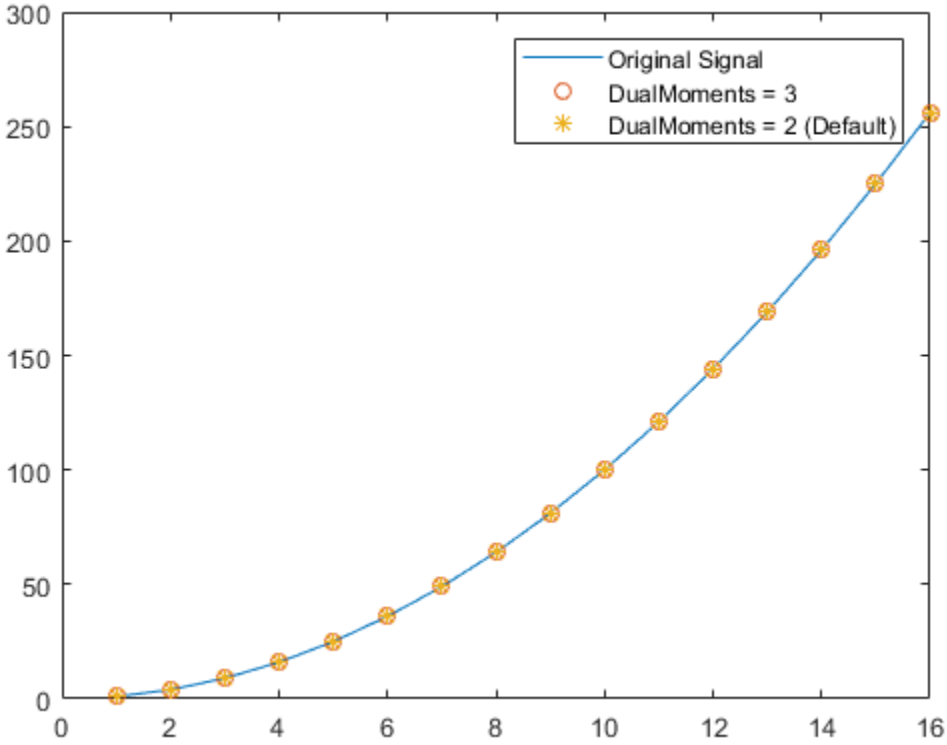
```
fprintf('  - Nondefault dual moments: %0.2f\n',mean(abs(y3-x)));
```

```
  - Nondefault dual moments: 0.00
```

```
fprintf('  - Default dual moments: %0.2f\n\n',mean(abs(y2-x)));
```

```
  - Default dual moments: 0.00
```

```
hold off
```



Specify Nondefault Resolution Levels

Resolution levels are the number of cascaded local polynomial smoothing operations. The details at each resolution level are obtained by predicting one half the samples based on a local polynomial interpolation of the other half. The difference between the predicted and actual values are the details at each resolution level. The scaling coefficients at each coarser resolution level are smoother versions of the higher resolution scaling coefficients. Only the final-level scaling coefficients are retained.

Increasing the number of resolution levels enables you to analyze narrowband coefficients for a computational and memory cost.

Create a dual-tone input signal, `x`, that contains high and low frequencies.

```
fs = 1000;
t = (0:1/fs:10)';
x = sin(499*pi.*t) + sin(2*pi.*t);
```

Use `mlpt` to obtain coefficients for minimum and maximum resolution levels. Print the computation time.

```
tic
[w1,~,nj1,m1] = mlpt(x,t,1);
computationTime1 = toc;
fprintf('Level one computation time: %0.2f\n',computationTime1)
```

```
Level one computation time: 8.57
```

```
tic
[w13,~,nj13,m13] = mlpt(x,t,13);
computationTime13 = toc;
fprintf('Level thirteen computation time: %0.2f\n',computationTime13)
```

```
Level thirteen computation time: 12.84
```

Use Default Time Instants

If your time instants are not known or specified, you can calculate the MLPT using default time instants.

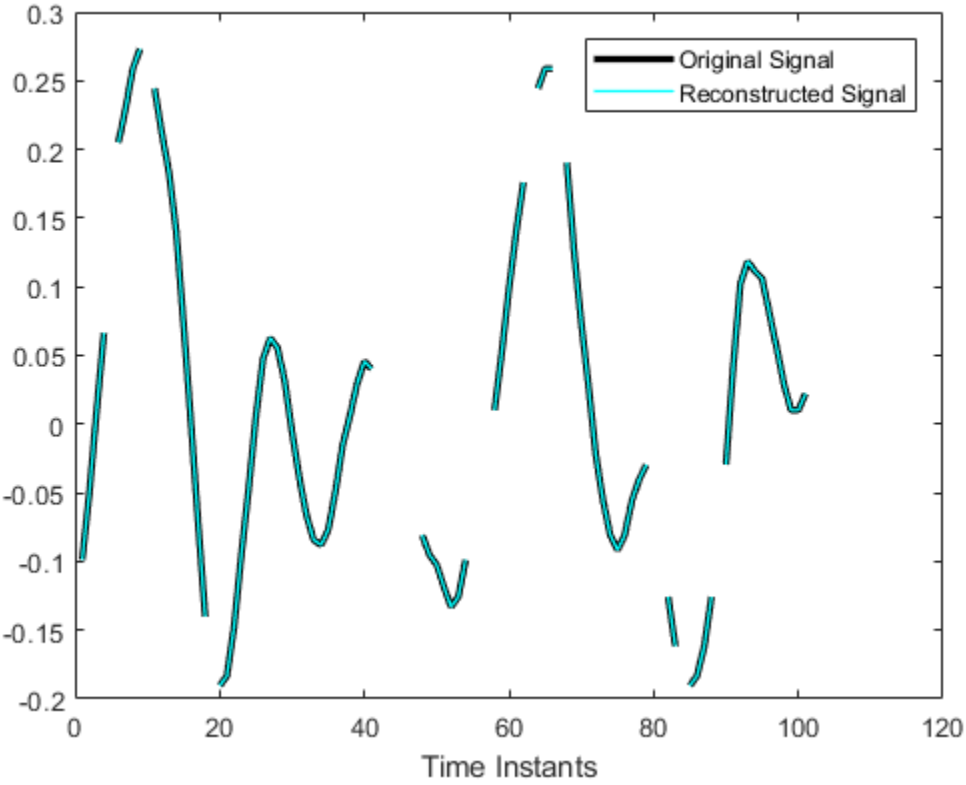
Load a data signal corrupted with NaNs and with unknown time instants. Calculate the MLPT without specifying time instants. The resulting implied time instants is a vector of valid indices of the corrupted signal.

```
load(fullfile(matlabroot,'examples','wavelet','CorruptedData.mat'));

[w,t,nj,scalingMoments] = mlpt(yCorrupt);
```

Calculate the inverse MLPT and visualize the results. Reinsert NaNs to visualize gaps in the signal.

```
z = imlpt(w,t,nj,scalingMoments);
```

```
zToPlot = NaN(numel(yCorrupt),1);
zToPlot(t) = z;

plot(yCorrupt,'k','LineWidth',2.5)
hold on
plot(zToPlot,'c','LineWidth',1)
hold off
legend('Original Signal','Reconstructed Signal')
xlabel('Time Instants')
```



- Smoothing Nonuniformly Sampled Data

Input Arguments

`x` — Input signal

vector | matrix

Input signal, specified as a vector or matrix.

- matrix — `x` must have at least two rows. `mlpt` operates independently on each column of `x`. The number of elements in `t` must equal the row dimension of `x`. Any `NaN`s in the columns of `x` must occur in the same rows.
- vector — `x` and `t` must have the same number of elements.

Data Types: `double`

`t` — Sampling instants

vector | `duration` array | `datetime` array

Sampling instants corresponding to the input signal, specified as a vector, `duration` array, or `datetime` array of monotonically increasing real values. The default value depends on the length of the input signal, `x`.

Data Types: `double` | `duration` | `datetime`

`numLevel` — Number of resolution levels

positive integer

Number of resolution levels, specified as a positive integer. The maximum value of `numLevel` depends on the shape of the input signal, `x`:

- matrix — `floor(log2(size(x,1)))`
- vector — `floor(log2(length(x)))`

If `numLevel` is not specified, `mlpt` uses the maximum value.

Data Types: `double`

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: `'DualMoments',3` computes a transform using three dual vanishing moments.

`DualMoments` — Number of dual vanishing moments

`2` (default) | `3` | `4`

Number of dual vanishing moments in the lifting scheme, specified as the comma-separated pair consisting of `'DualMoments'` and `2`, `3` or `4`.

Data Types: `double`

`PrimalMoments` — Number of primal vanishing moments

`2` (default) | `3` | `4`

Number of primal vanishing moments in the lifting scheme, specified as the comma-separated pair consisting of `'PrimalMoments'` and `2`, `3`, or `4`.

Data Types: `double`

`Prefilter` — Prefilter before `mlpt`

`'Haar'` (default) | `'UnbalancedHaar'` | `'None'`

Prefilter before `mlpt` operation, specified as the comma-separated pair consisting of `'Prefilter'` and`'Haar'` [1], `'UnbalancedHaar'`, or `'None'`.

Data Types: `char` | `string`

Output Arguments

`coefs` — MLPT coefficients

vector | matrix

MLPT coefficients, returned as a vector or matrix of coefficients, depending on the level to which the transform is calculated. `coefs` contains the approximation and detail coefficients.

Data Types: `double`

`T` — Sampling instants corresponding to output

vector | `duration` array

Sampling instants corresponding to output, returned as a vector or `duration` array of sample times obtained from `x` and `t`. The `imlpt` function requires `T` as an input. If the input `t` is a `datetime` or `duration` array, `t` is converted to units that allow for the stable computation of the `mlpt` and `imlpt`. Then `T` is returned as a `duration` array.

Data Types: `double` | `duration`

`coefsPerLevel` — Coefficients per resolution level

vector

Coefficients per resolution level, returned as a vector containing the number of coefficients at each resolution level in `coefs`. The elements of `coefsPerLevel` are organized as follows:

- `coefsPerLevel(1)` — Number of approximation coefficients at the coarsest resolution level.
- `coefsPerLevel(i)` — Number of detail coefficients at resolution level `i`, where `i = numLevel - i + 2` for `i = 2, ..., numLevel + 1`.

The smaller the index `i`, the lower the resolution. The MLPT is two times redundant in the number of detail coefficients, but not in the number of approximation coefficients.

Data Types: `double`

`scalingMoments` — Scaling function moments

matrix

Scaling function moments, returned as a `length(coefs)`-by-`P` matrix, where `P` is the number of primal moments specified by the `PrimalMoments` name-value pair.

Data Types: `double`

Algorithms

Maarten Jansen developed the theoretical foundation of the multiscale local polynomial transform (MLPT) and algorithms for its efficient computation [1][2][3]. The MLPT uses a lifting scheme, wherein a kernel function smooths fine-scale coefficients with a given

bandwidth to obtain the coarser resolution coefficients. The `mlpt` function uses only local polynomial interpolation, but the technique developed by Jansen is more general and admits many other kernel types with adjustable bandwidths [2].

References

[1] Jansen, M. "Multiscale Local Polynomial Smoothing in a Lifted Pyramid for Non-Equispaced Data". *IEEE Transactions on Signal Processing*. Vol. 61, Number 3, 2013, pp. 545–555.

[2] Jansen, M., and M. Amghar. "Multiscale local polynomial decompositions using bandwidths as scales". *Statistics and Computing* (forthcoming). 2016.

[3] Jansen, M., and Patrick Oonincx. *Second Generation Wavelets and Applications*. London: Springer, 2005.

See Also

`imlpt` | `mlptdenoise` | `mlptrecon`

Topics

Smoothing Nonuniformly Sampled Data

Introduced in R2017a

mlptdenoise

Denoise signal using multiscale local 1-D polynomial transform

Syntax

```
y = mlptdenoise(x,t)
y = mlptdenoise(x,t,numLevel)
y = mlptdenoise(___,Name,Value)
[y,T] = mlptdenoise(___)
[y,T,thresholdedCoefs] = mlptdenoise(___)
[y,T,thresholdedCoefs,originalCoefs] = mlptdenoise(___)
```

Description

`y = mlptdenoise(x,t)` returns a denoised version of input signal `x` sampled at the sampling instants, `t`. If `x` or `t` contain `NaN`s, the union of the `NaN`s in `x` and `t` is removed before obtaining the `mlpt`.

`y = mlptdenoise(x,t,numLevel)` denoises `x` down to `numLevel`.

`y = mlptdenoise(___,Name,Value)` specifies `mlpt` properties using one or more `Name,Value` pair arguments, and any of the previous syntaxes

`[y,T] = mlptdenoise(___)` also returns the time instants for the denoised signal.

`[y,T,thresholdedCoefs] = mlptdenoise(___)` also returns the thresholded multiscale local 1–D polynomial transform coefficients.

`[y,T,thresholdedCoefs,originalCoefs] = mlptdenoise(___)` also returns the original multiscale local 1–D polynomial transform coefficients.

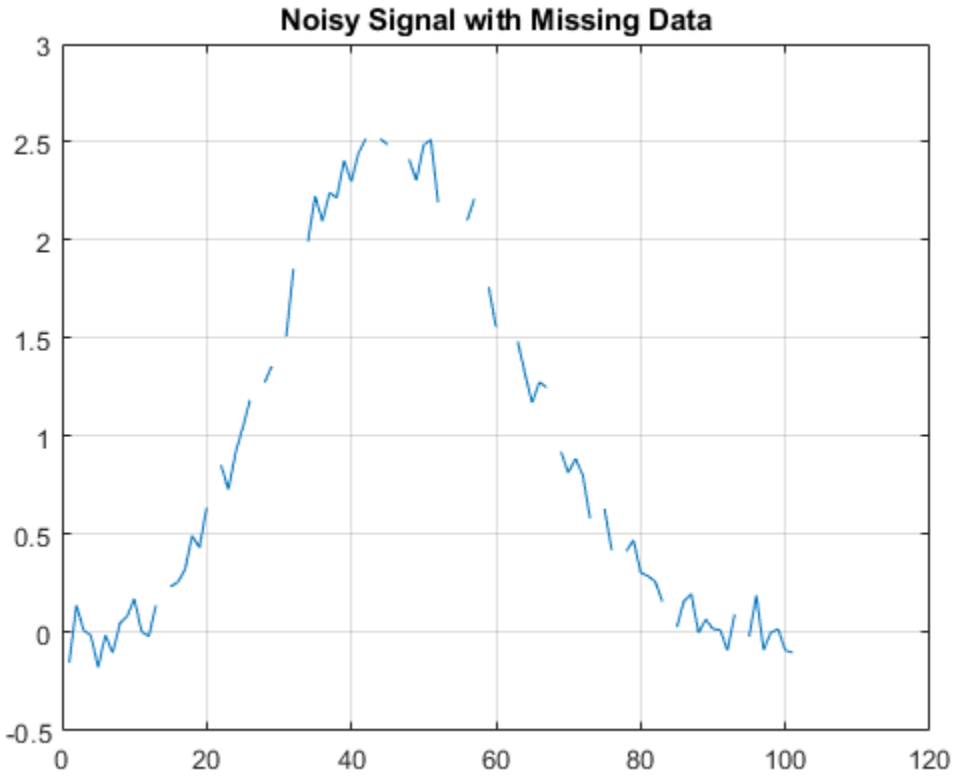
Examples

Specify Nondefault Denoising Method

Denoise a nonuniformly sampled spline signal with added noise using median smoothing and two primal vanishing moments. The nonuniformity of the signal is indicated by NaNs (missing data).

Load the data to your workspace and visualize it.

```
load nonuniformspline
plot(splinenoise)
grid on
title('Noisy Signal with Missing Data')
```

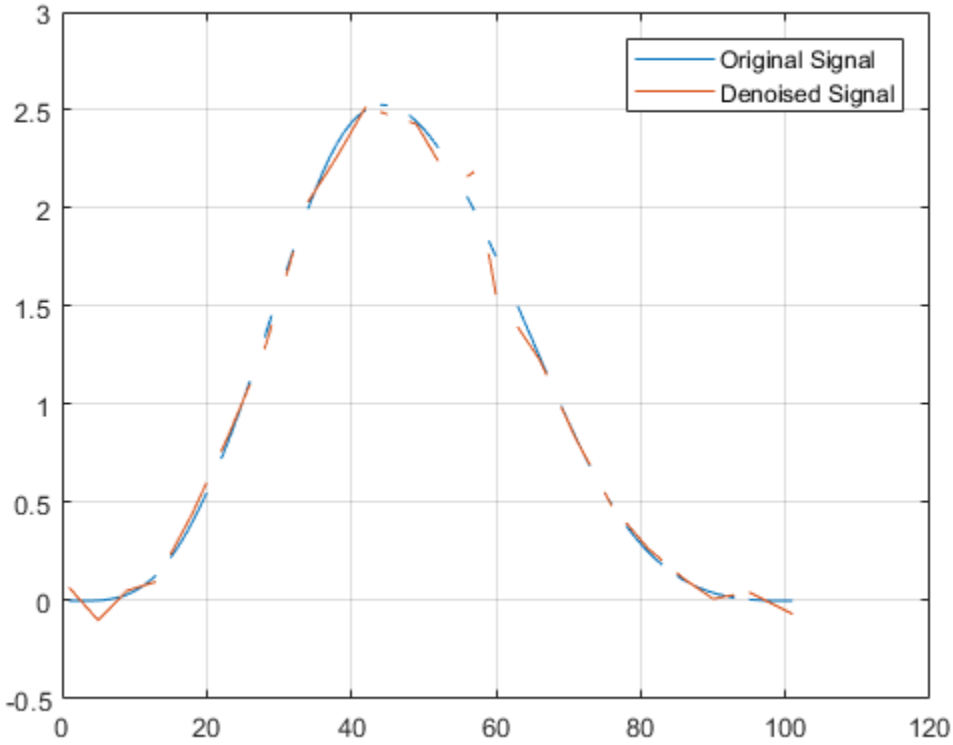


Denoise the data using the median denoising method.

```
xden = mlptdenoise(splinenoise,[],'DenoisingMethod','median');
```

Replace the original missing data in the correct position for plotting purposes. Visualize the original and denoised signals.

```
denoisedsig = NaN(size(splinenoise));
denoisedsig(~isnan(splinenoise)) = xden;
figure
plot([splinesig denoisedsig])
grid on
legend('Original Signal','Denoised Signal');
```

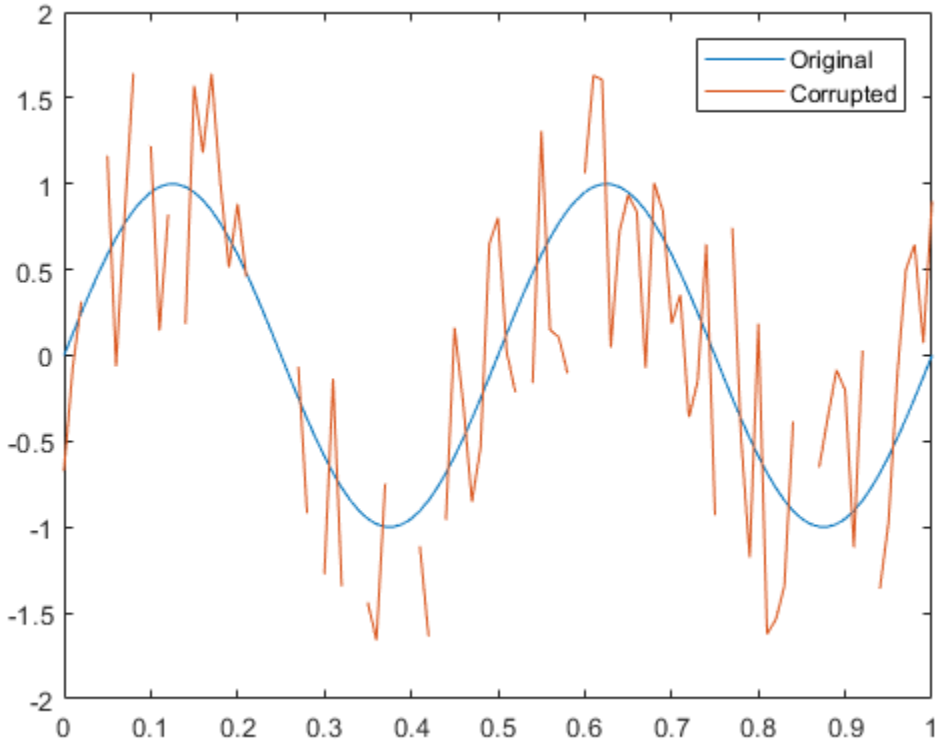


Denoise Using Multiscale Local Polynomial Transform

Reduce noise of signal using the multiscale local polynomial transform (MLPT).

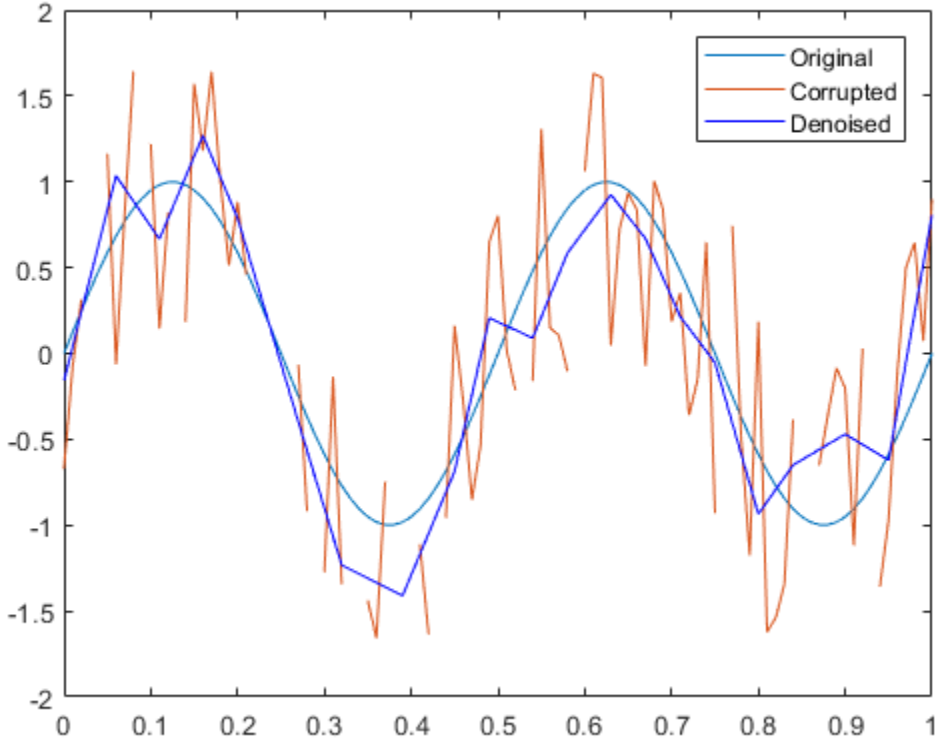
Load a pure sine wave signal with uniform sampling, and a corrupted version of the signal.

```
load(fullfile(matlabroot,'examples','wavelet','InputSamples.mat'))

plot(t,x)
hold on
plot(tCorrupt,xCorrupt)
legend('Original','Corrupted')
```



Use `mlptdenoise` to denoise the corrupted signal. Visually compare the corrupted and denoised signals against the original signal.

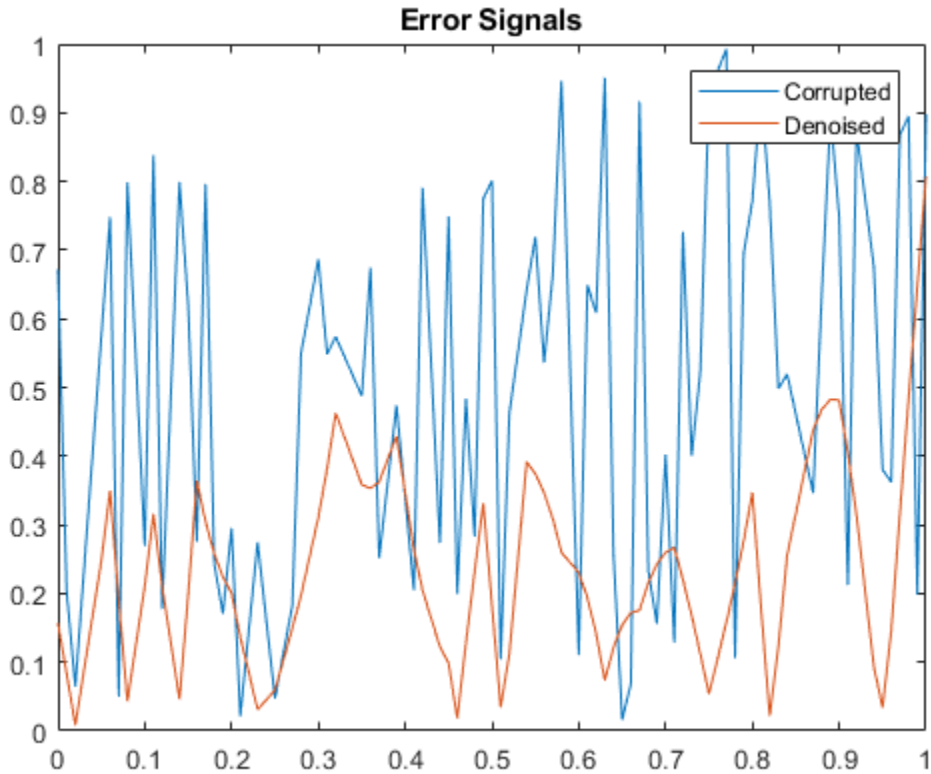
```
[xDenoised,tDenoised] = mlptdenoise(xCorrupt,tCorrupt);

plot(tDenoised,xDenoised,'b')
hold off
legend('Original','Corrupted','Denoised')
```



Compare the error signals associated with the corrupted signal and the denoised signal. Remove NaNs from the signals for visualization purposes.

```
x(samplesToErase) = [];
xCorrupt(samplesToErase) = [];
```

```
xCorruptError = abs(diff([x,xCorrupt],[],2));
yError = abs(diff([x,xDenoised],[],2));

plot(tDenoised,xCorruptError)
hold on
plot(tDenoised,yError)
title('Error Signals')
legend('Corrupted','Denoised')
hold off
```



Specify Nondefault Denoising Level

By default, `mlptdenoise` denoises a signal based on the two highest-level detail coefficients. In this example, you denoise a signal to different levels and visualize the effect.

Create a multitone signal.

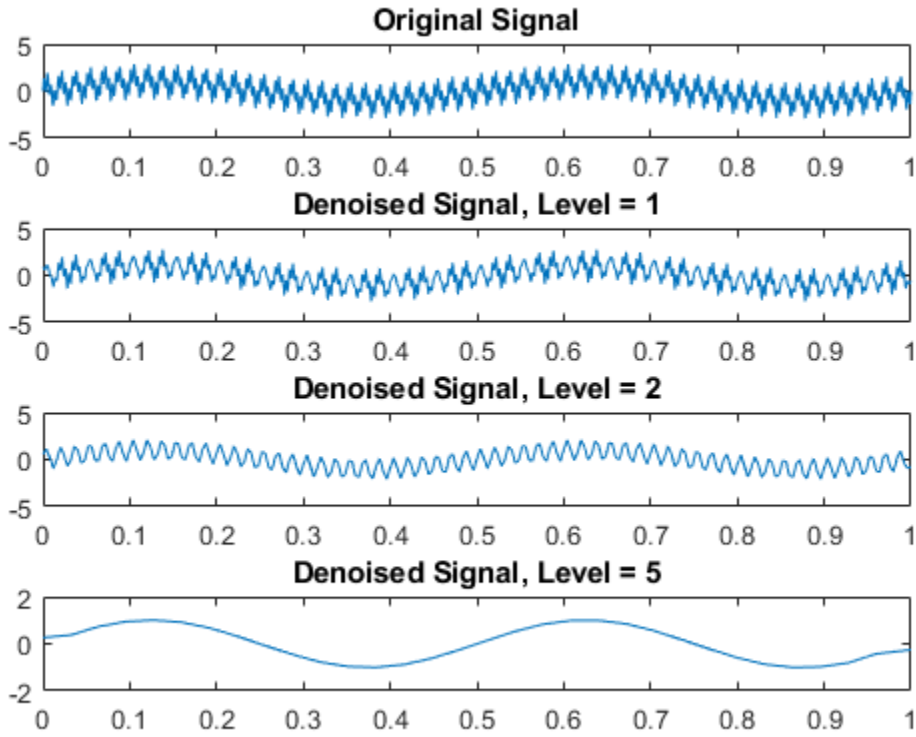
```
fs = 1000;
t = 0:1/fs:1;
x = sin(4*pi*t) + sin(120*pi*t) + sin(480*pi*t);
```

Denoise the signal to levels one, two, and five.

```
y1 = mlptdenoise(x,t,1);
y2 = mlptdenoise(x,t,2);
y5 = mlptdenoise(x,t,5);
```

Visualize the effect of `level` on the denoised signal.

```
subplot(4,1,1)
plot(t,x)
title('Original Signal')

subplot(4,1,2)
plot(t,y1)
title('Denoised Signal, Level = 1')

subplot(4,1,3)
plot(t,y2)
title('Denoised Signal, Level = 2')

subplot(4,1,4)
plot(t,y5)
title('Denoised Signal, Level = 5')
```



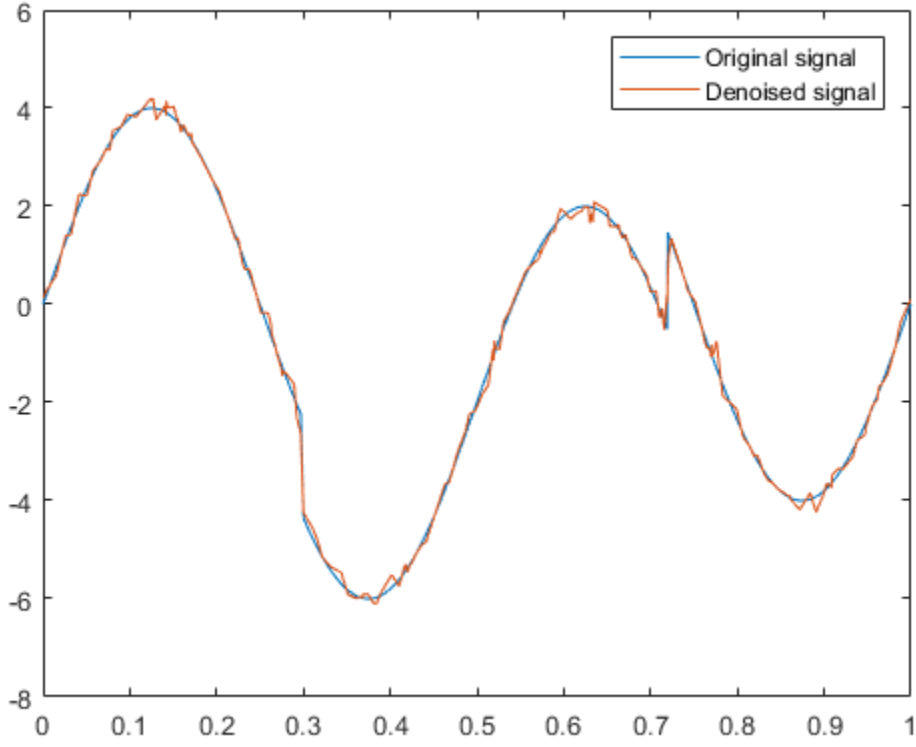
Compare Thresholded and Nonthresholded Coefficients

The `mlptdenoise` function performs the forward MLPT, thresholds the coefficients as specified by the `'DenoisingMethod'` name-value pair. Then `mlptdenoise` performs the inverse MLPT to return a denoised signal in the domain of your original signal.

You can optionally return the thresholded and original coefficients for inspection and analysis.

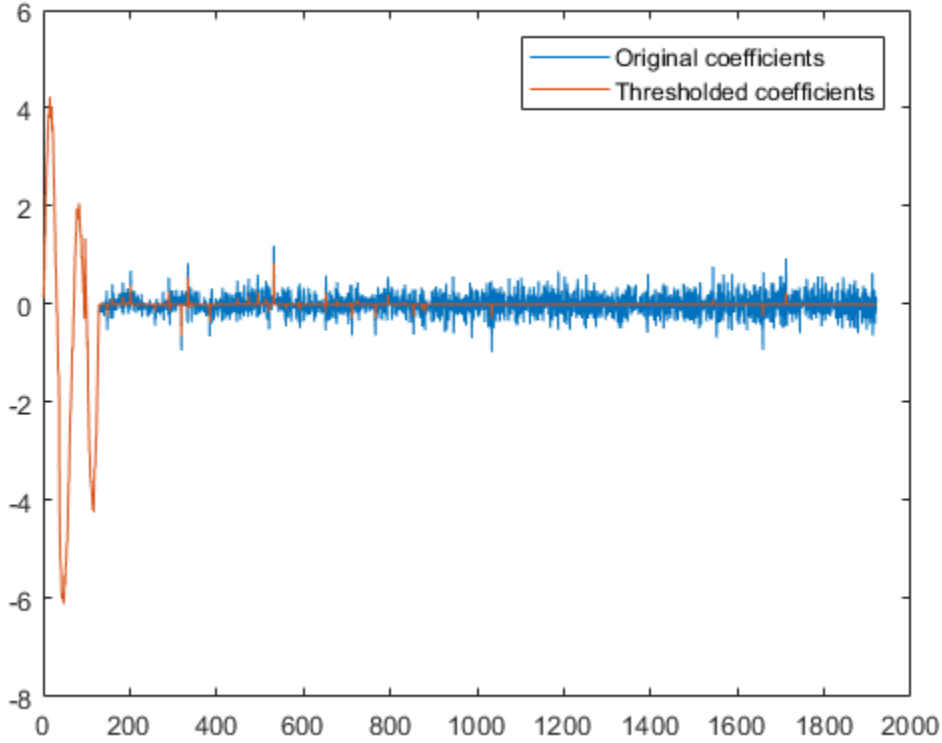
Denoise a nonuniformly sampled signal using Stein's unbiased risk method. Return the denoised signal, the associated time instants, the thresholded MLPT coefficients, and the original MLPT coefficients. Plot the original and denoised signals.

```
load nonuniformheavisine;

[xDenoised,t,wThrolded,wOriginal] = mlptdenoise(x,t,3,'denoisingmethod','SURE');

plot(t,[f,xDenoised])
legend('Original signal','Denoised signal')
```



Plot the original MLPT coefficients and the thresholded MLPT coefficients for comparison.

```
plot([wOriginal,wThrolded])
legend('Original coefficients','Thresholded coefficients')
```



- Smoothing Nonuniformly Sampled Data

Input Arguments

x — Input signal

vector | matrix

Input signal, specified as a vector or matrix.

- matrix — `x` must have at least two rows. `mlpt` operates independently on each column of `x`. The number of elements in `t` must equal the row dimension of `x`. Any `NaN`s in the columns of `x` must occur in the same rows.
- vector — `x` and `t` must have the same number of elements.

Data Types: `double`

`t` — Sampling instants

vector | `duration` array | `datetime` array

Sampling instants corresponding to the input signal, specified as a vector, `duration` array, or `datetime` array of monotonically increasing real values. The default value depends on the length of the input signal, `x`.

Data Types: `double` | `duration` | `datetime`

`numLevel` — Number of resolution levels

2 (default) | positive integer

Number of resolution levels, specified as a positive integer. The maximum value of `numLevel` depends on the shape of the input signal, `x`:

- matrix — `floor(log2(size(x,1)))`
- vector — `floor(log2(length(x)))`

`mlptdenoise` denoises `x` by thresholding all detail coefficients of an MLPT calculated for `numLevel` resolution levels.

Data Types: `double`

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: `'DualMoments',3` computes a transform using three dual vanishing moments.

`DualMoments` — Number of dual vanishing moments
`2` (default) | `3` | `4`

Number of dual vanishing moments in the lifting scheme, specified as the comma-separated pair consisting of `'DualMoments'` and `2`, `3` or `4`.

Data Types: `double`

`PrimalMoments` — Number of primal vanishing moments
`2` (default) | `3` | `4`

Number of primal vanishing moments in the lifting scheme, specified as the comma-separated pair consisting of `'PrimalMoments'` and `2`, `3`, or `4`.

Data Types: `double`

`Prefilter` — Prefilter before `mlpt`
`'Haar'` (default) | `'UnbalancedHaar'`

Prefilter before `mlpt` operation, specified as the comma-separated pair consisting of`'Prefilter'` and `'Haar'` or `'UnbalancedHaar'`. If no prefilter is specified, `'Haar'` is used by default.

Data Types: `char` | `string`

`DenoisingMethod` — Denoising method applied to MLPT detail coefficients
`'Bayesian'` (default) | `'Median'` | `'SURE'` | `'FDR'`

Denoising method applied to MLPT detail coefficients, specified as the comma-separated pair consisting of `'DenoisingMethod'` and `'Bayesian'`, `'Median'`, `'SURE'`, or `'FDR'`.

Note `'FDR'` has an optional argument for the Q-value. Q is the proportion of false positives and is specified as a real-valued scalar between zero and one. To specify `'FDR'` with a Q-value, use a cell array, where the second element is the Q-value, for example `'DenoisingMethod',{'FDR',0.01}`. If unspecified, Q defaults to `0.05`.

Data Types: `char` | `string`

Output Arguments

`y` — Denoised version of the input signal

vector | matrix

Denoised version of the input signal, returned as a vector or matrix. The size of y depends on the size of `x` and the union of `NaN`s in `x` and `t`.

By default, the `mlpt` is denoised based on the two highest resolution detail coefficients, unless `x` has fewer than four samples. If `x` has fewer than four samples, the `mlpt` is denoised based only on the highest resolution detail coefficients.

Data Types: `double`

`T` — Sampling instants corresponding to output

vector | `duration` array

Sampling instants corresponding to the output, returned as a vector or `duration` array obtained from `x` and the input `t`. If the input `t` is a `datetime` or `duration` array, `t` is converted to units that enable stable `mlpt` and `implt` computation. Then `T` is returned as a `duration` array.

Data Types: `double` | `duration`

`thresholdedCoefs` — Thresholded MLPT coefficients

vector | matrix

Thresholded MLPT coefficients, returned as a vector or matrix. The size of `thresholdedCoefs` depends on the size of `x` and the level to which the transform is calculated.

Data Types: `double`

`originalCoefs` — Original MLPT coefficients

vector | matrix

Original MLPT coefficients, returned as a vector or matrix. The size of `originalCoefs` depends on the size of `x` and the level to which the transform is calculated.

Data Types: `double`

Algorithms

Maarten Jansen developed the theoretical foundation of the multiscale local polynomial transform (MLPT) and algorithms for its efficient computation [1][2][3]. The MLPT uses a lifting scheme, wherein a kernel function smooths fine-scale coefficients with a given bandwidth to obtain the coarser resolution coefficients. The `mlpt` function uses only local polynomial interpolation, but the technique developed by Jansen is more general and admits many other kernel types with adjustable bandwidths [2].

References

[1] Jansen, M. "Multiscale Local Polynomial Smoothing in a Lifted Pyramid for Non-Equispaced Data". *IEEE Transactions on Signal Processing*, Vol. 61, Number 3, 2013, pp. 545–555.

[2] Jansen, M., and M. Amghar. "Multiscale local polynomial decompositions using bandwidths as scales". *Statistics and Computing* (forthcoming). 2016.

[3] Jansen, M., and Patrick Oonincx. *Second Generation Wavelets and Applications*. London: Springer, 2005.

See Also

`imlpt` | `mlpt` | `mlptrecon`

Topics

Smoothing Nonuniformly Sampled Data

Introduced in R2017a

mlptrecon

Reconstruct signal using inverse multiscale local 1-D polynomial transform

Syntax

```
y = mlptrecon(type,coefs,T,coefsPerLevel,scalingMoments,
reconstructionLevel)
y = mlptrecon(___,Name,Value)
```

Description

`y = mlptrecon(type,coefs,T,coefsPerLevel,scalingMoments, reconstructionLevel)` returns an approximation to the inverse multiscale 1-D polynomial transform (MLPT) of `coefs`.

`y = mlptrecon(___,Name,Value)` specifies `mlptrecon` properties using one or more `Name,Value` pair arguments and the input arguments from the previous syntax.

Examples

Detect and Localize High-Frequency Content

Create a low-frequency signal with high-frequency blips.

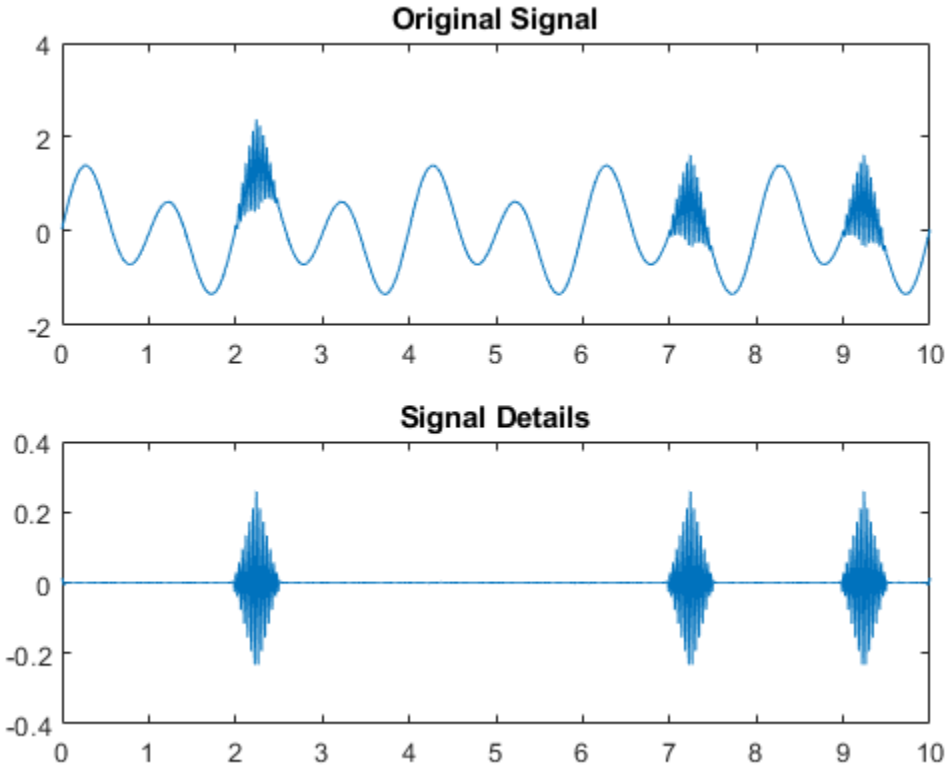
```
t = (0:0.01:10)';
x = sin(2*pi.*t) + 0.5*sin(pi.*t+0.1);
bliptime = (0:0.01:0.5)';
blip = sin(50*pi.*bliptime).*triang(numel(bliptime));
for i = [200,700,900]
    x(i:i+numel(bliptime)-1) = x(i:i+numel(bliptime)-1)+blip;
end
```

Perform a multilevel polynomial transform. Perform the inverse multilevel polynomial transform using the detail coefficients.

```
[w,t,nj,scalingmoments] = mlpt(x,t);
yDetails = mlptrecon('d',w,t,nj,scalingmoments,1);
```

Plot the original signal and the processed signal.

```
subplot(2,1,1)
plot(t,x)
title('Original Signal')

subplot(2,1,2)
plot(t,yDetails)
title('Signal Details')
```

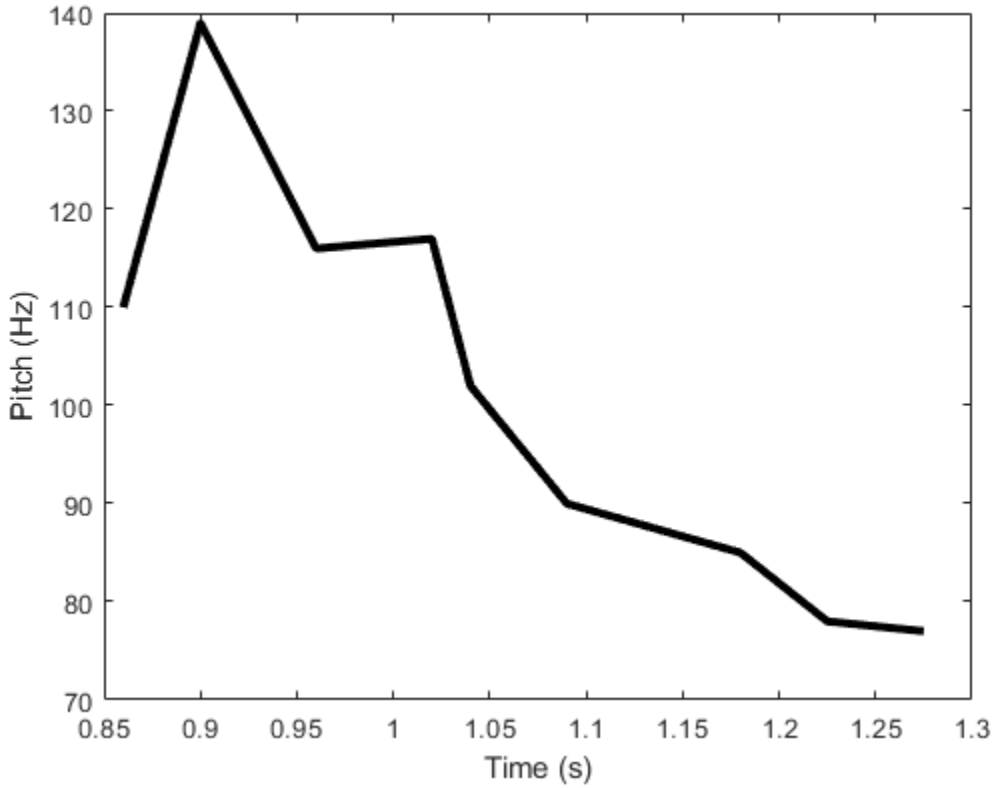


Approximate Data by Choosing Reconstruction Coefficients

Approximate data using multiscale local polynomial transform (MLPT) reconstruction. Use `mlptrecon` to approximate a corrupted and sparsely sampled pitch contour.

Load input data and visualize it.

```
load(fullfile(matlabroot,'examples','wavelet','CorruptedPitchData.mat'));
plot(time,pitchContour,'k','linewidth',3)
hold on
xlabel('Time (s)')
ylabel('Pitch (Hz)')
```



Compute the MLPT of the pitch contour.

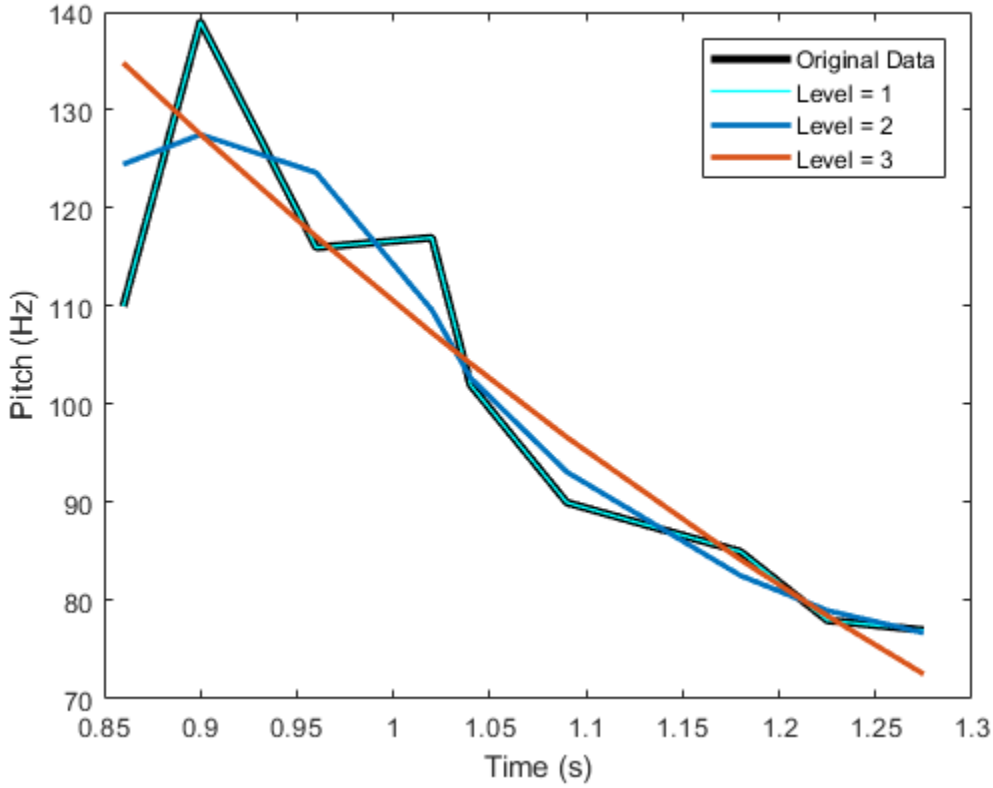
```
[w,t,nj,scalingMoments] = mlpt(pitchContour,time, ...
    'DualMoments',3, ...
    'PrimalMoments',4, ...
    'PreFilter','none');
```

Use `mlptrecon` to reconstruct the signal using the approximation coefficents at different levels.

```
y = zeros(numel(t),3);
for level = 1:3
    y(:,level) = mlptrecon('a',w,t,nj,scalingMoments,level,'DualMoments',3);
end
```

Plot the reconstructed signals. Level two obtains the best smoothed estimate.

```
plot(t,y(:,1),'c','linewidth',1)
plot(t,y(:,2),'linewidth',2)
plot(t,y(:,3),'linewidth',2)
legend('Original Data','Level = 1','Level = 2','Level = 3')
hold off
```



- Smoothing Nonuniformly Sampled Data

Input Arguments

`type` — Type of coefficients

`'a' | 'd'`

Type of coefficients used to reconstruct the signal, specified as `'a'` or `'d'`.

- `'a'` — Approximation coefficients

- `'d'` — Detail coefficients

Approximation coefficients are a lowpass representation of the input. At each level, the approximation coefficients are divided into coarser approximation and detail coefficients.

Data Types: `char` | `string`

`coefs` — MLPT coefficients

vector | matrix

MLPT coefficients, specified as a vector or matrix of MLPT coefficients returned by the `mlpt` function.

Data Types: `double`

`T` — Sampling instants corresponding to output

vector | `duration` array

Sampling instants corresponding to `y`, specified as a vector or `duration` array of increasing values returned by the `mlpt` function.

Data Types: `double` | `duration`

`coefsPerLevel` — Coefficients per resolution level

vector

Coefficients per resolution level, specified as a vector containing the number of coefficients at each resolution level in `coefs`. `coefsPerLevel` is an output argument of the `mlpt` function.

The elements of `coefsPerLevel` are organized as follows:

- `coefsPerLevel(1)` — Number of approximation coefficients at the coarsest resolution level.
- `coefsPerLevel(i)` — Number of detail coefficients at resolution level `i`, where `i = numLevel - i + 2` for `i = 2,..., numLevel + 1`. `numLevel` is the number of resolution levels used to calculate the MLPT. `numLevel` is inferred from `coefsPerLevel`: `numLevel = length(coefsPerLevel-1)`.

The smaller the index `i`, the lower the resolution. The MLPT is two times redundant in the number of detail coefficients, but not in t the number of approximation coefficients.

Data Types: `double`

`scalingMoments` — Scaling function moments
matrix

Scaling function moments, specified as a `length(coefs)`-by-`P` matrix, where `P` is the number of primal moments specified by the MLPT.

Data Types: `double`

`reconstructionLevel` — Resolution level used for reconstruction
positive integer

Resolution level used for reconstruction, specified as a positive integer less than or equal to `length(coefsPerLevel-1)`. `length(coefsPerLevel-1)` is the number of resolution levels used to calculate the MLPT. Increasing the value of `reconstructionLevel` corresponds to reconstructing your signal with coarser resolution approximations.

Data Types: `double`

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: `'DualMoments',3` computes a transform using three dual vanishing moments.

`DualMoments` — Number of dual vanishing moments
`2` (default) | `3` | `4`

Number of dual vanishing moments in the lifting scheme, specified as the comma-separated pair consisting of `'DualMoments'` and `2`, `3` or `4`. The number of dual moments must match the number used by `mlpt`.

Data Types: `double`

Output Arguments

`y` — Reconstructed approximation or details of signal

vector | matrix

Reconstructed approximation or details of signal, returned as a vector or matrix, depending on the inputs to the `mlpt` function.

Data Types: `double`

Algorithms

Maarten Jansen developed the theoretical foundation of the multiscale local polynomial transform (MLPT) and algorithms for its efficient computation [1][2][3]. The MLPT uses a lifting scheme, wherein a kernel function smooths fine-scale coefficients with a given bandwidth to obtain the coarser resolution coefficients. The `mlpt` function uses only local polynomial interpolation, but the technique developed by Jansen is more general and admits many other kernel types with adjustable bandwidths [2].

References

[1] Jansen, M. "Multiscale Local Polynomial Smoothing in a Lifted Pyramid for Non-Equispaced Data." *IEEE Transactions on Signal Processing*. Vol. 61, Number 3, 2013, pp. 545–555.

[2] Jansen, M. and M. Amghar. "Multiscale local polynomial decompositions using bandwidths as scales". *Statistics and Computing* (forthcoming). 2016.

[3] Jansen, M. and Patrick Oonincx. *Second Generation Wavelets and Applications*. London: Springer, 2005.

See Also

`imlpt` | `mlpt` | `mlptdenoise`

Topics

Smoothing Nonuniformly Sampled Data

Introduced in R2017a

modwpt

Maximal overlap discrete wavelet packet transform

Syntax

```
wpt = modwpt(x)
wpt = modwpt(x,wname)
wpt = modwpt(x,lo,hi)
wpt = modwpt(___,lev)

[wpt,packetlevs] = modwpt(___)
[wpt,packetlevs,cfreq] = modwpt(___)
[wpt,packetlevs,cfreq,energy] = modwpt(___)
[wpt,packetlevs,cfreq,energy,relenergy] = modwpt(___)

[___] = modwpt(___,Name,Value)
```

Description

`wpt = modwpt(x)` returns the terminal nodes for the maximal overlap discrete wavelet packet transform (MODWPT) for the 1-D real-valued signal, `x`.

Note The output of the MODWPT is time-delayed compared to the input signal. Most filters used to obtain the MODWPT have a nonlinear phase response, which makes compensating for the time delay difficult. This is true for all orthogonal scaling and wavelet filters, except the Haar wavelet. It is possible to time-align the coefficients with the signal features, but the result is an approximation, not an exact alignment with the original signal. The MODWPT partitions the energy among the wavelet packets at each level. The sum of the energy over all the packets equals the total energy of the input signal. The output of MODWPT is useful for applications where you want to analyze the energy levels in different packets.

The MODWPT details (`modwptdetails`) are the result of zero-phase filtering of the signal. The features in the MODWPT details align exactly with features in the input signal. For a given level, summing the details for each sample returns the exact original

signal. The output of the MODWPT details is useful for applications that require time-alignment, such as nonparametric regression analysis.

`wpt = modwpt(x,wname)` returns the MODWPT using the orthogonal wavelet filter specified by the `wname`.

`wpt = modwpt(x,lo,hi)` returns the MODWPT using the orthogonal scaling filter, `lo`, and wavelet filter, `hi`.

`wpt = modwpt(___,lev)` returns the terminal nodes of the wavelet packet tree at positive integer level `lev`.

`[wpt,packetlevs] = modwpt(___)` returns a vector of transform levels corresponding to the rows of `wpt`.

`[wpt,packetlevs,cfreq] = modwpt(___)` returns the center frequencies of the approximate passbands corresponding to the rows of `wpt`.

`[wpt,packetlevs,cfreq,energy] = modwpt(___)` returns the energy (squared L2 norm) of the wavelet packet coefficients for the nodes in `wpt`.

`[wpt,packetlevs,cfreq,energy,relenergy] = modwpt(___)` returns the relative energy for the wavelet packets in `wpt`.

`[___] = modwpt(___,Name,Value)` returns the MODWPT with additional options specified by one or more `Name,Value` pair arguments.

Examples

MODWPT Using Default Wavelet

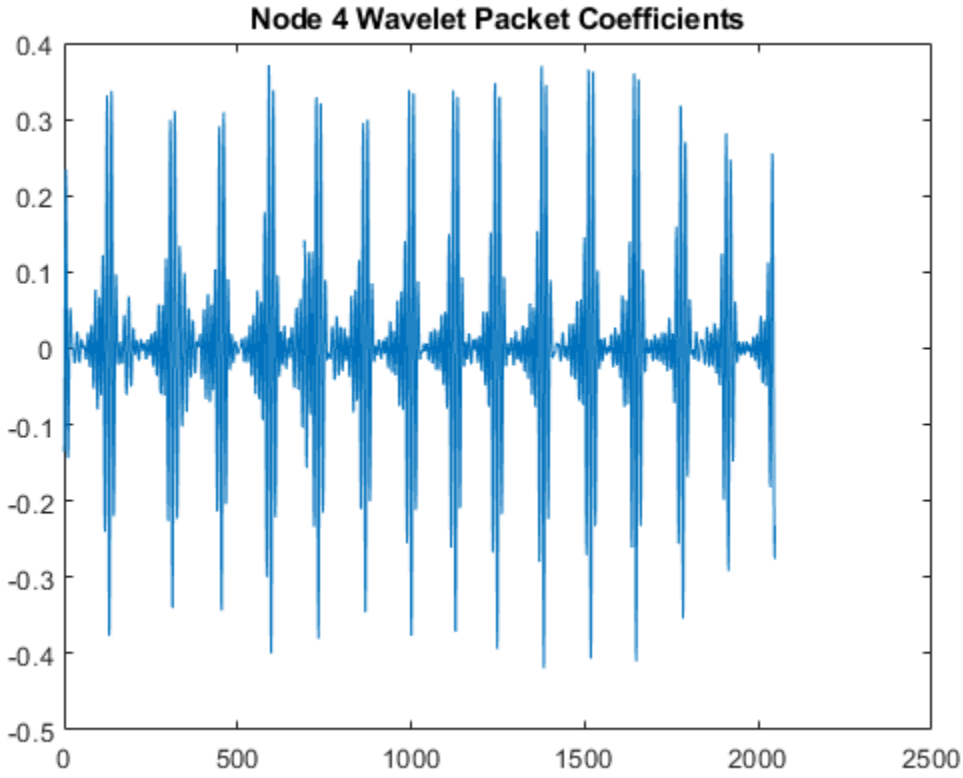
Obtain the MODWPT of an electrocardiogram (ECG) signal using the default length 18 Fejer-Korovkin (`'fk18'`) wavelet.

```
load wecg;
wpt = modwpt(wecg);
```

`wpt` is a 16-by-2048 matrix containing the sequency-ordered wavelet packet coefficients for the wavelet packet transform nodes. In this case, the nodes are at level 4. Each node

corresponds to an approximate passband filtering of $[nf_s/2^5, (n+1)f_s/2^5)$, where n = 0,...,15, and f_s is the sampling frequency. Plot the wavelet packet coefficients at node (4,2), which is level 4, node 2.

```
plot(wpt(3,:))
title('Node 4 Wavelet Packet Coefficients')
```



MODWPT Using Daubechies Extremal Phase Wavelet with Two Vanishing Moments

Obtain the MODWPT of Southern Oscillation Index data with the Daubechies extremal phase wavelet with two vanishing moments (`'db2'`).

```
load soi;
wsoi = modwpt(soi,'db2');
```

Verify that the size of the resulting transform contains 16 nodes. Each node is in a separate row.

```
size(wsoi)

ans =

          16       12998
```

MODWPT Using Scaling and Wavelet Filters

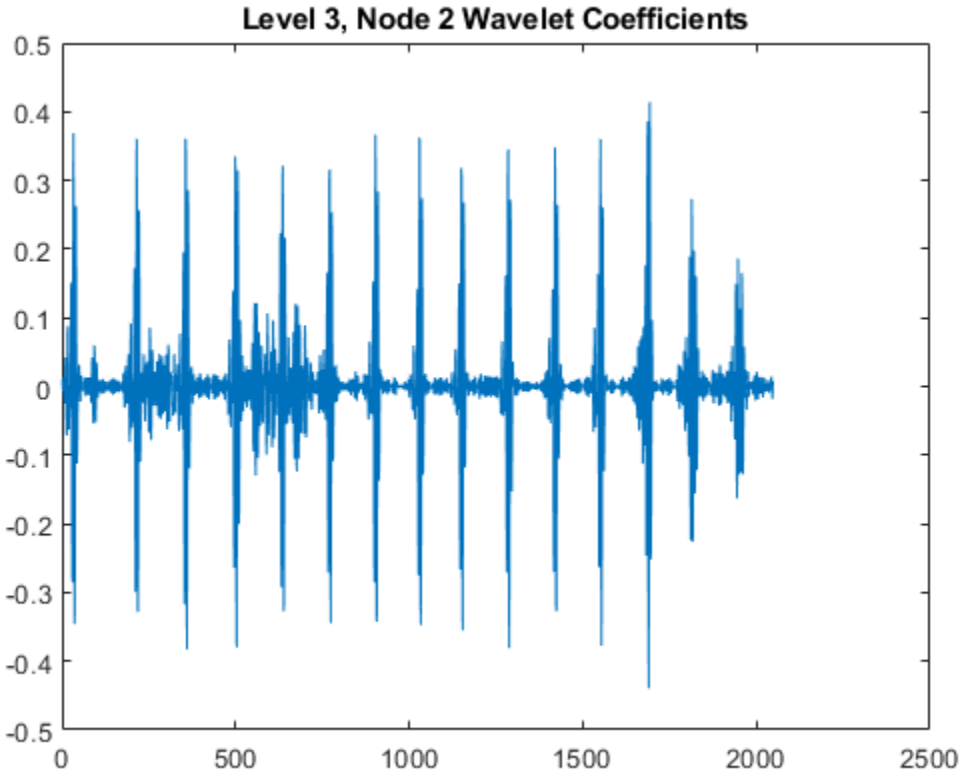
Obtain the MODWPT of an ECG waveform using the Fejer-Korovkin length 18 scaling and wavelet filters.

```
load wecg;
[lo,hi] = wfilters('fk18');
wpt = modwpt(wecg,lo,hi);
```

MODWPT Full Packet Tree and Passband Center Frequencies

Obtain the MODWPT and full wavelet packet tree of an ECG waveform using the default length 18 Fejer-Korovkin (`'fk18'`) wavelet. Extract and plot the node coefficients at level 3, node 2.

```
load wecg;
[wpt,packetlevels,cfreq] = modwpt(wecg,'FullTree',true);
p3 = wpt(packetlevels==3,:);
plot(p3(3,:))
title('Level 3, Node 2 Wavelet Coefficients')
```



Display the center frequencies at level 3.

```
cfreq(packetlevels==3,:)
```

```
ans =

    0.0313
    0.0938
    0.1563
    0.2188
    0.2813
    0.3438
    0.4063
```

```
    0.4688
```

MODWPT Energy and Relative Energy

Obtain and plot the MODWPT energy and relative energy of an ECG waveform.

```
load wecg;
[wpt,~,cfreq,energy,relenergy] = modwpt(wecg);
```

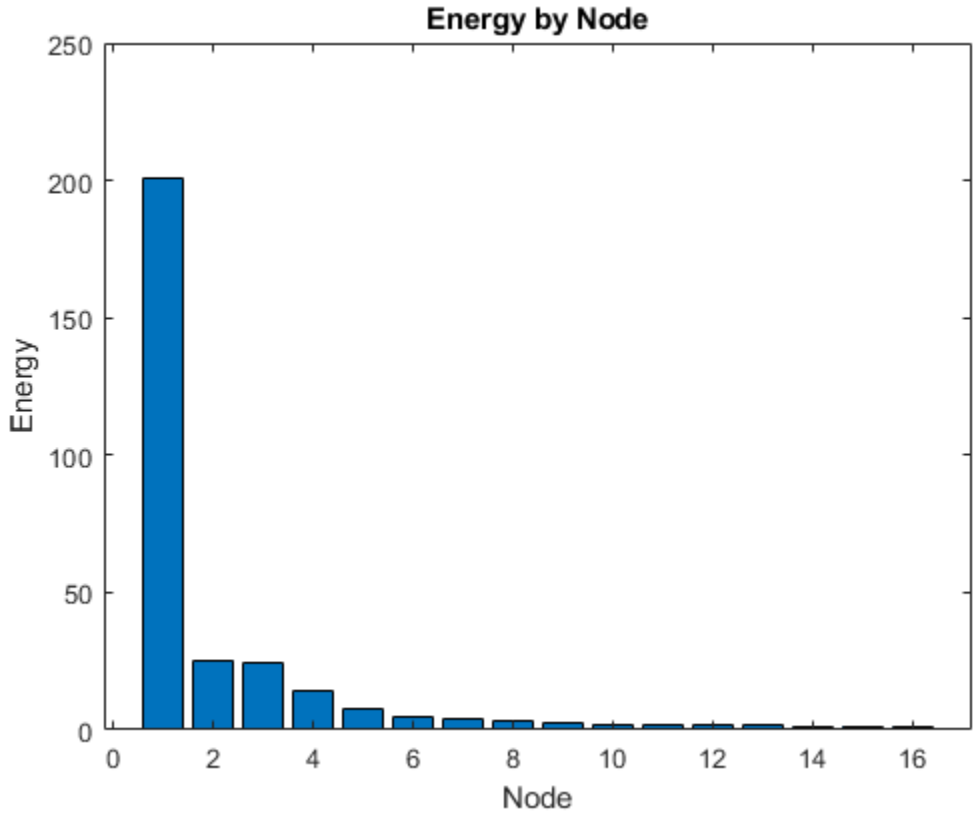
Show that the sum of the MODWPT energies is equal to the sum of the energy in the original signal. The difference between the total MODWPT energy and the signal energy is small enough to be considered insignificant.

```
sum(energy)-(bandpower(wecg)*length(wecg))
```

```
ans = 3.6120e-09
```

Plot the MODWPT energy by node.

```
figure;
bar(1:16,energy);
xlabel('Node')
ylabel('Energy')
title('Energy by Node')
```

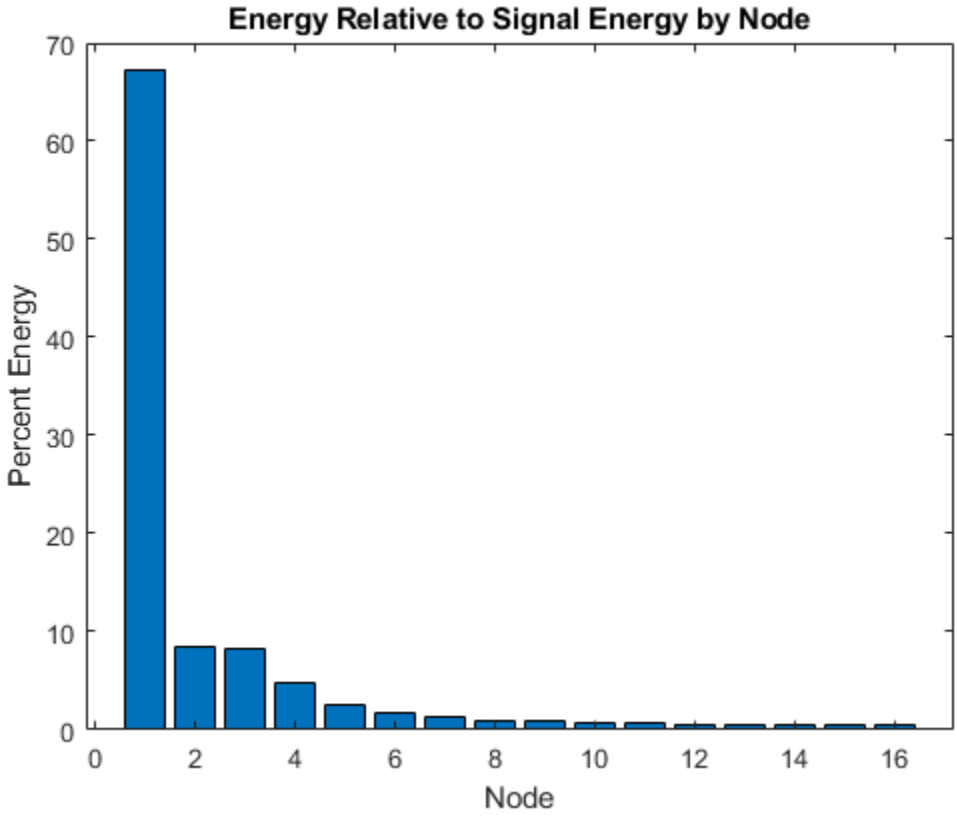



```
sprintf('Total power in passband is %.2f', energy(1))
```

```
ans =
'Total power in passband is 200.84'
```

Plot the relative energy and show the percentage of signal energy in the first passband [0,5.6250].

```
figure;
bar(1:16,relenergy*100);
xlabel('Node');
ylabel('Percent Energy');
title('Energy Relative to Signal Energy by Node');
```




```
sprintf('Percentage of signal power in passband is %.1f',(relenergy(1)*100))
```

```
ans =
'Percentage of signal power in passband is 67.3'
```

Time-Aligned MODWPT

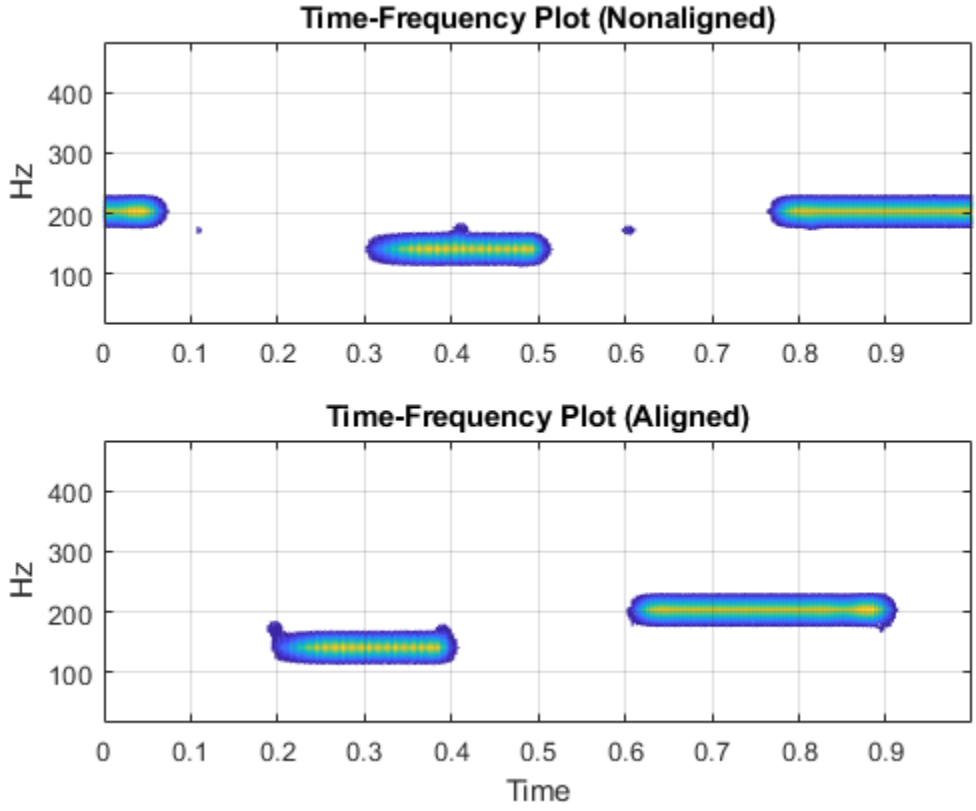
Obtain the time-aligned MODWPT of two intermittent sine waves in noise. The sine wave frequencies are 150 Hz and 200 Hz. The data is sampled at 1000 Hz.

```
dt = 0.001;
t = 0:dt:1-dt;
```

```
x = cos(2*pi*150*t).*(t>=0.2 & t<0.4)+ sin(2*pi*200*t).*(t>0.6 & t<0.9);
y = x+0.05*randn(size(t));
[wpta,~,Falign] = modwpt(x,'TimeAlign',true);
[wptn,~,Fnon] = modwpt(x);
```

Compare the nonaligned and time-aligned time-frequency plots.

```
subplot(2,1,1);
contour(t,Fnon.*(1/dt),abs(wptn).^2);
grid on;
ylabel('Hz');
title('Time-Frequency Plot (Nonaligned)');
subplot(2,1,2)
contour(t,Falign.*(1/dt),abs(wpta).^2);
grid on;
xlabel('Time');
ylabel('Hz');
title('Time-Frequency Plot (Aligned)');
```



Input Arguments

`x` — Input signal

real-valued vector

Input signal, specified as a real-valued row or column vector. `x` must have at least two elements.

Data Types: `double`

`wname` — Analyzing wavelet filter

`fk18` (default) |

Analyzing wavelet filter specified as a that corresponds to an orthogonal wavelet. If you specify the scaling (`lo`) and wavelet (`hi`) filters, `modwpt` ignores the `wname` input.

Valid orthogonal wavelet families begin with one of the following s, followed by an integer, *N*, for example, `sym4`. Note, however, that `'haar'` is not followed by an integer.

- `'haar'` — Haar wavelet, which is the same as Daubechies wavelet with one vanishing moment, `'db1'`.
- `'db`*N*`'` — Daubechies wavelet with *N* vanishing moments
- `'sym`*N*`'` — Symlets wavelet with *N* vanishing moments
- `'coif`*N*`'` — Coiflets wavelet with *N* vanishing moments
- `'fk`*N*`'` — Fejer-Korovkin wavelet with *N* coefficients

To check if your wavelet is orthogonal, use `wavemngr('type',wname)` and verify that it returns `1` as the wavelet type. To determine valid values for *N*, use `waveinfo`, for example, `waveinfo('fk')`.

`lo` — Scaling filter

even-length real-valued vector

Scaling filter, specified as an even-length real-valued vector. `lo` must satisfy the conditions necessary to generate an orthogonal scaling function. You cannot specify both the scaling-wavelet filters and the `wname` input.

`hi` — Wavelet filter

even-length real-valued vector

Wavelet filter, specified as an even-length real-valued vector. `hi` must satisfy the conditions necessary to generate an orthogonal wavelet. You cannot specify both the scaling-wavelet filters and the `wname` input.

`lev` — Transform level

positive integer

Transform level, specified as a positive integer less than or equal to `floor(log2(numel(x)))`.

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: `'Fulltree',true` returns the full wavelet packet tree

`FullTree` — Full packet tree
`false` (default) | `true`

Option to return the full wavelet packet tree, specified as the comma-separated pair consisting of `'FullTree'` and either `false` or `true`. If you specify `false`, then `modwpt` returns only the terminal (final-level) wavelet packet nodes. If you specify `true`, then `modwpt` returns the full wavelet packet tree down to the specified level.

Example: `'Fulltree',true`

`TimeAlign` — Signal time alignment
`false` (default) | `true`

Option to time align wavelet packet coefficients with signal features, specified as the comma-separated pair consisting of `'TimeAlign'` and either `true` to time align or `false` to not align.

The scaling and wavelet filters have a time delay. Circularly shifting the wavelet packet coefficients in all nodes aligns the signal and wavelet coefficients in time. If you want to reconstruct the signal, such as by using `imodwpt`, do not shift the coefficients because time alignment is done during the inversion process.

Example: `'TimeAlign',true`

Output Arguments

`wpt` — Wavelet packet transform
matrix

Wavelet packet tree, returned as a matrix with each row containing the sequency-ordered wavelet packet coefficients. By default, `wpt` contains only the terminal level for

the MODWPT. The default terminal level is either level 4 or `floor(log2(numel(x)))`, whichever is smaller. At level 4, `wpt` is a 16-by-`numel(x)` matrix. For the full tree, at level j, `wpt` is a 2^{j+2}-2-by-`numel(x)` matrix, with each row containing the packet coefficients by level and index. The approximate passband for the nth row of `wpt` at level j is $\left[\frac{n-1}{2^{(j+1)}}, \frac{n}{2^{(j+1)}}\right)$ cycles/sample, where $n = 1,2,...2^j$.

`packetlevs` — Transform levels

vector

Transform levels, returned as a vector. The levels correspond to the rows of `wpt`. If `wpt` contains only the terminal level coefficients, `packetlevs` is a vector of constants equal to the terminal level. If `wpt` contains the full wavelet packet table, `packetlevs` is a vector with 2^j elements for each level, j. To select all the wavelet packet nodes at a particular level, use `packetlevs` with logical indexing.

`cfreq` — Center frequencies of passbands

vector

Center frequencies of the approximate passbands in the `wpt` rows, returned as a vector. The center frequencies are in cycles/sample. To convert the units to cycles/unit time, multiply `cfreq` by the sampling frequency.

`energy` — Energy of the wavelet packet coefficients

vector

Energy of the wavelet packet coefficients for the `wpt` nodes, returned as a vector. The sum of the energies (squared L2 norms) for the wavelet packets at each level equals the energy in the signal.

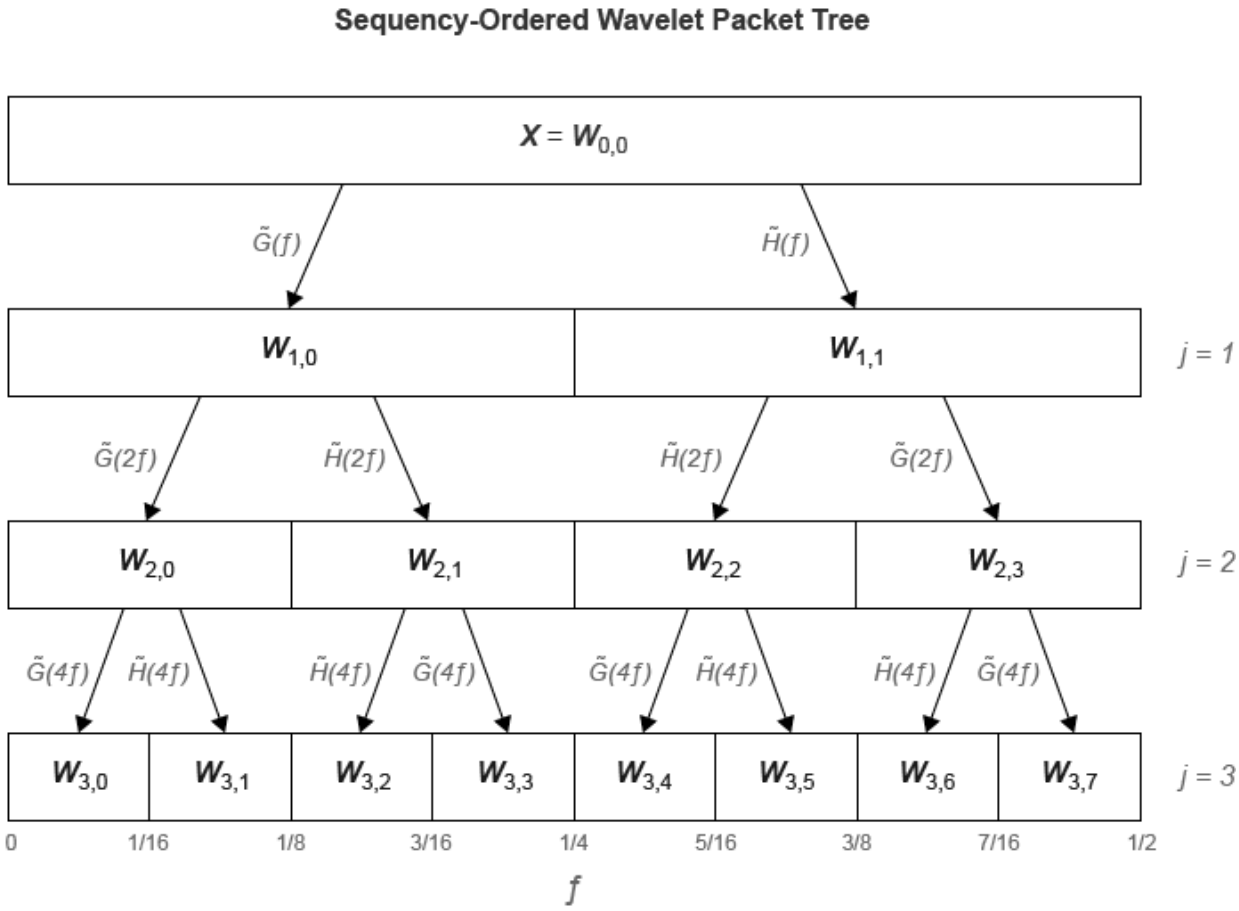
`relenergy` — Relative energy

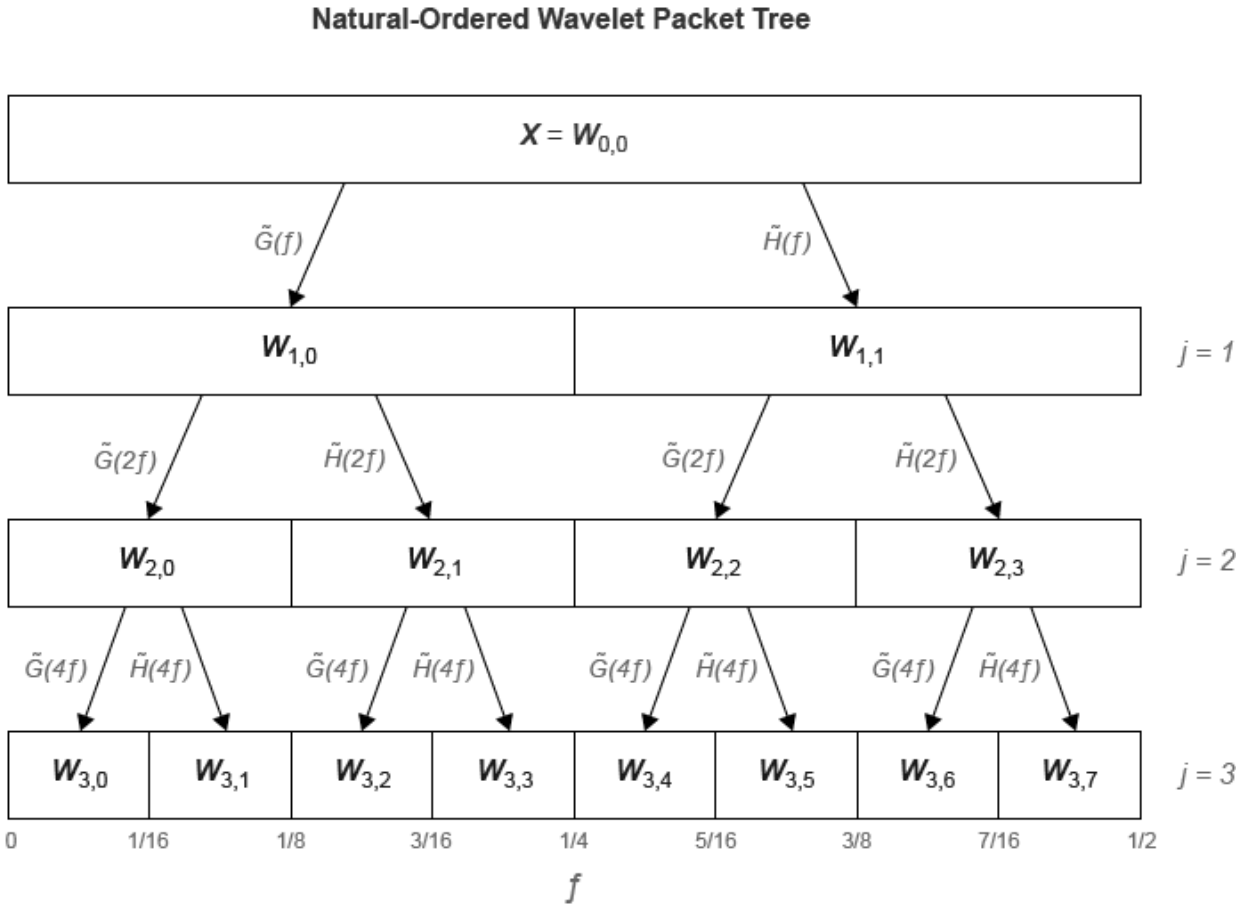
vector

Relative energy for each level, returned as a vector. The relative energy is the proportion of energy in each wavelet packet by level, relative to the total energy of that level. The sum of relative energies in all packets at each level equals 1.

Algorithms

The `modwpt` performs a discrete wavelet packet transform and produces a sequency-ordered wavelet packet tree. Compare the sequency-ordered and normal (Paley)-ordered trees.





References

[1] Percival, D. B., and A. T. Walden. *Wavelet Methods for Time Series Analysis*. Cambridge, UK: Cambridge University Press, 2000.

[2] Walden, A.T., and A. Contreras Cristan. "The phase-corrected undecimated discrete wavelet packet transform and its application to interpreting the timing of events." *Proceedings of the Royal Society of London A*. Vol. 454, Issue 1976, 1998, pp. 2243-2266.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`imodwpt` | `modwptdetails`

Introduced in R2016a

modwptdetails

Maximal overlap discrete wavelet packet transform details

Syntax

```
w = modwptdetails(x)
w = modwptdetails(x,wname)
w = modwptdetails(x,lo,hi)
w = modwptdetails(___,lev)

[w,packetlevs] = modwptdetails(___)
[w,packetlevs,cfreq] = modwptdetails(___)

[___] = modwptdetails(___,Name,Value)
```

Description

`w = modwptdetails(x)` returns the maximal overlap discrete wavelet packet transform (MODWPT) details for the 1-D real-valued signal, `x`. The MODWPT details provide zero-phase filtering of the signal. By default, `modwptdetails` returns only the terminal nodes, which are at level 4 or at level `floor(log2(numel(x)))`, whichever is smaller.

Note To decide whether to use `modwptdetails` or `modwpt`, consider the type of data analysis you need to perform. For applications that require time alignment, such as nonparametric regression analysis, use `modwptdetails`. For applications where you want to analyze the energy levels in different packets, use `modwpt`. For more information, see "Algorithms" on page 1-550

`w = modwptdetails(x,wname)` uses the orthogonal wavelet filter specified by the character vector `wname`.

`w = modwptdetails(x,lo,hi)` uses the orthogonal scaling filter, `lo`, and wavelet filter, `hi`.

`w = modwptdetails(___,lev)` returns the terminal nodes of the wavelet packet tree at positive integer level `lev`.

`[w,packetlevs] = modwptdetails(___)` returns a vector of transform levels corresponding to the rows of `w`.

`[w,packetlevs,cfreq] = modwptdetails(___)` returns, `w`, the center frequencies of the approximate passbands corresponding to the MODWPT details in .

`[___] = modwptdetails(___,Name,Value)` returns the MODWPT with additional options specified by one or more `Name,Value` pair arguments.

Examples

MODWPT Details Using Default Wavelet

Obtain the MODWPT of an electrocardiogram (ECG) signal using the default length 18 Fejer-Korovkin (`'fk18'`) wavelet and the default level, 4.

```
load wecg;
wptdetails = modwptdetails(wecg);
```

Demonstrate that summing the MODWPT details over each sample reconstructs the signal. The largest absolute difference between the original signal and the reconstruction is on the order of 10^{-11}, which demonstrates perfect reconstruction.

```
xrec = sum(wptdetails);
max(abs(wecg-xrec'))
```

```
ans = 1.7903e-11
```

MODWPT Details for Two Sine Waves

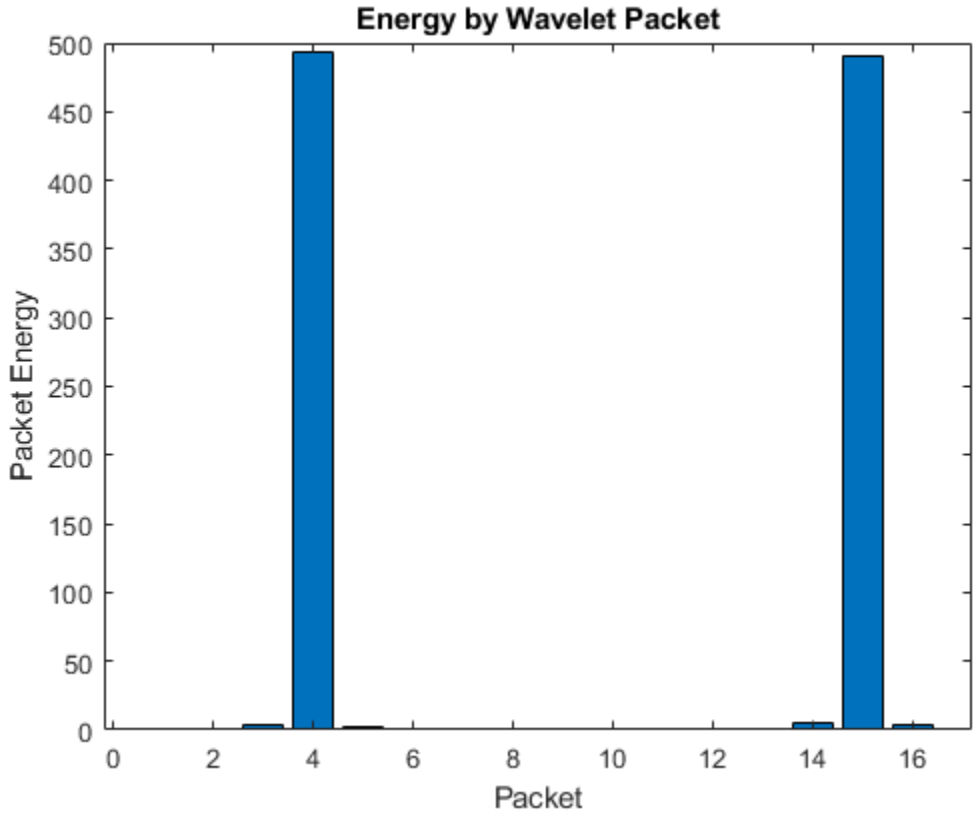
Obtain the MODWPT details for a signal containing 100 Hz and 450 Hz sine waves. Each row of the `modwptdetails` output corresponds to a separate frequency band.

```
dt = 0.001;
fs = 1/dt;
```

```
t = 0:dt:1;
x = (sin(2*pi*100*t)+sin(2*pi*450*t));
[lo,hi] = wfilters('fk22');
wptdetails = modwptdetails(x,lo,hi);
```

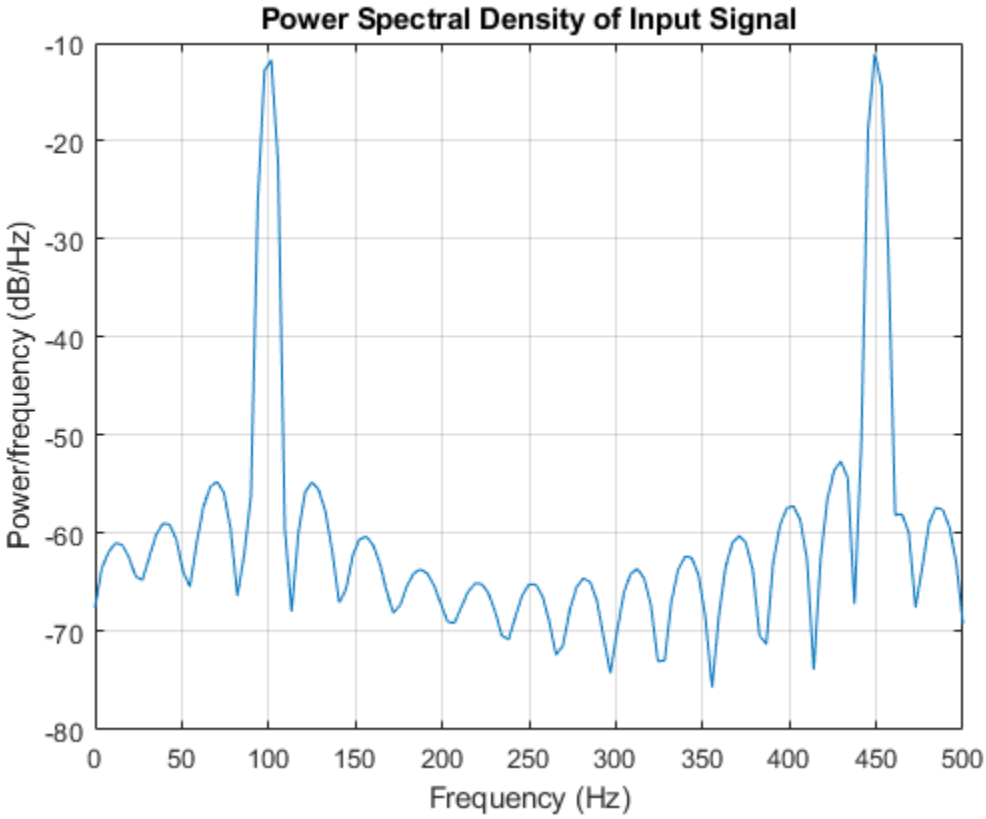
Use `modwpt` to obtain the energy and center frequencies of the signal. Plot the energy in the wavelet packets. The fourth and fifteenth frequency bands contain most of the energy. Other frequency bands have significantly less energy. The frequency ranges of fourth and fifteenth bands are approximately 94-125 Hz and 438-469 Hz, respectively.

```
[wpt,~,cfreqs,energy] = modwpt(x,lo,hi);
figure
bar(1:16,energy);
xlabel('Packet')
ylabel('Packet Energy')
title('Energy by Wavelet Packet')
```



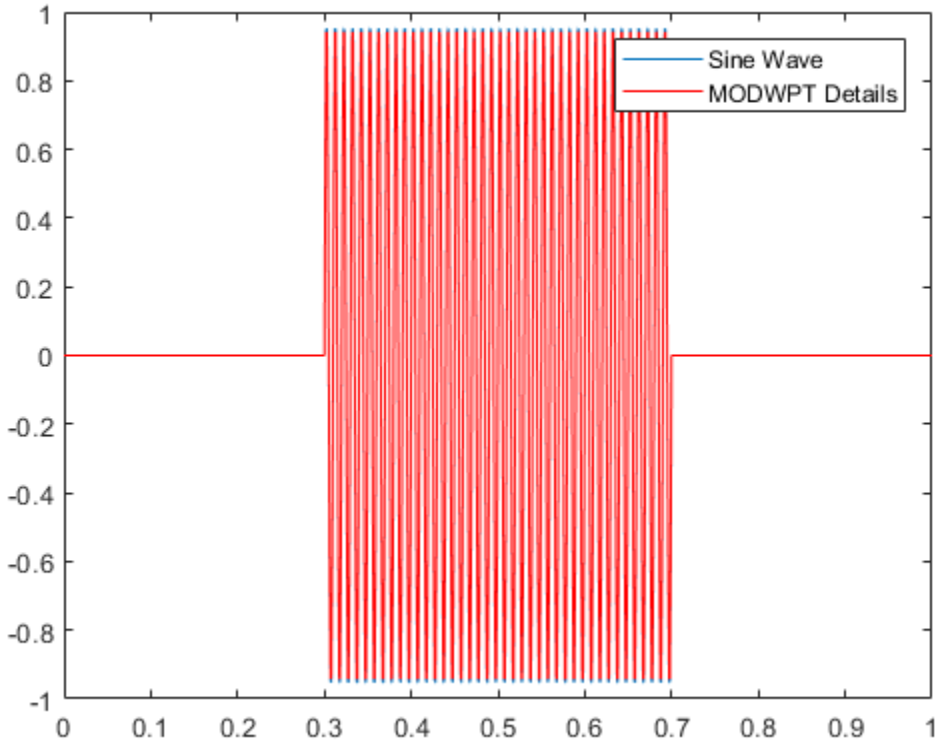
Plot the power spectral density of the input signal.

```
pwelch(x,[],[],[],fs,'onesided');
title('Power Spectral Density of Input Signal')
```



Show that the MODWPT details have zero-phase shift from the 100 Hz input sine.

```
p4 = wptdetails(4,:);
plot(t,sin(2*pi*100*t).*(t>0.3 & t<0.7))
hold on
plot(t,p4.*(t>0.3 & t<0.7),'r')
legend('Sine Wave','MODWPT Details')
```



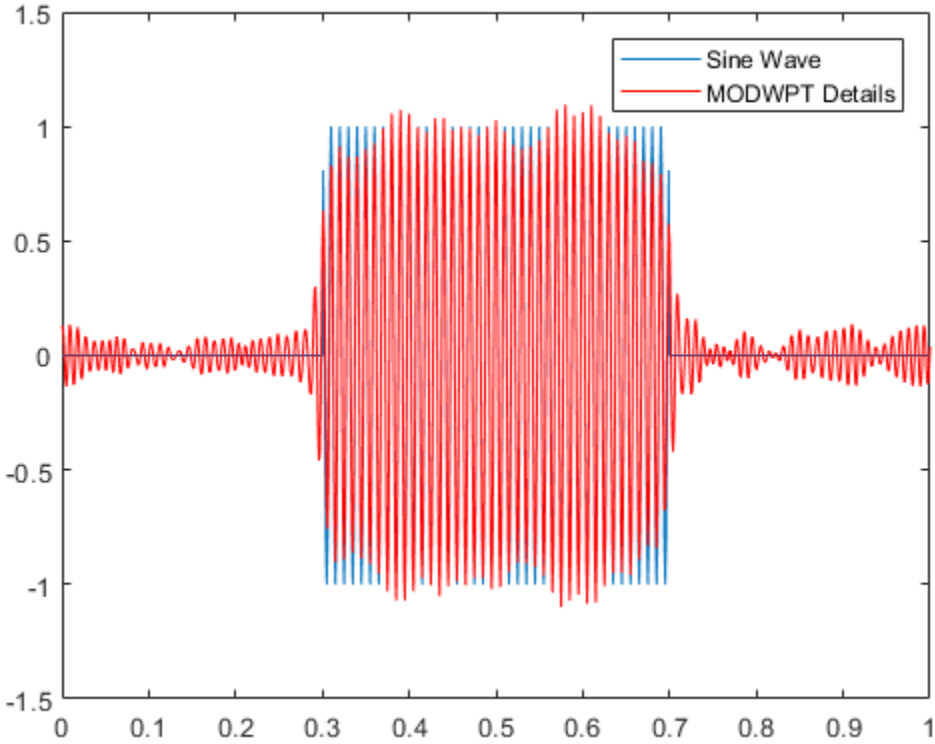
MODWPT Details for Noisy Sine Wave

Obtain the MODWPT details for a 100 Hz time-localized sine wave in noise. The sampling rate is 1000 Hz. Obtain the MODWPT at level 4 using the length 22 Fejer-Korovkin (`'fk22'`) wavelet.

```
dt = 0.001;
t = 0:dt:1;
x = cos(2*pi*100*t).*(t>0.3 & t<0.7)+0.25*randn(size(t));
wptdetails = modwptdetails(x,'fk22');
p4 = wptdetails(4,:);
```

Plot the MODWPT details for level 4, packet number 4. The MODWPT details represent zero-phase filtering of the input signal with an approximate passband of $[3Fs/2^5, 4Fs/2^5)$, where Fs is the sampling frequency.

```
plot(t,cos(2*pi*100*t).*(t>0.3 & t<0.7));
hold on
plot(t,p4,'r')
legend('Sine Wave','MODWPT Details')
```



MODWPT Details Using Scaling and Wavelet Filters

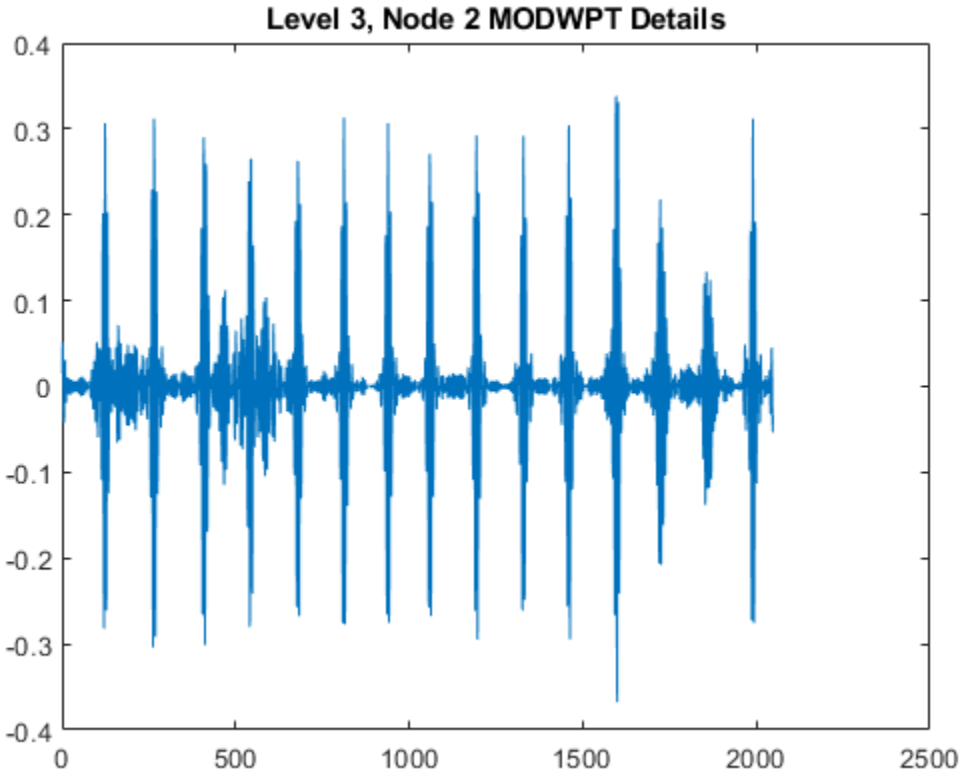
Obtain the MODWPT details of an ECG waveform using the length 18 Fejer-Korovkin scaling and wavelet filters.

```
load wecg;
[lo,hi] = wfilters('fk18');
wpt = modwptdetails(wecg,lo,hi);
```

MODWPT Details for Full Packet Tree

Obtain the MODWPT details for the full wavelet packet tree of an ECG waveform. Use the default length 18 Fejer-Korovkin (`'fk18'`) wavelet. Extract and plot the node coefficients at level 3, node 2.

```
load wecg;
[w,packetlevels] = modwptdetails(wecg,'FullTree',true);
p3 = w(packetlevels==3,:);
plot(p3(3,:))
title('Level 3, Node 2 MODWPT Details')
```



Input Arguments

`x` — Input signal

real-valued vector

Input signal, specified as a real-valued row or column vector. `x` must have at least two elements.

Data Types: `double`

`wname` — Analyzing wavelet filter

`fk18` (default) | character vector

Analyzing wavelet filter, specified as a character vector that corresponds to an orthogonal wavelet.

Valid orthogonal wavelet families begin with one of the following character vectors, followed by an integer, *N*. For example, `sym4`.

- `'haarN'` — Haar wavelet with *N* vanishing moments
- `'dbN'` — Daubechies wavelet with *N* vanishing moments
- `'symN'` — Symlets wavelet with *N* vanishing moments
- `'coifN'` — Coiflets wavelet with *N* vanishing moments
- `'fkN'` — Fejer-Korovkin wavelet with *N* coefficients

To check if your wavelet is orthogonal, use `wavemngr('type',wname)` and verify that it returns `1` as the wavelet type. To determine valid values for *N*, use `waveinfo`. For example, `waveinfo('fk')`.

`lo` — Scaling filter

even-length real-valued vector

Scaling filter, specified as an even-length real-valued vector. `lo` must satisfy the conditions necessary to generate an orthogonal scaling function. You can specify the `lo` and `hi` scaling-wavelet filter pair only if you do not specify `wname`.

`hi` — Wavelet filter

even-length real-valued vector

Wavelet filter, specified as an even-length real-valued vector. `hi` must satisfy the conditions necessary to generate an orthogonal wavelet. You can specify the `lo` and `hi` scaling-wavelet filter pair only if you do not specify `wname`.

`lev` — Transform level

positive integer

Transform level, specified as a positive integer less than or equal to `floor(log2(numel(x)))`.

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: `'Fulltree',true` returns the full wavelet packet tree

`FullTree` — Option to return full packet tree details
`false` (default) | `true`

Option to return full wavelet packet tree details, specified as the comma-separated pair consisting of `'FullTree'` and either `false` or `true`. If you specify `false`, then `modwptdetails` returns details about only the terminal (final-level) wavelet packet nodes. If you specify `true`, then `modwptdetails` returns details about the full wavelet packet tree down to the default or specified level.

For the full wavelet packet tree, `w` is a 2^{j+1}-2-by-`numel(x)` matrix. Each level, j, has 2^j wavelet packet details.

Output Arguments

`w` — Wavelet packet tree details
matrix

Wavelet packet tree details, returned as a matrix with each row containing the sequency-ordered wavelet packet details for the terminal nodes. The terminal nodes are at level 4 or at level `floor(log2(numel(x)))`, whichever is smaller. The MODWPT details are zero-phase-filtered projections of the signal onto the subspaces corresponding to the wavelet packet nodes. The sum of the MODWPT details over each sample reconstructs the original signal.

For the default terminal nodes, w is a 2^j-by-`numel(x)` matrix. For the full packet table, at level j, `w` is a 2^{j+1}-2-by-`numel(x)` matrix of sequency-ordered wavelet packet coefficients by level and index. The approximate passband for the nth row of `w` at level j is

$$\left[\frac{n-1}{2^{(j+1)}},\frac{n}{2^{(j+1)}}\right)$$ cycles per sample, where $n = 1,2,...,2^j$.

`packetlevs` — Transform levels
vector

Transform levels, returned as a vector. The levels correspond to the rows of `w`. If `w` contains only the terminal level coefficients, `packetlevs` is a vector of constants equal to the terminal level. If `w` contains the full wavelet packet tree of details, `packetlevs` is a vector with 2^{j-1} elements for each level, j. To select all the MODWPT details at a particular level, use `packetlevs` with logical indexing.

`cfreq` — Center frequencies of passbands
vector

Center frequencies of the approximate passbands in the `w` rows, returned as a vector. The center frequencies are in cycles per sample. To convert the units to cycles per unit time, multiply `cfreq` by the sampling frequency.

Algorithms

The MODWPT details (`modwptdetails`) are the result of zero-phase filtering of the signal. The features in the MODWPT details align exactly with features in the input signal. For a given level, summing the details for each sample returns the exact original signal.

The output of the MODWPT (`modwpt`) is time delayed compared to the input signal. Most filters used to obtain the MODWPT have a nonlinear phase response, which makes compensating for the time delay difficult. All orthogonal scaling and wavelet filters have this response, except the Haar wavelet. It is possible to time align the coefficients with the signal features, but the result is an approximation, not an exact alignment with the original signal. The MODWPT partitions the energy among the wavelet packets at each level. The sum of the energy over all the packets equals the total energy of the input signal.

References

[1] Percival, D. B., and A. T. Walden. *Wavelet Methods for Time Series Analysis*. Cambridge, UK: Cambridge University Press, 2000.

[2] Walden, A.T., and A. Contreras Cristan. “The phase-corrected undecimated discrete wavelet packet transform and its application to interpreting the timing of

events.” *Proceedings of the Royal Society of London A*. Vol. 454, Issue 1976, 1998, pp. 2243-2266.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`imodwpt` | `modwpt`

Introduced in R2016a

modwt

Maximal overlap discrete wavelet transform

Syntax

```
w = modwt(x)
w = modwt(x,wname)
w = modwt(x,Lo,Hi)
w = modwt(___,lev)
w = modwt(___,'reflection')
```

Description

`w = modwt(x)` returns the maximal overlap discrete wavelet transform (MODWT) of the 1-D real-valued signal, `x`.

`w = modwt(x,wname)` uses the orthogonal wavelet, `wname`, for the MODWT.

`w = modwt(x,Lo,Hi)` uses the scaling filter, `Lo`, and wavelet filter, `Hi`, to compute the MODWT. These filters must satisfy the conditions for an orthogonal wavelet. You cannot specify `Lo` and `Hi` if you specify `wname`.

`w = modwt(___,lev)` computes the MODWT down to the specified level, `lev`, using any of the arguments from previous syntaxes.

`w = modwt(___,'reflection')` computes the MODWT using reflection boundary handling. Other inputs can be any of the arguments from previous syntaxes. Before computing the wavelet transform, `modwt` extends the signal symmetrically at the right boundary to twice the signal length, `[x flip(x)]`. The number of wavelet and scaling coefficients that `modwt` returns is equal to twice the length of the input signal. By default, the signal is extended periodically.

Examples

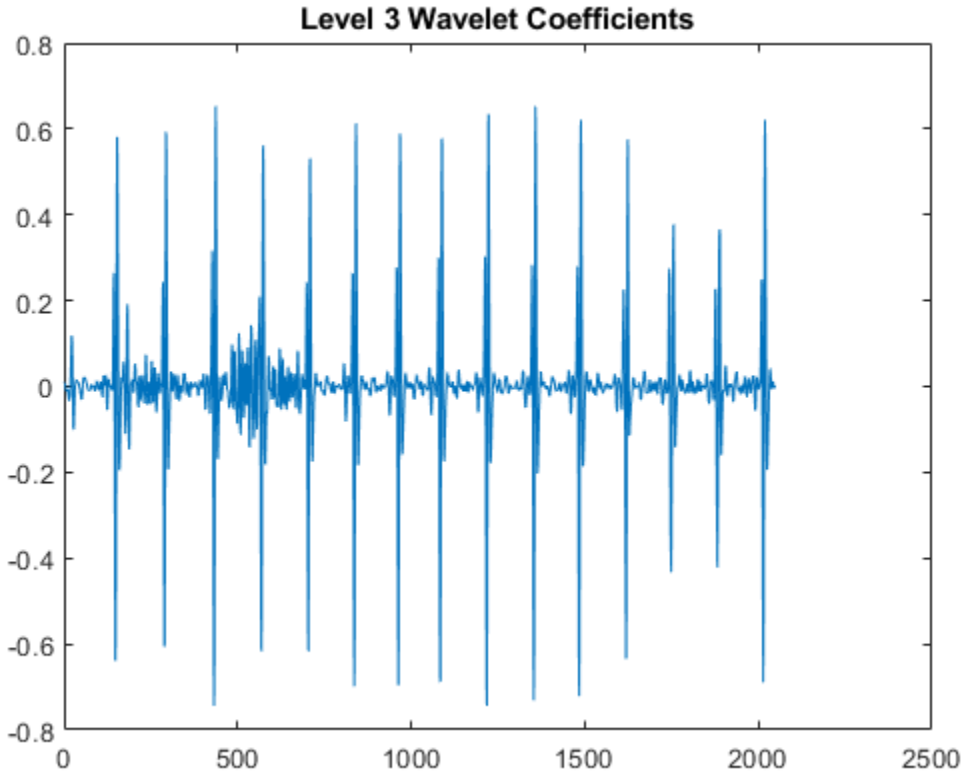
MODWT Using Default Wavelet

Obtain the MODWT of an electrocardiogram (ECG) signal using the default `sym4` wavelet down to the maximum level.

```
load wecg;
wtecg = modwt(wecg);
```

wtecg is 12-by-2048 matrix. The first eleven rows are the wavelet coefficients for scales 2^1 to 2^{11}. The final row contains the scaling coefficients at scale 2^{11}. Plot the detail (wavelet) coefficients for scale 2^3.

```
plot(wtecg(3,:))
title('Level 3 Wavelet Coefficients')
```



MODWT Using Daubechies Extremal Phase Wavelet with Two Vanishing Moments

Obtain the MODWT of Southern Oscillation Index data with the `'db2'` wavelet down to the maximum level.

```
load soi;
wsoi = modwt(soi,'db2');
```

MODWT Using Scaling and Wavelet Filters

Obtain the MODWT of the Deutsche Mark - U.S. Dollar exchange rate data using the Fejer-Korovkin length 8 scaling and wavelet filters.

```
load DM_USD;
[Lo,Hi] = wfilters('fk8');
wdm = modwt(DM_USD,Lo,Hi);
```

MODWT to a Specified Level

Obtain the MODWT of an ECG signal down to scale 2^4, which corresponds to level four. Use the default `'sym4'` wavelet.

```
load wecg;
wtecg = modwt(wecg,4);
```

`wtecg` is a 5-by-2048 matrix. The row size is L+1, where, in this case, the level (L) is 4. The column size matched the number of input samples.

MODWT with Reflection Boundary

Obtain the MODWT of an ECG signal using reflection boundary handling. Use the default `'sym4'` wavelet and obtain the transform down to level 4.

```
load wecg;
wtecg = modwt(wecg,4,'reflection');
```

`wtecg` has 4096 columns, which is twice the length of the input signal, `wecg`.

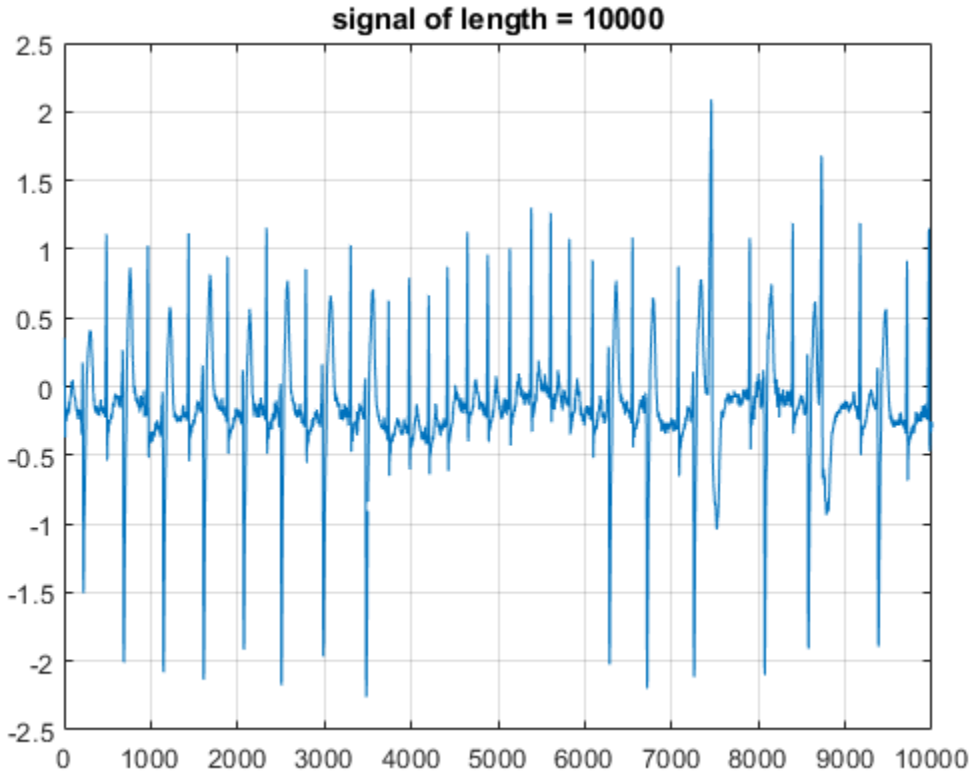
Comparing MODWT and MODWTMRA

This example demonstrates the differences between the functions MODWT and MODWTMRA. The MODWT partitions a signal's energy across detail coefficients and

scaling coefficients. The MODWTMRA projects a signal onto wavelet subspaces and a scaling subspace.

Choose the 'sym6' wavelet. Load and plot an ECG waveform. The ECG data is taken from the MIT-BIH Arrythmia Database.

```
load mit200
wv = 'sym6';
plot(ecgsig)
grid on
title(['signal of length = ',num2str(length(ecgsig))])
```



Take the MODWT of the signal.

```
wtecg = modwt(ecgsig,wv);
```

The input data are samples of a function $f(x)$ evaluated at N-many time points. The function can be expressed as a linear combination of the scaling function $\phi(x)$ and wavelet $\psi(x)$ at varying scales and translations:

$f(x) = \sum_{k=0}^{N-1} c_k 2^{-J_0/2} \phi(2^{-J_0}x - k) + \sum_{j=1}^{J_0} f_j(x)$ where

$f_j(x) = \sum_{k=0}^{N-1} d_{j,k} 2^{-j/2} \psi(2^{-j}x - k)$ and J_0 is the number of levels of wavelet decomposition. The first sum is the coarse scale approximation of the signal, and the $f_j(x)$ are the details at successive scales. MODWT returns the N-many coefficients $\{c_k\}$ and the $(J_0 \times N)$-many detail coefficients $\{d_{j,k}\}$ of the expansion. Each row in `wtecg` contains the coefficients at a different scale.

When taking the MODWT of a signal of length N, there are $\text{floor}(\log_2(N))$-many levels of decomposition (by default). Detail coefficients are produced at each level. Scaling coefficients are returned only for the final level. In this example, since $N = 10000$, $J_0 = \text{floor}(\log 2(10000)) = 13$ and the number of rows in `wtecg` is $J_0 + 1 = 13 + 1 = 14$.

The MODWT partitions the energy across the various scales and scaling coefficients:

$||X||^2 = \sum_{j=1}^{J_0} ||W_j||^2 + ||V_{J_0}||^2$ where X is the input data, W_j are the detail coefficients at scale j, and V_{J_0} are the final-level scaling coefficients.

Compute the energy at each scale, and evaluate their sum.

```
energy_by_scales = sum(wtecg.^2,2);
Levels = {'D1';'D2';'D3';'D4';'D5';'D6';'D7';'D8';'D9';'D10';'D11';'D12';'D13';'A13'};
energy_table = table(Levels,energy_by_scales);
disp(energy_table)
```

```
    Levels    energy_by_scales
    ______    ________________

    'D1'       0.31592
    'D2'        2.6504
    'D3'        28.802
    'D4'        159.37
    'D5'         300.5
```

```
    'D6'       431.33
    'D7'       444.93
    'D8'       182.37
    'D9'       45.381
    'D10'      11.578
    'D11'      19.809
    'D12'      4.5406
    'D13'       3.308
    'A13'      192.46

energy_total = varfun(@sum,energy_table(:,2))

energy_total=1x1 table
    sum_energy_by_scales
    ____________________

    1827.3
```

Confirm the MODWT is energy-preserving by computing the energy of the signal and comparing it with the sum of the energies over all scales.

```
energy_ecg = sum(ecgsig.^2);
max(abs(energy_total.sum_energy_by_scales-energy_ecg))

ans = 4.0870e-09
```

Take the MODWTMRA of the signal.

```
mraecg = modwtmra(wtecg,wv);
```

MODWTMRA returns the projections of the function $f(x)$ onto the various wavelet subspaces and final scaling space. That is, MODWTMRA returns $\sum_{k=0}^{N-1} c_k 2^{-J_0/2} \phi(2^{-J_0}x - k)$ and the J_0-many $\{f_j(x)\}$ evaluated at N-many time points. Each row in `mraecg` is a projection of $f(x)$ onto a different subspace. This means the original signal can be recovered by adding all the projections. This is *not* true in the case of the MODWT. Adding the coefficients in `wtecg` will *not* recover the original signal.

Choose a time point, add the projections of $f(x)$ evaluated at that time point and compare with the original signal.

```
time_point = 1000;
abs(sum(mraecg(:,time_point))-ecgsig(time_point))
```

```
ans = 3.0970e-13
```

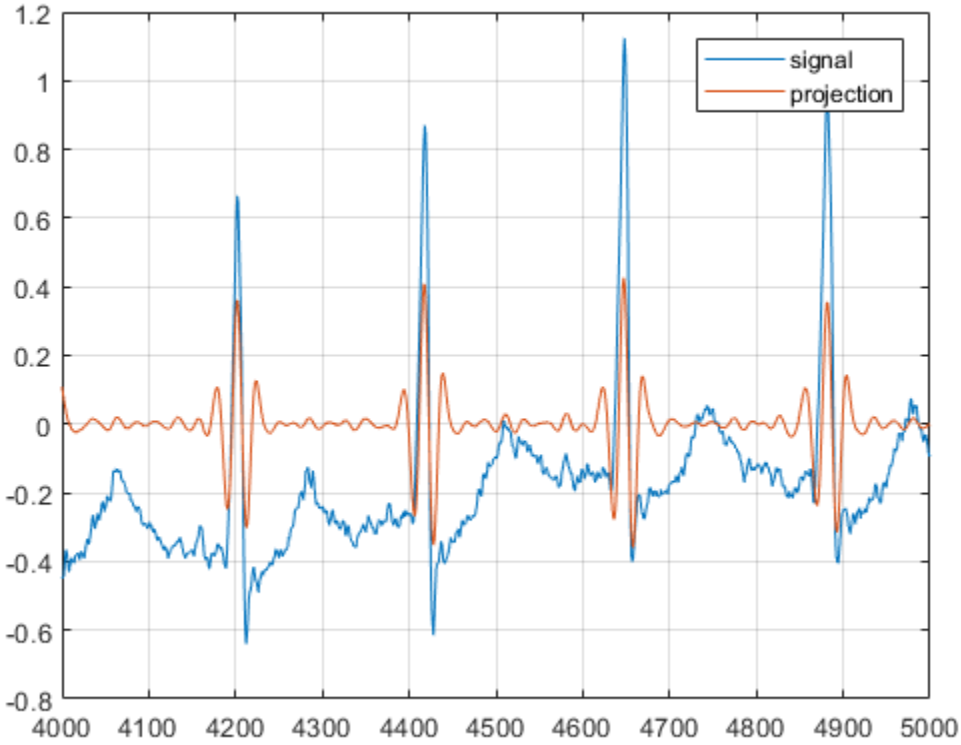
Confirm that, unlike MODWT, MODWTMRA is not an energy-preserving transform.

```
energy_ecg = sum(ecgsig.^2);
energy_mra_scales = sum(mraecg.^2,2);
energy_mra = sum(energy_mra_scales);
max(abs(energy_mra-energy_ecg))
```

```
ans = 534.7949
```

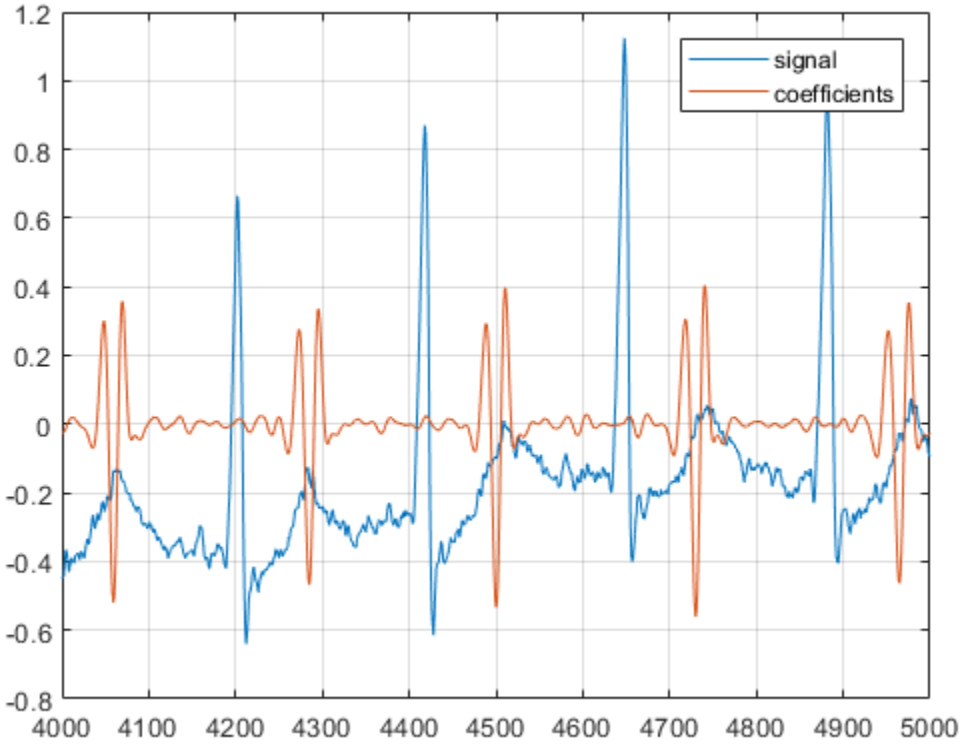
The MODWTMRA is a zero-phase filtering of the signal. Features will be time-aligned. Demonstrate this by plotting the original signal and one of its projections. To better illustrate the alignment, zoom in.

```
figure
plot(ecgsig)
hold on
plot(mraecg(4,:),'-')
grid on
xlim([4000 5000])
legend('signal','projection')
```



Make a similar plot using the MODWT coefficients at the same scale. Note that features will not be time-aligned.The MODWT is *not* a zero-phase filtering of the input.

```
figure
plot(ecgsig)
hold on
plot(wtecg(4,:),'-')
grid on
xlim([4000 5000])
legend('signal','coefficients')
```



References

Goldberger A. L., L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. Ch. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C-K Peng, H. E. Stanley. "PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals." Circulation 101. Vol.23, e215-e220, 2000. `http://circ.ahajournals.org/cgi/content/full/101/23/e215`

Moody, G. B. "Evaluating ECG Analyzers". `http://www.physionet.org/physiotools/wfdb/doc/wag-src/eval0.tex`

Moody G. B., R. G. Mark. "The impact of the MIT-BIH Arrhythmia Database." IEEE Eng in Med and Biol. Vol. 20, Number 3, 2001), pp. 45-50 .

- “Wavelet Analysis of Financial Data”

Input Arguments

`x` — Input signal

real-valued vector

Input signal, specified as a row or column vector. `x` must have at least two elements.

By default, `modwt` computes the wavelet transform down to level `floor(log2(length(x)))` using the Daubechies least-asymmetric wavelet with four vanishing moments (`'sym4'`) and periodic boundary handling.

Data Types: `double`

`wname` — Analyzing wavelet

`'sym4'` (default) | character vector

Analyzing wavelet, specified as a character vector that corresponds to an orthogonal wavelet. Valid orthogonal wavelet families begin with one of the following character vectors, followed by an integer, *N*, which indicates the number of vanishing moments, for example, `sym`*N*. For `'fk'`, *N* is the number of coefficients.

- `'haar'` — Haar wavelet
- `'db'` — Daubechies wavelet
- `'sym'` — Symlets wavelet
- `'coif'` — Coiflets wavelet
- `'fk'` — Fejer-Korovkin wavelet

To check if your wavelet is orthogonal, use `wavemngr('type',wname)` and verify that it returns `1` as the wavelet type. To determine valid values for *N*, use `waveinfo`, for example, `waveinfo('db')`.

`Lo` — Scaling filter

even-length real-valued vector

Scaling filter, specified as an even-length real-valued vector. `Lo` must satisfy the conditions necessary to generate an orthogonal scaling function. You can specify `Lo` only if you do not specify `wname`.

`Hi` — Wavelet filter

even-length real-valued vector

Wavelet filter, specified as an even-length real-valued vector. `Hi` must satisfy the conditions necessary to generate an orthogonal wavelet. You can specify `Hi` only if you do not specify `wname`.

`lev` — Transform level

positive integer

Transform level, specified as a positive integer less than or equal to `floor(log2(length(x)))`.

Output Arguments

`w` — Wavelet transform

matrix

Wavelet transform , returned as an L+1-by-N matrix containing wavelet coefficients and final-level scaling coefficients. L is the level of the MODWT. N is equal to the input signal length unless you specify `'reflection'` boundary handling, in which case N is twice the length of the input signal. The kth row of `w` contains the wavelet coefficients for scale 2^k (wavelet scale $2^{(k-1)}$). The final, (L+1)th, row of `w` contains the scaling coefficients for scale 2^L.

Algorithms

The standard algorithm for the MODWT implements the circular convolution directly in the time domain. This implementation of the MODWT performs the circular convolution in the Fourier domain. The wavelet and scaling filter coefficients at level j are computed by taking the inverse discrete Fourier transform (DFT) of a product of DFTs. The DFTs in the product are the signal's DFT and the DFT of the j^{th} level wavelet or scaling filter.

Let H_k and G_k denote the length N DFTs of the MODWT wavelet and scaling filters, respectively. Let j denote the level and N denote the sample size.

The j[th] level wavelet filter is defined by

$$\frac{1}{N}\sum_{k=0}^{N-1} H_{j,k} e^{i2\pi nk/N}$$

where

$$H_{j,k} = H_{2^{j-1}k \bmod N} \prod_{m=0}^{j-2} G_{2^m k \bmod N}$$

The j[th] level scaling filter is

$$\frac{1}{N}\sum_{k=0}^{N-1} G_{j,k} e^{i2\pi nk/N}$$

where

$$G_{j,k} = \prod_{m=0}^{j-1} G_{2^m k \bmod N}$$

References

[1] Percival, D. B., and A. T. Walden. *Wavelet Methods for Time Series Analysis*. Cambridge, UK: Cambridge University Press, 2000.

[2] Percival, D. B., and H. O. Mofjeld. “Analysis of subtidal coastal sea level fluctuations using wavelets.”*Journal of the American Statistical Association*. Vol. 92, pp 868–880.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`imodwt` | `modwtcorr` | `modwtmra` | `modwtvar` | `modwtxcorr`

Topics

"Wavelet Analysis of Financial Data"

Introduced in R2015b

modwtcorr

Multiscale correlation using the maximal overlap discrete wavelet transform

Syntax

```
wcorr = modwtcorr(w1,w2)
wcorr = modwtcorr(w1,w2,wav)

[wcorr,wcorrci] = modwtcorr(___)
[wcorr,wcorrci] = modwtcorr(___,conflevel)
[wcorr,wcorrci,pval] = modwtcorr(___)
[wcorr,wcorrci,pval,nj] = modwtcorr(___)

wcorrtable = modwtcorr(___,'table')

[___] = modwt(___,'reflection')

modwtcorr(___)
```

Description

`wcorr = modwtcorr(w1,w2)` returns the wavelet correlation by scale for the maximal overlap discrete wavelet transforms (MODWTs) specified in `w1` and `w2`. `wcorr` is an *M*-by-1 vector of correlation coefficients, where *M* is the number of levels with nonboundary wavelet coefficients. If the final level has enough nonboundary coefficients, `modwtcorr` returns the scaling correlation in the final row of `wcorr`.

`wcorr = modwtcorr(w1,w2,wav)` uses the wavelet `wav` to determine the number of boundary coefficients by level.

`[wcorr,wcorrci] = modwtcorr(___)` returns in `wcorrci` the lower and upper 95% confidence bounds for the correlation coefficients of `wcorr`, using any arguments from the previous syntaxes.

`[wcorr,wcorrci] = modwtcorr(___,conflevel)` uses `conflevel` for the coverage probability of the confidence interval. `conflevel` is a real scalar strictly greater than 0

and less than 1. If `conflevel` is unspecified or specified as empty, the coverage probability defaults to 0.95.

`[wcorr,wcorrci,pval] = modwtcorr(___)` returns the p-values for the null hypothesis test that the correlation coefficient in `wcorr` is equal to zero. `pval` is an *M*-by-2 matrix, where *M* is the number of levels with nonboundary wavelet coefficients. T

`[wcorr,wcorrci,pval,nj] = modwtcorr(___)` returns the number of nonboundary coefficients used in the computation of the correlation estimates by level, `nj`.

`wcorrtable = modwtcorr(___,'table')` returns an *M*-by-6 table with the correlation, confidence bounds, p-value, and adjusted p-value. The table also lists the number of nonboundary coefficients by level. The row names of the table `wcorrtable` designate the type and level of each estimate. For example, `D1` designates that the row corresponds to a wavelet or detail estimate at level 1 and `S6` designates that the row corresponds to the scaling estimate at level 6. The scaling correlation is only computed for the final level of the MODWT and only when there are nonboundary scaling coefficients. You can specify the `'table'` flag anywhere after the input transforms `w1` and `w2`. You must enter the entire character vector `'table'`. If you specify `'table'`, `modwtcorr` only outputs one argument.

`[___] = modwt(___,'reflection')` reduces the number of wavelet and scaling coefficients at each scale by half before computing the correlation. Use this option only when you obtain the MODWT of `w1` and `w2` were obtained using the `'reflection'` boundary condition. You must enter the entire character vector `'reflection'`. If you added a wavelet named 'reflection' using the wavelet manager, you must rename that wavelet prior to using this option.

`modwtcorr` supports only unbiased estimates of the wavelet correlation. For these estimates, the algorithm must removed the extra coefficients obtained using the `'reflection'` boundary condition. Specifying the `'reflection'` option in `modwtcorr` is identical to first obtaining the MODWT of `w1` and `w2` using the default `'periodic'` boundary handling and then computing the wavelet correlation estimates.

`modwtcorr(___)` with no output arguments plots the wavelet correlations by scale with lower and upper confidence bounds. By default, the coverage probability is 0.95. Scales with NaNs for the confidence bounds and the scaling correlation are excluded.

Examples

Correlation by Scale

Find the correlation by scale for monthly DM-USD exhange rate returns from 1970 to 1998. The return data are log transformed. Use the Daubechies wavelet with two vanishing moments ('db2') to obtain the MODWT down to level 6. Then, obtain the correlation data.

```
load DM_USD;
load JY_USD;
wdm = modwt(DM_USD,'db2',6);
wjy = modwt(JY_USD,'db2',6);
wcorr = modwtcorr(wdm,wjy,'db2')
```

```
wcorr =

    0.5854
    0.5748
    0.6264
    0.4948
    0.3787
    0.9072
    0.7976
```

`wcorr` contains seven elements. The first six elements are the correlation coefficients for the wavelet (detail) levels one to six. The final element is the correlation for the scaling (lowpass) level six.

Multiscale Correlation

Obtain the MODWT of the Southern Oscillation Index and Truk Island daily pressure data sets. Tabulate the correlation between the two data sets by level.

```
load soi;
load truk;
wsoi = modwt(soi);
wtruk = modwt(truk);
wcorr = modwtcorr(wsoi,wtruk)
```

```
wcorr =

    0.1749
    0.2936
    0.0914
    0.0883
    0.2667
    0.0894
   -0.0415
    0.4825
    0.4394
    0.7433
```

Show that the number of nonboundary coefficients, in this case, is less than the maximal length of the input. The MODWT is computed down to level thirteen, which is the maximal level for the length of the input. Level thirteen contains thirteen wavelet coefficient vectors and one scaling coefficient vector.

```
size(wsoi,1)
```

```
ans = 14
```

The multiscale correlations are computed only down to level ten because the levels after than do not contain nonboundary coefficients. For unbiased estimates, you must use nonboundary coefficients only.

```
numel(wcorr)
```

```
ans = 10
```

Confidence Intervals for Correlation

Obtain the MODWT of the monthly US-DM and US-JPY exchange return data from 1970 to 1998. The return data are log transformed. Use the Daubechies wavelet with two degrees of freedom ('db2') and obtain wavelet and obtain the MODWT of each series down to level six. Obtain the correlation estimates by scale and the 95% confidence intervals.

```
load DM_USD
load JY_USD
wdm = modwt(DM_USD,'db2',6);
wjy = modwt(JY_USD,'db2',6);
```

```
[wcorr,wcorrci] = modwtcorr(wdm,wjy,'db6');
[wcorr wcorrci]

ans =

    0.5855    0.4780    0.6756
    0.5753    0.4139    0.7017
    0.6257    0.4007    0.7796
    0.5578    0.1661    0.7974
    0.7202    0.1657    0.9287
```

The width of the confidence interval increases as you go down in level.

Confidence Intervals with 0.99 Coverage Probability

Specify the coverage probability for the confidence intervals. Obtain the 99% confidence intervals for the US-DM and US-JY exchange returns.

```
load DM_USD;
load JY_USD;
wdm = modwt(DM_USD,'db2',6);
wjy = modwt(JY_USD,'db2',6);
[wcorr,wcorrci] = modwtcorr(wdm,wjy,'db2',0.99);
[wcorr wcorrci]

ans =

    0.5854    0.4407    0.7005
    0.5748    0.3557    0.7340
    0.6264    0.3169    0.8153
    0.4948   -0.0646    0.8176
    0.3787   -0.5191    0.8792
    0.9072   -0.3006    0.9975
    0.7976   -0.6227    0.9941
```

P-values for Correlation

Return *p*-values for the test of zero correlation by scale. Obtain the MODWT of the DM-USD and JY-USD exchange return data down to level six using the Daubechies wavelet with two degrees of freedom ('db2') wavelet. Compute the correlation by scale and return the *p*-values.

```
load DM_USD;
load JY_USD;
wdm = modwt(DM_USD,'db2',6);
wjy = modwt(JY_USD,'db2',6);
[wcorr,wcorrci,pval] = modwtcorr(wdm,wjy,'db2');
format longe
pval
```

```
pval =
     2.694174887029593e-17     4.889927419958712e-16
     7.125460513473891e-09     6.466355415977556e-08
     7.012389783536512e-06     4.242495819039590e-05
     2.258540027996925e-02     1.024812537703605e-01
     2.805930327935258e-01     7.275376493146417e-01
     3.348079529469842e-02     1.215352869197552e-01
     1.059217509938027e-01     3.204132967562530e-01
```

```
format
```

The first column contains the *p*-value and the second column contains the adjusted *p*-value based on the false discovery rate.

Multiscale Correlation in Tabular Form

Output results from `modwtcorr` in tabular form. Obtain the MODWT of the DM-USD and JY-USD exchange returns down to level six using the Daubechies wavelet with two degrees of freedom ('db2'). Output the results in a table.

```
load DM_USD;
load JY_USD;
wdm = modwt(DM_USD,'db2',6);
wjy = modwt(JY_USD,'db2',6);
corrtable = modwtcorr(wdm,wjy,'db2','table')
```

```
corrtable =

  7x6 table

          NJ      Lower        Rho       Upper       Pvalue      AdjustedPvalue
          ___    ________    _______    _______    __________    ______________

    D1    344     0.47797    0.58542    0.67561    2.6942e-17    4.8899e-16
    D2    338     0.41329    0.57483    0.70129    7.1255e-09    6.4664e-08
    D3    326     0.40163    0.62641    0.78001    7.0124e-06    4.2425e-05
    D4    302    0.080255     0.4948    0.76342      0.022585       0.10248
    D5    254    -0.32954    0.37865    0.81417       0.28059       0.72754
    D6    158     0.12469    0.90716    0.99393      0.033481       0.12154
    S6    158    -0.28573    0.79761    0.98601       0.10592       0.32041
```

Correlation with Reflection Boundary Conditions

Obtain multiscale correlation estimates when using `'reflection'` boundary handling. Obtain the MODWT of the Southern Oscillation Index and Truk Islands pressure data sets using `'reflection'` boundary handling for both data sets.

```
load soi
load truk
wsoi = modwt(soi,'fk4',6,'reflection');
wtruk = modwt(truk,'fk4',6,'reflection');
corrtable = modwtcorr(wsoi,wtruk,'fk4',0.95,'reflection','table')
```

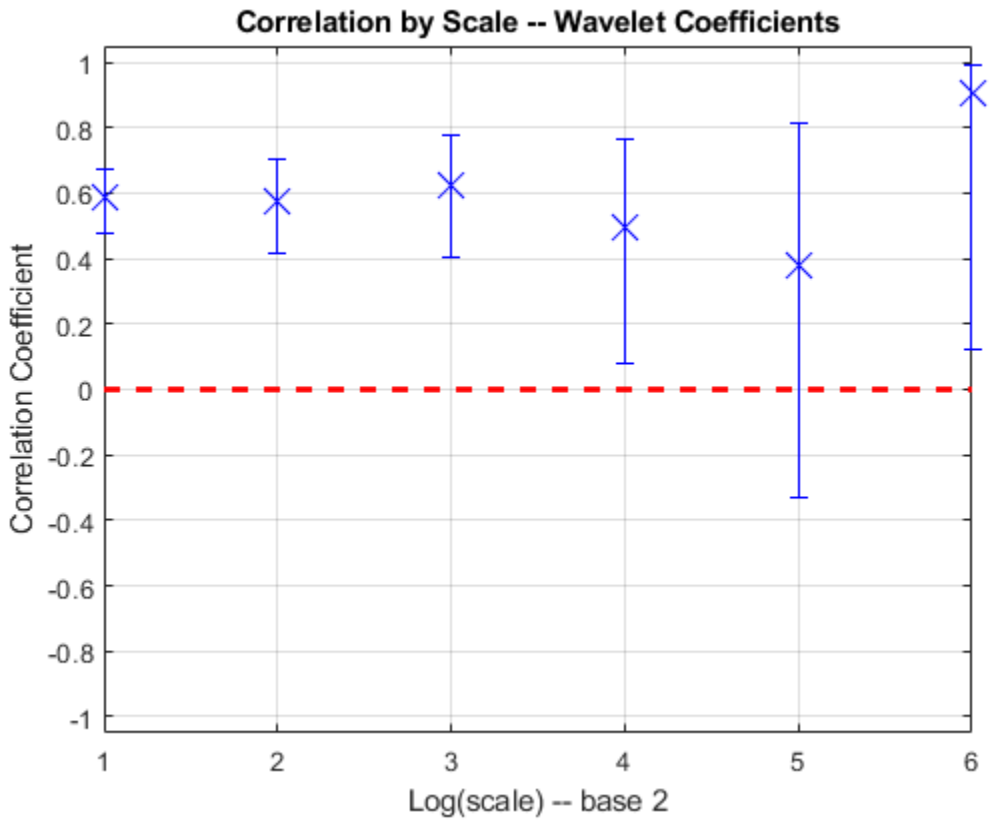
```
corrtable =

  7x6 table

          NJ        Lower        Rho       Upper       Pvalue      AdjustedPvalue
         _____    _________    _______    _______    __________    ______________

    D1    12995      0.16942    0.19294    0.21624    1.5466e-55    2.8071e-54
    D2    12989      0.21426    0.24683    0.27885    2.7037e-46    2.4536e-45
    D3    12977     0.057885    0.10623    0.15407     1.789e-05     6.494e-05
    D4    12953     0.048034    0.11645    0.18378    0.00088579     0.0026795
    D5    12905      0.13281     0.2272     0.3175    3.7566e-06    1.7046e-05
    D6    12809    -0.019835     0.1182    0.25181      0.093044       0.24125
```

```
    S6     12809       0.26664     0.39003     0.50084     8.8066e-09      5.328e-08
```

Plot Correlation with Confidence Intervals

Plot the multiscale correlation of the DM-USD and JY-USD exchange returns down to level six. Use modwtcorr with no output arguments.

```
load DM_USD;
load JY_USD;
wdm = modwt(DM_USD,'db2',6);
wjy = modwt(JY_USD,'db2',6);
modwtcorr(wdm,wjy,'db2')
```



- “Wavelet Cross-Correlation for Lead-Lag Analysis”
- “Wavelet Analysis of Financial Data”

Input Arguments

`w1` — MODWT transform of signal 1

matrix

MODWT transform of signal 1, specified as a matrix. `w1` is the output of `modwt`. It must be the same size and must have been obtained using the same analyzing wavelet.

Data Types: `double`

`w2` — MODWT transform of signal 2

matrix

MODWT transform of signal 2, specified as a matrix. `w2` is the output of `modwt`. It must be the same size and must have been obtained using the same analyzing wavelet.

`wav` — Wavelet

`'sym4'` (default) | character vector | positive even scalar

Wavelet, specified as a character vector indicating a valid wavelet name or as a positive even scalar indicating the length of the wavelet and scaling filters. `wav` must be the same wavelet and length used to obtain the MODWTs of `w1` and `w2`. For a list of valid wavelets, see `modwt`. If unspecified or specified as an empty, `[]`, `wav` defaults to the symlets wavelet with four degrees of freedom,`'sym4'`.

`conflevel` — Confidence level

0.95 (default) | positive scalar less than 1

Confidence level, specified as a positive scalar less than 1. `conflevel` determines the coverage probability of the confidence intervals in `wcorrci` and in the table, if you specify `'table'` as an input. If unspecified, or if specified as empty, `[]`, `conflevel` defaults to 0.95.

Output Arguments

`wcorr` — Correlation coefficients by scale

vector

Correlation coefficients by scale, returned as a vector. `wcorr` is an *M*-by-1 vector of correlation coefficients, where *M* is the number of levels with nonboundary wavelet coefficients. `modwtcorr` returns correlation estimates only where there are nonboundary coefficients. This condition is satisfied when the transform level is not greater than `floor(log2(N/(L-1)+1))`, where *N* is the length of the original signal and *L* is the filter length. If the final level has enough nonboundary coefficients, `modwtcorr` returns the scaling correlation in the final row of `wcorr`. By default, `modwtcorr` uses the symlet wavelet with four degrees of freedom `'sym4'` to determine the boundary coefficients.

`wcorrci` — Confidence intervals by scale

matrix

Confidence intervals by scale, returned as a matrix. The matrix is of size *M*-by-2 , where *M* is the number of levels with nonboundary wavelet coefficients. The first column contains the lower confidence bound and the second column contains the upper confidence bound. The `conflevel` determines the coverage probability.

Confidence bounds are computed using Fisher's Z-transformation. The standard error of Fisher's Z statistic is the square root of $(N - 3)$. In this case, *N* is the equivalent number of coefficients in the critically sampled discrete wavelet transform (DWT), `floor(size(w1,2)/2^LEV)`, where `LEV` is the level of the wavelet transform. `modwtcorr` returns NaNs for the confidence bounds when N^{-3} is less than or equal to zero.

`pval` — *P*-values for null hypothesis test

matrix

P-values for null hypothesis test, returned as a matrix. `pval` is an *M*-by-2 matrix.

- The first column of `pval` is the *p*-value computed using the standard *t*-statistic test for a correlation coefficient of zero.
- The second column of `pval` contains the adjusted *p*-value using the false discovery procedure of Benjamini & Yekutieli under arbitrary dependence assumptions.

The degrees of freedom, $(N - 2)$, for the t-statistic are determined by the equivalent number of coefficients in the critically sampled DWT, `floor(size(w1,2)/2^LEV)`, where `LEV` is the levels of the wavelet transform. `modwtcorr` returns NaNs when N^{-3} is less than or equal to zero.

`nj` — Number of nonboundary coefficients

vector

Number of nonboundary coefficients by scale, returned as a vector.

`wcorrtable` — Correlation table

table

Correlation table, returned as a MATLAB table. The table contains six variables:

- **NJ** — Number of nonboundary coefficients by level.
- **Lower** — Lower confidence bound for the coverage probability specified by `conflevel`.
- **Rho** — Correlation coefficient.
- **Upper** — Upper confidence bound for the coverage probability specified by `conflevel`.
- **Pvalue** — P-value for hypothesis test. The null hypothesis is that the correlation coefficient is equal to zero.
- **AdjustedPvalue** — P-value adjusted for multiple comparisons. The p-values are adjusted using false discovery rate under dependency assumptions.

References

[1] Percival, D. B., and Walden, A. T. *Wavelet Methods for Time Series Analysis*. Cambridge, U.K: Cambridge University Press, 2000.

[2] Whitcher, B., P. Guttorp, and D. B. Percival. "Wavelet analysis of covariance with application to atmospheric time series." *Journal of Geophysical Research*, Vol. 105, 2000, pp. 14941–14962.

See Also

`imodwt` | `modwt` | `modwtmra` | `modwtvar` | `modwtxcorr`

Topics

"Wavelet Cross-Correlation for Lead-Lag Analysis"
"Wavelet Analysis of Financial Data"

Introduced in R2015b

modwtmra

Multiresolution analysis based on MODWT

Syntax

```
mra = modwtmra(w)
mra = modwtmra(w,wname)
mra = modwtmra(w,Lo,Hi)
mra = modwtmra(___,'reflection')
```

Description

`mra = modwtmra(w)` returns the multiresolution analysis (MRA) of the maximal overlap discrete wavelet transform (MODWT) matrix, `w`. The MODWT matrix, `w`, is the output of the `modwt` function.

`mra = modwtmra(w,wname)` constructs the MRA using the wavelet corresponding to `wname`. The `wname` wavelet must be the same wavelet used to obtain the MODWT.

`mra = modwtmra(w,Lo,Hi)` constructs the MRA using the scaling filter `Lo` and wavelet filter `Hi`. The `Lo` and `Hi` filters must be the same filters used to obtain the MODWT.

`mra = modwtmra(___,'reflection')` uses the reflection boundary condition in the construction of the MRA using any of the arguments from previous syntaxes. If you specify `'reflection'`, `modwtmra` assumes that the length of the original signal is one half the number of columns in the input coefficient matrix.

Examples

Perfect Reconstruction with the MODWTMRA

Obtain the MODWTMRA of a simple time-series signal and demonstrate perfect reconstruction.

Create a time-series signal

```
t = 1:10;
x =sin(2*pi*200*t);
```

Obtain the MODWT and the MODWTMRA and sum the MODWTMRA rows.

```
m = modwt(x);
mra = modwtmra(m);
xrec = sum(mra);
```

Use the maximum of the absolute values to show that the difference between the original signal and the reconstruction is extremely small. The largest absolute value is on the order of 10^{-25}, which demonstrates perfect reconstruction.

```
max(abs(x-xrec))
```

```
ans = 5.5738e-25
```

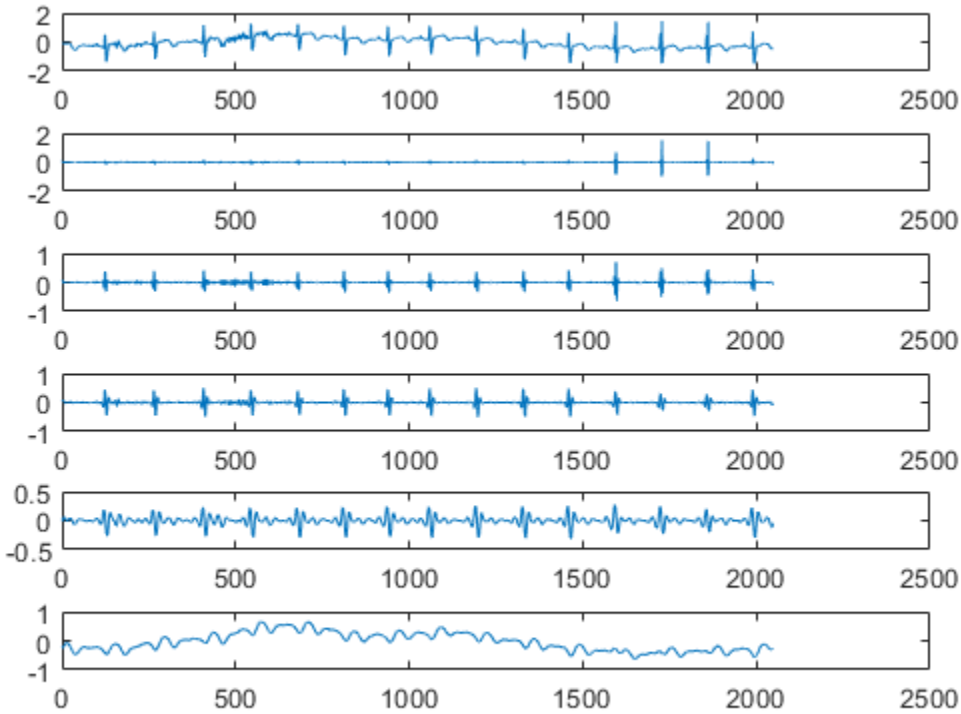
MRA Using Non-Default Wavelet

Construct an MRA of an ECG signal down to level four using the 'db2' wavelet.

```
load wecg;
lev = 4;
wtecg = modwt(wecg,'db2',lev);
mra = modwtmra(wtecg,'db2');
```

Plot the ECG waveform and the MRA.

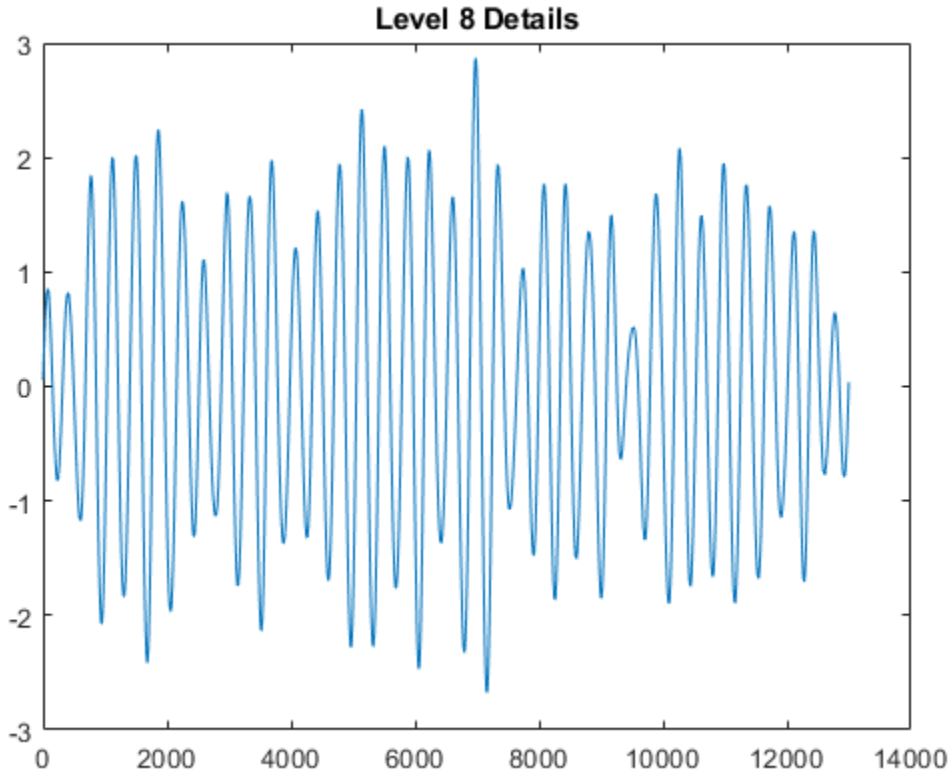
```
subplot(6,1,1)
plot(wecg)
    for kk = 2:lev+2
        subplot(6,1,kk)
        plot(mra(kk-1,:))
    end
```



MRA Using the Default Wavelet

Construct a multiresolution analysis for the Southern Oscillation Index data. The sampling period is one day. Plot the level eight details corresponding to a scale of 2^8 days. The details at this scale capture oscillations on a scale of approximately one year.

```
load soi
wtsoi = modwt(soi);
mrasoi = modwtmra(wtsoi);
plot(mrasoi(8,:))
title('Level 8 Details')
```



MRA Using Minimum Bandwidth Scaling and Wavelet Filters

Obtain the MRA for the Deutsch Mark - U.S. Dollar exchange rate data using the minimum bandwidth scaling and wavelet filters with four coefficients.

```
load DM_USD;
Lo = [0.4801755, 0.8372545, 0.2269312, -0.1301477];
Hi = qmf(Lo);
wdm = modwt(DM_USD,Lo,Hi);
mra = modwtmra(wdm,Lo,Hi);
```

MRA Using Reflection Boundary

Obtain the MRA for an ECG signal using `'reflection'` boundary handling.

```
load wecg;
wtecg = modwt(wecg,'reflection');
mra = modwtmra(wtecg,'reflection');
```

Show that the number of columns in the MRA is equal to the number of elements in the original signal.

```
isequal(size(mra,2),numel(wecg))
```

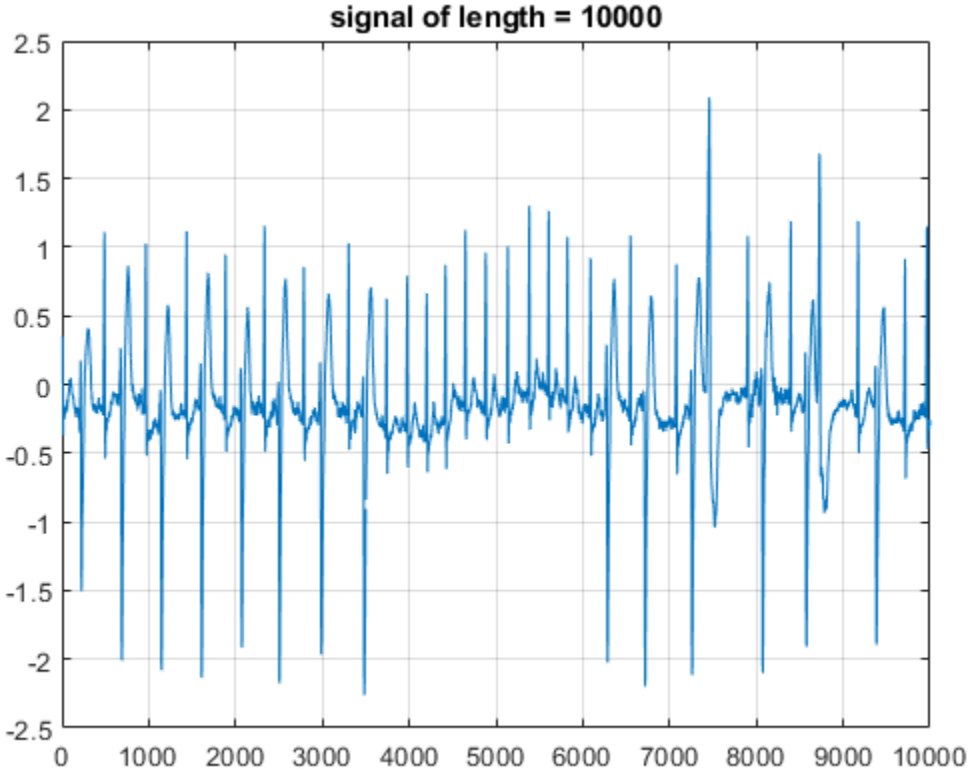
```
ans =

  logical

   1
```

Comparing MODWT and MODWTMRA

This example demonstrates the differences between the functions MODWT and MODWTMRA. The MODWT partitions a signal's energy across detail coefficients and scaling coefficients. The MODWTMRA projects a signal onto wavelet subspaces and a scaling subspace.

Choose the 'sym6' wavelet. Load and plot an ECG waveform. The ECG data is taken from the MIT-BIH Arrythmia Database.

```
load mit200
wv = 'sym6';
plot(ecgsig)
grid on
title(['signal of length = ',num2str(length(ecgsig))])
```



Take the MODWT of the signal.

```
wtecg = modwt(ecgsig,wv);
```

The input data are samples of a function $f(x)$ evaluated at N-many time points. The function can be expressed as a linear combination of the scaling function $\phi(x)$ and wavelet $\psi(x)$ at varying scales and translations:

$f(x) = \sum_{k=0}^{N-1} c_k 2^{-J_0/2} \phi(2^{-J_0}x - k) + \sum_{j=1}^{J_0} f_j(x)$ where

$f_j(x) = \sum_{k=0}^{N-1} d_{j,k} 2^{-j/2} \psi(2^{-j}x - k)$ and J_0 is the number of levels of wavelet decomposition. The first sum is the coarse scale approximation of the signal, and the

$f_j(x)$ are the details at successive scales. MODWT returns the N-many coefficients $\{c_k\}$ and the $(J_0 \times N)$-many detail coefficients $\{d_{j,k}\}$ of the expansion. Each row in `wtecg` contains the coefficients at a different scale.

When taking the MODWT of a signal of length N, there are $\text{floor}(\log_2(N))$-many levels of decomposition (by default). Detail coefficients are produced at each level. Scaling coefficients are returned only for the final level. In this example, since $N = 10000$, $J_0 = \text{floor}(\log 2(10000)) = 13$ and the number of rows in `wtecg` is $J_0 + 1 = 13 + 1 = 14$.

The MODWT partitions the energy across the various scales and scaling coefficients: $||X||^2 = \sum_{j=1}^{J_0} ||W_j||^2 + ||V_{J_0}||^2$ where X is the input data, W_j are the detail coefficients at scale j, and V_{J_0} are the final-level scaling coefficients.

Compute the energy at each scale, and evaluate their sum.

```
energy_by_scales = sum(wtecg.^2,2);
Levels = {'D1';'D2';'D3';'D4';'D5';'D6';'D7';'D8';'D9';'D10';'D11';'D12';'D13';'A13'};
energy_table = table(Levels,energy_by_scales);
disp(energy_table)
```

```
    Levels    energy_by_scales
    ______    ________________

    'D1'       0.31592
    'D2'        2.6504
    'D3'        28.802
    'D4'        159.37
    'D5'         300.5
    'D6'        431.33
    'D7'        444.93
    'D8'        182.37
    'D9'        45.381
    'D10'       11.578
    'D11'       19.809
    'D12'       4.5406
    'D13'        3.308
    'A13'       192.46
```

```
energy_total = varfun(@sum,energy_table(:,2))
```

```
energy_total=1x1 table
    sum_energy_by_scales
    ____________________

    1827.3
```

Confirm the MODWT is energy-preserving by computing the energy of the signal and comparing it with the sum of the energies over all scales.

```
energy_ecg = sum(ecgsig.^2);
max(abs(energy_total.sum_energy_by_scales-energy_ecg))
```

```
ans = 4.0870e-09
```

Take the MODWTMRA of the signal.

```
mraecg = modwtmra(wtecg,wv);
```

MODWTMRA returns the projections of the function $f(x)$ onto the various wavelet subspaces and final scaling space. That is, MODWTMRA returns $\sum_{k=0}^{N-1} c_k 2^{-J_0/2} \phi(2^{-J_0}x - k)$ and the J_0-many $\{f_j(x)\}$ evaluated at N-many time points. Each row in `mraecg` is a projection of $f(x)$ onto a different subspace. This means the original signal can be recovered by adding all the projections. This is *not* true in the case of the MODWT. Adding the coefficients in `wtecg` will *not* recover the original signal.

Choose a time point, add the projections of $f(x)$ evaluated at that time point and compare with the original signal.

```
time_point = 1000;
abs(sum(mraecg(:,time_point))-ecgsig(time_point))
```

```
ans = 3.0970e-13
```

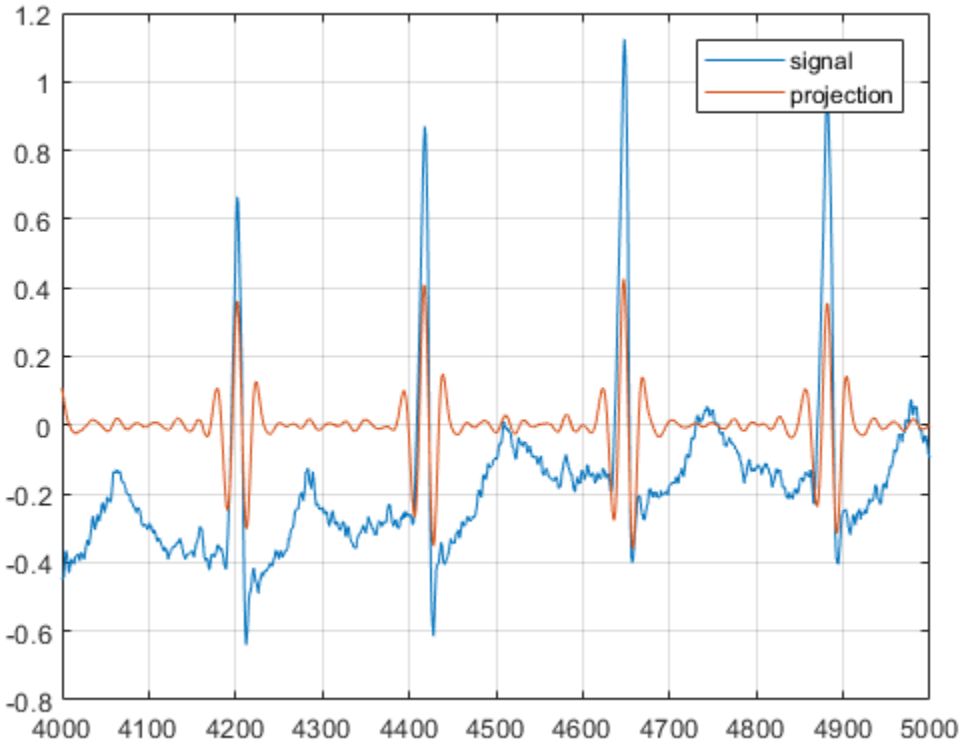
Confirm that, unlike MODWT, MODWTMRA is not an energy-preserving transform.

```
energy_ecg = sum(ecgsig.^2);
energy_mra_scales = sum(mraecg.^2,2);
energy_mra = sum(energy_mra_scales);
max(abs(energy_mra-energy_ecg))
```

```
ans = 534.7949
```

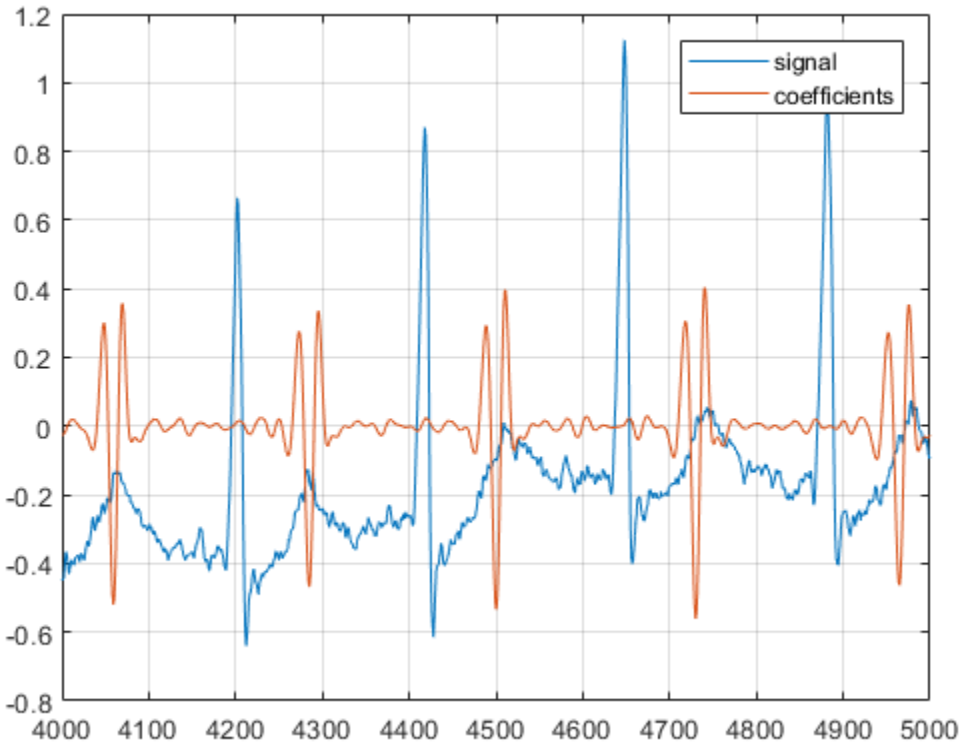
The MODWTMRA is a zero-phase filtering of the signal. Features will be time-aligned. Demonstrate this by plotting the original signal and one of its projections. To better illustrate the alignment, zoom in.

```
figure
plot(ecgsig)
hold on
plot(mraecg(4,:),'-')
grid on
xlim([4000 5000])
legend('signal','projection')
```



Make a similar plot using the MODWT coefficients at the same scale. Note that features will not be time-aligned.The MODWT is *not* a zero-phase filtering of the input.

```
figure
plot(ecgsig)
hold on
plot(wtecg(4,:),'-')
grid on
xlim([4000 5000])
legend('signal','coefficients')
```



References

Goldberger A. L., L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. Ch. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C-K Peng, H. E. Stanley. "PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals."

Circulation 101. Vol.23, e215-e220, 2000. http://circ.ahajournals.org/cgi/content/full/101/23/e215

Moody, G. B. "Evaluating ECG Analyzers". http://www.physionet.org/physiotools/wfdb/doc/wag-src/eval0.tex

Moody G. B., R. G. Mark. "The impact of the MIT-BIH Arrhythmia Database." IEEE Eng in Med and Biol. Vol. 20, Number 3, 2001), pp. 45-50 .

Input Arguments

`w` — MODWT transform

matrix

Maximal overlap discrete wavelet transform, specified as a matrix. `w` is the output of `modwt`.

The input `w` is an L+1-by-N matrix containing the MODWT of an N-point input signal down to level L. By default, `modwtmra` assumes that you obtained the MODWT using the symlet wavelet with four vanishing moments, `'sym4'`, and using periodic boundary handling.

Data Types: `double`

`wname` — Synthesis wavelet

`'sym4'` (default) | character vector

Synthesis wavelet, specified as a character vector. The synthesis wavelet must be the same wavelet used to obtain the MODWT with the `modwt` function.

`Lo` — Scaling filter

even-length real-valued vector

Scaling filter, specified as an even-length real-valued vector. You can specify `Lo` only if you do not specify `wname`. `Lo` must be the same scaling filter used to obtain the MODWT with the `modwt` function.

`Hi` — Wavelet filter

even-length real-valued vector

Wavelet filter, specified as an even-length real-valued vector. You can specify `Hi` only if you do not specify `wname`. `Hi` must be the same wavelet filter used to obtain the MODWT with the `modwt` function.

Output Arguments

`mra` — Multiresolution analysis

matrix

Multiresolution analysis, returned as a matrix. By default, the `mra` is the same size as the input transform matrix `w`. If you specify reflection boundary handling, then `mra` has one half the number of columns as the input matrix `w`.

The output `mra` is an L+1-by-N matrix. The k^{th} row of `mra` contains the details for the k^{th} level. The $(L+1)^{th}$ row of `mra` contains the L^{th} level smooth.

References

[1] Percival, D. B., and Walden, A. T. *Wavelet Methods for Time Series Analysis*. Cambridge, U.K: Cambridge University Press, 2000.

[2] Whitcher, B., P. Guttorp, and D. B. Percival. "Wavelet analysis of covariance with application to atmospheric time series." *Journal of Geophysical Research*, Vol. 105, 2000, pp. 14941–14962.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`imodwt` | `modwt`

Introduced in R2015b

modwtvar

Multiscale variance of maximal overlap discrete wavelet transform

Syntax

```
wvar = modwtvar(w)
wvar = modwtvar(w,wname)
[wvar,wvarci] = modwtvar(___)

[___] = modwtvar(w,wname,___,conflevel)
[___] = modwtvar(w,wname,___,Name,Value,)
[wvar,wvarci,nj] = modwtvar(w,wname,___)

wvartable = modwtvar(w,wname,'table')

modwtvar(___)
```

Description

`wvar = modwtvar(w)` returns unbiased estimates of the wavelet variance by scale for the maximal overlap discrete wavelet transform (MODWT). The default wavelet type is `sym4`.

`wvar = modwtvar(w,wname)` uses the wavelet `wname` to determine the number of boundary coefficients by level for unbiased estimates.

`[wvar,wvarci] = modwtvar(___)` returns the 95% confidence intervals for the variance estimates by scale.

`[___] = modwtvar(w,wname,___,conflevel)` uses `conflevel` for the coverage probability of the confidence interval.

`[___] = modwtvar(w,wname,___,Name,Value,)` returns wavelet variance with additional options specified by one or more `Name,Value` pair arguments.

`[wvar,wvarci,nj] = modwtvar(w,wname, ___ )` returns the number of coefficients used to form the variance and confidence intervals by level.

`wvartable = modwtvar(w,wname,'table')`, where `'table'` returns a MATLAB table, `wvartable`, containing the number of MODWT coefficients by level, the confidence boundaries, and the variance estimates. You can place `'table'` anywhere after input `w`, except between the name and value of another `Name,Value` pair.

`modwtvar( ___ )` with no output arguments plots the wavelet variances by scale with lower and upper confidence bounds. The scaling variance is not included in the plot because the scaling variance can be much larger than the wavelet variances.

Examples

Wavelet Variance Using Default Wavelet

Obtain the MODWT of the Southern Oscillation Index data using the default symlets wavelet with 4 vanishing moements. Compute the unbiased estimates of the wavelet variance by scale.

```
load soi
wsoi = modwt(soi);
wvar = modwtvar(wsoi)
```

```
wvar =

    0.3568
    0.9026
    1.1576
    1.0952
    0.9678
    0.5478
    0.6353
    1.9570
    0.8398
    0.8247
```

Wavelet Variance Using Specified Wavelet

Obtain the MODWT of the Southern Oscillation Index data using the Daubechies wavelet with 2 vanishing moments ('db2'). Compute the unbiased estimates of the wavelet variance by scale.

```
load soi
wsoi = modwt(soi,'db2');
wvar = modwtvar(wsoi,'db2')
```

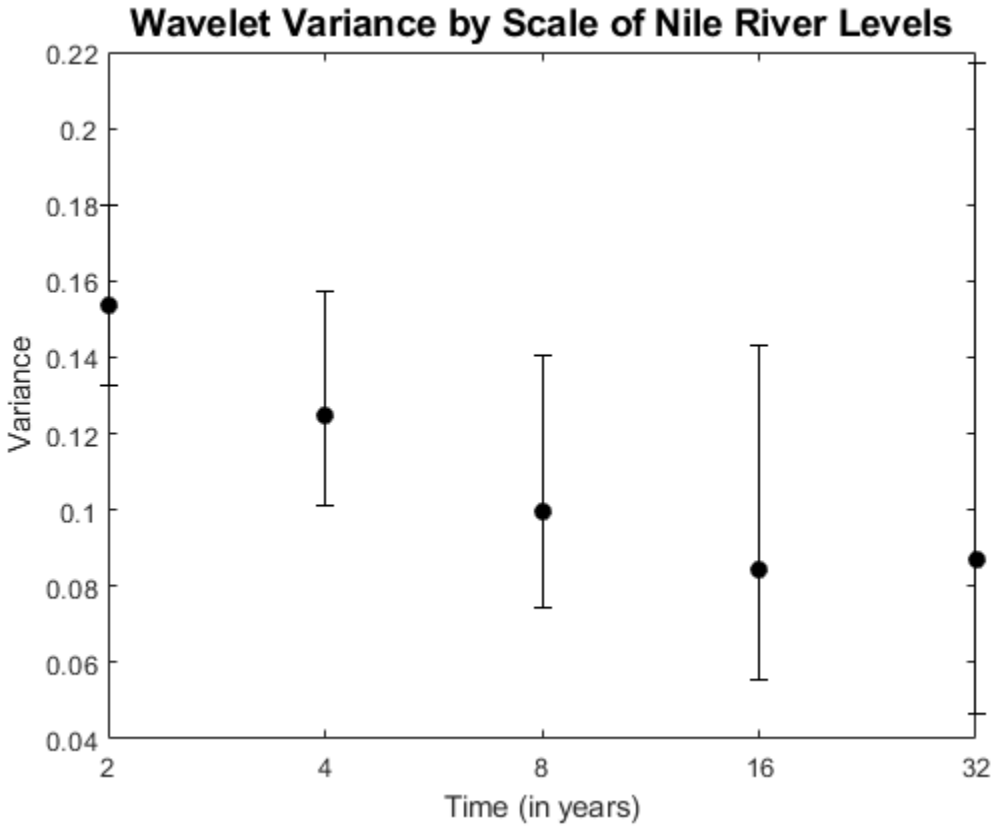
```
wvar =

    0.4296
    0.9204
    1.1370
    1.0847
    0.9255
    0.5932
    0.7630
    1.6672
    0.8048
    0.7555
```

Variance Estimates and Confidence Intervals Using MODWTVAR

Obtain the MODWT of the Nile River minimum level data using the Fejer- Korovkin wavelet with eight coefficients down to level five. Use `modwtvar` to obtain and plot the variance estimates and 95% confidence intervals.

```
load nileriverminima;
wtnile = modwt(nileriverminima,'fk8',5);
[wnilevar,wvarci] = modwtvar(wtnile,'fk8');

errlower = (wnilevar-wvarci(:,1));
errupper = (wvarci(:,2)-wnilevar);
errorbar(1:5,wnilevar(1:5),errlower(1:5),...
    errupper(1:5),'ko','markerfacecolor','k')
hold on
title('Wavelet Variance by Scale of Nile River Levels','fontsize',14);
ylabel('Variance');
xlabel('Time (in years)');
```

```
ax = gca;
ax.XTick = [1:5];
ax.XTickLabel = {'2','4','8','16','32'};
hold off
```



Wavelet Confidence Intervals

Show how different confidence level values affect the width of the confidence intervals. An increased confidence level value increases the confidence interval width.

Obtain the MODWT of the Southern Oscillation Index data using the Fejer-Korovkin wavelet with eight coefficients.

```
load soi;
wsoi = modwt(soi,'fk8');
```

Obtain the width of the.90, .95, and .99 confidence intervals for each level.

```
[~,wvarci90] = modwtvar(wsoi,'fk8',0.90);
w90 = wvarci90(:,2)-wvarci90(:,1);
[~,wvarci95] = modwtvar(wsoi,'fk8',0.95);
w95 = wvarci95(:,2)-wvarci95(:,1);
[~,wvarci99] = modwtvar(wsoi,'fk8',0.99);
w99 = wvarci99(:,2)-wvarci99(:,1);
```

Compare the three columns. The first column shows the .90 confidence level values, the second the .95 values, and the third the .99 values. Each row is the width of the interval at each wavelet scale. You can see that the width of the confidence interval increases with larger confidence level values.

```
[w90,w95,w99]

ans =

    0.0195    0.0233    0.0306
    0.0739    0.0880    0.1158
    0.1347    0.1606    0.2113
    0.1798    0.2145    0.2826
    0.2304    0.2751    0.3634
    0.1825    0.2184    0.2900
    0.2858    0.3435    0.4613
    1.5445    1.8757    2.5837
    1.0625    1.3262    1.9551
    2.8460    3.9883    7.8724
```

Compare Chi2Eta2 and Gaussian Confidence Intervals

Specify non-default confidence methods using name-value pairs to compare the width of their confidence levels. Note that for Gaussian confidence level intervals, it is possible to obtain negative lower confidence bounds.

Obtain the MODWT of the Southern Oscillation Index data using the Fejer-Korovkin wavelet with eight coefficients.

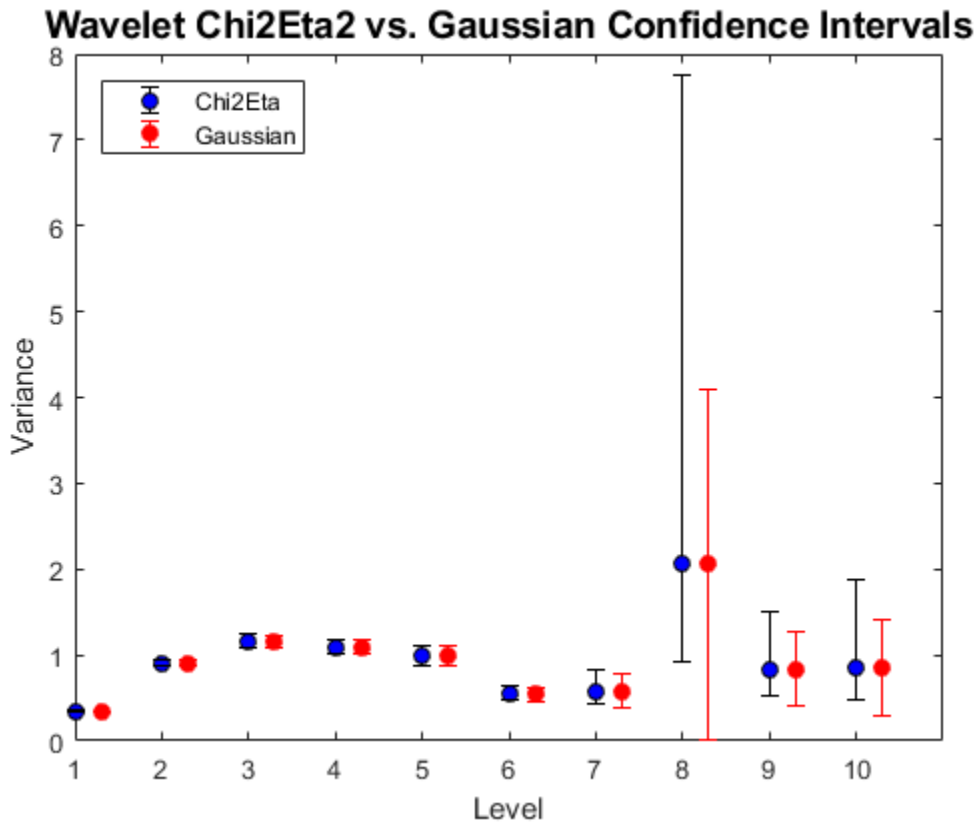
```
load soi;
wsoi = modwt(soi,'fk8');
```

Use the Chi2Eta and Gaussian confidence methods to obtain the variances and confidence interval bounds for each method.

```
[wvar_c,wvarci_c] = modwtvar(wsoi,'fk8',[],'ConfidenceMethod','chi2eta1');
[wvar_g,wvarci_g] = modwtvar(wsoi,'fk8',[],'ConfidenceMethod','gaussian');
```

Compute the upper and lower errors for each confidence interval and plot the results. Note that the Gaussian intervals are slightly shifted to enable better visualization.

```
errlower_c = wvar_c-wvarci_c(:,1);
errupper_c = wvarci_c(:,2)-wvar_c;

errlower_g = wvar_g(:,1)-wvarci_g(:,1);
errupper_g = wvarci_g(:,2)-wvar_g;

errorbar(1:10,wvar_c(1:10),errlower_c(1:10),...
    errupper_c(1:10),'ko','markerfacecolor','b')
hold on;
xoffset = (1.3:10.3);
errorbar(xoffset,wvar_g(1:10),errlower_g(1:10),...
    errupper_g(1:10),'ro','markerfacecolor','r')

title('Wavelet Chi2Eta2 vs. Gaussian Confidence Intervals','fontsize',14);
ylabel('Variance');
xlabel('Level')
ax = gca;
ax.XTick = [1:10];
legend('Chi2Eta','Gaussian','Location','northwest');
hold off
```



Compare Number of Coefficients for Unbiased and Biased Variance Estimates

Compare the number of coefficients for unbiased and biased wavelet variance estimates. For the unbiased (default) estimates, the number of nonboundary coefficients decreases by scale. For biased estimates, the number of coefficients matches the number of input rows and is constant for every scale.

Obtain the MODWT of the Southern Oscillation Index data using the Fejer-Korovkin wavelet with eight coefficients. Compute the unbiased and biased estimates of the wavelet variance down to level ten. The number of coefficients used in the unbiased estimates decrease by scale.

```
load soi
wsoi = modwt(soi,'fk8');
[wvar_unb,wvarci_unb,nj_unb] = modwtvar(wsoi,'fk8');
[wvar_b,wvarci_b,nj_b] = modwtvar(wsoi,'fk8',[],'EstimatorType','biased');
[nj_unb(1:10),nj_b(1:10)]

ans =

       12991       12998
       12977       12998
       12949       12998
       12893       12998
       12781       12998
       12557       12998
       12109       12998
       11213       12998
        9421       12998
        5837       12998
```

Table of Wavelet Variance Estimates Using Gaussian Confidence Intervals

Compute the MODWT of the Southern Oscillation Index data using the Fejer- Korovkin wavelet with eight coefficients. Compute a variance table for the data.

```
load soi;
wsoi = modwt(soi,'fk8');
[wvartable] = modwtvar(wsoi,'fk8',0.90,'ConfidenceMethod','Gaussian',...
    'table')


wvartable =

  10x4 table

            NJ       Lower      Variance      Upper
          _____    ________    _________    _______

    D1    12991     0.3291     0.33848      0.34786
    D2    12977    0.87172      0.9034      0.93508
    D3    12949     1.1041      1.1628       1.2216
    D4    12893     1.0204      1.0933       1.1662
```

```
D5     12781     0.8833    0.98255      1.0818
D6     12557    0.47178    0.54152     0.61125
D7     12109    0.41916    0.57934     0.73951
D8     11213    0.33639      2.055      3.7736
D9      9421     0.4752    0.83369      1.1922
D10     5837    0.37485    0.84386      1.3129
```

The resulting table contains the number of nonboundary coefficients, the lower and upper conficence level bounds, and the variance estimate for each level.

- "Wavelet Analysis of Financial Data"

Input Arguments

`w` — MODWT transform matrix

matrix

MODWT transform, specified as a matrix. `w` is the output of `modwt`.

Data Types: `double`

`wname` — Wavelet

`'sym4'` (default) | character vector | positive even scalar

Wavelet, specified as a character vector corresponding to a valid wavelet, or as a positive even scalar indicating the length of the wavelet and scaling filters. The wavelet filter length must match the length used in the MODWT of the input.

If you use `Name,Value` pairs arguments or the `'table'` syntax and you did not specify a `wname` , you must use `[]` as the second argument.

`conflevel` — Confidence level

0.95 (default) | real scalar greater than 0 and less than 1

Confidence level, specified as a real scalar value greater than 0 and less than 1. The confidence level determines the coverage probability of the confidence intervals. The confidence levels are also shown in `wvartable`, if you specify `'table'` as an input.

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: `'EstimatorType','biased'` specifies a biased estimator.

`EstimatorType` — Estimator
`'unbiased'` (default) | `'biased'`

Type of estimator used for variance estimates and confidence bounds, specified as the comma-separated pair consisting of `'EstimatorType'` and one of these values.

- 'unbiased' — Unbiased estimator, which identifies and removes boundary coefficients prior to computing the variance estimates and confidence bounds. Unbiased estimates are used more frequently for wavelet variance computations.
- 'biased' — Biased estimator, which uses all coefficients to compute the variance estimates and confidence bounds.

`ConfidenceMethod` — Confidence method
`'chi2eta3'` (default) | `'chi2eta1'` | `'gaussian'`

Confidence method used to compute the confidence intervals, specified as the comma-separated pair consisting of `'ConfidenceMethod'` and one of these values:

`'chi2eta3'`	Chi-square probability density method three, which determines the degrees of freedom.[1].
`'chi2eta1'`	Chi-square probability density method one, which determines the degrees of freedom [1].
`'gaussian'`	Gaussian method [1] . This method can result in negative lower bounds.

See “Algorithm” on page 1-603 for information on each of these confidence methods.

`Boundary` — Boundary condition
`'periodic'` (default) | `'reflection'`

Boundary condition used to compute the variance estimates and confidence bounds, specified as the comma-separated pair consisting of `'Boundary'` and one of these values:

`'periodic'`	Periodic boundary handling, which does not change the original signal before computing the MODWT. If `modwt` uses periodic boundary handling, you must specify `'Boundary','periodic'` for `modwtvar` to obtain a correct estimate.
`'reflection'`	Reflection boundary handling. If the MODWT uses reflection boundary handling, you must also specify `'Boundary','reflection'` for `modwtvar` to obtain a correct unbiased estimate. The MODWT, with reflection boundary handling, extends the original signal symmetrically at the right boundary to twice the signal length. The MODWTVAR algorithm has to know about this extended signal to calculate the correct unbiased estimate. For biased estimators, all the coefficients are used to form the variance estimates and confidence intervals regardless of the boundary handling.

Output Arguments

`wvar` — Wavelet variance estimates

matrix

Wavelet variance estimates, returned as vector. The number of elements in `wvar` depends on the number of scales in the input matrix and, for unbiased estimates, on the wavelet length. For the unbiased case, `modwtvar` returns estimates only where nonboundary coefficients exist. This condition is satisfied when the transform level is not greater than `floor(log2(N/(L-1)+1))`, where N is the input signal length and L is the length of the wavelet filter. The number of biased estimates equals the input signal length. If the final level has sufficient nonboundary coefficients, `modwtvar` returns the scaling variance in the final element of `wvar`.

`wvarci` — Confidence intervals for each variance estimate

matrix

Confidence bounds, expressed as upper and lower confidence bounds, for the variance estimates in `wvar`, returned as a matrix. The default is 95% confidence bounds, but you can use a different value using the `conflevel` input argument. The confidence bounds matrix is *M*-by-2, where *M* is the number of levels. For unbiased estimates, the number of levels is limited by the number of nonboundary coefficients. For biased estimates, all levels are used. The first column of the confidence interval matrix contains the lower confidence bound and the second column contains the upper confidence bound. By default, `modwtvar` calculates the confidence intervals using the chi-square probability density, with the equivalent degrees of freedom estimated using the `'Chi2Eta3'` confidence method.

`nj` — Number of coefficients by level

vector

Number of nonboundary coefficients by scale, returned as a vector. For unbiased estimates, `nj` is the number of nonboundary coefficients and decreases by level. For biased estimates, `nj` is a vector of constants equal to the number of columns in the input matrix.

`wvartable` — Variance table

table

Variance table, returned as a MATLAB table. The four variables in the table are:

- **NJ** — Number of MODWT coefficients by level. For biased estimates, NJ is the number of coefficients in the MODWT. For unbiased estimates, NJ is the number of nonboundary coefficients.
- **Lower** — Lower confidence bound for the variance estimate.
- **Variance** — Variance estimate by level.
- **Upper** — Upper confidence bound for the variance estimate.

The row names of `wvartable` indicate the type and level of each estimate. For example, D1 indicates that the row corresponds to a wavelet or detail estimate at level 1. S6 indicates that the row corresponds to the scaling estimate at level 6. The scaling variance is computed for the final level of the MODWT. For unbiased estimates, `modwtvar` computes the scaling variance only when nonboundary scaling coefficients exist.

Algorithms

The following expressions define the variance and confidence methods used in the MODWTVAR. The variables are

- N_j — Number of coefficients at level j
- v^2 — Variance
- j — Level
- $W_{j,t}$ — Wavelet coefficients

The variance estimate is

$$\hat{v}_j^2 = \frac{1}{N_j}\sum_{t=0}^{N_j-1} W_{j,t}^2$$

The degrees of freedom for the Chi2Eta1 (`chi2eta1`) method are defined as

$$\eta_1 = \frac{N_j \hat{v}_j^4}{\hat{A}_j}$$

where

$$\hat{A}_j = \frac{1}{2}\int_{-1/2}^{1/2}\left[\hat{S}_j^{(p)}(f)\right]^2 df$$

In this equation, $\hat{S}_j^{(p)}$ is the spectral density function estimate of the wavelet coefficients at level j.

The chi-square statistic is

$$\frac{\eta_1 N_j \hat{v}_j^2}{v_j^2} \sim \mathrm{X}^2_{\eta_1}$$

The degrees of freedom for the Chi2Eta3 (`chi2eta3`) method are defined as

$$\eta_3 = \max\left(\frac{N_j}{2^j}, 1\right)$$

The chi-square statistic is

$$\frac{\eta_3 N_j \hat{v}_j^2}{v_j^2} \sim \chi^2_{\eta_3}$$

For the Gaussian method, the statistic

$$\frac{N_j^{1/2}\left((v_j^2 - v_j^2)\right)}{\left(2A_j\right)^{1/2}}$$

is distributed as `N(0,1)`. The variable $\hat{A}_j$ is as described for `chi2eta1`.

References

[1] Percival, D. B., and A. T. Walden. *Wavelet Methods for Time Series Analysis*. Cambridge, UK: Cambridge University Press, 2000.

[2] Percival, D. B., D. Mondal, "A Wavelet Variance Primer." *Handbook of Statistics, Volume. 300, Time Series Analysis: Methods and Applications*, (T. S. Rao, S. S. Rao, and C. R. Rao, eds.). Oxford, UK: Elsevier, 2012, pp. 623–658.

[3] Cornish, C. R., C. S. Bretherton, and D. B. Percival. "Maximal Overlap Wavelet Statistical Analysis with Application to Atmospheric Turbulence." *Boundary-Layer Meteorology*. Vol. 119, Number 2, 2005, pp. 339–374.

See Also

`imodwt` | `modwt` | `modwtcorr` | `modwtmra` | `modwtxcorr`

Topics

“Wavelet Analysis of Financial Data”

Introduced in R2015b

modwtxcorr

Wavelet cross-correlation sequence estimates using the maximal overlap discrete wavelet transform (MODWT)

Syntax

```
xcseq = modwtxcorr(w1,w2)
xcseq = modwtxcorr(w1,w2,wav)

[xcseq,xcseqci] = modwtxcorr(___)
[xcseq,xcseqci] = modwtxcorr(w1,w2,wav,conflevel)
[xcseq,xcseqci,lags] = modwtxcorr(___)

[___] = modwtxcorr(___,'reflection')
```

Description

`xcseq = modwtxcorr(w1,w2)` returns the wavelet cross-correlation sequence estimates for the maximal overlap discrete wavelet transform (MODWT) transforms specified in `w1` and `w2`. `xcseq` is a cell array of vectors where the elements in each cell correspond to cross-correlation sequence estimates. If there are enough nonboundary coefficients at the final level, `modwtxcorr` returns the scaling cross-correlation sequence estimate in the final cell of `xcseq`.

`xcseq = modwtxcorr(w1,w2,wav)` uses the wavelet `wav` to determine the number of boundary coefficients by level.

`[xcseq,xcseqci] = modwtxcorr(___)` returns in `xcseqci` the 95% confidence intervals for the cross-correlation sequence estimates in `xcseq`, using any arguments from the previous syntaxes.

`[xcseq,xcseqci] = modwtxcorr(w1,w2,wav,conflevel)` uses `conflevel` for the coverage probability of the confidence interval. `conflevel` is a real scalar strictly greater than 0 and less than 1. If `conflevel` is unspecified or specified as empty, the coverage probability defaults to 0.95.

`[xcseq,xcseqci,lags] = modwtxcorr(___)` returns the lags for the wavelet cross-correlation sequence estimates in a cell array of column vectors.

`[___] = modwtxcorr(___,'reflection')` reduces the number of wavelet and scaling coefficients at each scale by half before computing the cross-correlation sequences. Specifying the `'reflection'` option in `modwtxcorr` is equivalent to first obtaining the MODWT of `w1 w2` with `'periodic'` boundary handling and then computing the wavelet cross-correlation sequence estimates. Use this option only when you obtain the MODWT of `w1` and `w2` using the `'reflection'` boundary condition. You must enter the entire character vector `'reflection'`. If you added a wavelet named `'reflection'` using the wavelet manager, you must rename that wavelet prior to using this option. `'reflection'` may be placed in any position in the input argument list after the input transforms `w1 w2`.

Examples

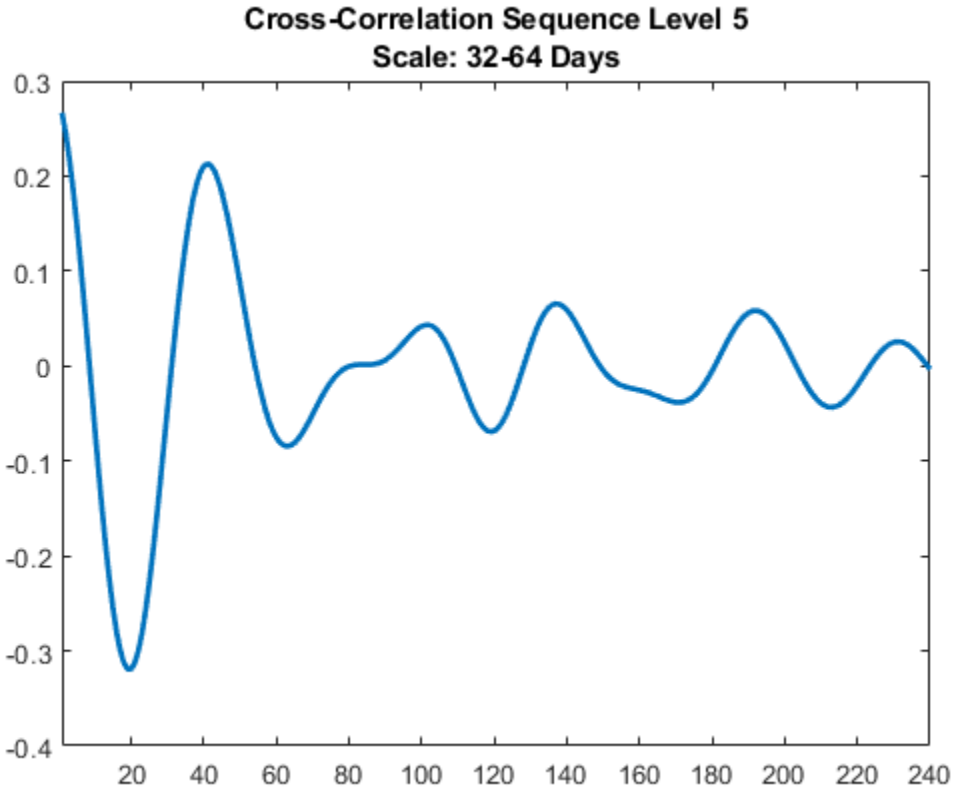
Cross-Correlation Sequence

Obtain the MODWT of the Southern Oscillation Index and Truk Islands pressure data. The sampling period is one day.

```
load soi
load truk
wsoi = modwt(soi);
wtruk = modwt(truk);
```

Compute the wavelet cross-correlation sequences. Examine the level-five cross-correlation sequence corresponding to a scale of 32-64 days. Determine the index corresponding to a lag of zero and plot out to 240 lags.

```
xcseq = modwtxcorr(wsoi,wtruk);
zerolag = floor(numel(xcseq{5})/2)+1;
plot(xcseq{5}(zerolag:zerolag+240),'linewidth',2)
set(gca,'xlim',[1 240]);
title({'Cross-Correlation Sequence Level 5'; 'Scale: 32-64 Days'});
hold off
```



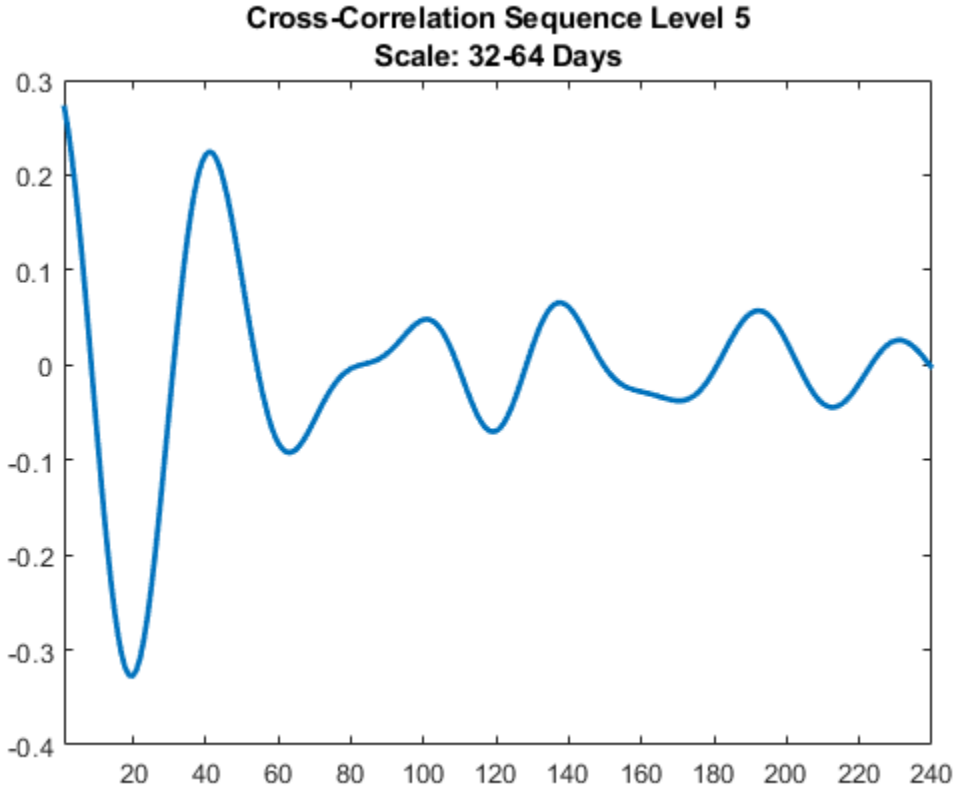
Cross-Correlation Sequence with Fejer-Korovkin Wavelet

Obtain the MODWT of the Southern Oscillation Index and Truk Islands pressure data using the Fejer-Korovkin wavelet filter with 8 coefficients. The sampling period of the data is one day.

```
load soi
load truk
wsoi = modwt(soi,'fk8');
wtruk = modwt(truk,'fk8');
```

Compute the wavelet cross-correlation sequences. Examine the level-five cross-correlation sequence corresponding to a scale of 32-64 days. Determine the index corresponding to a lag of zero and plot out to 240 lags.

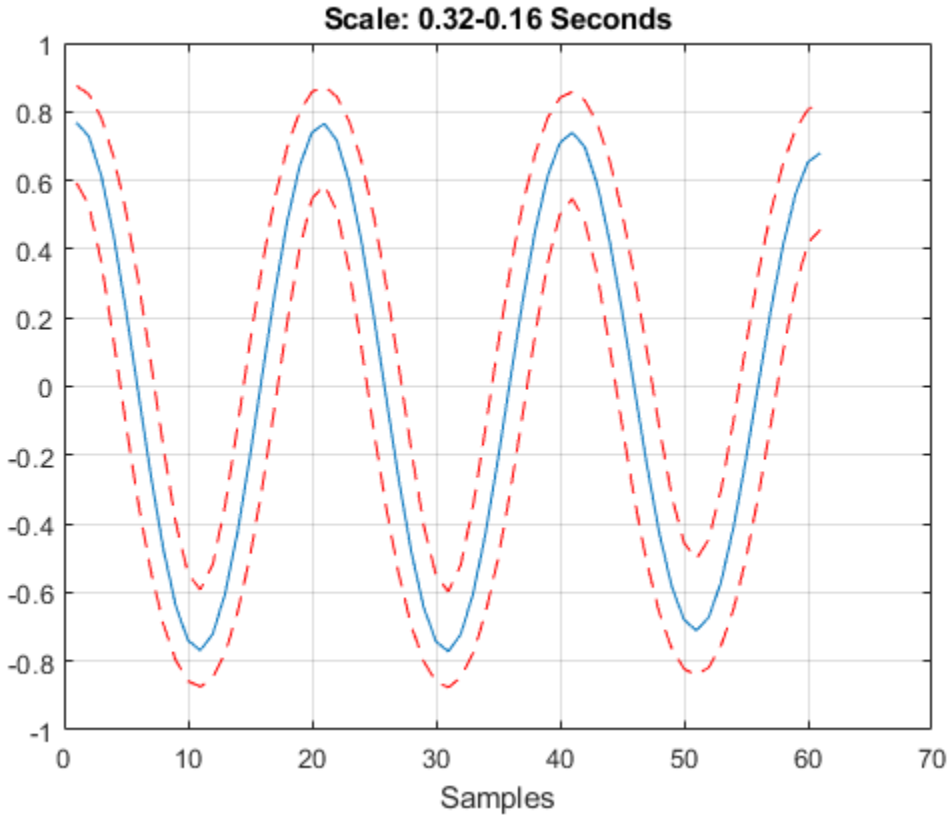
```
xcseq = modwtxcorr(wsoi,wtruk,'fk8');
zerolag = floor(numel(xcseq{5})/2)+1;
plot(xcseq{5}(zerolag:zerolag+240),'linewidth',2)
set(gca,'xlim',[1 240]);
title({'Cross-Correlation Sequence Level 5'; 'Scale: 32-64 Days'});
hold off
```



Cross-Correlation Confidence Intervals by Scale

Plot the wavelet cross-correlation with 95% confidence intervals at scale 4 for two 5-Hz sine wave signals with additive noise.

```
dt = 0.01;
t = 0:dt:6;
x = cos(2*pi*5*t)+1.5*randn(size(t));
y = cos(2*pi*5*t-pi)+2*randn(size(t));
wx = modwt(x,'fk14',5);
wy = modwt(y,'fk14',5);
modwtcorr(wx,wy,'fk14')
j = 4;
[xcseq,xcseqci] = modwtxcorr(wx,wy,'fk14');
zerolag = floor(numel(xcseq{j})/2)+1;
lagidx = zerolag-30:zerolag+30;
plot(xcseq{j}(lagidx));
hold on;
grid
plot(xcseqci{j}(lagidx,:),'r--');
xlabel('Samples');
title('Scale: 0.32-0.16 Seconds');
```



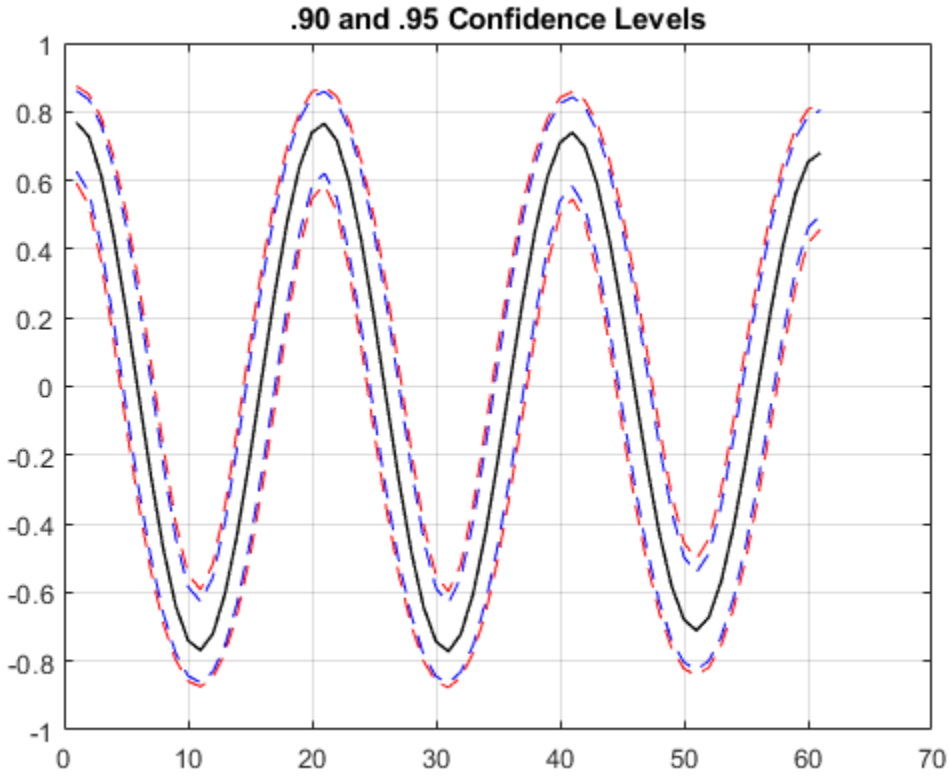
Cross-Correlation .90 and .95 Confidence Intervals Comparison

Compare the .90 and .95 (default) confidence intervals for the wavelet cross-correlation at level four.

Obtain the MODWT for two noisy sine waves using the Fejer-Korovkin with 14 coefficients, and specify the level to use.

```
dt = 0.01;
t = 0:dt:6;
x = cos(2*pi*5*t)+1.5*randn(size(t));
y = cos(2*pi*5*t-pi)+2*randn(size(t));
```

```
wx = modwt(x,'fk14',4);
wy = modwt(y,'fk14',4);
lev = 4;

[xcseq,xcseqci] = modwtxcorr(wx,wy,'fk14');
[xcseq90,xcseqci90] = modwtxcorr(wx,wy,'fk14',0.90);

zerolag = floor(numel(xcseq{lev})/2)+1;
zerolag90 = floor(numel(xcseq90{lev})/2)+1;

lagidx = zerolag-30:zerolag+30;
lagidx90 = zerolag90-30:zerolag90+30;

plot(xcseqci{lev}(lagidx,:),'--r');
hold on
plot(xcseqci90{lev}(lagidx90,:),'--b');
plot(xcseq{lev}(lagidx),'-k','LineWidth',1);
grid
title('.90 and .95 Confidence Levels')
```



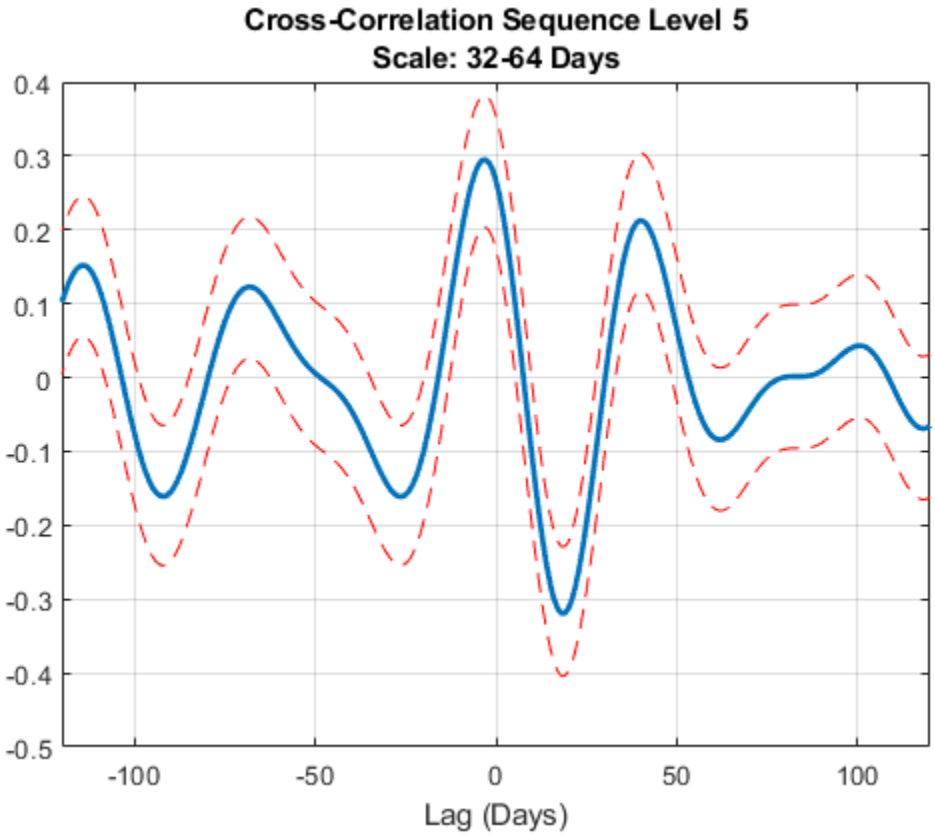
Notice that the .95 confidence interval width (in red) is larger than the .90 confidence interval width (in blue).

Plot Cross-Correlation Sequences by Lag

Plot the cross-correlation sequence estimate for the Southern Oscillation Index and Truk Island pressure data. Estimate 95% confidence intervals for scale of 2^5 days.

```
load soi
load truk
wsoi = modwt(soi);
```

```
wtruk = modwt(truk);
[xcseq,xcseqci,lags] = modwtxcorr(wsoi,wtruk);
plot(lags{5},xcseq{5},'linewidth',2)
hold on
plot(lags{5},xcseqci{5},'r--')
set(gca,'xlim',[-120 120]);
xlabel('Lag (Days)');
grid
title({'Cross-Correlation Sequence Level 5'; 'Scale: 32-64 Days'});
hold off
```



Cross-Correlation with Reflection Boundary

Obtain the MODWT of 36 years of Southern Oscillation Index and Truk Islands pressure data with both periodic and reflection boundary conditions. The `modwt` with the `'reflection'` option extends the input signal symmetrically at the right boundary. The input signal is then twice its original length. MODWTXCORR with the reflection boundary handling reduces the number of wavelet and scaling coefficients at each half before computing the cross-correlation sequences. The size of the cross-correlation sequences is the same as acquiring the MODWT with the default periodic boundary condition.

```
load soi
load truk
```

Obtain the MODWT with the default periodic boundary condition.

```
wsoi_default = modwt(soi);
wtruk_default = modwt(truk);
```

Obtain the MODWT with the reflection boundary condition.

```
wsoi_reflect = modwt(soi,'reflection');
wtruk_reflect = modwt(truk,'reflection');
```

Obtain the cross correlation sequences.

```
xcseq_default = modwtxcorr(wsoi_default,wtruk_default);
xcseq_reflect = modwtxcorr(wsoi_reflect,wtruk_reflect,'reflection');
```

Compare the number of elements in the cell array output for both boundary conditions.

```
cellfun(@numel,xcseq_reflect)

ans =

       25981
       25953
       25897
       25785
       25561
       25113
       24217
       22425
       18841
```

```
       11673


cellfun(@numel,xcseq_default)

ans =

       25981
       25953
       25897
       25785
       25561
       25113
       24217
       22425
       18841
       11673
```

- “Wavelet Analysis of Financial Data”

Input Arguments

`w1` — MODWT transform of signal 1

matrix

MODWT transform of signal 1, specified as a matrix of doubles. The input `w1` must be the same size as `w2` and must have been obtained with the same wavelet. By default, `modwtxcorr` assumes that you obtained the MODWT using the symlet wavelet with four vanishing moments, `'sym4'`).

`w2` — MODWT transform of signal 2

matrix

MODWT transform of signal 2, specified as a matrix of doubles. The input `w2` must be the same size as `w1` and must have been obtained with the same wavelet. By default, `modwtxcorr` assumes that you obtained the MODWT using the symlet wavelet with four vanishing moments (`'sym4'`).

`wav` — Wavelet

`'sym4'` (default) | character vector | positive even integer

Wavelet, specified as a character vector, indicating a valid wavelet, or as a positive even integer indicating the length of the wavelet and scaling filters. If `wav` is unspecified or specified as an empty, `[]`, `wav` defaults to `'sym4'`.

`conflevel` — Confidence level
0.95 (default) | positive scalar less than 1

Confidence level, specified as a positive scalar less than 1. `conflevel` determines the coverage probability of the confidence intervals in `xcseqci`. If unspecified, or specified as empty, `[]`, `conflevel` defaults to 0.95.

Output Arguments

`xcseq` — Cross-correlation sequences by scale
cell array of vectors

Cross-correlation sequences by scale, returned as a cell array of vectors. The vectors are of size $2NJ$-by-1, where NJ is the number of nonboundary coefficients by level (scale). This level is the minimum of `size(w1,1)` and `floor(log2(N/(L-1)+1))` where N is the length of the data and L is the filter length. If there are enough nonboundary coefficients at the final level, `modwtxcorr` returns the scaling cross-correlation sequence estimate in the final cell of `xcseq`.

`xcseqci` — Confidence intervals by scale
cell array of matrices

Confidence intervals by scale, returned as a cell array of matrices. The size of each matrix is $(2NJ\text{-}1)$-by-2, where NJ is the number of nonboundary coefficients by level (scale).

- For the .95 default value, the first column of the i^{th} element of `xcseqci` contains the lower 95% confidence bound for the cross-correlation coefficient at each lag.
- For the .95 default value, the second column contains the upper 95% confidence bound.

Confidence bounds are computed using Fisher's Z-transformation. The standard error of Fisher's Z statistic is the square root of $N–3$. In this case, N is the equivalent number of coefficients in the critically sampled discrete wavelet transform (DWT), `floor(size(w1,2)/2^LEV)`, where `LEV` is the level of the wavelet transform.

`modwtcorr` returns NaNs for the confidence bounds when N^{-3} is less than or equal to zero.

`lags` — Lags for the cross-correlation sequences

cell array of vectors

Lags for the cross-correlation sequences, returned as a cell array of vectors. `lags` is a cell array of column vectors the same length as `xcseq`. The elements in each cell of `xcseq` correspond to the cross-correlation sequence estimates at lags from -(*NJ*-1) to (*NJ*-1), where *NJ* is the number of nonboundary coefficients at level *J*. The 0^{th} lag element is located at the index floor((2*NJ-1)/2)+1.

References

[1] >Percival, D. B., and Walden, A. T. *Wavelet Methods for Time Series Analysis*. Cambridge, U.K: Cambridge University Press, 2000.

[2] Whitcher, B., P. Guttorp, and D. B. Percival. "Wavelet analysis of covariance with application to atmospheric time series." *Journal of Geophysical Research*, Vol. 105, 2000, pp. 14941–14962.

See Also

`imodwt` | `modwt` | `modwtcorr` | `modwtmra` | `modwtvar`

Topics

“Wavelet Analysis of Financial Data”

Introduced in R2015b

morlet

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.

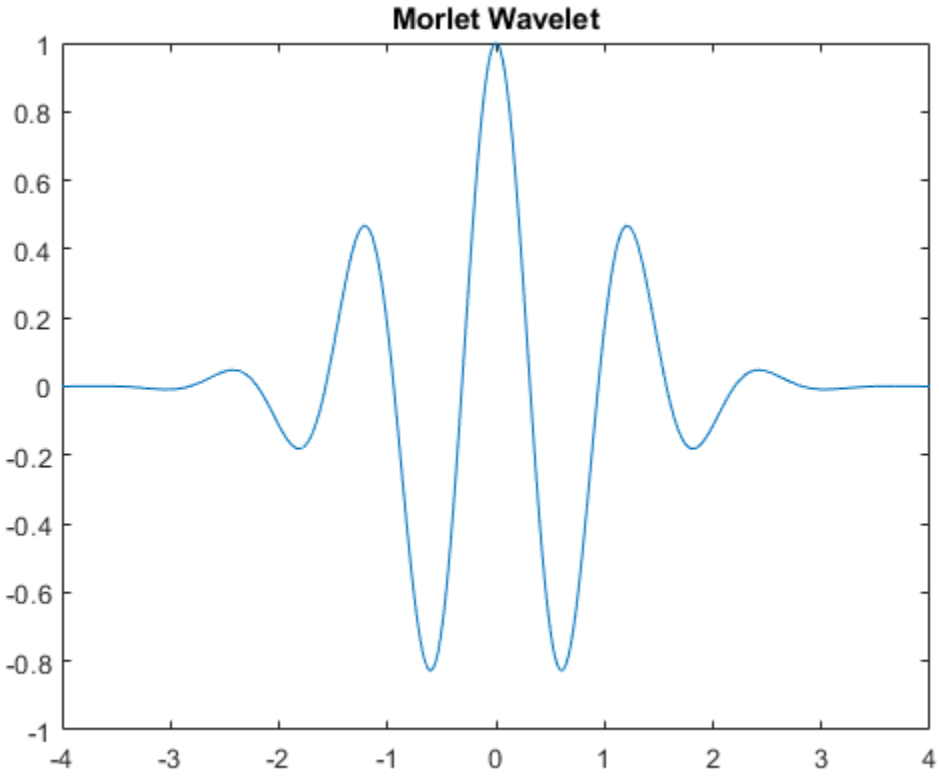
$$\psi(x) = e^{-x^2/2}\cos(5x)$$

Examples

Morlet Wavelet

This example shows how to create a Morlet wavelet. The wavelet has an effective support of [-4,4]. Use 1,000 sample points.

```
lb = -4;
ub = 4;
n = 1000;
[psi,xval] = morlet(lb,ub,n);
plot(xval,psi)
title('Morlet Wavelet');
```



See Also

`waveinfo`

Introduced before R2006a

mswcmp

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 ≤ PARAM ≤ 10
'scarceme'	Scarce medium, 1.5 ≤ PARAM ≤ 2.5
'scarcelo'	Scarce low, 1 ≤ PARAM ≤ 2

PARAM is a sparsity parameter, and it should be such that: 1 ≤ PARAM ≤ 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 ≤ PARAM ≤ 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 ≤ j ≤ 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

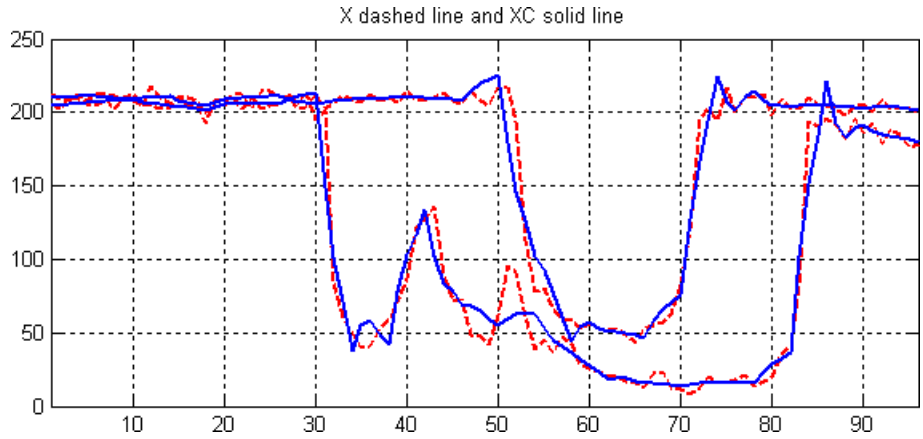
LEV. Before performing a compression or computing thresholds, the multisignal matrix X is decomposed at level LEV using the wavelet WNAME, in the direction DIRDEC.

[...] = 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 character vector '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,'N0_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')
```



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`

Introduced in R2007a

mswcmpscr

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)` ($2 \le j \le$ `NbCFS`), and `L2SCR(:,1)=100`.

- `N0SCR(i,j)` is the percentage of zeros corresponding to a threshold equal to `CFS(i,j-1)` ($2 \leq j \leq$ `NbCFS`), and `N0SCR(:,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 character vector `'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,N0SCR_S] = mswcmpscr(dec,'s');
[THR_H,L2SCR_H,N0SCR_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

`ddencmp` | `mdwtdec` | `mdwtrec` | `wdencmp`

Introduced in R2007a

mswcmptp

Multisignal 1-D compression thresholds and performances

Syntax

```
[THR_VAL,L2_Perf,N0_Perf] = mswcmptp(DEC,METH)
[THR_VAL,L2_Perf,N0_Perf] = mswcmptp(DEC,METH,PARAM)
```

Description

`[THR_VAL,L2_Perf,N0_Perf] = mswcmptp(DEC,METH)` or `[THR_VAL,L2_Perf,N0_Perf] = mswcmptp(DEC,METH,PARAM)` computes the vectors `THR_VAL`, `L2_Perf` and `N0_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.
- `N0_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 character vector `'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 thresholds and exact performances
% obtained after a compression using the method 'N0_perf' and
% requiring a percentage of zeros near 95% for the wavelet
% coefficients.
[THR_VAL,L2_Perf,N0_Perf] = mswcmptp(dec,'N0_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

`ddencmp` | `mdwtdec` | `mdwtrec` | `wdencmp`

Introduced in R2007a

mswden

Multisignal 1-D denoising using wavelets

Syntax

```
[XD,DECDEN,THRESH] = mswden('den',...)
[XD,THRESH] = mswden('densig',...)
[DECDEN,THRESH] = mswden('dendec',...)
THRESH = mswden('thr',...)
[...] = mswden(OPTION,DIRDEC,X,WNAME,LEV,METH,PARAM)
[...] = mswden(...,S_OR_H)
[...] = mswden(...,S_OR_H,KEEPAPP)
[...] = 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 ℜ≤ PARAM ℜ≤ 10
'penalme'	Penal medium, 1.5 ℜ≤ PARAM ℜ≤ 2.5
'penallo'	Penal low, 1 ℜ≤ PARAM ℜ≤ 2

PARAM is a sparsity parameter, and it should be such that: 1 ≤ PARAM ≤ 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 ≤ j ≤ 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:

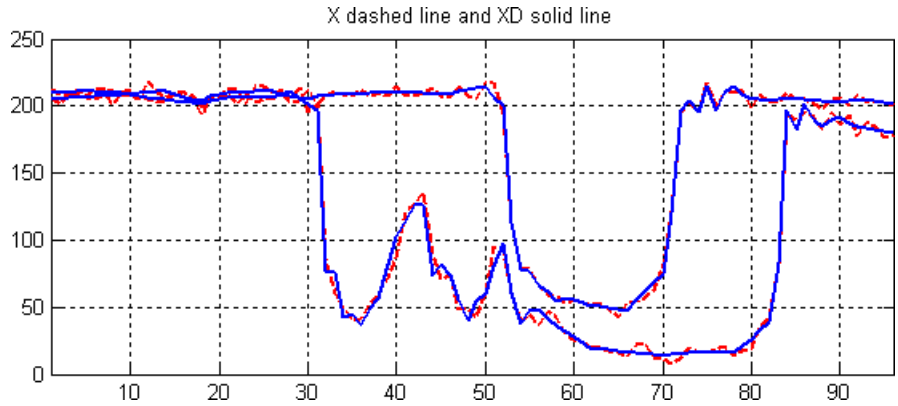
`[...] = mswden(...,S_OR_H)` or
`[...] = mswden(...,S_OR_H,KEEPAPP)` or
`[...] = mswden(...,S_OR_H,KEEPAPP,IDXSIG)`

- `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 character vector `'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')
```



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`

Introduced in R2007a

mswthresh

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)` ≤ `LT`.

`Y = mswthresh(X,SORH,T,'c')` performs a columnwise thresholding and `size(X,2)` ≤ `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

`mswcmp` | `mswden` | `wden` | `wdencmp` | `wpdencmp` | `wthresh`

Introduced in R2007a

nodeasc

Node ascendants

Syntax

```
A = nodeasc(T,N)
```

Description

`nodeasc` is a tree-management utility.

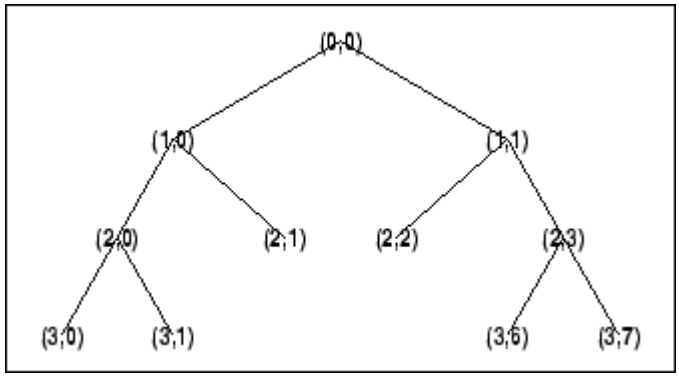
`A = nodeasc(T,N)` returns the indices of all the ascendants of the node *N* in the tree *T* where `N` can be the index node or the depth and position of the node. `A` is a column vector with `A`(1) = index of node *N*.

`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.

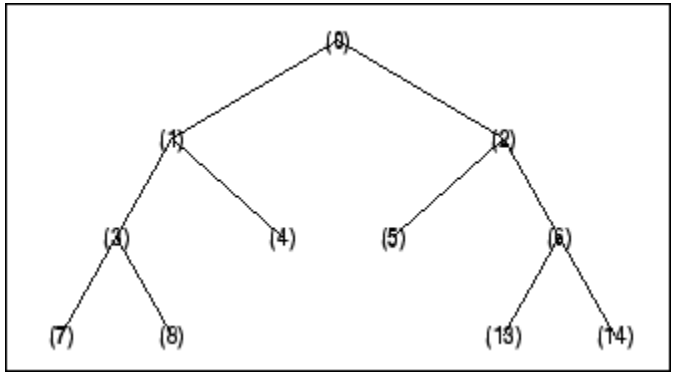
The nodes are numbered from left to right and from top to bottom. The root index is 0.

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(t,[2 2])
ans =
     5
     2
     0

nodeasc(t,[2 2],'deppos')
ans =
     2     2
     1     1
     0     0
```

See Also

`nodedesc` | `nodepar` | `wtreemgr`

Introduced before R2006a

nodedesc

Node descendants

Syntax

```
D = nodedesc(T,N)
D = nodedesc(T,N,'deppos')
```

Description

`nodedesc` is a tree-management utility.

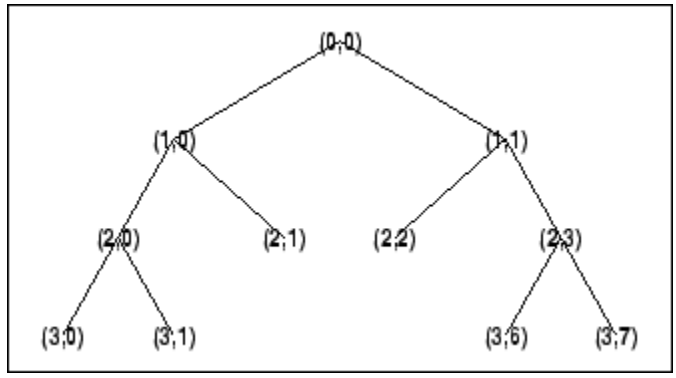
`D = nodedesc(T,N)` returns the indices of all the descendants of the node *`N`* in the tree *`T`* where *`N`* can be the index node or the depth and position of node. `D` is a column vector with `D(1)` = index of node *`N`*.

`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.

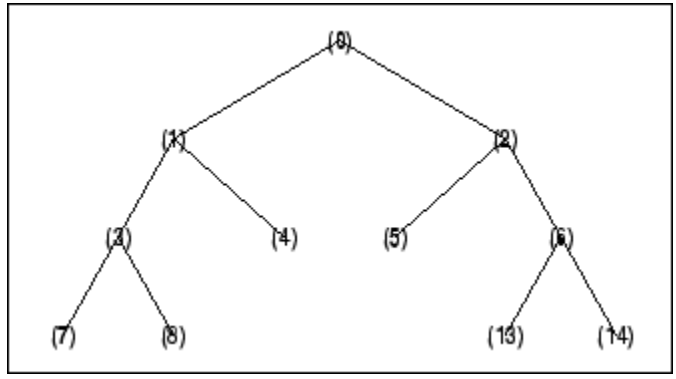
The nodes are numbered from left to right and from top to bottom. The root index is 0.

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

Introduced before R2006a

nodejoin

Recompose node

Syntax

```
T = nodejoin(T,N)
T = nodejoin(T)
T = nodejoin(T,0)
```

Description

`nodejoin` is a tree-management utility.

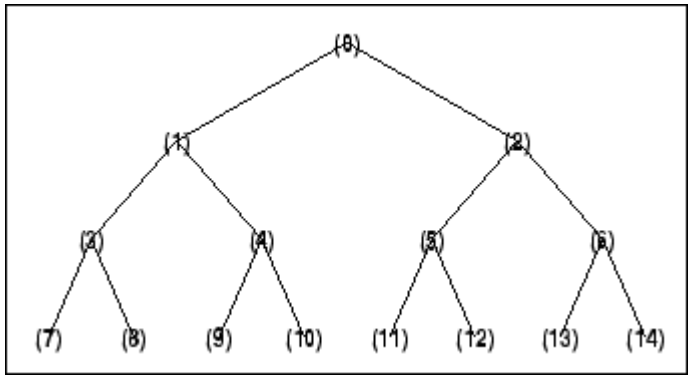
`T = nodejoin(T,N)` returns the modified tree *T* corresponding to a recomposition of the node *N*.

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

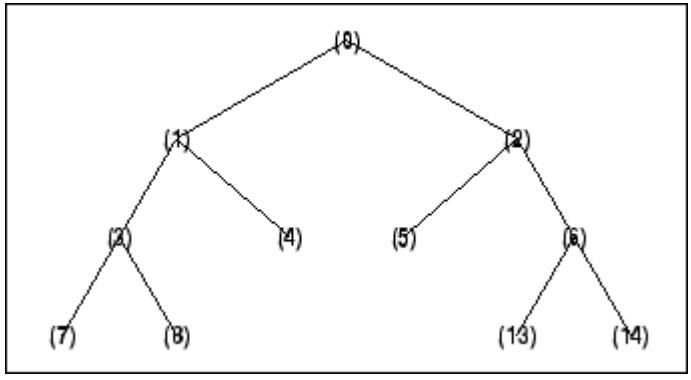
`T = nodejoin(T)` is equivalent to `T = nodejoin(T,0)`.

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

Introduced before R2006a

nodepar

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.

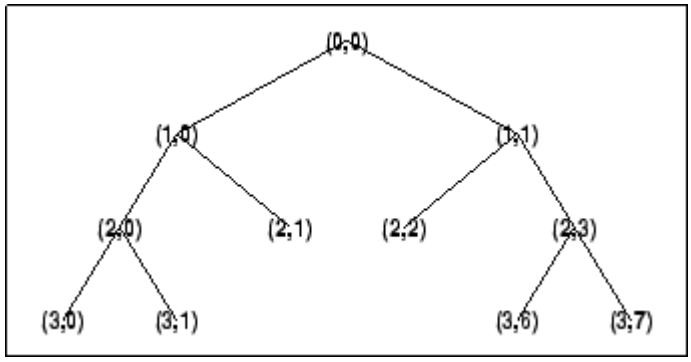
`nodepar(T,0)` or `nodepar(T,[0,0])` returns `-1`.

`nodepar(T,0,'deppos')` or `nodepar(T,[0,0],'deppos')` returns `[-1,0]`.

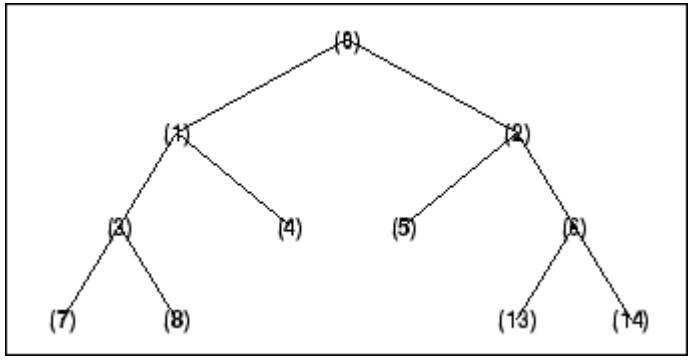
The nodes are numbered from left to right and from top to bottom. The root index is 0.

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
    3
    6
```

See Also

`nodeasc` | `nodedesc` | `wtreemgr`

Introduced before R2006a

nodesplt

Split (decompose) node

Syntax

```
T = nodesplt(T,N)
```

Description

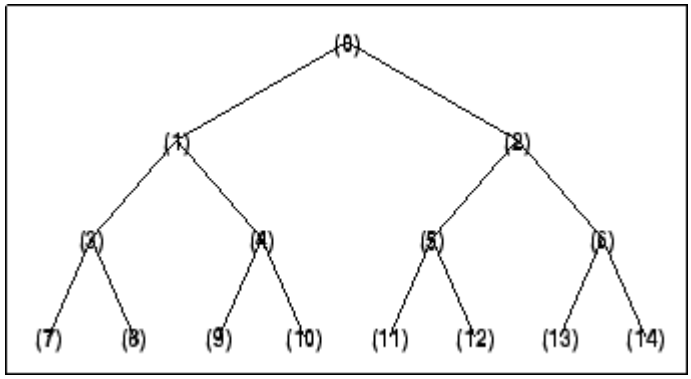
`nodesplt` is a tree-management utility.

`T = nodesplt(T,N)` returns the modified tree *T* corresponding to the decomposition of the node *N*.

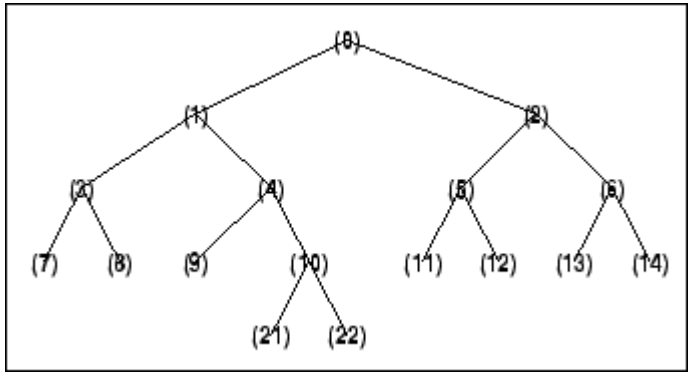
The nodes are numbered from left to right and from top to bottom. The root index is 0.

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

Introduced before R2006a

noleaves

Determine nonterminal nodes

Syntax

```
N = noleaves(T)
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.

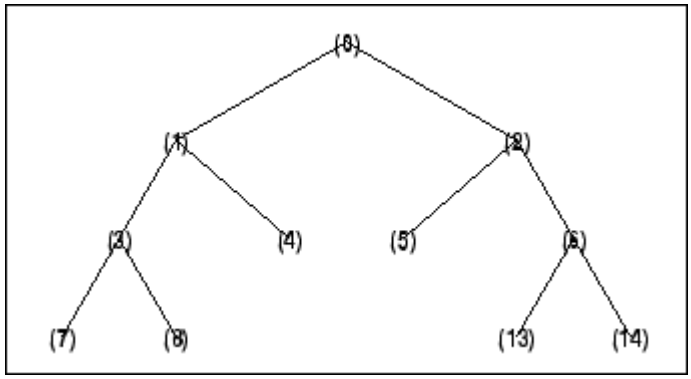
The nodes are ordered from left to right as in tree `T`.

`N = noleaves(T,'dp')` returns a matrix `N`, which contains the depths and positions of nonterminal nodes.

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

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 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`

Introduced before R2006a

ntnode

Number of terminal nodes

Syntax

```
NB = ntnode(T)
```

Description

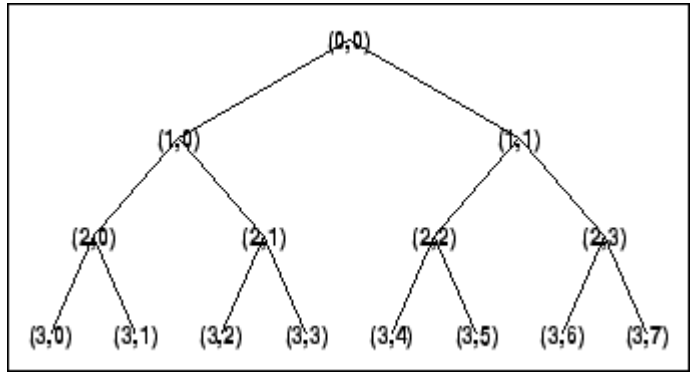
`ntnode` is a tree-management utility.

`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

Introduced before R2006a

ntree

NTREE constructor

Syntax

```
T = ntree(ORD,D)
T = ntree
T = ntree(2,0)
T = ntree(ORD)
T = ntree(ORD,0)
T = ntree(ORD,D,S)
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`.

`T = ntree` is equivalent to `T = ntree(2,0)`.

`T = ntree(ORD)` is equivalent to `T = ntree(ORD,0)`.

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.

The root of the tree can be split and it has `ORD` children. You can split the `j`-th child if `S(j) = 1`.

Each node that you can split has the same property as the root node.

With `T = ntree(ORD,D,S,U)` you can, in addition, set a userdata field.

Inputs can be given in another way:

`T = ntree('order',ORD,'depth',D,'spsch',S,'ud',U)`. For "missing" inputs the defaults are `ORD = 2` and `D = 0 , S = ones([1:ORD]) , U = {}`.

`[T,NB] = ntree( ... )` returns also the number of terminal nodes (leaves) of `T`.

For more information on object fields, type `help ntree/get`.

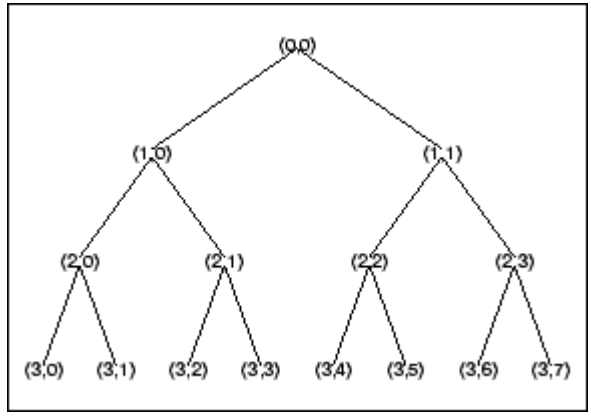
Class NTREE (Parent class: WTBO)

Fields

`wtbo`	Parent object
`order`	Tree order
`depth`	Tree depth
`spsch`	Split scheme for nodes
`tn`	Column vector with terminal node indices

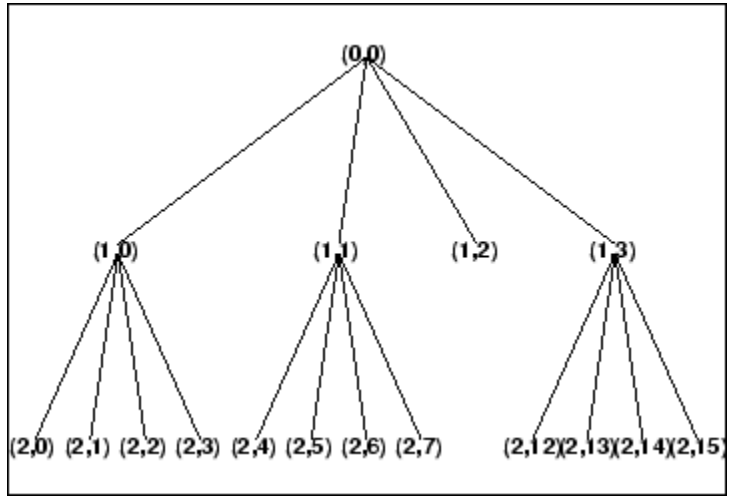
Examples

```
% Create binary tree (tree of order 2) of depth 3.
t2 = ntree(2,3);

% Plot tree t2.
plot(t2)
```

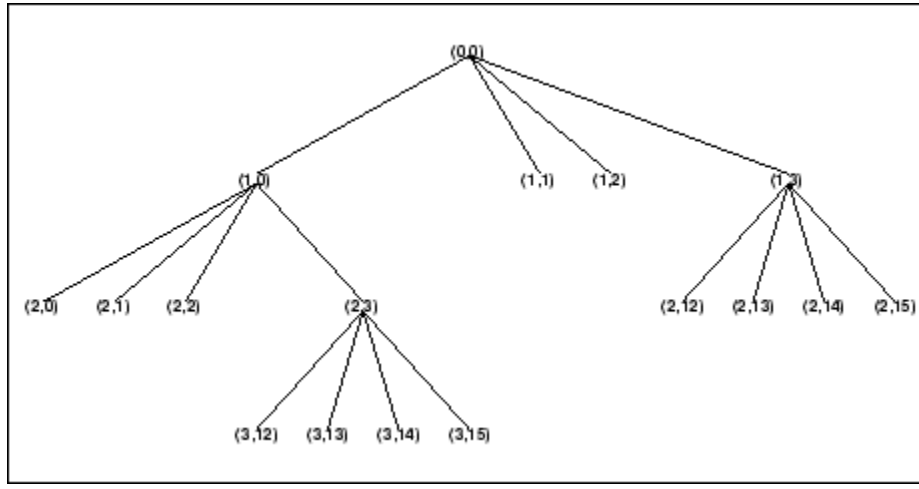



```
% Create a quadtree (tree of order 4) of depth 2.
t4 = ntree(4,2,[1 1 0 1]);
```

```
% 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`

Introduced before R2006a

orthfilt

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 $2N$
- Of sum 1
- Of norm $\frac{1}{\sqrt{2}}$

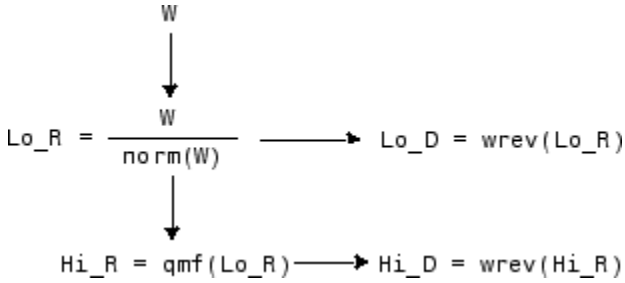
For example, for the `db3` scaling filter,

```
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:



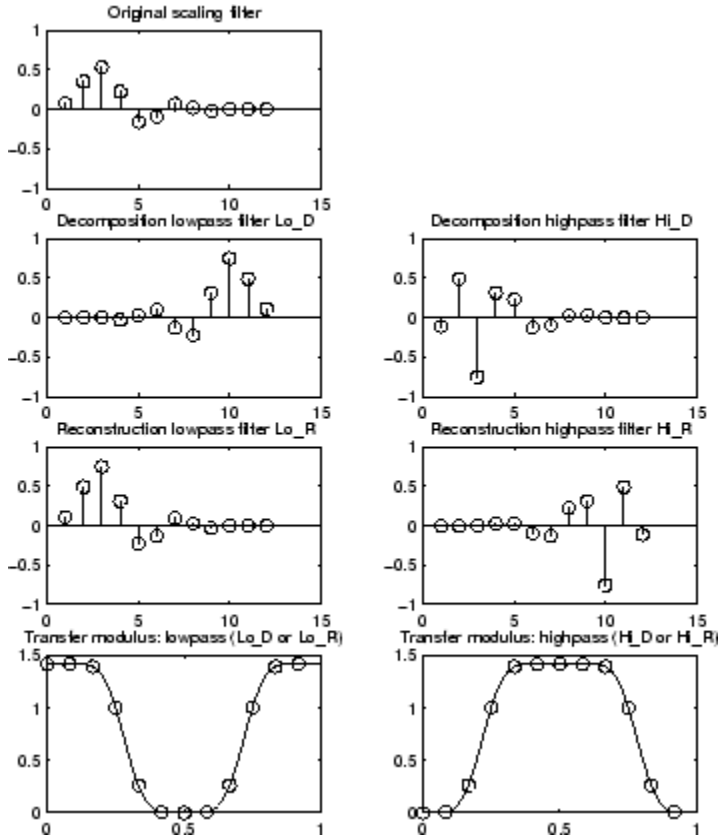
where `qmf` is such that `Hi_R` and `Lo_R` are quadrature mirror filters (i.e., `Hi_R(k) = (-1)^k Lo_R(2N + 1 - k)`, for `k = 1, 2, Ä, 2N`), and where `wrev` flips the filter coefficients. So `Hi_D` and `Lo_D` are also quadrature mirror filters. The computation of these filters is performed using `orthfilt`.

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.
```



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`

Introduced before R2006a

otnodes

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 Arguments

`WPT`

Binary wavelet packet tree. You can use `treeord` to determine the order of your wavelet packet tree.

`dp`

Character vector indicating that the Paley-ordered or sequency-ordered nodes are returned in depth-position format.

Output Arguments

`Tn_Pal`

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.

Examples

Order Terminal Nodes

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 =

     3
     4
     5
     6


Tn_Seq =

     3
```

```
     4
     6
     5
```

Return Permutations for Ordering

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))
```

```
ans =

  logical

   1


ans =

  logical

   1
```

Order Terminal Nodes by Depth and Position

Order terminal nodes by depth and position.

```
x = randn(8,1);
wpt = wpdec(x,2,'haar');
[DP_Pal,DP_Seq] = otnodes(wpt,'dp')
```

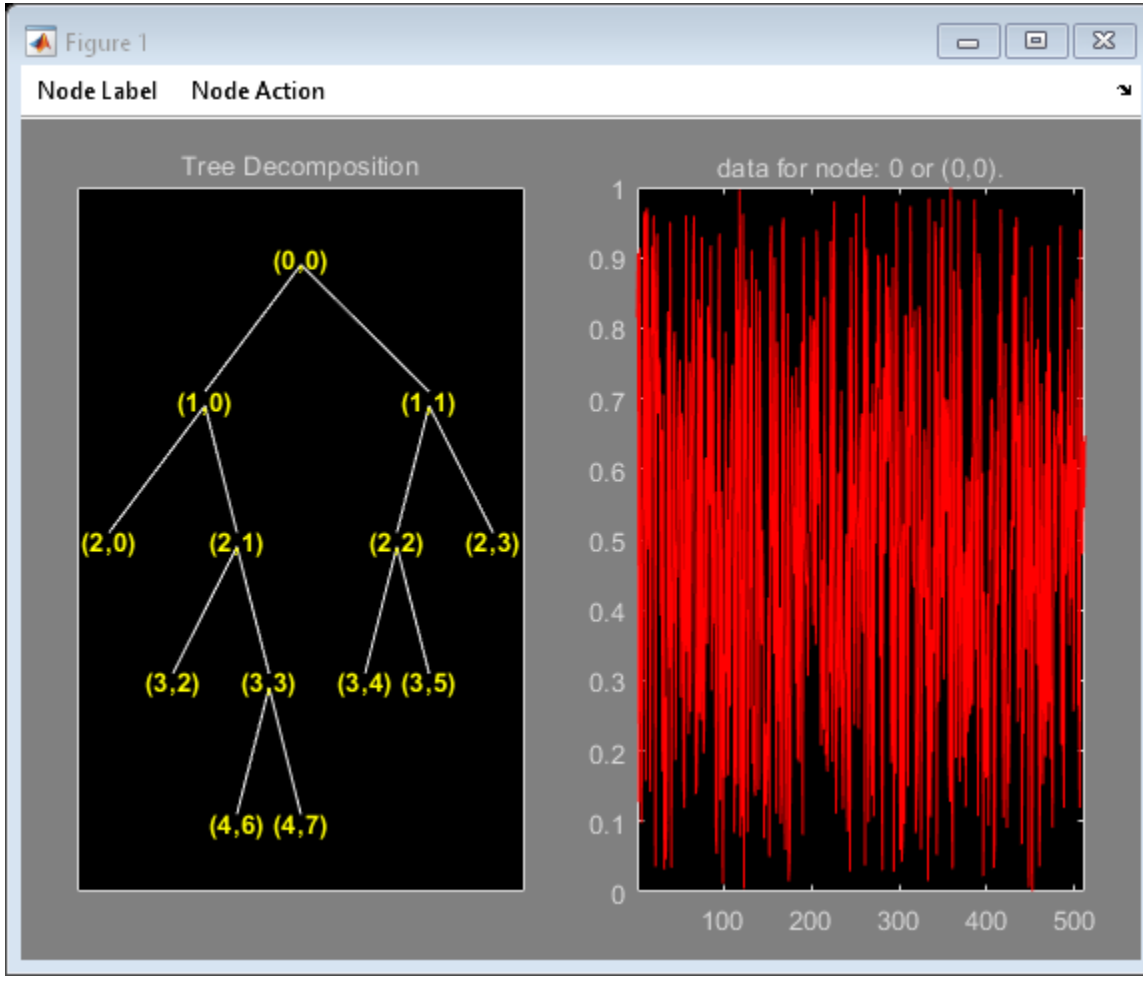
```
DP_Pal =
```

```
     2     0
     2     1
     2     2
     2     3


DP_Seq =

     2     0
     2     1
     2     3
     2     2
```

Order Terminal Nodes from Wavelet Packet Tree

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)
```

```
tn_Pal =

     3
     9
    21
    22
    11
    12
```

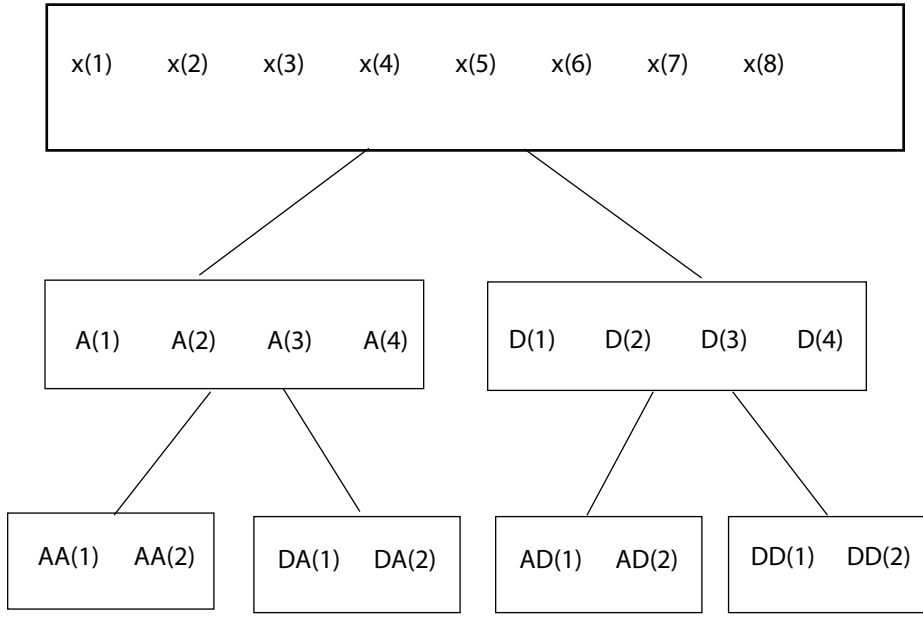
```
     6


tn_Seq =

     3
    21
    22
     9
     6
    12
    11


I =

     1
     3
     4
     2
     7
     6
     5


J =

     1
     4
     2
     3
     7
     6
     5
```

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.

References

Wickerhauser, M.V. *Lectures on Wavelet Packet Algorithms*, Technical Report, Washington University, Department of Mathematics, 1992.

See Also

`leaves` | `treeord`

Introduced in R2010b

pat2cwav

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'`.

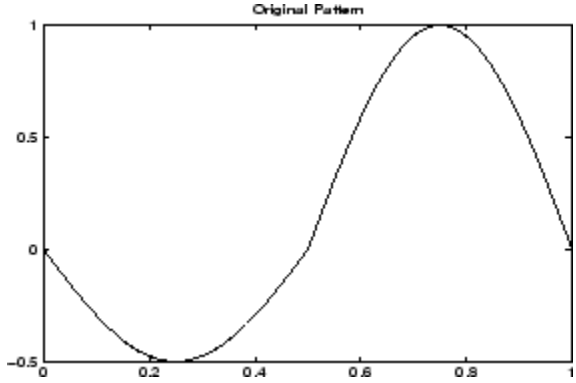
When `METHOD` is equal to `'polynomial'`

- 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.

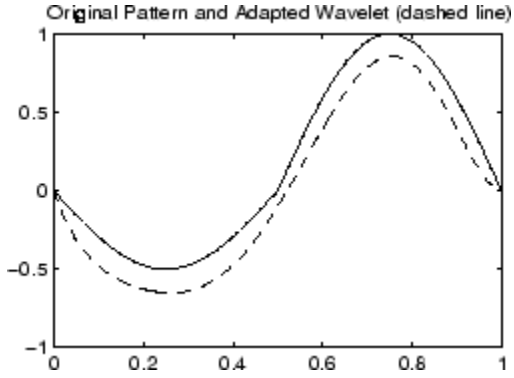
Examples

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')
```




```
[psi,xval,nc] = pat2cwav(Y, 'polynomial',6, 'continuous') ;
plot(X,Y,'-',xval,nc*psi,'--'),
title('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.

Introduced before R2006a

plot

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

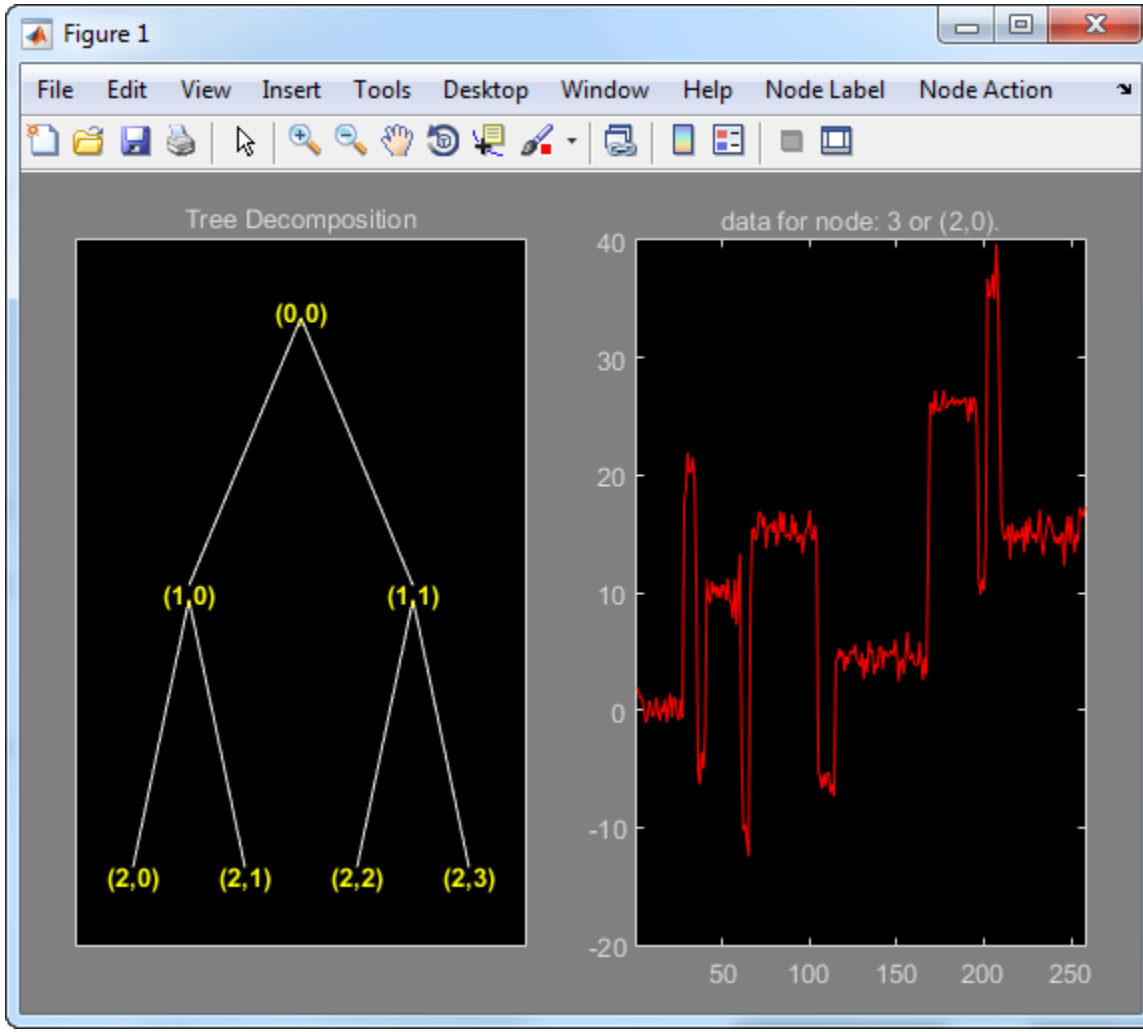
```
NEWT = plot(DUMMY,'read',FIG)
```

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:

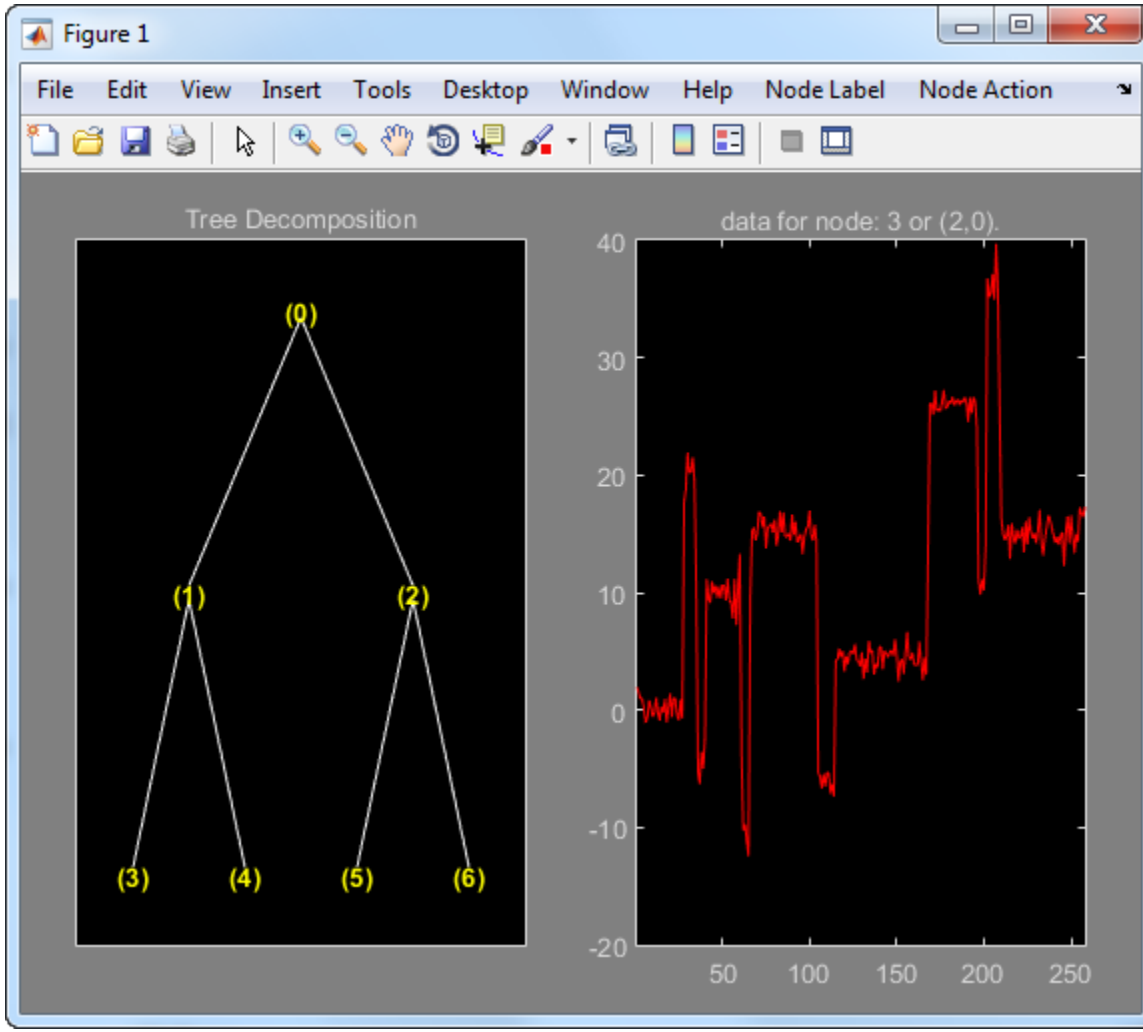
```
NEWT = plot(ntree,'read',FIG)
NEWT = plot(dtree,'read',FIG)
NEWT = plot(wptree,'read',FIG)
```

Examples

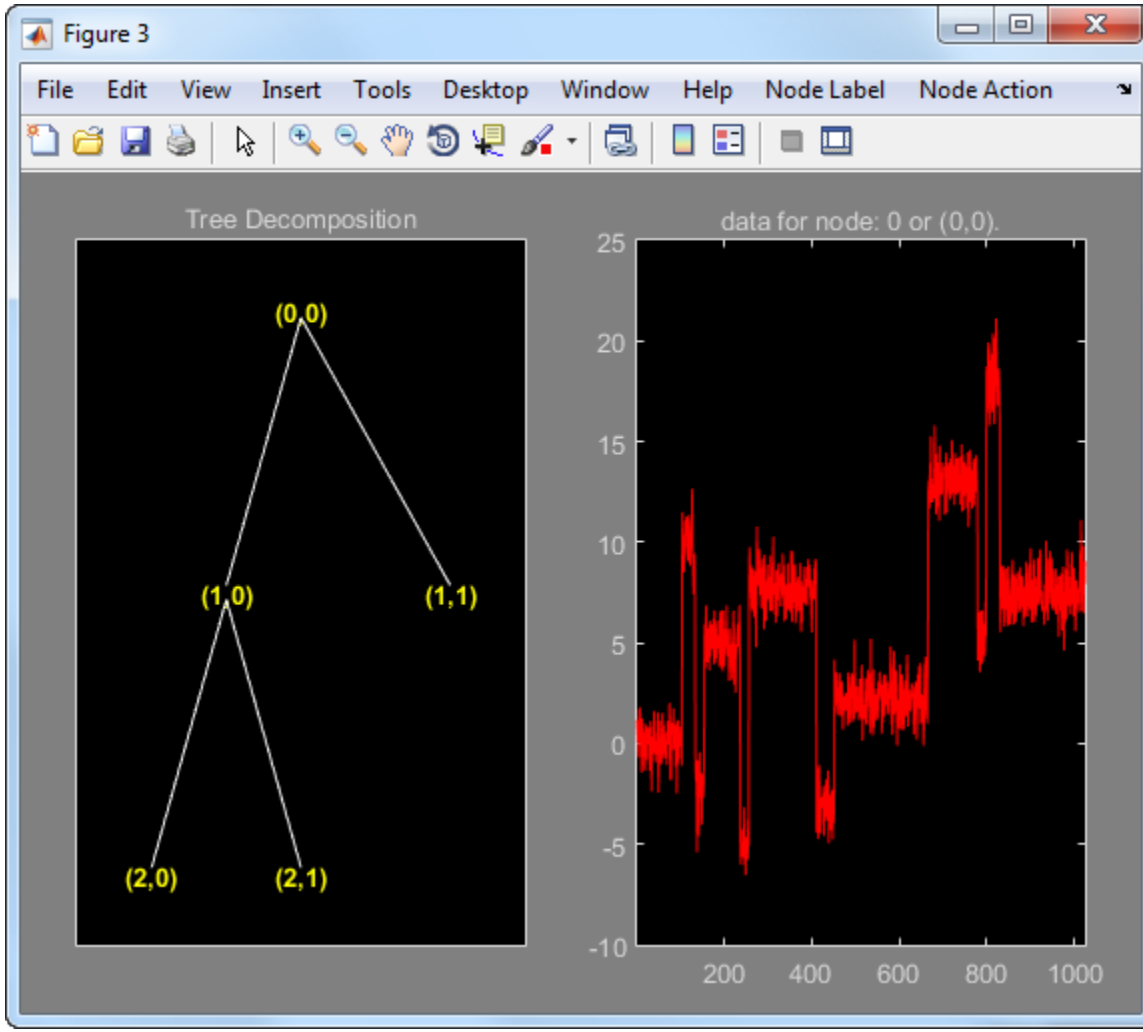
```
% Create a wavelet packets tree (1-D)
load noisbloc
x = noisbloc;
t = wpdec(x,2,'db2');

% Plot tree t.
plot(t)
```




```
% Change Node Label from Depth_Position to Index.

% Click the node (3). You get the following figure.
```



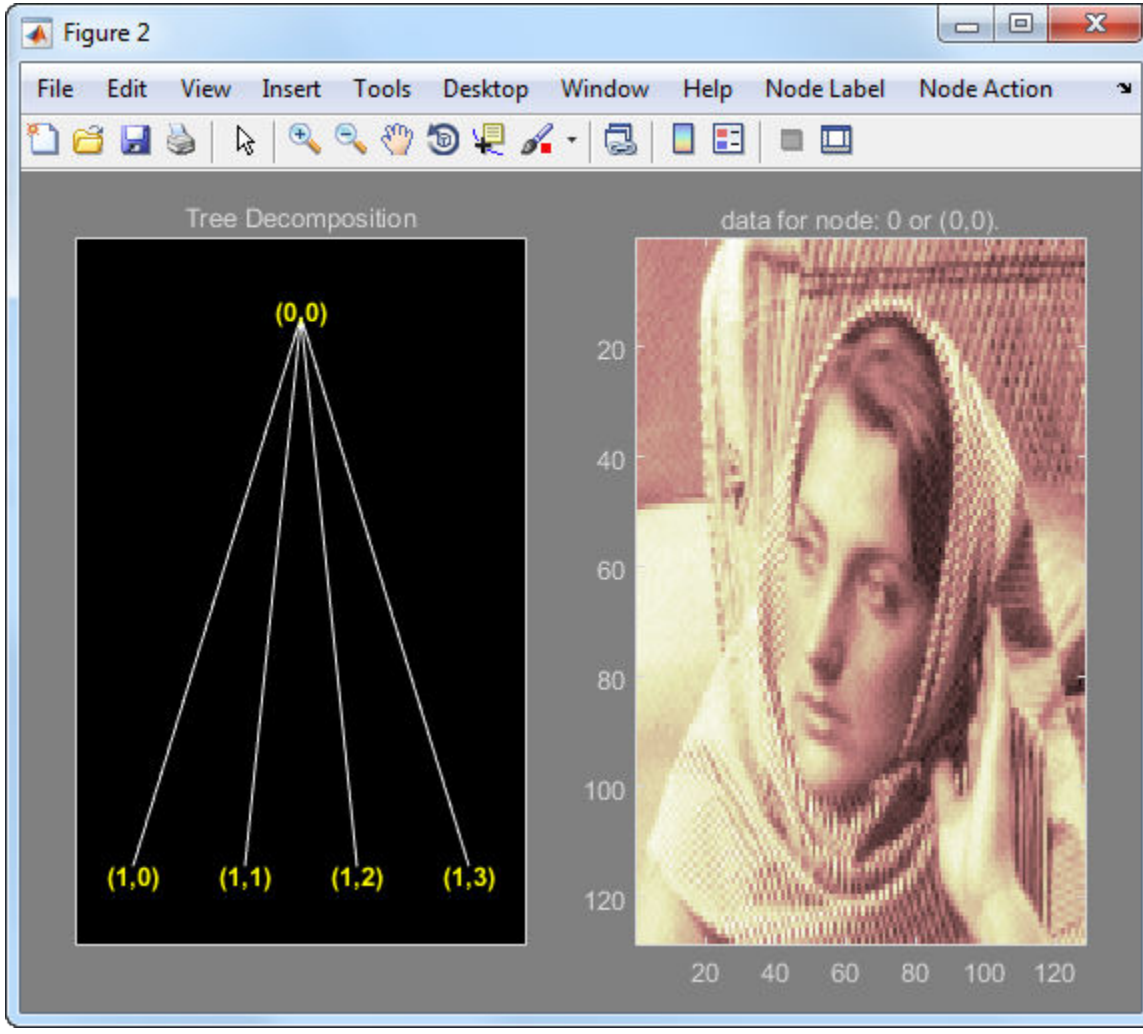
Now set the **Node Label** back to **Depth_Position**. Change **Node Action** to **Split-Merge**. Click on the `(1,1)` node.



The above figure now shows the discrete wavelet transform down to level 2.

```
% 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.
```



Introduced before R2006a

plotdt

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`.

Examples

Plot Complex Dual-Tree Wavelet Transform of 1-D Signal

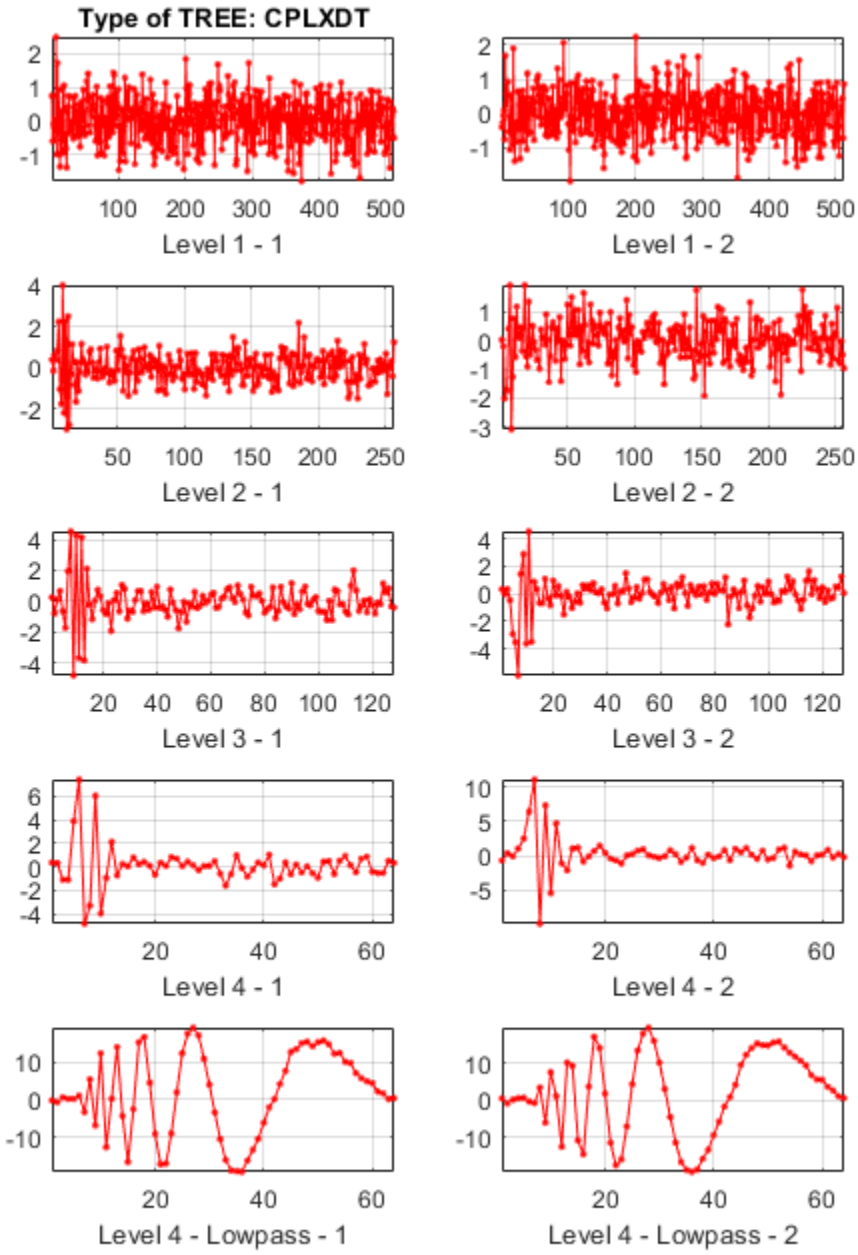
Plot the complex dual-tree wavelet transform of the noisy Doppler signal.

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)
```


Type of TREE: CPLXDT
Level 1 - 1
Level 1 - 2
Level 2 - 1
Level 2 - 2
Level 3 - 1
Level 3 - 2
Level 4 - 1
Level 4 - 2
Level 4 - Lowpass - 1
Level 4 - Lowpass - 2

Plot Complex Oriented Dual-Tree Wavelet Transform of 2-D Image

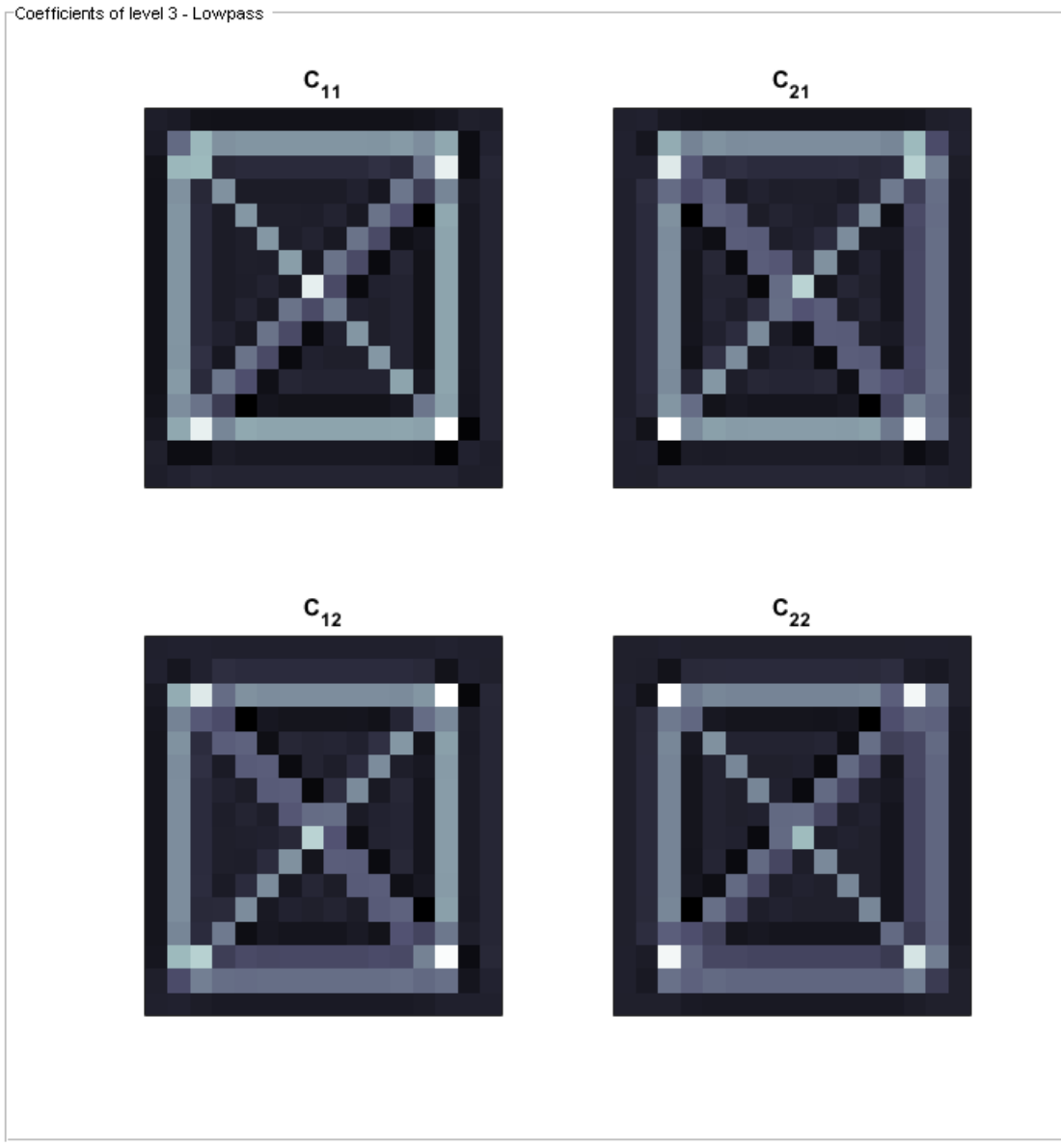
Plot the complex oriented dual-tree wavelet transform of an image.

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)
```


Coefficients of level 3 - Lowpass
C_{11}
C_{21}
C_{12}
C_{22}
Level 3 - Lowpass

Select the desired level detail coefficients from the drop-down list.

- “Analytic Wavelets Using the Dual-Tree Wavelet Transform”

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.

See Also

`dddtree` | `dddtree2` | `dddtreecfs`

Topics

“Analytic Wavelets Using the Dual-Tree Wavelet Transform”
“Critically Sampled and Oversampled Wavelet Filter Banks”

Introduced in R2013b

qmf

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 elements of the reversed vector filter coefficients `X` if `P` is `0`. If `P` is `1`, the signs of the odd index elements are reversed. Changing `P` changes the phase of the Fourier transform of the resulting wavelet filter by π radians.

`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 with norm equal to 1 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 `db10`):

$$|F_0(z)|^2 + |F_1(z)|^2 = 2.$$

Examples

Create a Quadrature Mirror Filter

This example shows how to create a quadrature mirror filter associated with the `db10` wavelet.

Compute the scaling filter associated with the `db10` wavelet.

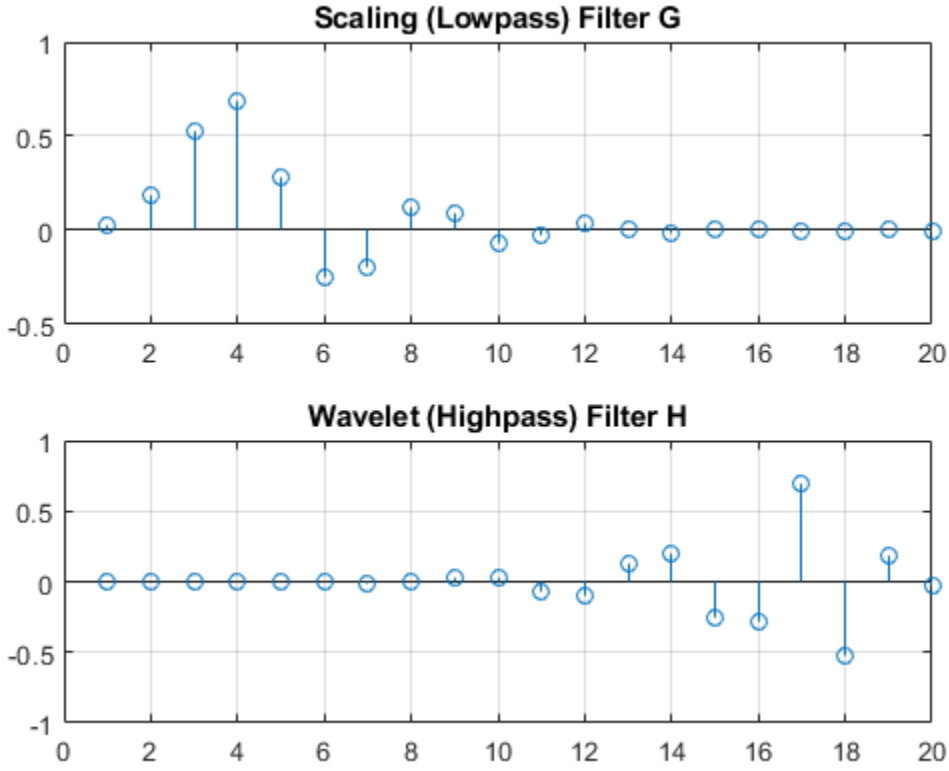
```
sF = dbwavf('db10');
```

`dbwavf` normalizes the filter coefficients so that the norm is equal to $1/\sqrt{2}$. Normalize the coefficients so that the filter has norm equal to 1.

```
G = sqrt(2)*sF;
```

Obtain the wavelet filter coefficients by using `qmf`. Plot the filters.

```
H = qmf(G);
subplot(2,1,1)
stem(G)
title('Scaling (Lowpass) Filter G')
grid on
subplot(2,1,2)
stem(H)
title('Wavelet (Highpass) Filter H')
grid on
```



Set the DWT extension mode to Periodization. Generate a random signal of length 64. Perform a single-level wavelet decomposition of the signal using `G` and `H`.

```
dwtmode('per')
```

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

```
n = 64;
rng 'default'
sig = randn(1,n);
[a,d] = dwt(sig,G,H);
```

The lengths of the approximation and detail coefficients are both 32. Confirm that the filters preserve energy.

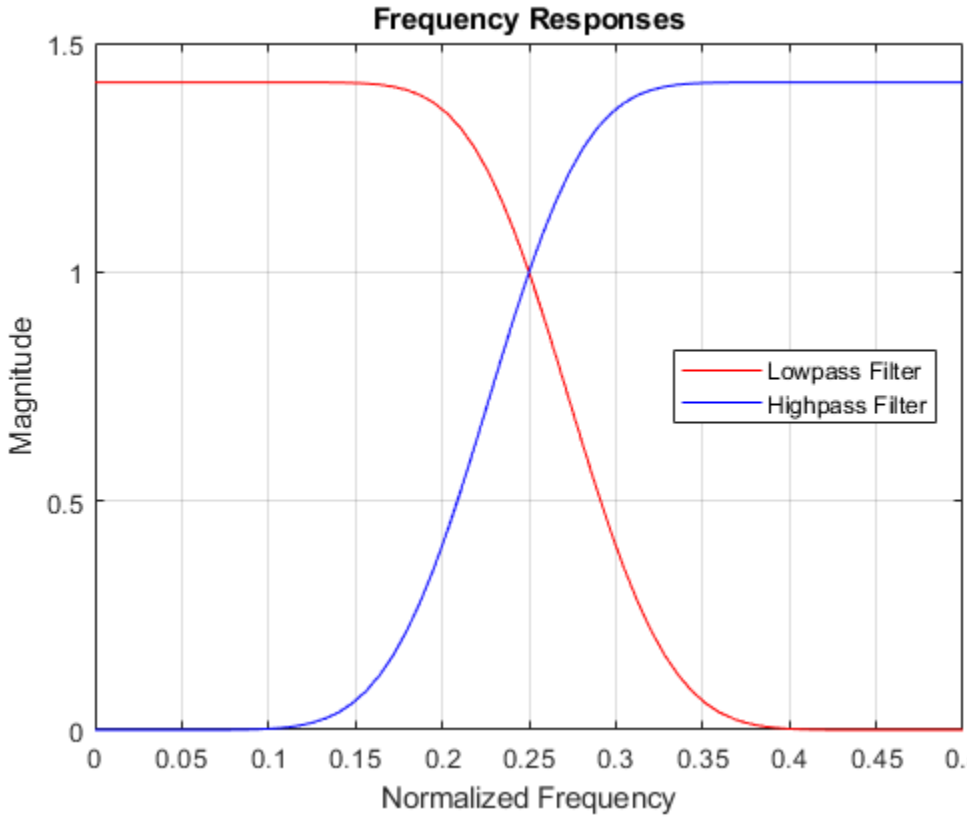
```
[sum(sig.^2) sum(a.^2)+sum(d.^2)]

ans =

   92.6872   92.6872
```

Compute the frequency responses of `G` and `H`. Zeropad the filters when taking the Fourier transform.

```
n = 128;
F = 0:1/n:1-1/n;
Gdft = fft(G,n);
Hdft = fft(H,n);
```

Plot the magnitude of each frequency response.

```
figure
plot(F(1:n/2+1),abs(Gdft(1:n/2+1)),'r')
hold on
plot(F(1:n/2+1),abs(Hdft(1:n/2+1)),'b')
grid on
title('Frequency Responses')
xlabel('Normalized Frequency')
ylabel('Magnitude')
legend('Lowpass Filter','Highpass Filter','Location','east')
```



Confirm the sum of the squared magnitudes of the frequency responses of `G` and `H` at each frequency is equal to 2.

```
sumMagnitudes = abs(Gdft).^2+abs(Hdft).^2;
[min(sumMagnitudes) max(sumMagnitudes)]
```

```
ans =

    2.0000    2.0000
```

Confirm that the filters are orthonormal.

```
df = [G;H];
id = df*df'
```

```
id =

    1.0000    0.0000
    0.0000    1.0000
```

Controlling the Phase of a Quadrature Mirror Filter

This example shows the effect of setting the phase parameter of the `qmf` function.

Obtain the decomposition low-pass filter associated with a Daubechies wavelet.

```
lowfilt = wfilters('db4');
```

Use the `qmf` function to obtain the decomposition low-pass filter for a wavelet. Then, compare the signs of the values when the `qmf` phase parameter is set to 0 or 1. The reversed signs indicates a phase shift of π radians, which is the same as multiplying the DFT by $e^{i\pi}$.

```
p0 = qmf(lowfilt,0)
p1 = qmf(lowfilt,1)
```

```
p0 =

  Columns 1 through 7

    0.2304   -0.7148    0.6309    0.0280   -0.1870   -0.0308    0.0329

  Column 8

    0.0106


p1 =

  Columns 1 through 7

   -0.2304    0.7148   -0.6309   -0.0280    0.1870    0.0308   -0.0329

  Column 8
```

```
   -0.0106
```

Compute the magnitudes and display the difference between them. Unlike the phase, the magnitude is not affected by the sign reversals.

```
abs(p0)-abs(p1)
```

```
ans =

     0     0     0     0     0     0     0     0
```

References

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

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Introduced before R2006a

rbiowavf

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 character vector `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.

Examples

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))
```

```
ans = 0
```

```
max(abs(sqrt(2)*RF-LoR))
```

```
ans = 0
```

See Also

`biorfilt` | `waveinfo`

Introduced before R2006a

read

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.

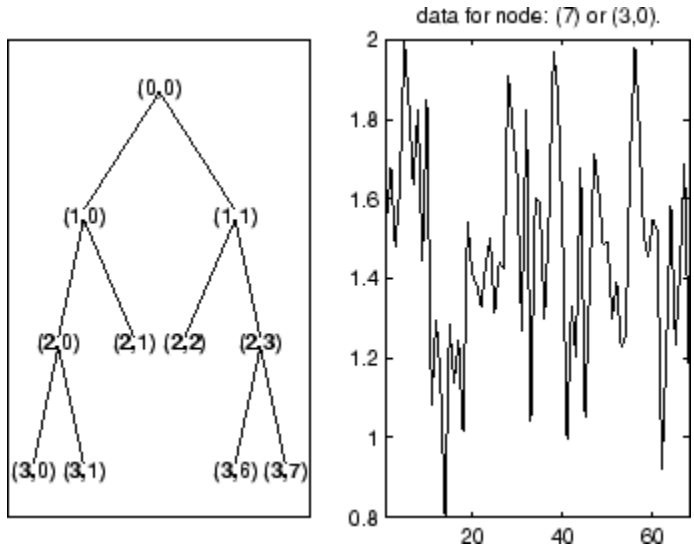
The valid choices for *`'PropName'`* and *`'PropParam'`* are listed in this table.

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);
```



See Also

`disp` | `get` | `set` | `wptree` | `write`

Introduced before R2006a

readtree

Read wavelet packet decomposition tree from figure

Syntax

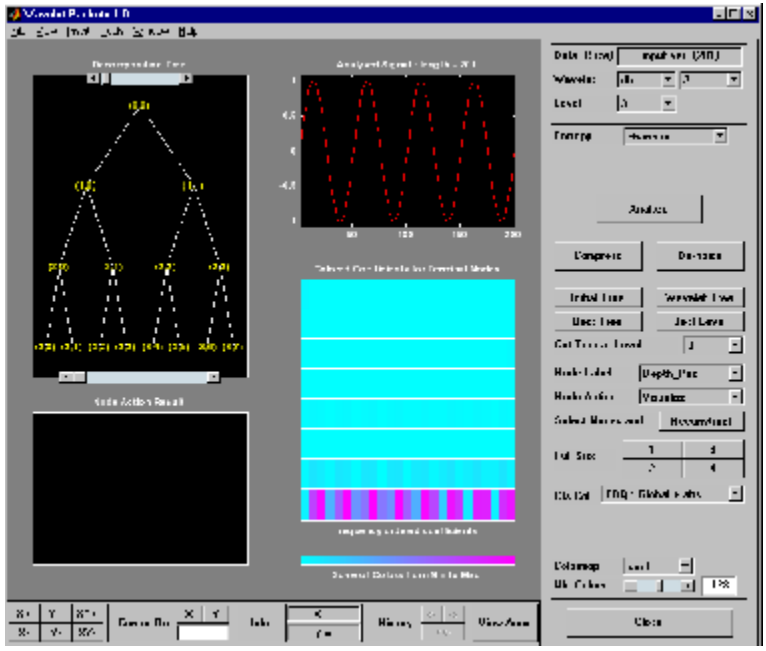
```
T = readtree(F)
```

Description

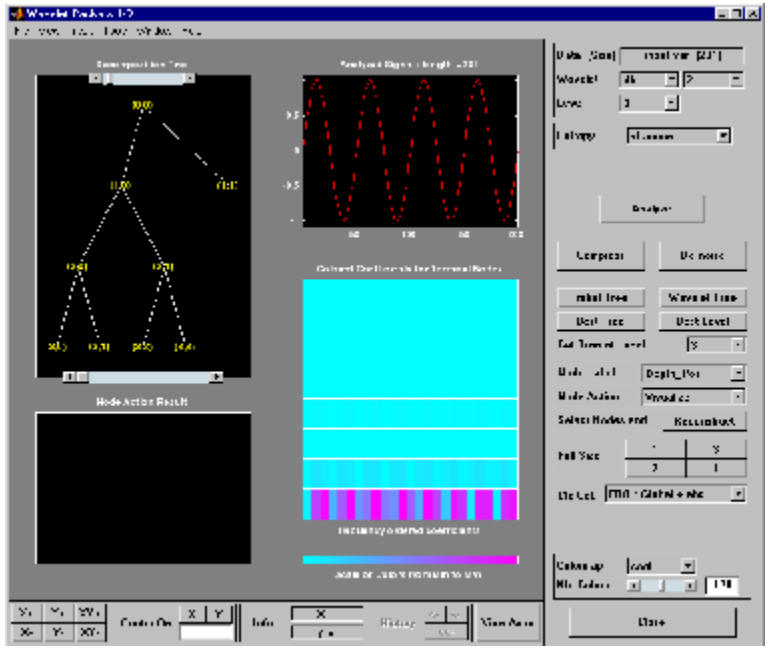
`T = readtree(F)` reads the wavelet packet decomposition tree from the figure whose handle is *F*.

Examples

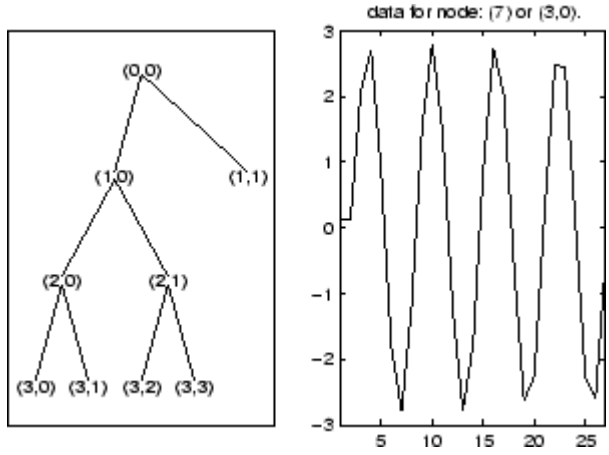
```
% Create a wavelet packet tree.
x   = sin(8*pi*[0:0.005:1]);
t   = wpdec(x,3,'db2');

% 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).
```



See Also

`drawtree`

Introduced before R2006a

scal2frq

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` and the wavelet function `'wname'` (see `wavefun` for more information) and the sampling period `DELTA`.

`scal2frq(A,'wname')` is equivalent to `scal2frq(A,'wname',1)`.

There is only an approximate answer for the relationship between scale and frequency.

In wavelet analysis, the way to relate scale to frequency is to determine the center frequency of the wavelet, F_c, and use the following relationship:

$$F_a = \frac{F_c}{a}$$

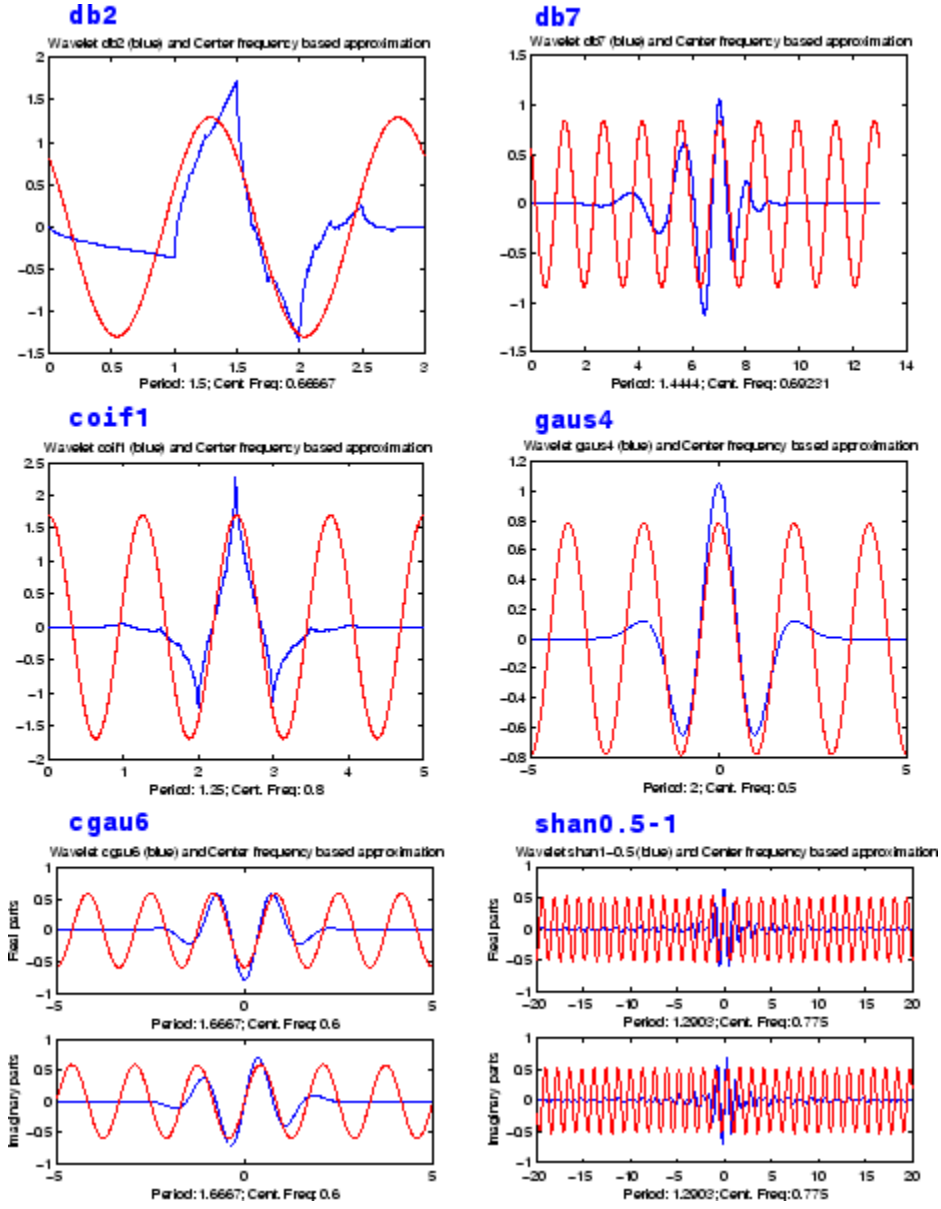
where

- a is a scale.
- F_c is the center frequency of the 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 Fourier transform of the wavelet modulus is F_c. `centfrq` computes the center frequency for a specified wavelet. From the above relationship, it can be seen that scale is inversely proportional to pseudo-frequency. For example, if the

scale increases, the wavelet becomes more spread out, resulting in a lower pseudo-frequency.

Some examples of the correspondence between the center frequency and the wavelet are shown in the following figure.



Center Frequencies for Real and Complex Wavelets

As you can see, the center frequency-based approximation captures the main wavelet oscillations. The center frequency is a convenient and simple characterization of the dominant frequency of the wavelet.

Examples

Scales To Frequencies

Construct a vector of scales with 10 voices per octave over five octaves. Assume the data are sampled at 10 kHz.

```
voicesperoctave = 10;
numoctaves = 5;
a0 = 2^(1/voicesperoctave);
Fs = 1e4;
scales = ...
    a0.^(voicesperoctave:1/voicesperoctave:numoctaves*voicesperoctave);
```

Convert the scales to approximate frequencies in hertz for the Morlet wavelet.

```
Frq = scal2frq(scales,'morl',1/Fs);
```

Determine the corresponding periods. Construct a table with the scales, the corresponding frequencies, and periods. Display the smallest 20 scales along with their corresponding frequencies and periods.

```
Frq = Frq(:);
scales = scales(:);
T = [scales.*(1/Fs) Frq 1./Frq];
T = array2table(T,'VariableNames',{'Scale','Frequency','Period'});
T(1:20,:)
```

```
ans =

  20x3 table

      Scale       Frequency      Period
    __________    _________    __________

        0.0002    4062.5       0.00024615
    0.00020139    4034.4       0.00024787
```

```
0.00020279    4006.6       0.00024959
 0.0002042    3978.9       0.00025133
0.00020562    3951.4       0.00025307
0.00020705    3924.1       0.00025483
0.00020849      3897       0.00025661
0.00020994    3870.1       0.00025839
 0.0002114    3843.4       0.00026019
0.00021287    3816.8         0.000262
0.00021435    3790.4       0.00026382
0.00021585    3764.3       0.00026566
0.00021735    3738.3        0.0002675
0.00021886    3712.4       0.00026936
0.00022038    3686.8       0.00027124
0.00022191    3661.3       0.00027312
0.00022346      3636       0.00027502
0.00022501    3610.9       0.00027694
0.00022658      3586       0.00027886
0.00022815    3561.2        0.0002808
```

Plot CWT with Frequencies in a Contour Plot

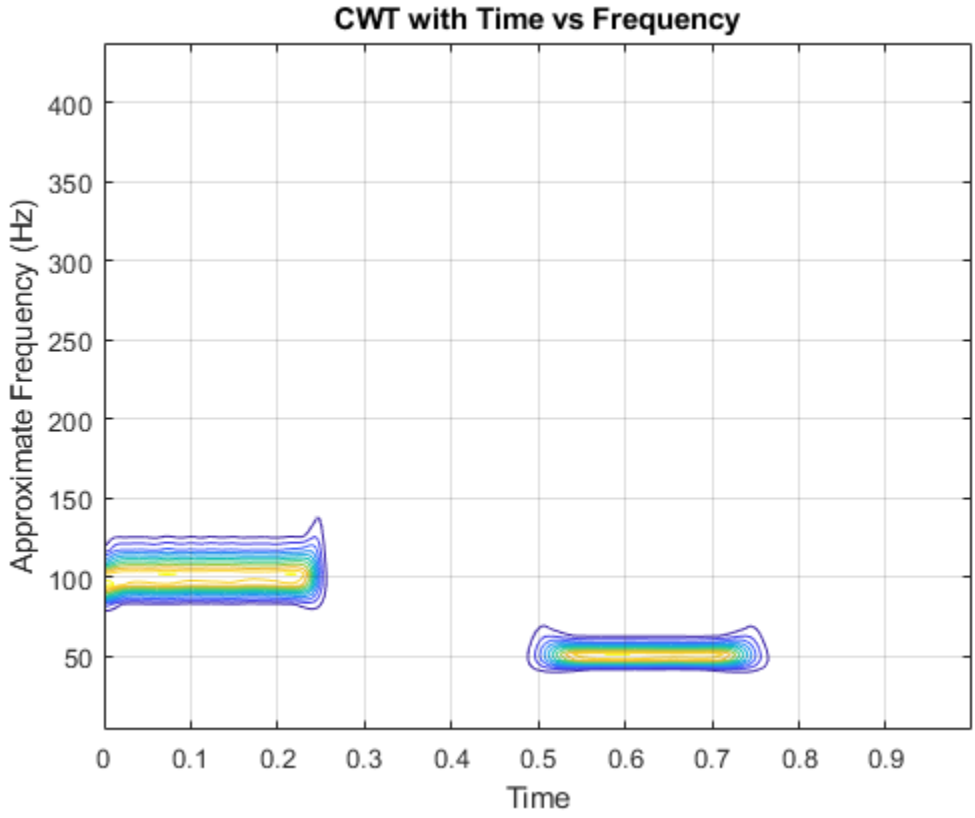
The example shows how to create a contour plot of the CWT using approximate frequencies in Hz.

Create a signal consisting of two sine waves with disjoint support in additive noise. Assume the signal is sampled at 1 kHz.

```
Fs = 1000;
t = 0:1/Fs:1-1/Fs;
x = 1.5*cos(2*pi*100*t).*(t<0.25)+1.5*cos(2*pi*50*t).*(t>0.5 & t<=0.75);
x = x+0.05*randn(size(t));
```

Obtain the CWT of the input signal and plot the result.

```
[cfs,f] = cwt(x,Fs);
contour(t,f,abs(cfs).^2);
axis tight;
grid on;
xlabel('Time');
ylabel('Approximate Frequency (Hz)');
title('CWT with Time vs Frequency');
```



References

Abry, P. (1997), *Ondelettes et turbulence. Multirésolutions, algorithmes de décomposition, invariance d'échelles*, Diderot Editeur, Paris.

See Also

`centfrq`

Introduced before R2006a

set

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`

Introduced before R2006a

shanwavf

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. The complex Shannon wavelet is 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

Complex Shannon Wavelet

Obtain and plot a complex Shannon wavelet. Set the bandwidth and center frequency parameters.

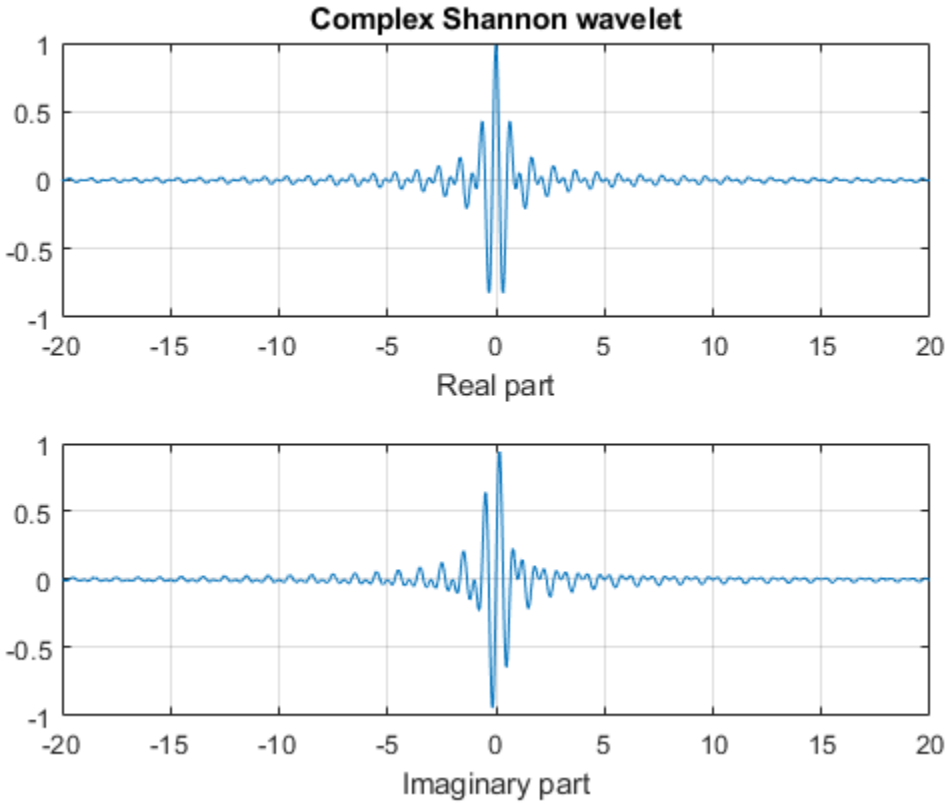
```
fb = 1; fc = 1.5;
```

Set the effective support and number of sample points.

```
lb = -20;
ub = 20;
n = 1000;
```

Obtain the complex-valued Shannon wavelet and plot the real and imaginary parts.

```
[psi,x] = shanwavf(lb,ub,n,fb,fc);
subplot(211)
plot(x,real(psi))
title('Complex Shannon wavelet')
xlabel('Real part');
grid on;
subplot(212)
plot(x,imag(psi))
xlabel('Imaginary part')
grid on;
```



References

Teolis, A. (1998), *Computational signal processing with wavelets*, Birkäuser, p. 62.

See Also

`waveinfo`

Introduced before R2006a

swt

Discrete stationary wavelet transform 1-D

Syntax

```
SWC = swt(X,N,'wname')
SWC = swt(X,N,Lo_D,Hi_D)
[SWA,SWD] = swt(___)
```

Description

`swt` performs a multilevel 1-D stationary wavelet decomposition using either an orthogonal or a biorthogonal wavelet. Specify the wavelet using its name (*'wname'*, see `wfilters` for more information) or its 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 ≤ i ≤ 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 periodic extension. The length of the approximation and detail coefficients computed at each level equals the length of the signal.

Examples

Multilevel Stationary Wavelet Decomposition

Perform a multilevel stationary wavelet decomposition of a signal.

Load a one-dimensional signal and acquire its length.

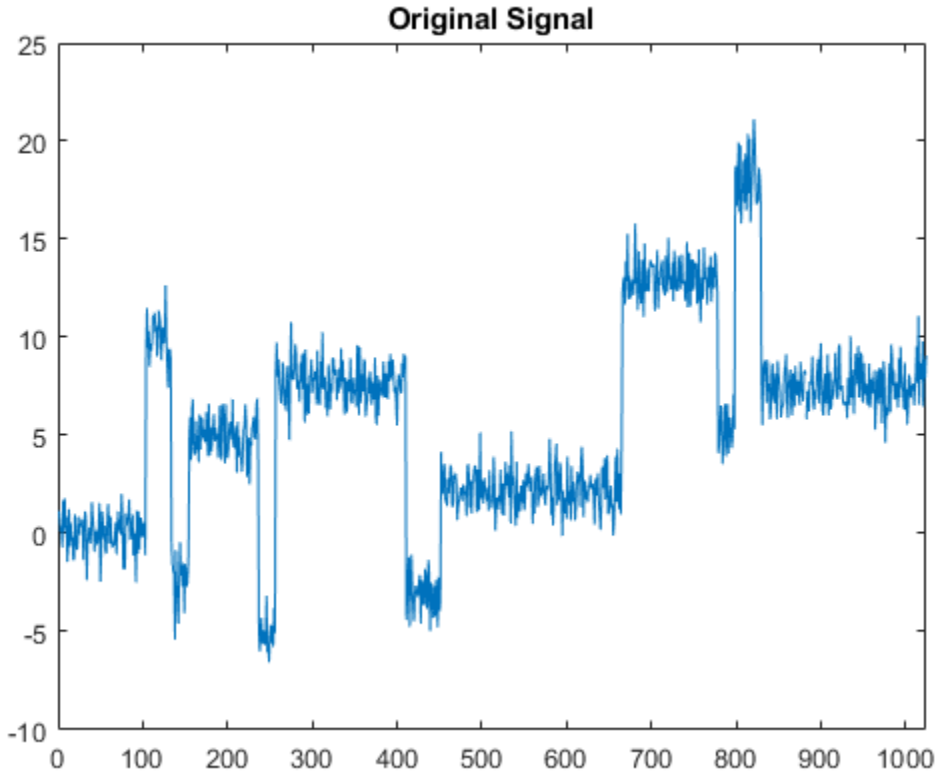
```
load noisbloc
s = noisbloc;
sLen = length(s);
```

Perform a stationary wavelet decomposition at level 3 of the signal using `'db1'`. Extract the detail and approximation coefficients at level 3.

```
[swa,swd] = swt(s,3,'db1');
swd3 = swd(3,:);
swa3 = swa(3,:);
```

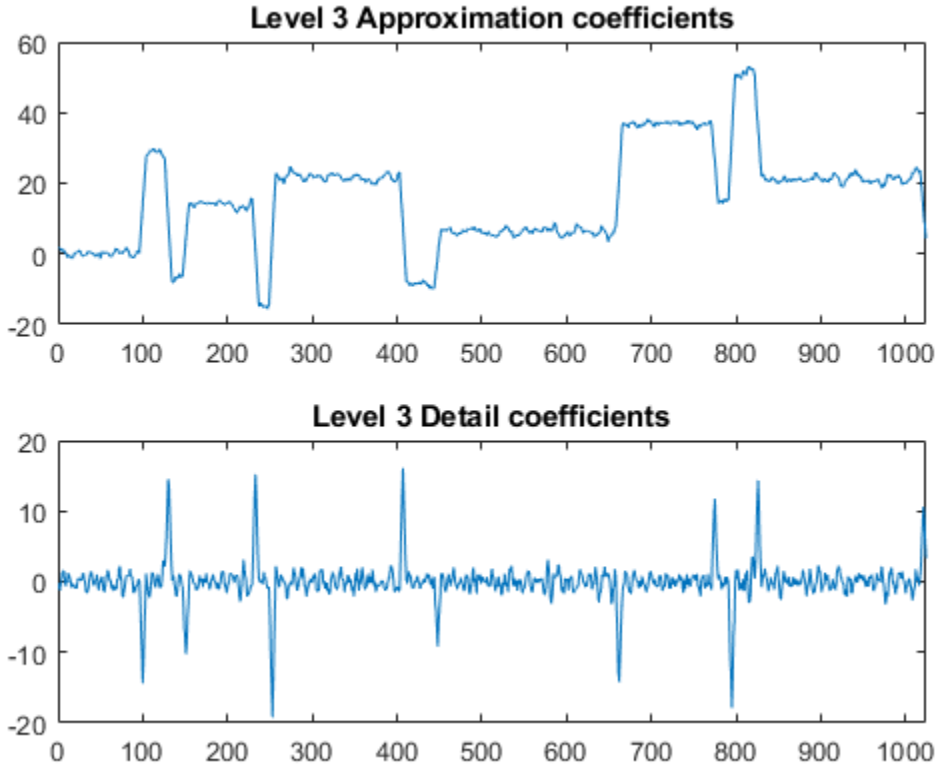
Plot the output of the decomposition.

```
plot(s)
xlim([0 sLen])
title('Original Signal')
```



Plot the level 3 approximation and detail coefficients.

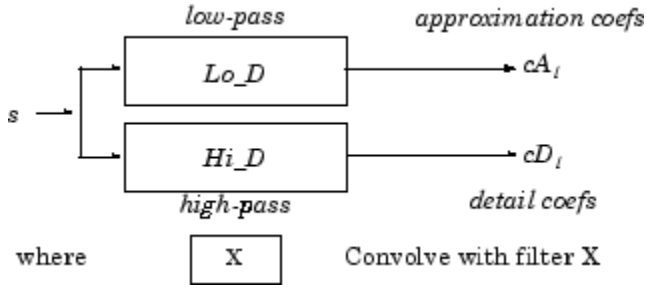
```
subplot(2,1,1)
plot(swa3)
xlim([0 sLen])
title('Level 3 Approximation coefficients')
subplot(2,1,2)
plot(swd3)
xlim([0 sLen])
title('Level 3 Detail coefficients')
```



Algorithms

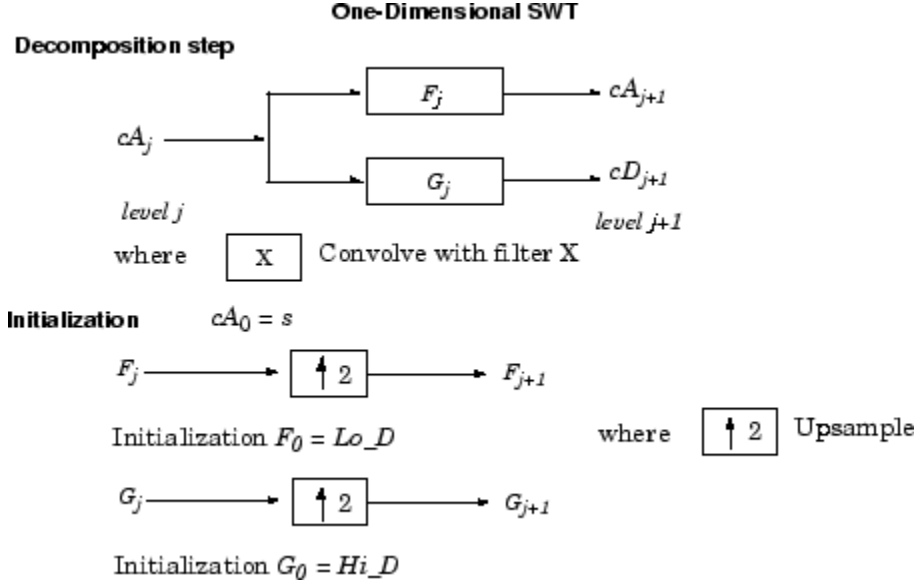
Given a signal s of length N, the first step of the SWT produces, starting from s, two sets of coefficients: approximation coefficients cA_1 and detail coefficients cD_1. These vectors are obtained by convolving s with the low-pass filter `Lo_D` for approximation, and with the high-pass filter `Hi_D` 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,



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` | `iswt` | `modwt` | `wavedec`

Introduced before R2006a

swt2

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 an orthogonal or a biorthogonal wavelet. Specify the wavelet using its name(`'wname'`, see `wfilters` for more information) or its decomposition filters.

`SWC = swt2(X,N,'wname')` or `[A,H,V,D] = swt2(X,N,'wname')` compute the stationary wavelet decomposition of the real-valued 2-D or 3-D matrix `X` at level `N`, using `'wname'`.

If `X` is a 3-D matrix, the third dimension of `X` must equal 3.

`N` must be a strictly positive integer (see `wmaxlev` for more information), and 2^N must divide `size(X,1)` and `size(X,2)`.

The dimension of `X` and level `N` determine the dimensions of the outputs.

- If `X` is a 2-D matrix and `N` is greater than 1, the 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)]
    ```

- If `X` is a 2-D matrix and `N` is equal to 1, the outputs `[A,H,V,D]` are 2-D arrays where `A` contains the approximation coefficients, and `H`, `V`, and `D` contain the horizontal, vertical, and diagonal detail coefficients, respectively.
- If `X` is a 3-D matrix of dimension `m`-by-`n`-by-3, and `N` is greater than 1, the outputs `[A,H,V,D]` are 4-D arrays of dimension `m`-by-`n`-by-3-by-`N`, which contain the coefficients:
  - For `1 ≤ i ≤ N` and `j = 1, 2, 3`, the output matrix `A(:,:,j,i)` contains the coefficients of approximation of level `i`.
  - The output matrices `H(:,:,j,i)`, `V(:,:,j,i)` and `D(:,:,j,i)` contain the coefficients of details of level `i` (horizontal, vertical, and diagonal):

    ```
    SWC = [H(:,:,1:3,1:N) ; V(:,:,1:3,1:N) ; D(:,:,1:3,1:N) ; A(:,:,1:3,N)]
    ```
- If `X` is a 3-D matrix of dimension `m`-by-`n`-by-3, and `N` is equal to 1, the outputs `[A,H,V,D]` are 4-D arrays of dimension `m`-by-`n`-by-1-by-3, which contain the coefficients:
  - For `j = 1, 2, 3`, the output matrix `A(:,:,1,j)` contains the approximation coefficients.
  - The output matrices `H(:,:,1,j)`, `V(:,:,1,j)` and `D(:,:,1,j)` contain the horizontal, vertical, and diagonal detail coefficients, respectively.

    ```
    SWC = [H(:,:,1,1:3) ; V(:,:,1,1:3) ; D(:,:,1,1:3) ; A(:,:,1,1:3)]
    ```

Note

- `swt2` uses double-precision arithmetic internally and returns double-precision coefficient matrices. `swt2` warns if there is a loss of precision when converting to double.
- To distinguish a single-level decomposition of a truecolor image from a multilevel decomposition of an indexed image, the approximation and detail coefficient arrays of truecolor images are 4-D arrays. See the Release Notes for details. Also see examples “Stationary Wavelet Transform of an Image” on page 1-728 and “Inverse Stationary Wavelet Transform of an Image” on page 1-733.

  If an `K`-level decomposition is performed, the dimensions of the `A`, `H`, `V`, and `D` coefficient arrays are `m`-by-`n`-by-3-by-`K`.

If a single-level decomposition is performed, the dimensions of the `A`, `H`, `V`, and `D` coefficient arrays are `m`-by-`n`-by-1-by-3. Since MATLABremoves singleton last dimensions by default, the third dimension of the coefficient arrays is singleton.

`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.

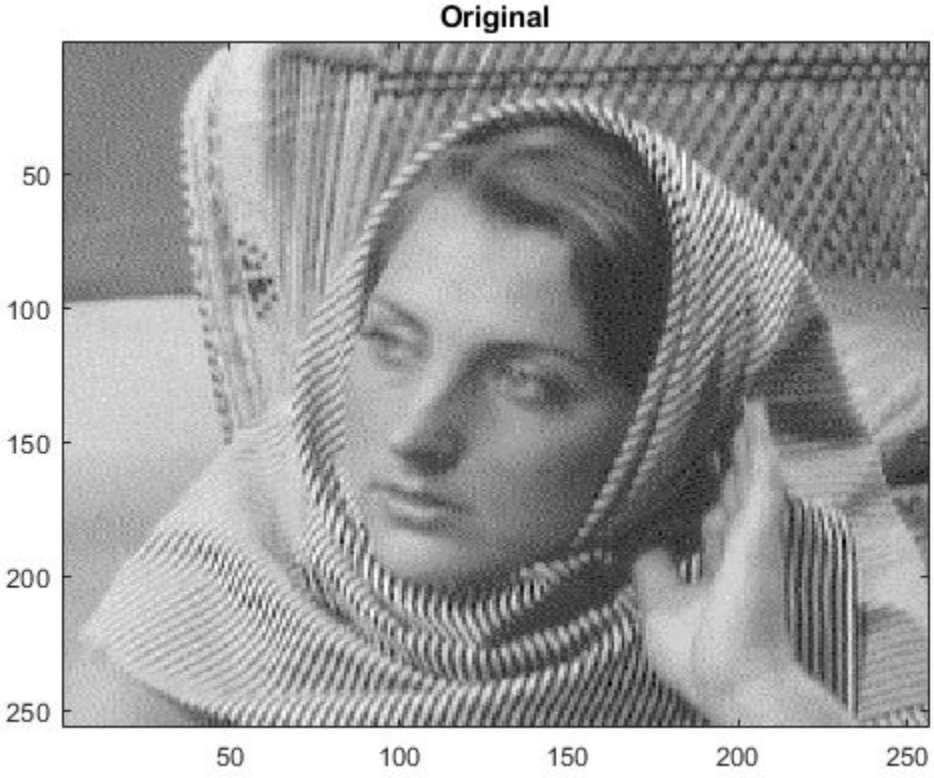
Note `swt2` is defined using periodic extension. The size of the approximation and details coefficients computed at each level equals the size of the input data.

Examples

Extract and Display Two-Dimensional Stationary Wavelet Decomposition

Extract and display images of the stationary wavelet decomposition level coefficients. First load and display the original image. Then perform the stationary wavelet decomposition of the image at level 2 using `db6`.

```
load woman
imagesc(X)
colormap(map)
title('Original')
```



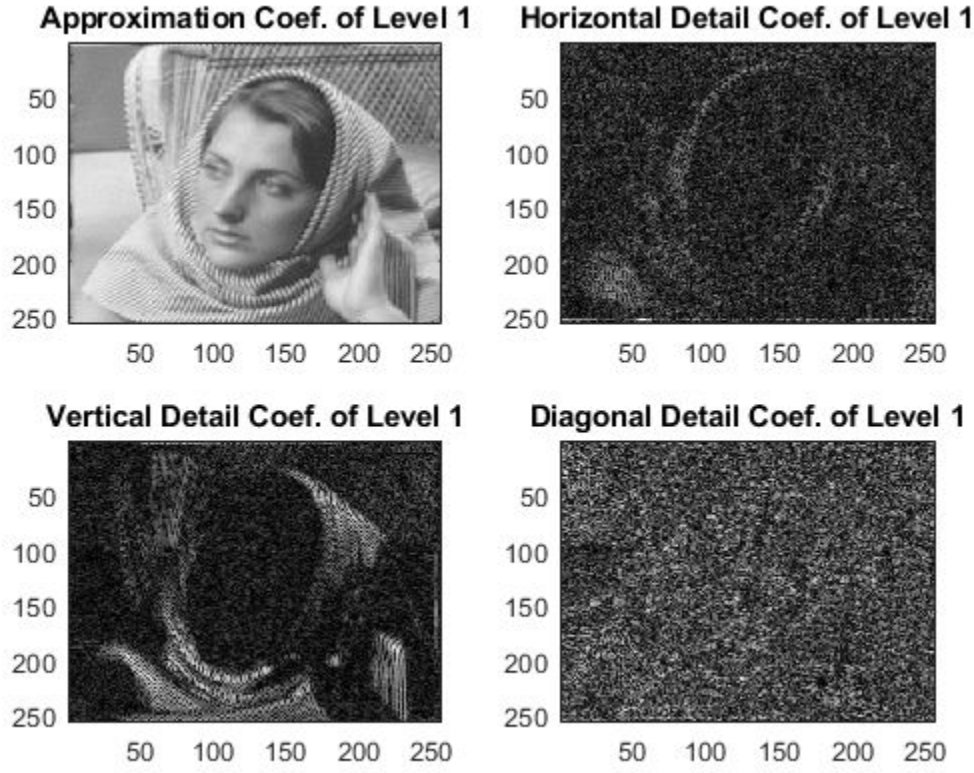

```
[ca,chd,cvd,cdd] = swt2(X,2,'db6');
```

Extract the Level 1 and Level 2 approximation and detail coefficients from the decomposition.

```
A1 = wcodemat(ca(:,:,1),255);
H1 = wcodemat(chd(:,:,1),255);
V1 = wcodemat(cvd(:,:,1),255);
D1 = wcodemat(cdd(:,:,1),255);

A2 = wcodemat(ca(:,:,2),255);
H2 = wcodemat(chd(:,:,2),255);
V2 = wcodemat(cvd(:,:,2),255);
D2 = wcodemat(cdd(:,:,2),255);
```

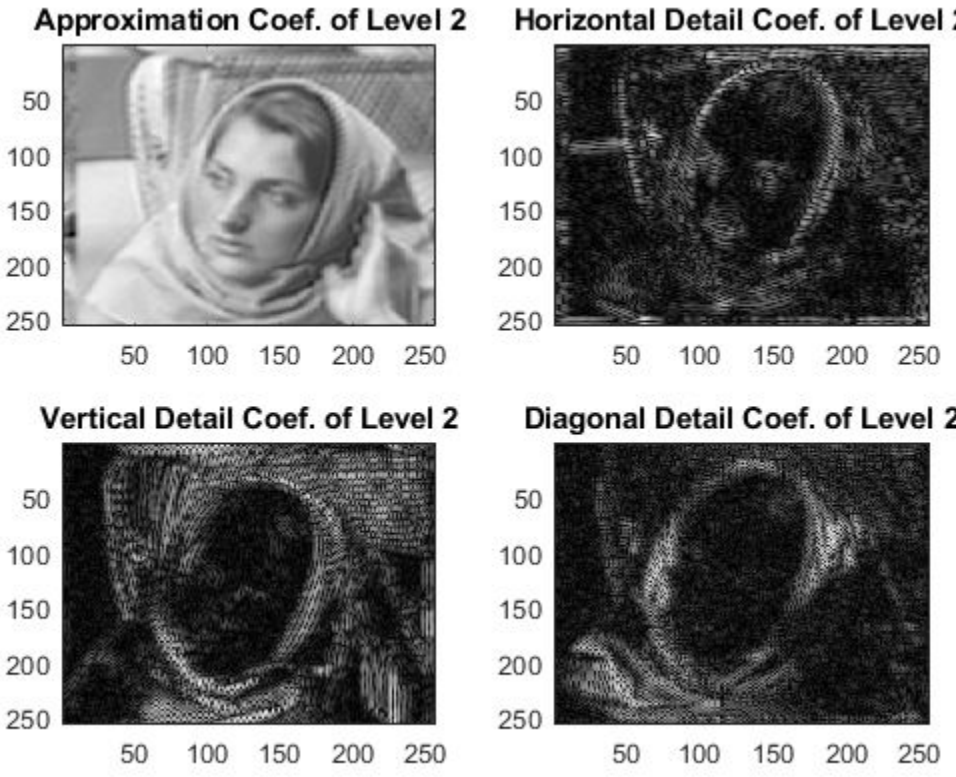
Display the approximation and detail coefficients from the two levels.

```
subplot(2,2,1)
imagesc(A1)
title('Approximation Coef. of Level 1')

subplot(2,2,2)
imagesc(H1)
title('Horizontal Detail Coef. of Level 1')

subplot(2,2,3)
imagesc(V1)
title('Vertical Detail Coef. of Level 1')

subplot(2,2,4)
imagesc(D1)
title('Diagonal Detail Coef. of Level 1')
```




```
subplot(2,2,1)
imagesc(A2)
title('Approximation Coef. of Level 2')

subplot(2,2,2)
imagesc(H2)
title('Horizontal Detail Coef. of Level 2')

subplot(2,2,3)
imagesc(V2)
title('Vertical Detail Coef. of Level 2')

subplot(2,2,4)
```

```
imagesc(D2)
title('Diagonal Detail Coef. of Level 2')
```

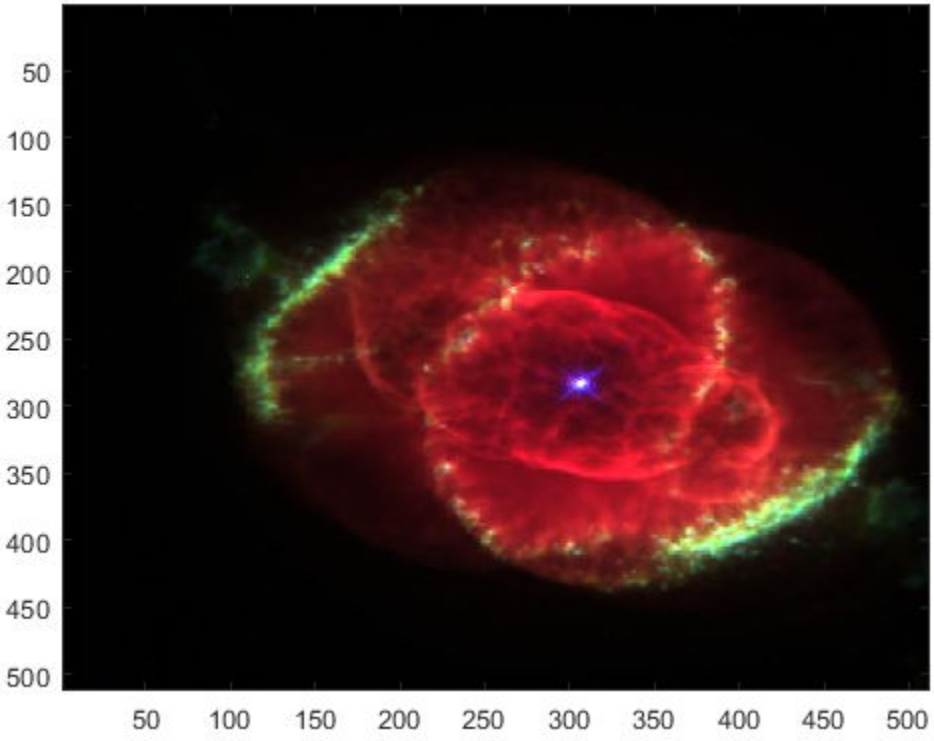


Stationary Wavelet Transform of an Image

In this example you obtain single-level and multilevel stationary wavelet decompositions of a truecolor image. You view results of each decomposition.

Load in a truecolor image. The image is a 3-D array of type `uint8`. Since `swt2` requires the first and second dimensions both be divisible by a power of 2, extract a portion of the image and view it.

```
imdata = imread('ngc6543a.jpg');
x = imdata(1:512,1:512,:);
imagesc(x)
```



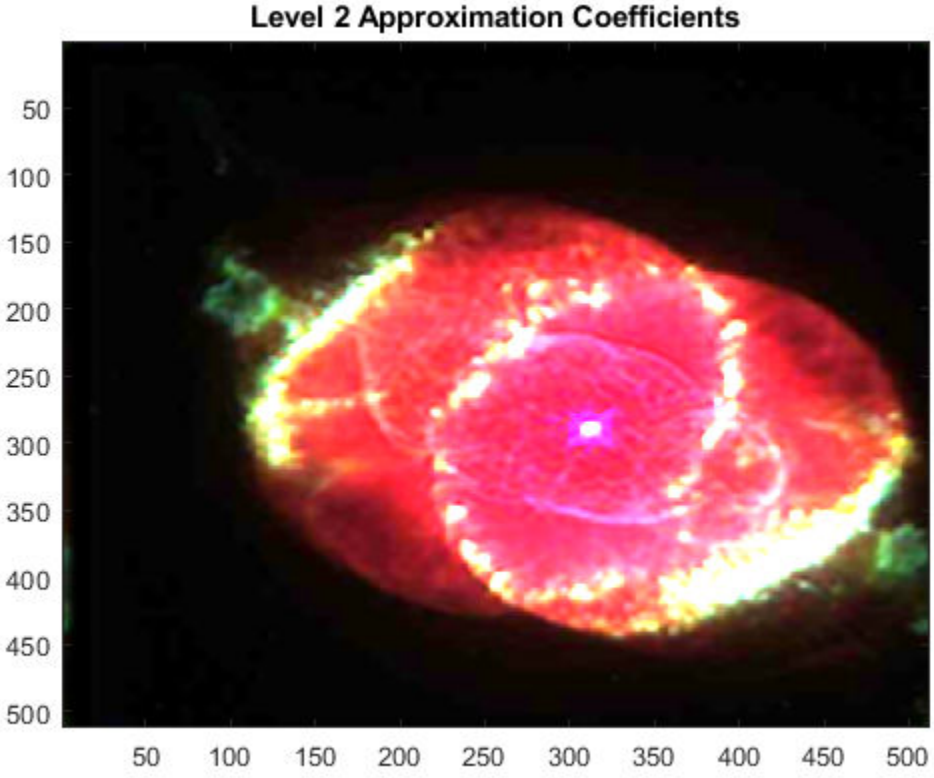
Obtain the 4-level stationary wavelet decomposition of the image using the `db4` wavelet. Return the approximation coefficients and the horizontal, vertical, and detail coefficients as separate arrays. Note the dimensions of the output arrays.

```
[a,h,v,d] = swt2(x,4,'db4');
size(a)

ans =
```

```
   512   512     3     4

size(h)

ans =

   512   512     3     4

size(v)

ans =

   512   512     3     4

size(d)

ans =

   512   512     3     4
```

The output arrays are all of type `double`. View the level 2 approximation coefficients. Since the approximation coefficients are of type `double`, cast them as `uint8`, which was the original datatype of the image.

```
figure
imagesc(uint8(a(:,:,:,2)))
title('Level 2 Approximation Coefficients')
```



Reconstruct the image by performing the inverse transform. Compute the difference between the original image and reconstruction. Since the reconstruction is of type `double`, cast the reconstruction as type `uint8` before computing the difference.

```
rec = iswt2(a,h,v,d,'db4');
maxdiff = max(abs(uint8(rec(:))-x(:)));
disp(['maximum difference = ' num2str(maxdiff)])
```

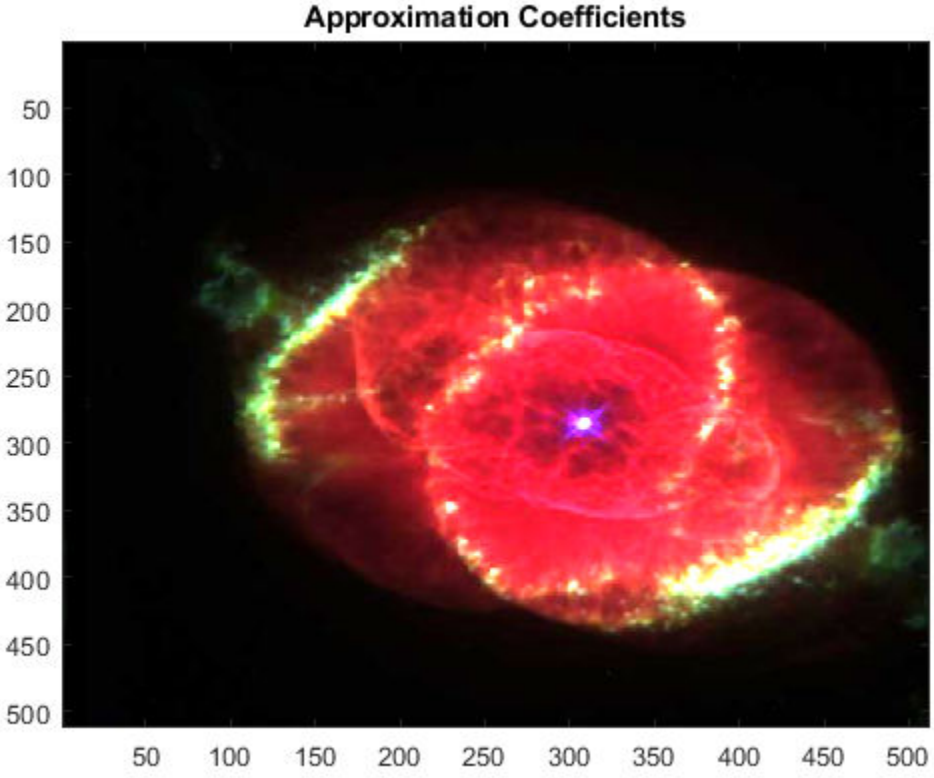
```
maximum difference = 0
```

Obtain the single-level stationary wavelet decomposition of the image using the `db4` wavelet. Return the approximation coefficients and horizontal, vertical, and detail coefficients in separate arrays. Note the dimensions of the output arrays.

```
[a,h,v,d] = swt2(x,1,'db4');
size(a)

ans =

   512   512     1     3


size(h)

ans =

   512   512     1     3


size(v)

ans =

   512   512     1     3


size(d)

ans =

   512   512     1     3
```

View the approximation coefficients. To prevent an error when using `imagesc`, first squeeze the approximation coefficients array to remove the singleton dimension.

```
asqueeze = squeeze(a);
size(asqueeze)

ans =

   512   512     3


figure
imagesc(uint8(asqueeze))
title('Approximation Coefficients')
```



Reconstruct the image by performing the inverse transform. Compute the difference between the original image and reconstruction. Since the reconstruction is of type `double`, cast the reconstruction as type `uint8` before computing the difference.

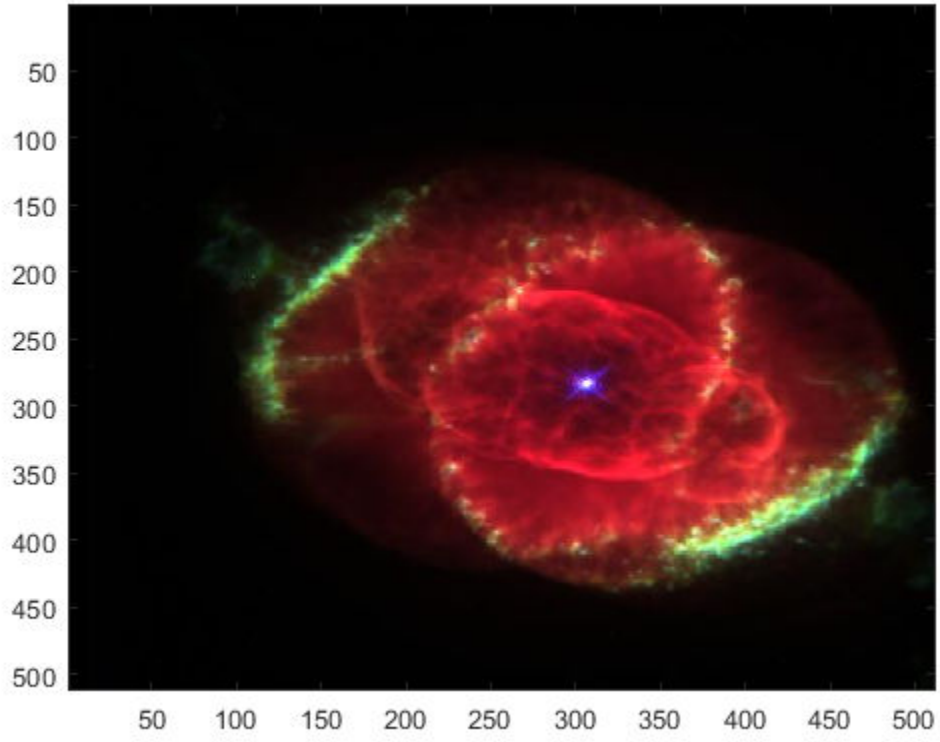
```
rec = iswt2(a,h,v,d,'db4');
maxdiff = max(abs(uint8(rec(:))-x(:)));
disp(['maximum difference = ' num2str(maxdiff)])

maximum difference = 0
```

Inverse Stationary Wavelet Transform of an Image

This example shows how to reconstruct a truecolor image from a single-level stationary wavelet decomposition using 3-D approximation and detail coefficient arrays.

Load in a truecolor image. The image is a 3-D array of type `uint8`. Since `swt2` requires the first and second dimensions both be divisible by a power of 2, extract a portion of the image and view it.

```
imdata = imread('ngc6543a.jpg');
x = imdata(1:512,1:512,:);
imagesc(x)
```



Obtain the single-level stationary wavelet decomposition of the image using the `db4` wavelet. Return the approximation coefficients and horizontal, vertical, and detail coefficients in separate arrays. Note the dimensions of the output arrays.

```
[a,h,v,d] = swt2(x,1,'db4');
size(a)

ans =

   512   512     1     3


size(h)

ans =

   512   512     1     3


size(v)

ans =

   512   512     1     3


size(d)

ans =

   512   512     1     3
```

Squeeze the coefficient arrays to remove their singleton dimensions. Note the dimensions of the squeezed arrays.

```
asq = squeeze(a);
hsq = squeeze(h);
vsq = squeeze(v);
dsq = squeeze(d);
size(asq)

ans =

   512   512     3
```

```
size(hsq)

ans =

   512   512     3


size(vsq)

ans =

   512   512     3


size(dsq)

ans =

   512   512     3
```

So that `iswt2` can properly reconstruct the trueimage from the new coefficient arrays, insert a singleton dimension by reshaping the squeezed arrays. Reconstruct the image with the reshaped coefficient arrays.

```
a2 = reshape(asq,[512,512,1,3]);
h2 = reshape(hsq,[512,512,1,3]);
v2 = reshape(vsq,[512,512,1,3]);
d2 = reshape(dsq,[512,512,1,3]);
rec = iswt2(a2,h2,v2,d2,'db4');
```

Compute the difference between the original image and reconstruction. Since the reconstruction is of type `double`, cast the reconstruction as type `uint8` before computing the difference.

```
maxdiff = max(abs(uint8(rec(:))-x(:)));
disp(['maximum difference = ' num2str(maxdiff)])

maximum difference = 0
```

Tips

When X represents an indexed image, X is an `m`-by-`n` matrix. If the level of decomposition `N` is greater than 1, the output arrays SWC or cA, cH, cV, and cD are `m`-by-`n`-by-`N` arrays. If the level of decomposition `N` is equal to 1, the output arrays SWC or cA, cH, cV, and cD are `m`-by-`n` 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. If the level of decomposition `N` is greater than 1, the output arrays SWC or cA, cH, cV, and cD are `m`-by-`n`-by-3-by-`N`. If the level of decomposition `N` is equal to 1, the output arrays SWC or cA, cH, cV, and cD are `m`-by-`n`-by-1-by-3.

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

Algorithms

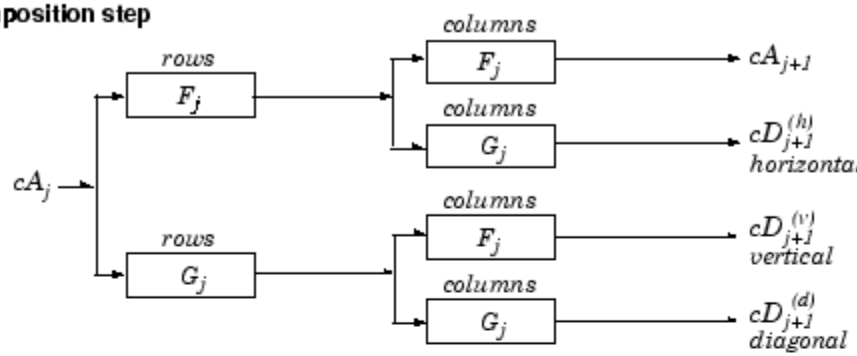
For images, an algorithm similar to the one-dimensional case is possible for two-dimensional wavelets and scaling functions obtained from one-dimensional ones by tensor product.

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

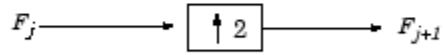


where

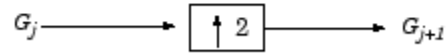
rows
X Convolve with filter X the rows of the entry

columns
X Convolve with filter X the columns of the entry

Initialization $cA_0 = s$ for the decomposition initialization



Initialization $F_0 = Lo\_D$ where 2 Upsample



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`

Introduced before R2006a

symaux

Symlet wavelet filter computation

The `symaux` function generates the scaling filter coefficients for the "least asymmetric" Daubechies wavelets.

Syntax

```
w = symaux(n)
w = symaux(___,sumw)
```

Description

`w = symaux(n)` is the order `n` Symlet scaling filter such that `sum(w) = 1`.

Note

- Instability may occur when `n` is too large. Starting with values of `n` in the 30s range, function output will no longer accurately represent scaling filter coefficients.
- As `n` increases, the time required to compute the filter coefficients rapidly grows.

`w = symaux(___,sumw)` is the order `n` Symlet scaling filter such that `sum(w) = sumw`.

`w = symaux(n,0)` is equivalent to `w = symaux(n,1)`.

Examples

Unit Norm Scaling Filter Coefficients

In this example you will generate symlet scaling filter coefficients whose norm is equal to 1.You will also confirm the coefficients satisfy a necessary relation. This example requires the Signal Processing Toolbox.

Compute the scaling filter coefficients of the order 10 symlet whose sum equals $\sqrt{2}$.

```
n = 10;
w = symaux(n,sqrt(2));
```

Confirm the sum of the cofficients is equal to $\sqrt{2}$ and the norm is equal to 1.

```
sqrt(2)-sum(w)
```

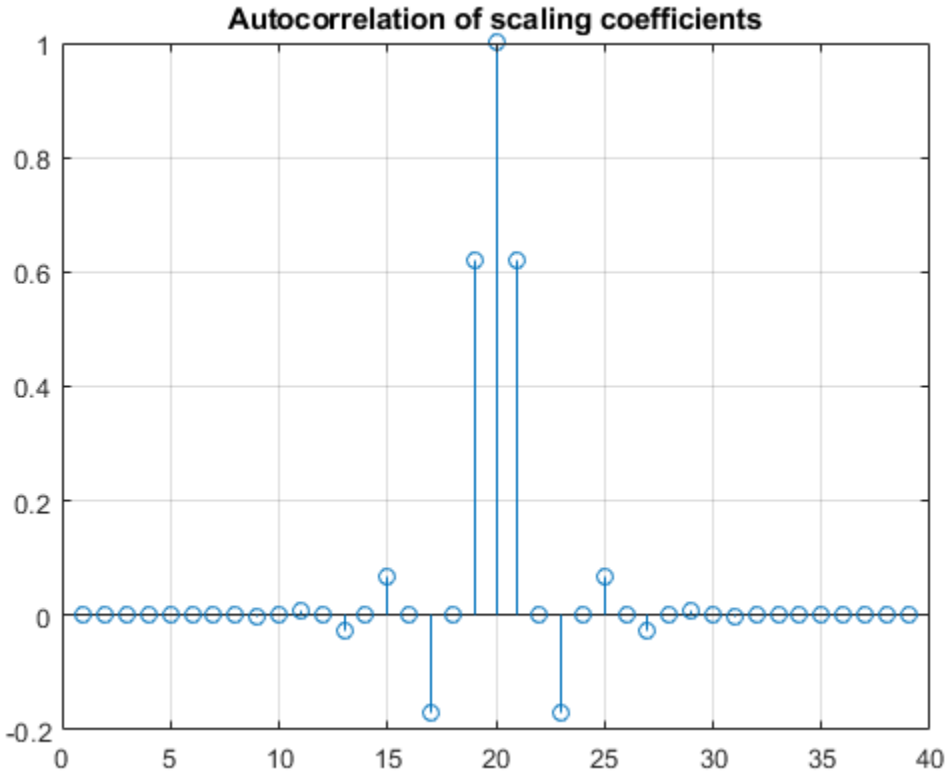
```
ans = -2.2204e-16
```

```
1-sum(w.^2)
```

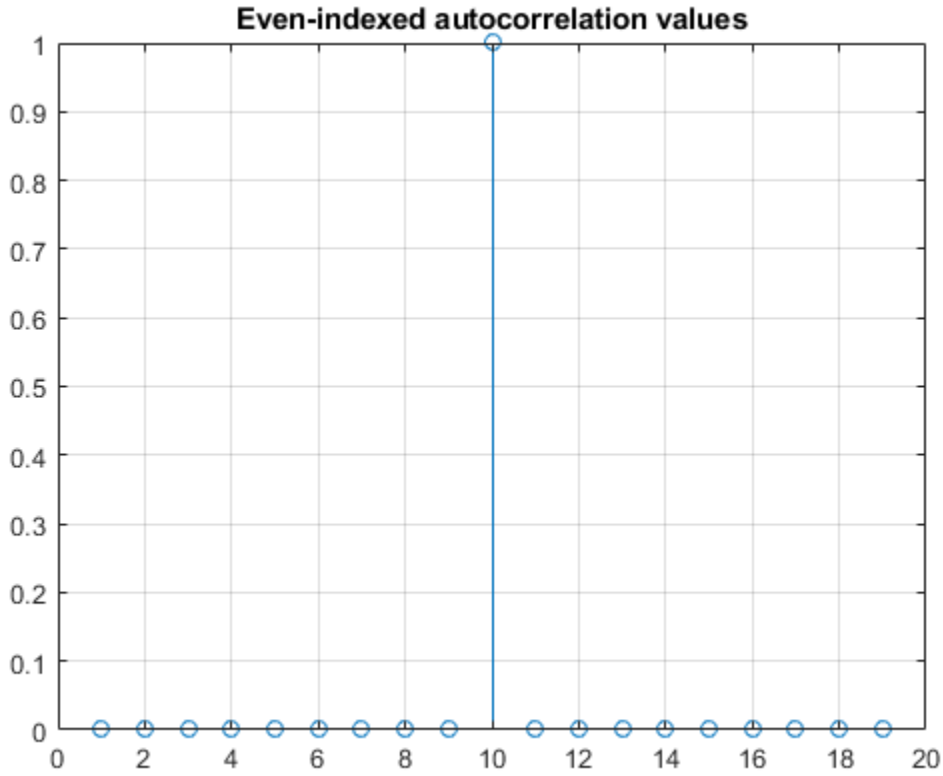
```
ans = 2.5202e-14
```

Since integer translations of the scaling function form an orthogonal basis, the coefficients satisfy the relation $\sum_n w(n)w(n-2k) = \delta(k)$. Confirm this by taking the autocorrelation of the coefficients and plotting the result.

```
corrw = xcorr(w,w);
stem(corrw)
grid on
title('Autocorrelation of scaling coefficients')
```




```
stem(corrw(2:2:end))
grid on
title('Even-indexed autocorrelation values')
```



Symlet and Daubechies Scaling Filters

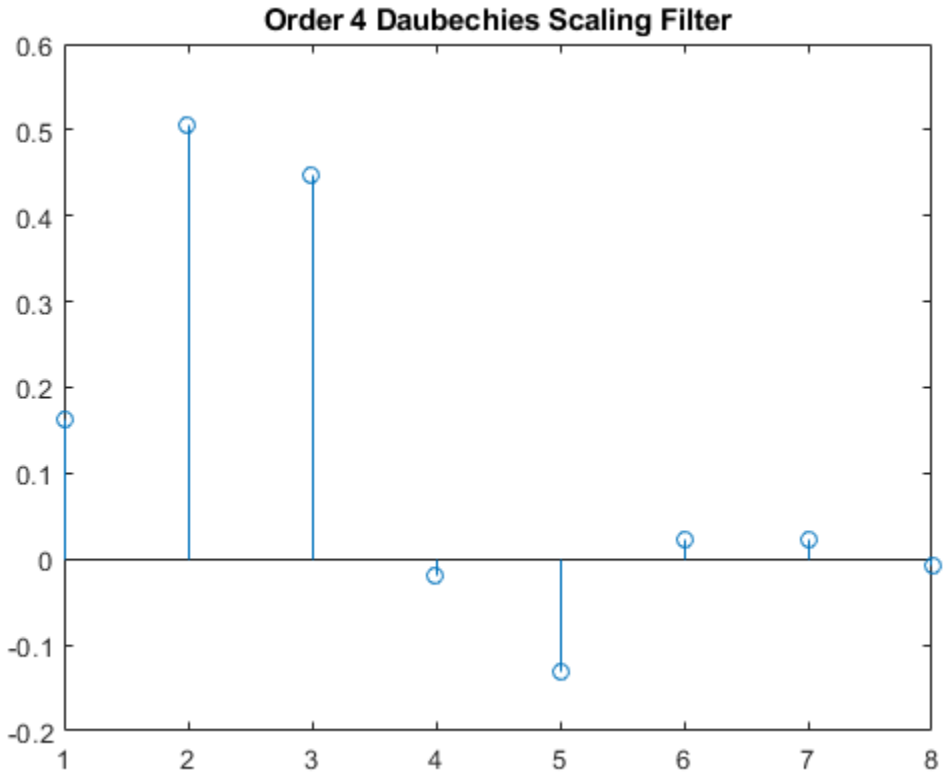
This example shows that symlet and Daubechies scaling filters of the same order are both solutions of the same polynomial equation.

Generate the order 4 Daubechies scaling filter and plot it.

```
wdb4 = dbaux(4)
```

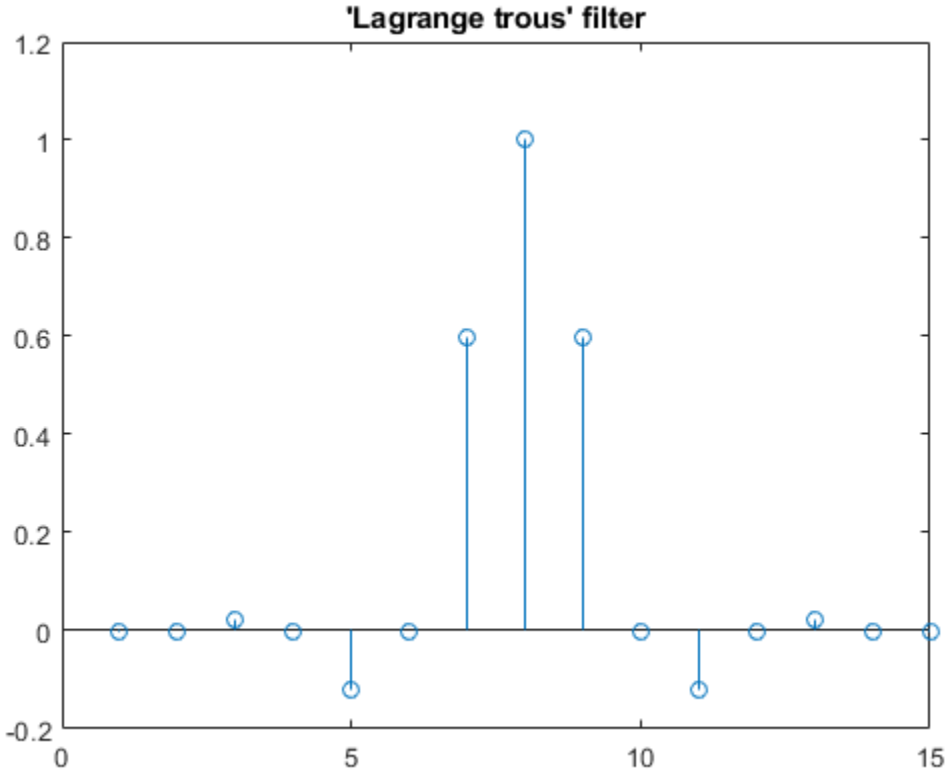
```
wdb4 =

  Columns 1 through 7

    0.1629    0.5055    0.4461   -0.0198   -0.1323    0.0218    0.0233

  Column 8

   -0.0075
```

```
stem(wdb4)
title('Order 4 Daubechies Scaling Filter')
```



wdb4 is a solution of the equation: P = conv(wrev(w),w)*2, where P is the "Lagrange trous" filter for N = 4. Evaluate P and plot it. P is a symmetric filter and wdb4 is a minimum phase solution of the previous equation based on the roots of P.

```
P = conv(wrev(wdb4),wdb4)*2;
stem(P)
title('''Lagrange trous'' filter')
```



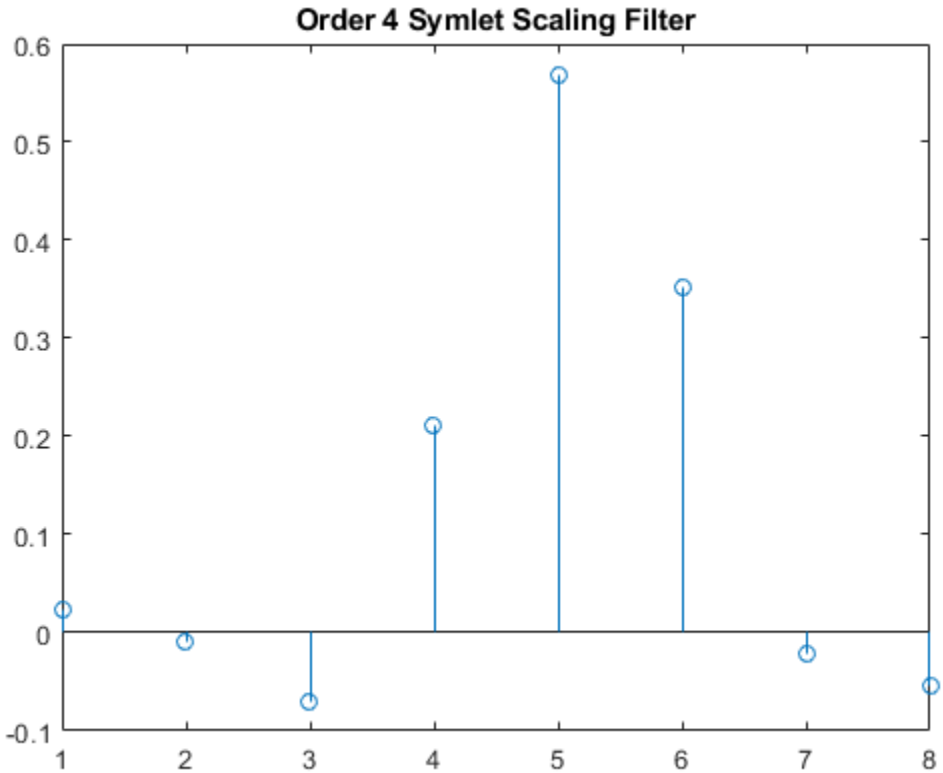
Generate wsym4, the order 4 symlet scaling filter and plot it. The Symlets are the "least asymmetric" Daubechies' wavelets obtained from another choice between the roots of P.

```
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
```

```
stem(wsym4)
title('Order 4 Symlet Scaling Filter')
```



Compute conv(wrev(wsym4),wsym4)*2 and confirm that wsym4 is another solution of the equation P = conv(wrev(w),w)*2.

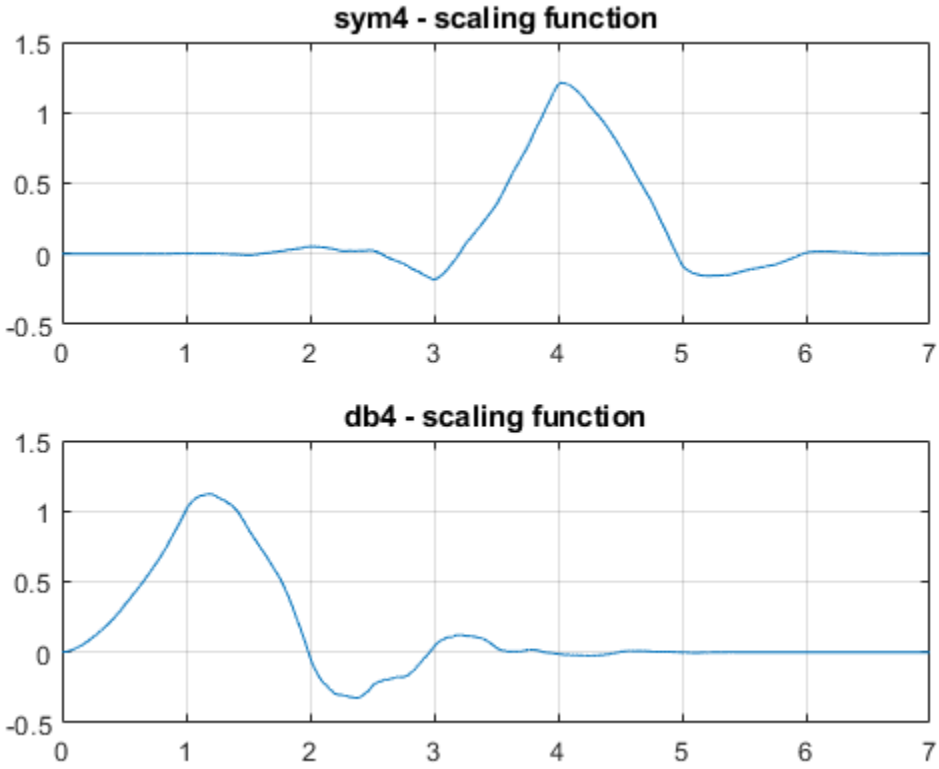
```
P_sym = conv(wrev(wsym4),wsym4)*2;
err = norm(P_sym-P)

err = 9.0732e-16
```

Least Asymmetric Wavelet and Phase

For a given support, the orthogonal wavelet with a phase response that most closely resembles a linear phase filter is called least asymmetric. Symlets are examples of least asymmetric wavelets. They are modified versions of the classic Daubechies db wavelets. In this example you will show that the order 4 symlet has a nearly linear phase response, while the order 4 Daubechies wavelet does not. This example requires the Signal Processing Toolbox.

First plot the order 4 symlet and order 4 Daubechies scaling functions. While neither is perfectly symmetric, note how much more symmetric the symlet is.

```
[phi_sym,~,xval_sym]=wavefun('sym4',10);
[phi_db,~,xval_db]=wavefun('db4',10);
subplot(2,1,1)
plot(xval_sym,phi_sym)
title('sym4 - scaling function')
grid on
subplot(2,1,2)
plot(xval_db,phi_db)
title('db4 - scaling function')
grid on
```



Generate the filters associated with the order 4 symlet and Daubechies wavelets.

```
scal_sym = symaux(4,sqrt(2));
scal_db = dbaux(4,sqrt(2));
```

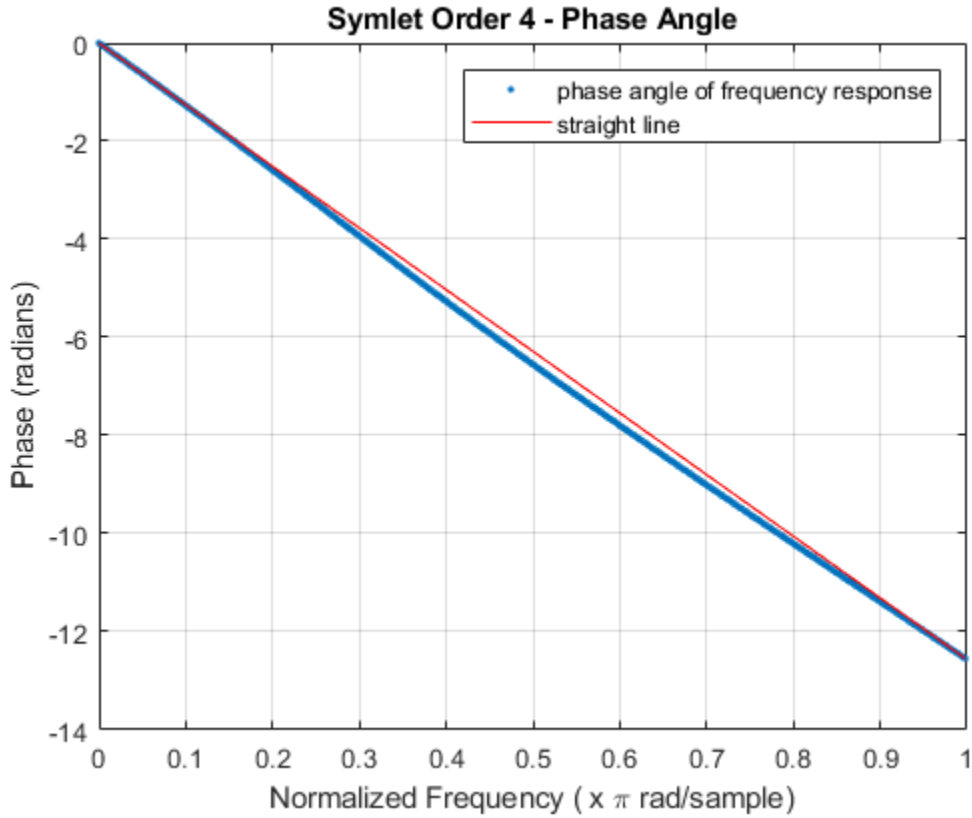
Compute the frequency response of the scaling synthesis filters.

```
[h_sym,w_sym] = freqz(scal_sym);
[h_db,w_db] = freqz(scal_db);
```

To avoid visual discontinuities, unwrap the phase angles of the frequency responses and plot them. Note how well the phase angle of the symlet filter approximates a straightline.

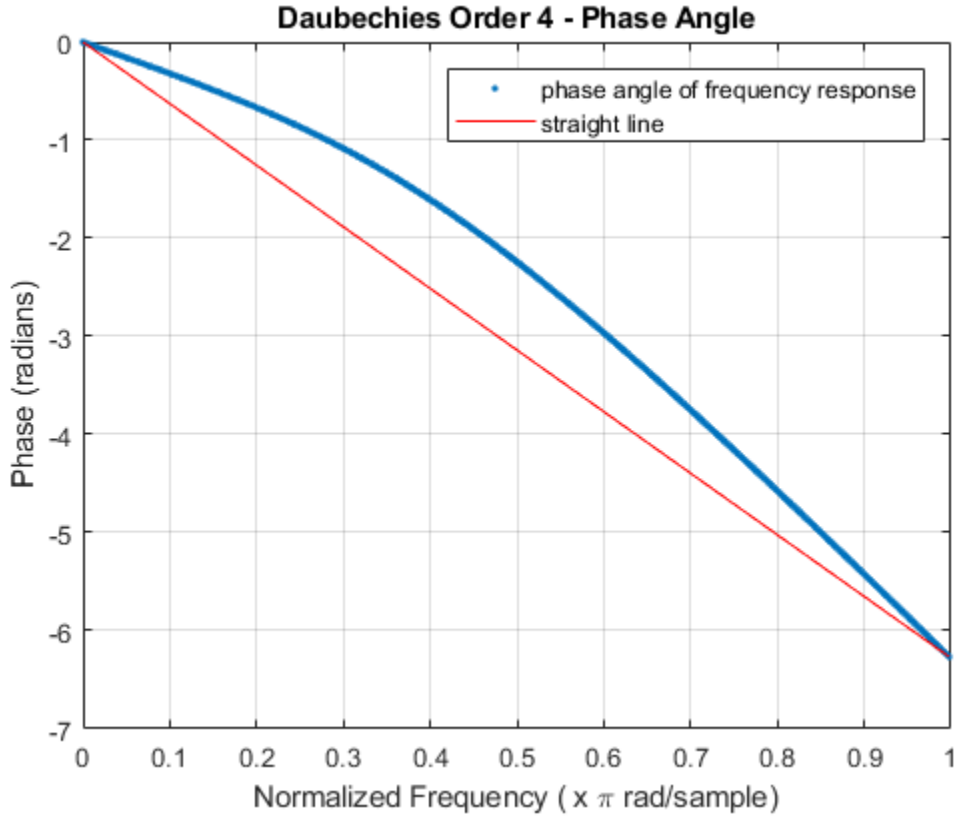
```
h_sym_u = unwrap(angle(h_sym));
h_db_u = unwrap(angle(h_db));
```

```
figure
plot(w_sym/pi,h_sym_u,'.')
hold on
plot(w_sym([1 end])/pi,h_sym_u([1 end]),'r')
grid on
xlabel('Normalized Frequency ( x \pi rad/sample)')
ylabel('Phase (radians)')
legend('phase angle of frequency response','straight line')
title('Symlet Order 4 - Phase Angle')
```




```
figure;
plot(w_db/pi,h_db_u,'.')
hold on
plot(w_db([1 end])/pi,h_db_u([1 end]),'r')
```

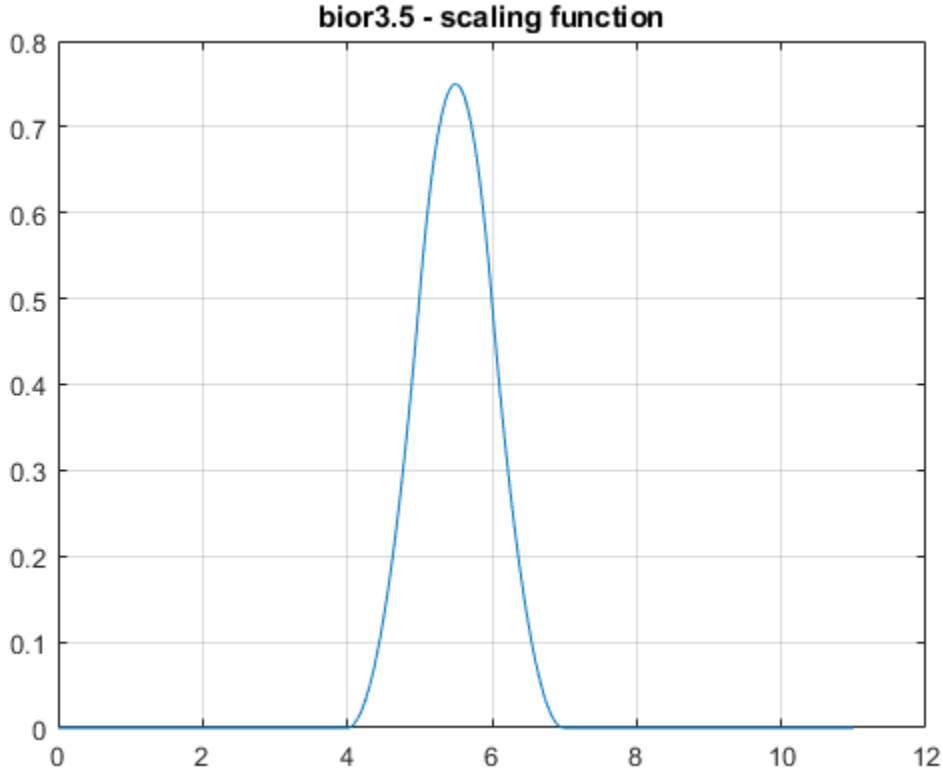
```
grid on
xlabel('Normalized Frequency ( x \pi rad/sample)')
ylabel('Phase (radians)')
legend('phase angle of frequency response','straight line')
title('Daubechies Order 4 - Phase Angle')
```



The sym4 and db4 wavelets are not symmetric, but the biorthogonal wavelet is. Plot the scaling the function associated with the bior3.5 wavelet. Compute the frequency response of the synthesis scaling filter for the wavelet and verify that it has linear phase.

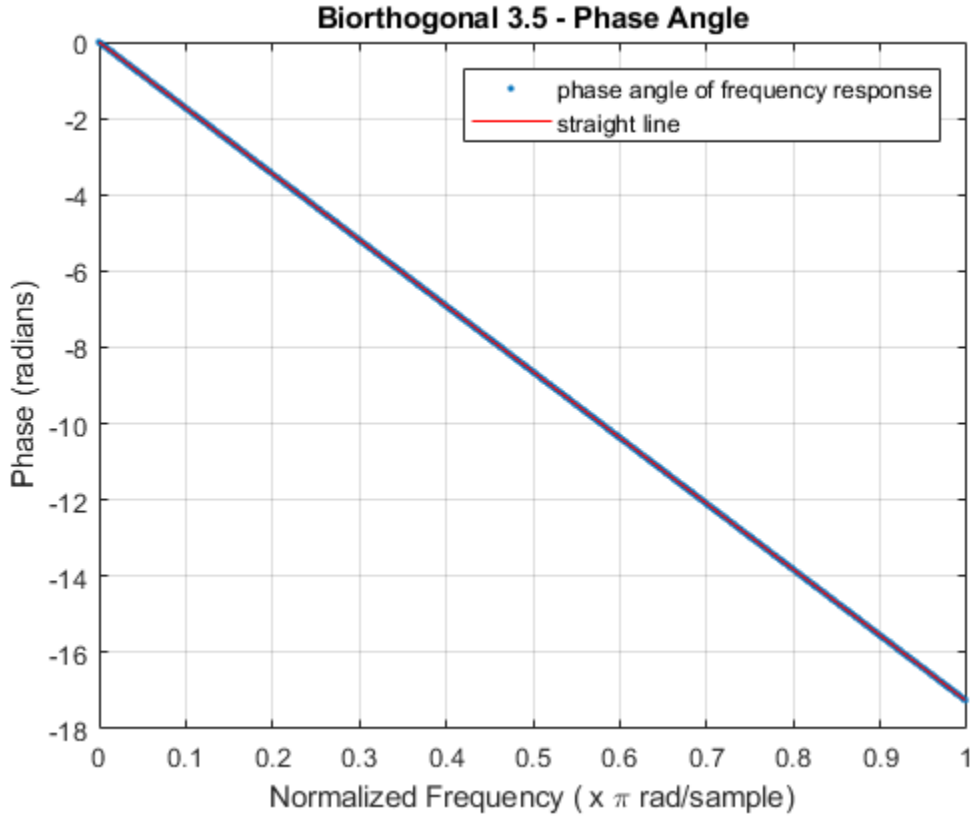
```
[~,~,phi_bior_r,~,xval_bior]=wavefun('bior3.5',10);
figure
plot(xval_bior,phi_bior_r)
```

```
title('bior3.5 - scaling function')
grid on
```




```
[LoD_bior,HiD_bior,LoR_bior,HiR_bior] = wfilters('bior3.5');
[h_bior,w_bior] = freqz(LoR_bior);
h_bior_u = unwrap(angle(h_bior));
figure
plot(w_bior/pi,h_bior_u,'.')
hold on
plot(w_bior([1 end])/pi,h_bior_u([1 end]),'r')
grid on
xlabel('Normalized Frequency ( x \pi rad/sample)')
ylabel('Phase (radians)')
```

```
legend('phase angle of frequency response','straight line')
title('Biorthogonal 3.5 - Phase Angle')
```



Extremal Phase

This example demonstrates that for a given support, the cumulative sum of the squared coefficients of a scaling filter increase more rapidly for an extremal phase wavelet than other wavelets.

First, set the order to 15 and generate the scaling filter coefficients for the Daubechies wavelet and Symlet. Both wavelets have support of length 29.

```
n = 15;
[~,~,LoR_db,~] = wfilters('db15');
[~,~,LoR_sym,~] = wfilters('sym15');
```

Next, generate the scaling filter coefficients for the order 5 Coiflet. This wavelet also has support of length 29.

```
[~,~,LoR_coif,~] = wfilters('coif5');
```

Confirm the sum of the coefficients for all three wavelets equals $\sqrt{2}$.

```
sqrt(2)-sum(LoR_db)
```

```
ans = 2.2204e-16
```

```
sqrt(2)-sum(LoR_sym)
```

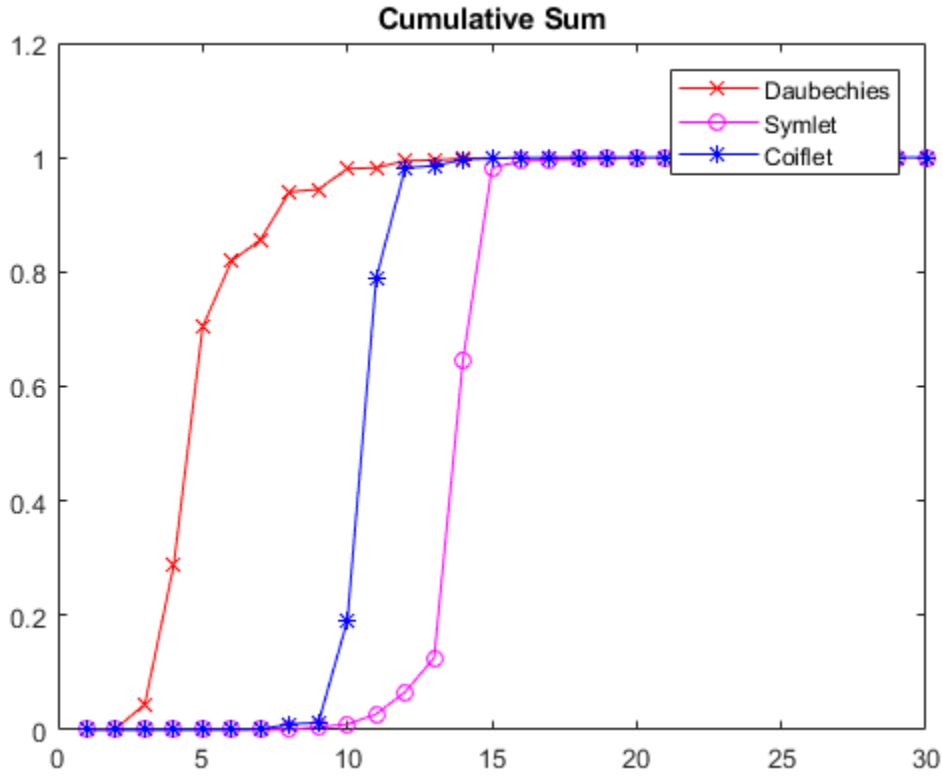
```
ans = 0
```

```
sqrt(2)-sum(LoR_coif)
```

```
ans = 2.2204e-16
```

Plot the cumulative sums of the squared coefficients. Note how rapidly the Daubechies sum increases. This is because its energy is concentrated at small abscissas. Since the Daubechies wavelet has extremal phase, the cumulative sum of its squared coefficients increases more rapidly than the other two wavelets.

```
plot(cumsum(LoR_db.^2),'rx-')
hold on
plot(cumsum(LoR_sym.^2),'mo-')
plot(cumsum(LoR_coif.^2),'b*-')
legend('Daubechies','Symlet','Coiflet')
title('Cumulative Sum')
```



Input Arguments

`n` — Order of symlet
positive integer

Order of the symlet, specified as a positive integer.

`sumw` — Sum of coefficients
1 (default) | positive real number

Sum of the scaling filter coefficients, specified as a positive real number. Set to `sqrt(2)` to generate vector of coefficients whose norm is 1.

Output Arguments

`w` — Filter coefficients

row vector

Vector of scaling filter coefficients of the order `n` symlet.

The scaling filter coefficients satisfy a number of properties. You can use these properties to check your results. See “Unit Norm Scaling Filter Coefficients” on page 1-740 for additional information.

Definitions

Least Asymmetric Wavelet

The Haar wavelet, also known as the Daubechies wavelet of order 1, `db1`, is the only compactly supported orthogonal wavelet that is symmetric, or equivalently has linear phase. No other compactly supported orthogonal wavelet can be symmetric. However, it is possible to derive wavelets which are minimally asymmetric, meaning that their phase will be very nearly linear. For a given support, the orthogonal wavelet with a phase response that most closely resembles a linear phase filter is called least asymmetric.

Constructing a compactly supported orthogonal wavelet basis involves choosing roots of a particular polynomial equation. Different choices of roots will result in wavelets whose phases are different. The example “Least Asymmetric Wavelet and Phase” on page 1-747 compares wavelets with different phases. The example “Symlet and Daubechies Scaling Filters” on page 1-743 shows that two different scaling filters can satisfy the same polynomial equation. For additional information, see Daubechies [1].

Extremal Phase

As mentioned in “Least Asymmetric Wavelet” on page 1-755, when constructing a wavelet, you must choose among a set of roots of a particular equation. Choosing roots that lie within the unit circle in the complex plane results in a filter with highly

nonlinear phase. Such a wavelet is said to have extremal phase, and has energy concentrated at small abscissas. Let $\{h_k\}$ denote the set of scaling coefficients associated with an extremal phase wavelet, where $k = 1,...,N$. Then for any other set of scaling coefficients $\{g_k\}$ resulting from a different choice of roots, the following inequality will hold for all $J = 1,...,N$:

$$\sum_{k=1}^{J} g_k^2 \le \sum_{k=1}^{J} h_k^2$$

The inequality is illustrated in the example "Extremal Phase" on page 1-752. The $\{h_k\}$ are sometimes called a *minimal delay filter* [2].

References

[1] Daubechies, I. (1992), *Ten Lectures on Wavelets*, CBMS-NSF conference series in applied mathematics, SIAM Ed.

[2] Oppenheim, Alan V., and Ronald W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.

See Also

`dbaux` | `symwavf` | `wfilters`

Introduced before R2006a

symwavf

Symlet wavelet filter

Syntax

```
F = symwavf(W)
```

Description

`F = symwavf(W)` returns the scaling filter associated with the symlet wavelet specified by the character vector `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`

Introduced before R2006a

thselect

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 character vector `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 Stein's 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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`wden`

Introduced before R2006a

tnodes

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.

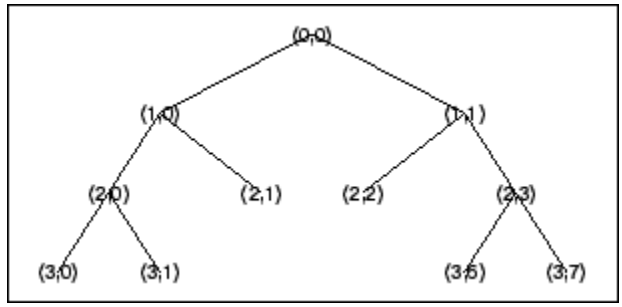
`N = tnodes(T,'deppos')` returns a matrix `N`, which contains the depths and positions of terminal nodes.

`N(i,1)` is the depth of the `i`-th terminal node. `N(i,2)` is the position of the `i`-th terminal node.

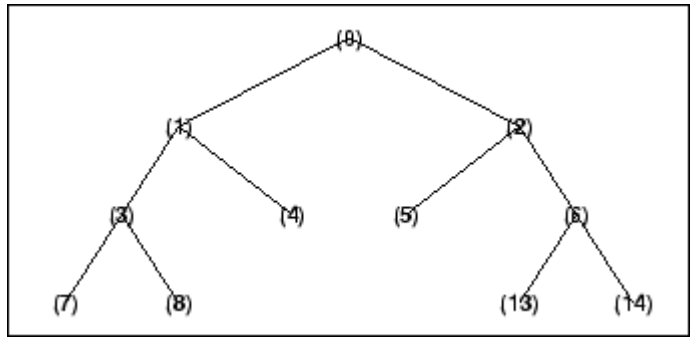
For `[N,K] = tnodes(T)` or `[N,K] = tnodes(T,'deppos'), M = N(K)` are the indices reordered as in tree *T*, from left to right.

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`

Introduced before R2006a

treedpth

Tree depth

Syntax

```
D = treedpth(T)
```

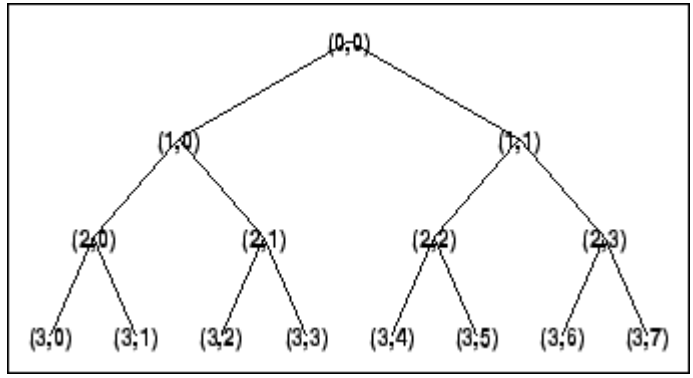
Description

`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`

Introduced before R2006a

treeord

Tree order

Syntax

```
ORD = treeord(T)
```

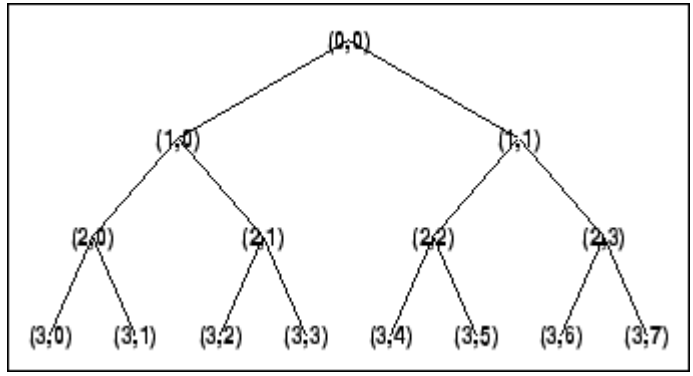
Description

`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`

Introduced before R2006a

upcoef

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 character vector 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.

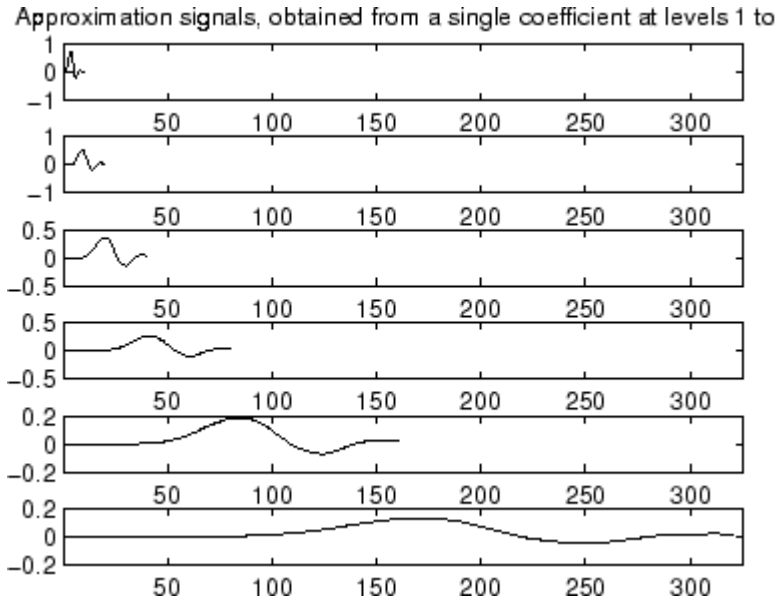
For `Y = upcoef(O,X,Lo_R,Hi_R,N)` or `Y = upcoef(O,X,Lo_R,Hi_R,N,L)`, `Lo_R` is the reconstruction low-pass filter and `Hi_R` is the reconstruction high-pass filter.

`Y = upcoef(O,X,'wname'')` is equivalent to `Y = upcoef(O,X,'wname'',1)`.

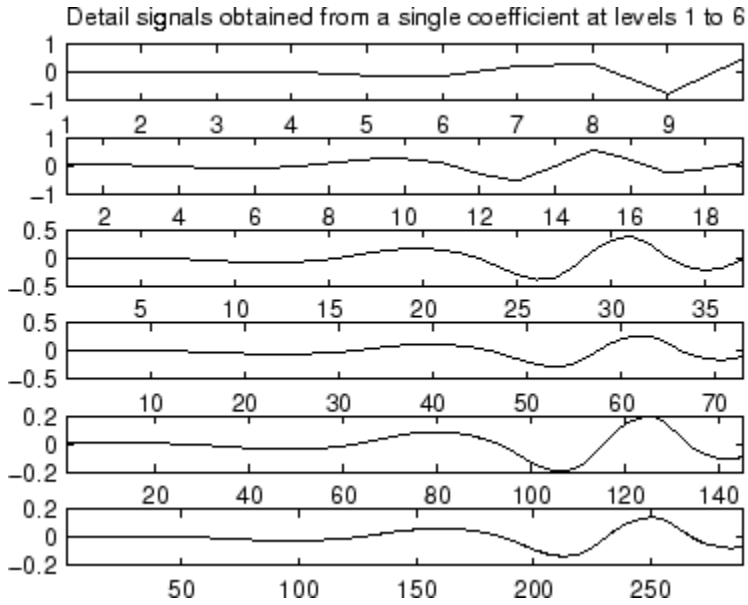
`Y = upcoef(O,X,Lo_R,Hi_R)` is equivalent to `Y = upcoef(O,X,Lo_R,Hi_R,1)`.

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.
```




```
% 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 ' ...
    'coefficient at levels 1 to 6'])
% Editing some graphical properties,
% the following figure is generated.
```



Algorithms

`upcoef` is equivalent to an `N` time repeated use of the inverse wavelet transform.

See Also

`idwt`

Introduced before R2006a

upcoef2

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 character vector 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`

Introduced before R2006a

upwlev

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 character vector containing the wavelet name, `C` is the original wavelet decomposition vector, and `L` the corresponding bookkeeping vector (for detailed storage information, see `wavedec`).

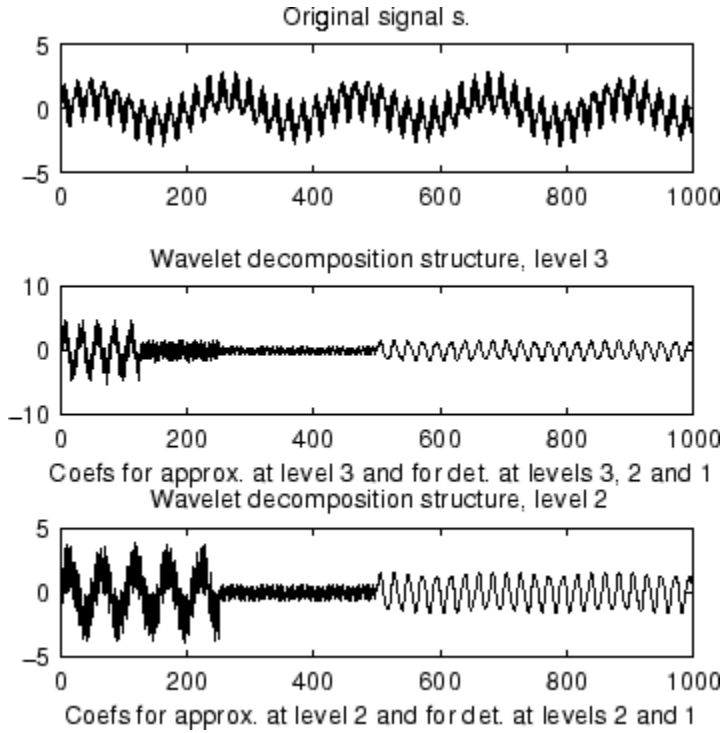
Instead of giving the wavelet name, you can give the filters.

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.
```



See Also

`idwt`

Topics

`upcoef`
`wavedec`

Introduced before R2006a

upwlev2

Single-level reconstruction of 2-D wavelet decomposition

Syntax

```
[NC,NS,cA] = upwlev2(C,S,'wname')
[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 character vector 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

Introduced before R2006a

wave2lp

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`

Introduced before R2006a

wavedec

Multilevel 1-D wavelet decomposition

Syntax

```
[C,L] = wavedec(X,N,'wname')
[C,L] = wavedec(X,N,Lo_D,Hi_D)
```

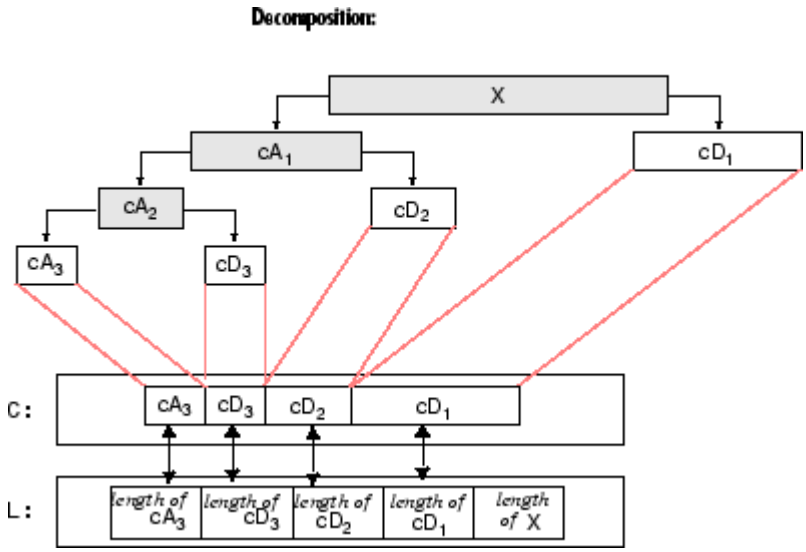
Description

`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'`. `wavedec` does not enforce a maximum level restriction. Use `wmaxlev` to ensure the wavelet coefficients are free from boundary effects. If boundary effects are not a concern in your application, a good rule is to set `N` less than or equal to `fix(log2(length(X)))`.

The output decomposition structure contains the wavelet decomposition vector `C` and the bookkeeping vector `L`, which contains the number of coefficients by level. The structure is organized as in this level-3 decomposition example.



`[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

Perform Multilevel One-Dimensional Wavelet Analysis

The current extension mode for this example is zero-padding, as specified using the `dwtmode` function.

Load original one-dimensional signal.

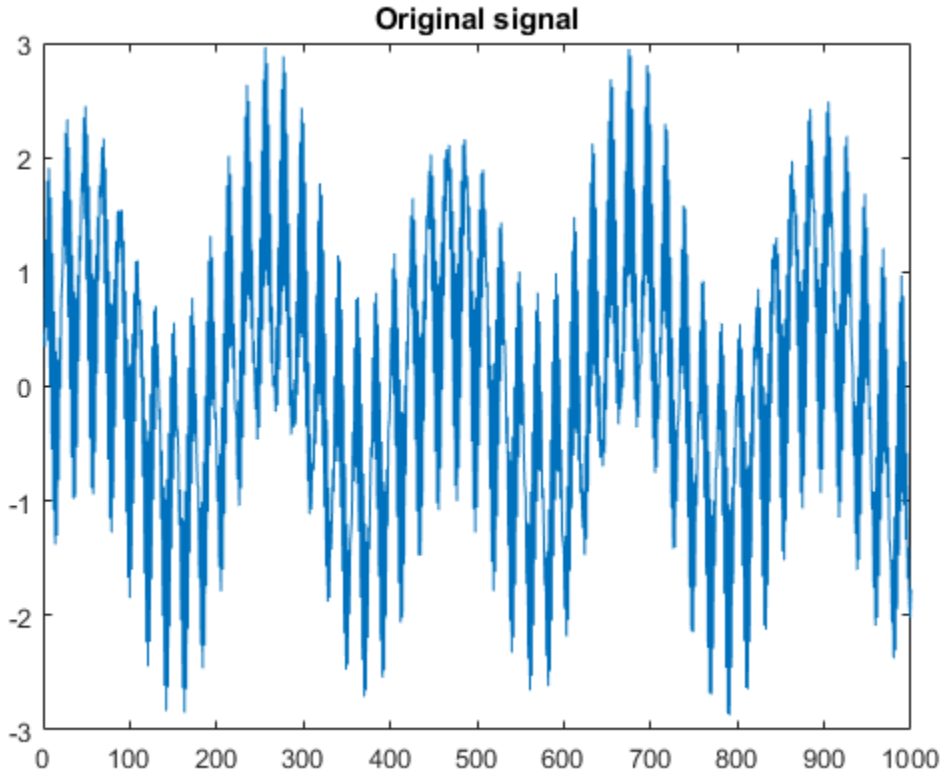
```
load sumsin;
s = sumsin;
```

Perform decomposition at level 3 of s using `db1`. Extract the detail coefficients at levels 1, 2, and 3 from the composition structure.

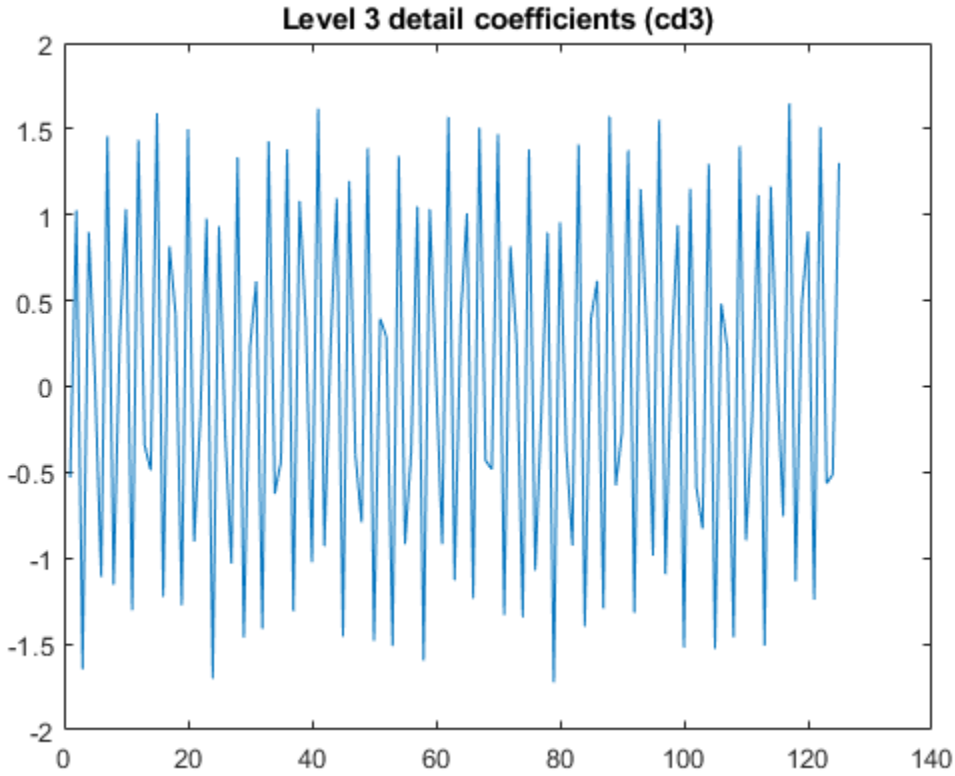
```
[c,l] = wavedec(s,3,'db1');
[cd1,cd2,cd3] = detcoef(c,l,[1 2 3]);
```

Plot the output of the decomposition.

```
plot(s)
title('Original signal')
```



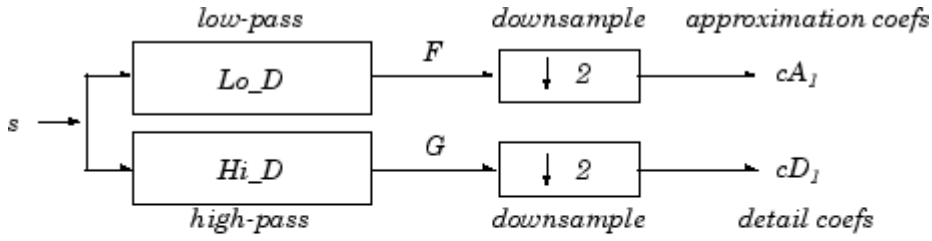

```
plot(cd3)
title('Level 3 detail coefficients (cd3)')
```



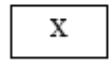
Algorithms

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).

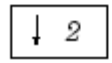
More precisely, the first step is



where



Convolve with filter X



Keep the even indexed elements
(We call this operation *downsampling*.)

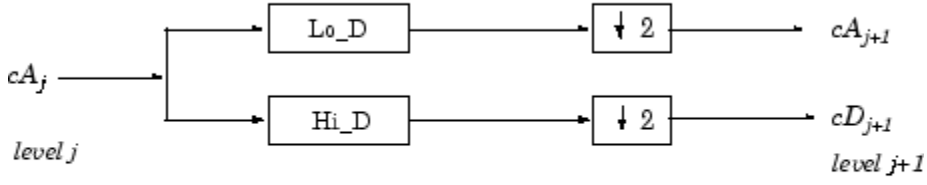
The length of each filter is equal to $2N$. If n = length(s), the signals F and G are of length $n + 2N - 1$ and the coefficients cA_1 and cD_1 are of length

$$\text{floor}\left(\frac{n-1}{2}\right)+N$$

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



where

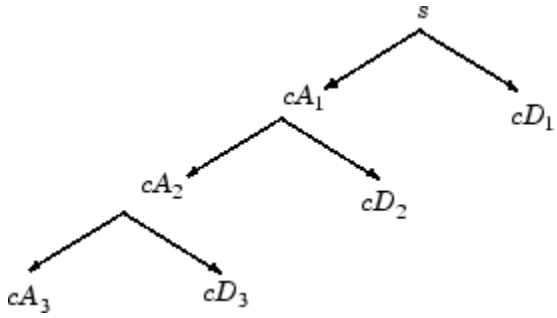
Convolve with filter X

Downsample

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.)

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Variable-size data support must be enabled.
- The input `'wname'` must be constant.

See Also

`dwt` | `waveinfo` | `waverec` | `wfilters` | `wmaxlev`

Introduced before R2006a

wavedec2

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 character vector `'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.

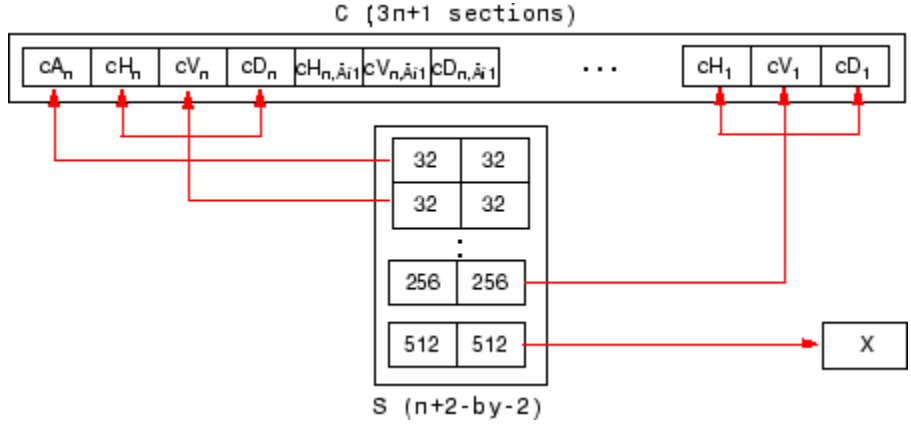
For `[C,S] = wavedec2(X,N,Lo_D,Hi_D)`, `Lo_D` is the decomposition low-pass filter and `Hi_D` is the decomposition high-pass filter.

Vector `C` is organized as a vector with 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`, and `D` are each a row vector. Each vector is the vector column-wise storage of a matrix.

- `A` contains the approximation coefficients
- `H` contains the horizontal detail coefficients
- `V` contains the vertical detail coefficients
- `D` contains the diagonal detail coefficients
-

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)`.



Examples

Decomposition Structure

This example shows the structure of `wavedec2` output matrices.

Load original image from the `woman.mat` file, which contains variables named `X` and `map`.

```
load woman;
```

Get current discrete wavelet transform extension mode.

```
origMode = dwtmode('status','nodisplay');
```

Change to periodic boundary handling.

```
dwtmode('per');
```

```
****************************************
```

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

Perform decomposition at level 2 of `X` using db1.

```
[c,s] = wavedec2(X,2,'db1');
```

Get the decomposition structure organization.

```
sizex = size(X)
```

```
sizex =

   256   256
```

```
sizec = size(c)
```

```
sizec =

           1       65536
```

Reset discrete wavelet transform extension mode to its original mode.

```
dwtmode(origMode);
```

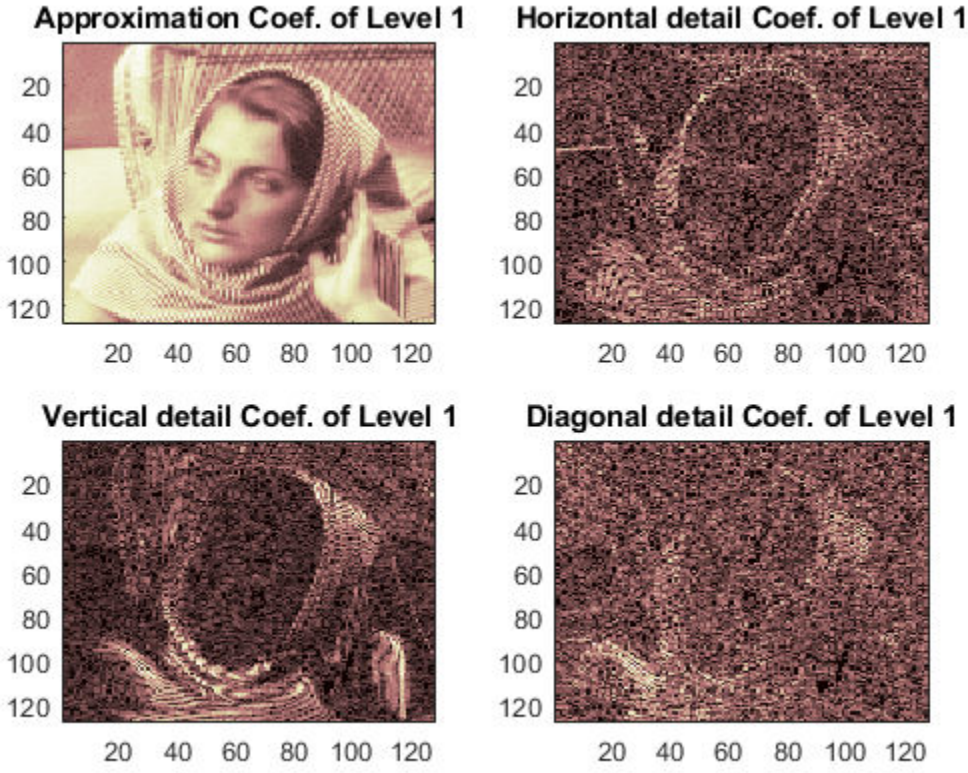
```
****************************************
**  DWT Extension Mode: Periodization  **
****************************************
```

Extract and Display Image Decomposition Level

Extract and display images of wavelet decomposition level details. The resulting images are similar to the visualizations in the *At level 2, with haar --> woman* indexed image example in the Wavelet 2-D interactive tool. Use `waveletAnalyzer` to launch this tool.

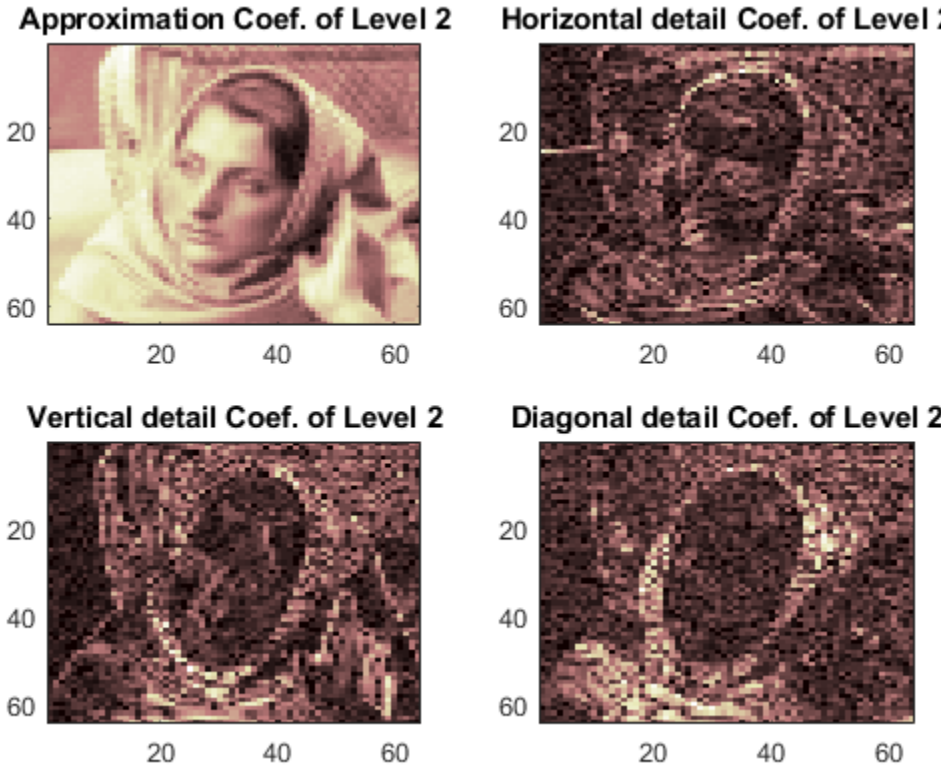
```
load woman;
[c,s]=wavedec2(X,2,'haar');

[H1,V1,D1] = detcoef2('all',c,s,1);
A1 = appcoef2(c,s,'haar',1);
```

```
V1img = wcodemat(V1,255,'mat',1);
H1img = wcodemat(H1,255,'mat',1);
D1img = wcodemat(D1,255,'mat',1);
A1img = wcodemat(A1,255,'mat',1);

[H2,V2,D2] = detcoef2('all',c,s,2);
A2 = appcoef2(c,s,'haar',2);
V2img = wcodemat(V2,255,'mat',1);
H2img = wcodemat(H2,255,'mat',1);
D2img = wcodemat(D2,255,'mat',1);
A2img = wcodemat(A2,255,'mat',1);

subplot(2,2,1);
imagesc(A1img);
colormap pink(255);
title('Approximation Coef. of Level 1');

subplot(2,2,2);
imagesc(H1img);
title('Horizontal detail Coef. of Level 1');

subplot(2,2,3);
imagesc(V1img);
title('Vertical detail Coef. of Level 1');

subplot(2,2,4);
imagesc(D1img);
title('Diagonal detail Coef. of Level 1');
```




```
figure;
subplot(2,2,1);
imagesc(A2img);
colormap pink(255);
title('Approximation Coef. of Level 2');

subplot(2,2,2)
imagesc(H2img);
title('Horizontal detail Coef. of Level 2');

subplot(2,2,3)
imagesc(V2img);
title('Vertical detail Coef. of Level 2');
```

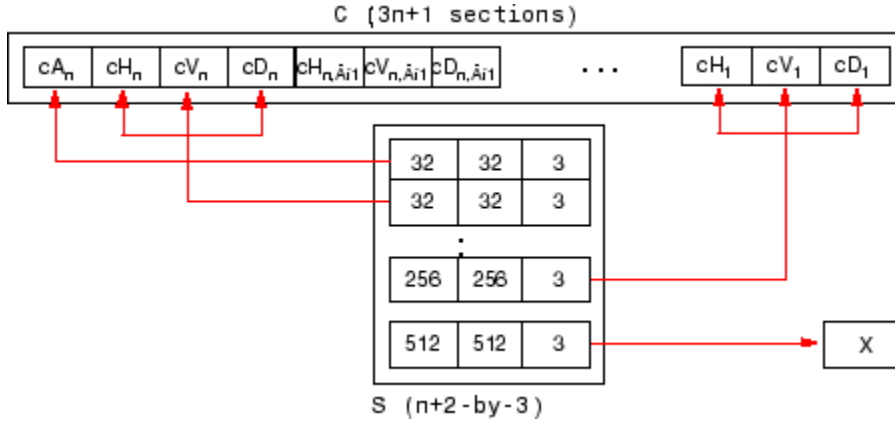
```
subplot(2,2,4)
imagesc(D2img);
title('Diagonal detail Coef. of Level 2');
```



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.



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

Algorithms

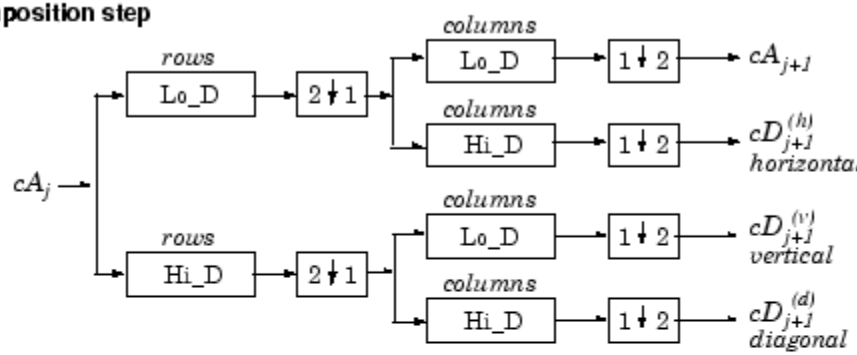
For images, an algorithm similar to the one-dimensional case is possible for two-dimensional wavelets and scaling functions obtained from one-dimensional ones by tensor product.

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

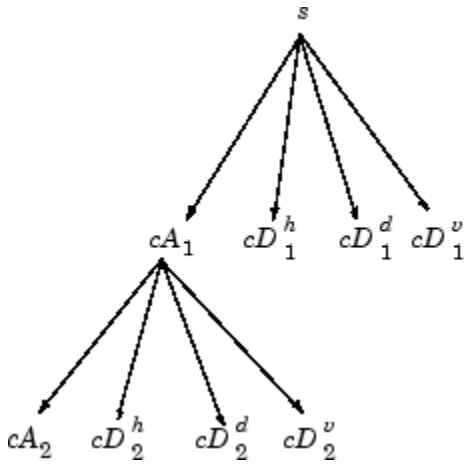
Decomposition step



where
- $2\downarrow 1$ Downsample columns: keep the even indexed columns
- $1\downarrow 2$ 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

So, for J=2, the two-dimensional wavelet tree has the form



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.)

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Variable-size data support must be enabled.
- The input `'wname'` must be constant.

See Also

`dwt` | `waveinfo` | `waverec2` | `wfilters` | `wmaxlev`

Introduced before R2006a

wavedec3

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 character vector `'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 the 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 character vectors 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 of integers 1 to 64 and make the data 3-D.

```
M = magic(8);
X = repmat(M,[1 1 8]);
```

Obtain the 3-D discrete wavelet transform at level 1 using the Haar wavelet and the default whole-point symmetric extension mode.

```
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 of integers 1 to 64 and make the data 3-D.

```
M = magic(8);
X = repmat(M,[1 1 8]);
```

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.

```
[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}(:))))

ans = 0

max(abs((wdec.dec{5}(:)-dwtOut.dec{1,1,2}(:))))

ans = 0
```

See Also

dwt3 | dwtmode | waveinfo | waverec3 | wfilters | wmaxlev

Introduced in R2010a

wavefun

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,

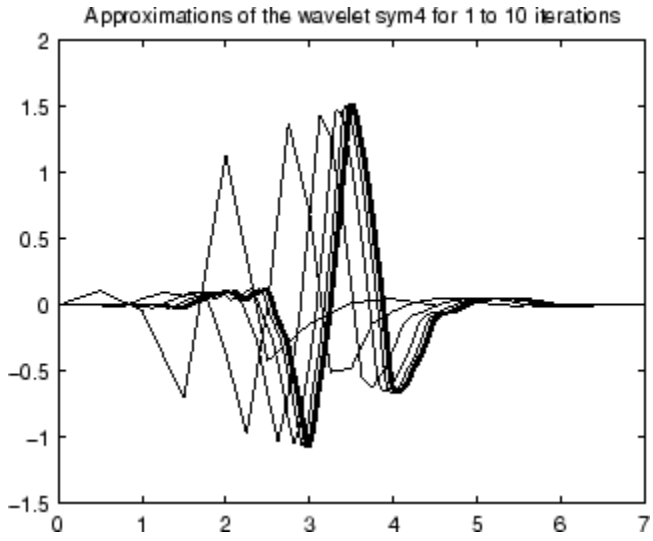
- `[...] = wavefun('wname',0)` is equivalent to
- `[...] = wavefun('wname',8,0)`.
- `[...] = wavefun('wname')` is equivalent to
- `[...] = wavefun('wname',8)`.

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
```



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 ϕ.

Since ϕ is also equal to $\phi_{0,0}$, this function is characterized by the following coefficients in the orthogonal framework:

- $<\phi, \phi_{0,n}> = 1$ only if $n = 0$ and equal to 0 otherwise
- $<\phi, \psi_{-j,k}> = 0$ 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 ϕ 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

$$\left|\phi(x) - 2^{\frac{j}{2}}\langle \phi, \phi_{-j,\, n2^{j-J}}\rangle\right| \le C.2^{-j\alpha}$$

where C is a constant, and α is a positive constant depending on the wavelet regularity.

Then using a good approximation of ϕ on dyadic rationals, we can use piecewise constant or piecewise linear interpolations η on dyadic intervals, for which uniform convergence occurs with similar exponential rate:

$$\|\phi - \eta\|_{\infty} \le C.2^{-j\alpha}$$

So using a J-step reconstruction scheme, we obtain an approximation that converges exponentially towards ϕ 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 ψ can also be expanded on the $(\phi_{-1,n)})_n$, 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`

Introduced before R2006a

wavefun2

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^{ITER} x 2^{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)`.

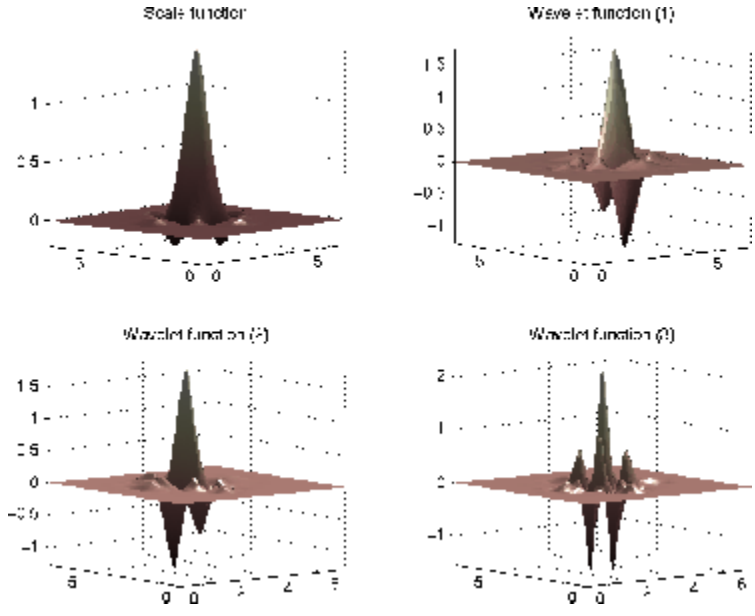
The output arguments are optional.

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);
```



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`

Introduced before R2006a

waveinfo

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 character vector `'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
`'fk'`	Fejer-Korovkin filters
`'rbio'`	Reverse biorthogonal wavelets
`'meyr'`	Meyer wavelet
`'dmey'`	Discrete approximation of Meyer wavelet
`'gaus'`	Gaussian wavelets
`'mexh'`	Mexican hat wavelet (also known as Ricker wavelet)
`'morl'`	Morlet wavelet
`'cgau'`	Complex Gaussian wavelets

Wavelet Family Short Name	Wavelet Family Name
'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`

Introduced before R2006a

Wavelet Analyzer

Analyze signals and images using wavelets

Description

The **Wavelet Analyzer** app is an interactive tool for using wavelets to visualize and analyze signals and images. Using this app, you can:

- Perform wavelet and wavelet packet analysis
- Denoise and compress signals and images
- Design custom wavelets
- Estimate density and regression
- Perform matching pursuit analysis
- Perform image fusion

Open the Wavelet Analyzer App

- MATLAB Toolstrip: On the **Apps** tab, under **Signal Processing and Communications**, click the app icon.
- MATLAB command prompt: Enter `waveletAnalyzer`

Examples

- "Continuous Wavelet Analysis of Noisy Sinusoid Using the Wavelet Analyzer App"
- "DFT-Based Continuous Wavelet Analysis Using the Wavelet Analyzer App"
- "Interactive 1-D Stationary Wavelet Transform Denoising"
- "Matching Pursuit Using Wavelet Analyzer App"

Definitions

Boundary Conditions

To change the way **Wavelet Analyzer** handles boundary conditions, use the `dwtmode` function.

See Also

Topics

"Continuous Wavelet Analysis of Noisy Sinusoid Using the Wavelet Analyzer App"
"DFT-Based Continuous Wavelet Analysis Using the Wavelet Analyzer App"
"Interactive 1-D Stationary Wavelet Transform Denoising"
"Matching Pursuit Using Wavelet Analyzer App"
"Continuous Wavelet Analysis"
"Discrete Wavelet Analysis"

Introduced before R2006a

waveletfamilies

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 **
character vector
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 **
character vector
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 **
character vector
cmorwavf
-8 8
------------------------
```

See Also

wavemngr

Introduced in R2008a

Wavelet Signal Denoiser

Visualize and denoise time series data

Description

The **Wavelet Signal Denoiser** app is an interactive tool for visualizing and denoising real-valued 1-D signals and comparing results. With the app, you can:

- Access all the signals in the MATLAB workspace.
- Easily adjust default parameters and apply different denoising techniques.
- Visualize and compare results.
- Export denoised signals to your workspace.
- Recreate the denoised signal in your workspace by generating a MATLAB script.

The **Wavelet Signal Denoiser** app provides a way to work with multiple versions of denoised data simultaneously.

A typical workflow for denoising a signal and comparing results using the app is:

1. Start the app and load a 1-D signal from the MATLAB workspace. The app provides an initial denoised version of your data using default parameters.
2. Adjust the denoising parameters and produce multiple versions of the denoised signal.
3. Compare results and export the desired denoised signal to your workspace.
4. To apply the same denoising parameters to other signals in your workspace, generate a MATLAB script and modify it as you see fit.

Open the Wavelet Signal Denoiser App

- MATLAB Toolstrip: On the **Apps** tab, under **Signal Processing and Communications**, click **Wavelet Signal Denoiser**  .
- MATLAB command prompt: Enter `waveletSignalDenoiser`.

Examples

- "Denoise a Signal with the Wavelet Signal Denoiser"

Parameters

`Wavelet` — Wavelet family

`sym` (default) | `bior` | `coif` | `db` | `fk`

Wavelet family used to denoise the signal, specified as one of the following:

- `sym` — Symlets
- `bior` — Biorthogonal spline wavelets
- `coif` — Coiflets
- `db` — Daubechies wavelets
- `fk` — Fejér-Korovkin wavelets

`Method` — Denoising method

`Bayes` (default) | `BlockJS` | `FDR` | `Minimax` | `SURE` | `UniversalThreshold`

Denoising method to apply, specified as one of the following:

- `Bayes` — Empirical Bayes
- `BlockJS` — Block James-Stein
- `FDR` — False Discovery Rate
- `Minimax` — Minimax Estimation
- `SURE` — Stein's Unbiased Risk Estimate
- `UniversalThreshold` — Universal Threshold $\sqrt{2\ln(\bullet)}$

`Rule` — Thresholding rule

`Median` (default) | `Mean` | `Soft` | `Hard`

Thresholding rule to use. The possible rules you can specify for different denoising methods are as follows:

- Block James-Stein — `James-Stein`
- Empirical Bayes — `Median`, `Mean`, `Soft`, `Hard`
- False Discovery Rate — `Hard`
- Minimax Estimation — `Soft`, `Hard`
- Stein's Unbiased Risk Estimate — `Soft`, `Hard`
- Universal Threshold —`Soft`, `Hard`

Tips

- To simultaneously denoise more than one signal, you can run multiple instances of the Wavelet Signal Denoiser.

See Also

Functions

`wdenoise`

Topics

“Denoise a Signal with the Wavelet Signal Denoiser”

Introduced in R2017b

wavemngr

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 (character vector)

`FSN` = Family Short Name (character vector 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 = ''`. Note that for this case you specify an empty character vector.

Examples:

`mexh`	`''`
`morl`	`''`

If the wavelet is member of a finite family of wavelets, `NUMS` is a character vector containing a space-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 character vector containing a space-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 `'charactervector'`; the default value is `'integer'`.

Examples:

`db`	`TYPNUMS = 'integer'`
`bior`	`TYPNUMS = 'real'`
`shan`	`TYPNUMS = 'charactervector'`

`FILE` = MAT-file or code file name (character vector). 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.

`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 character vector.

`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

Wavelet Names and Family Names

List the wavelet families available by default.

```
wavemngr('read')
```

```
ans =

  18x35 char array

    '==================================='
    '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      '
    'Fejer-Korovkin            fk        '
    '==================================='
```

List all wavelets.

```
wavemngr('read',1)
```

```
ans =

  71x44 char array

    '==================================          '
    '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                        '
'===================================          '
'Mexican_hat               mexh                  '
'===================================          '
'Morlet                    morl                  '
'===================================          '
'Complex Gaussian          cgau                  '
'-------------------------------              '
'cgau1    cgau2    cgau3    cgau4                        '
'cgau5    cgau6    cgau7    cgau8                        '
'===================================          '
'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**                      '
'===================================          '
'Fejer-Korovkin            fk                    '
'-------------------------------              '
'fk4    fk6    fk8    fk14                              '
'fk18    fk22                                    '
'===================================          '
```

Add Wavelet Families

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.

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 character vector of the form `sh.name` + `number` (for example, `'lem1'` or `'lem2'`) 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.

Note that `wavemngr` works on the current folder. If you add a new wavelet family, it is available in this folder only.

List the available wavelet 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. Regenerate the list of wavelet families.

```
wavemngr('del','Lemarie');
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'`. List the restored wavelets.

```
wavemngr('restore');
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 the initial wavelets. Restore the initial ASCII file `'wavelets.ini'` and the initial MAT-file `'wavelets.bin'`. Regenerate the list of wavelet families.

```
wavemngr('restore',0);
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
===================================
```

All command line capabilities are available for new families of wavelets. Create a new family. Compute the four associated filters and the scale and wavelet functions.

```
wavemngr('add','Lemarie','lem',1,'1 2 3','lemwavf');
[Lo_D,Hi_D,Lo_R,Hi_R] = wfilters('lem3');
[phi,psi,xval] = wavefun('lem3');
```

Add a new family of orthogonal wavelets in a form specialized 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',0);
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 shortcut in the order definition only if the possible orders are integers.

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.

Introduced before R2006a

wavenames

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')`.

Introduced before R2006a

waverec

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
```

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Variable-size data support must be enabled.

See Also

`appcoef` | `idwt` | `wavedec`

Introduced before R2006a

waverec2

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 character vector 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)`.

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
```

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Variable-size data support must be enabled.

See Also

`appcoef2` | `idwt2` | `wavedec2`

Introduced before R2006a

waverec3

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`, and `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 character vectors 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(:)))
```

```
err1 = 8.6057e-11
```

Verify that the data matrix is the sum of the approximation and the details from levels 2 and 1. Reconstruct the sum of components different from the lowpass component and check that `X = A + D`.

```
A = waverec3(wd,'LLL');
D = waverec3(wd,'d');
err2 = max(abs(X(:)-A(:)-D(:)))
```

```
err2 = 8.6054e-11
```

Compare waverec3 and idwt3

Compare level-1 reconstructions based on the filtering operations `'LLH'` using `idwt3` and `waverec3`.

```
M = magic(8);
X = repmat(M,[1 1 8]);
wd = wavedec3(X,2,'db2','mode','per');
dwtOut = dwt3(X,'db2');
Xr = idwt3(dwtOut,'LLH');
Xrec = waverec3(wd,'LLH',1);
norm(Xr(:)-Xrec(:))
```

```
ans = 2.7511e-14
```

See Also

`idwt3` | `wavedec3` | `waveinfo`

Introduced in R2010a

wavsupport

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

Return the lower bound and upper bound of the support for the Haar wavelet.

```
[LB, UB] = wavsupport('haar')
```

```
LB = -0.5000
```

```
UB = 0.5000
```

Compare `LB` and `UB` to the lower and upper bounds for orthogonal and biorthogonal wavelets (type 1 and type 2).

```
LowerBound = -0.5*(2-1);
UpperBound = 0.5*(2-1);
```

Support of Complex-Valued Gaussian Wavelet

Return the lower bound and upper bound of the support for the complex-valued Gaussia wavelet.

```
[LB,UB] = wavsupport('cgau3')
```

```
LB = -5
```

```
UB = 5
```

See Also

`wavemngr`

Introduced in R2010b

wbmpen

Penalized threshold for wavelet 1-D or 2-D de-noising

Syntax

```
THR = wbmpen(C,L,SIGMA,ALPHA)
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^*)|.

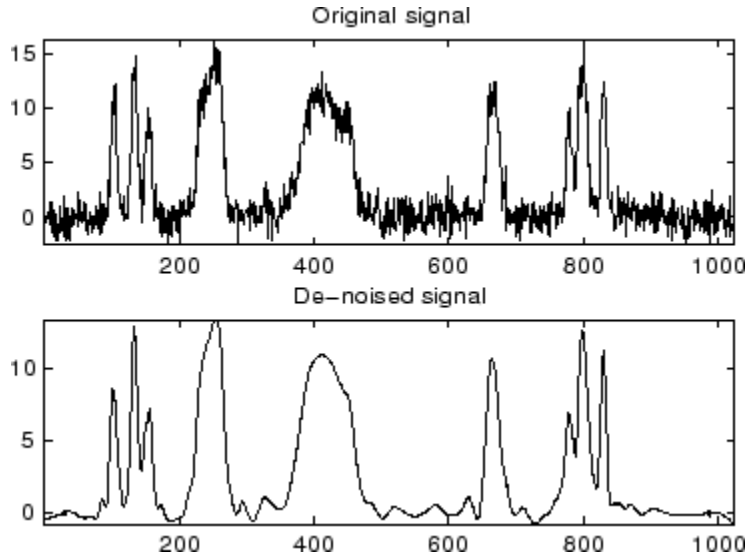
`wbmpen(C,L,SIGMA,ALPHA,ARG)` computes the global threshold and, in addition, plots three curves:

- `2*SIGMA^2*t*(ALPHA + log(n/t))`

- sum(c(k)^2,k¬≤t)
- crit(t)

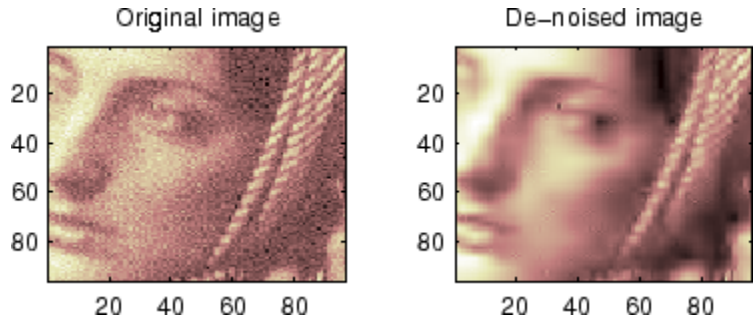
Examples

```
% 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')
```




```
% 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

Introduced before R2006a

wcodemat

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 character vectors 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

Extended Pseudocolor Matrix Scaling

Scale level-one approximation coefficients globally to the full range of the colormap.

Load an image.

```
load woman;
```

Get the range of the colormap.

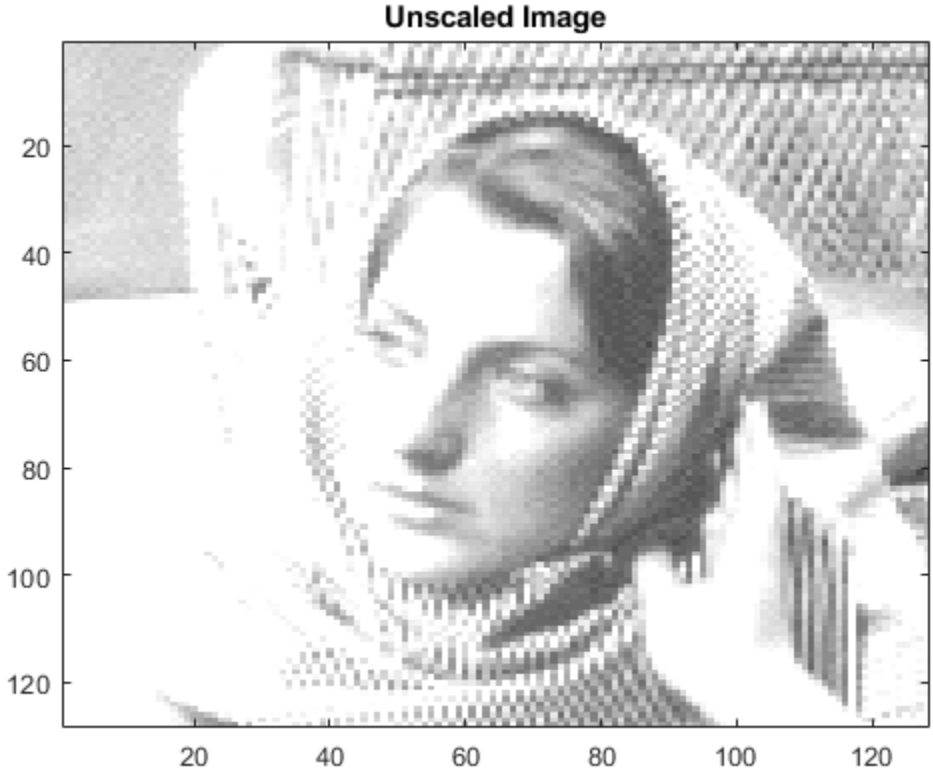
```
NBCOL = size(map,1);
```

Obtain the 2D dwt using the Haar wavelet.

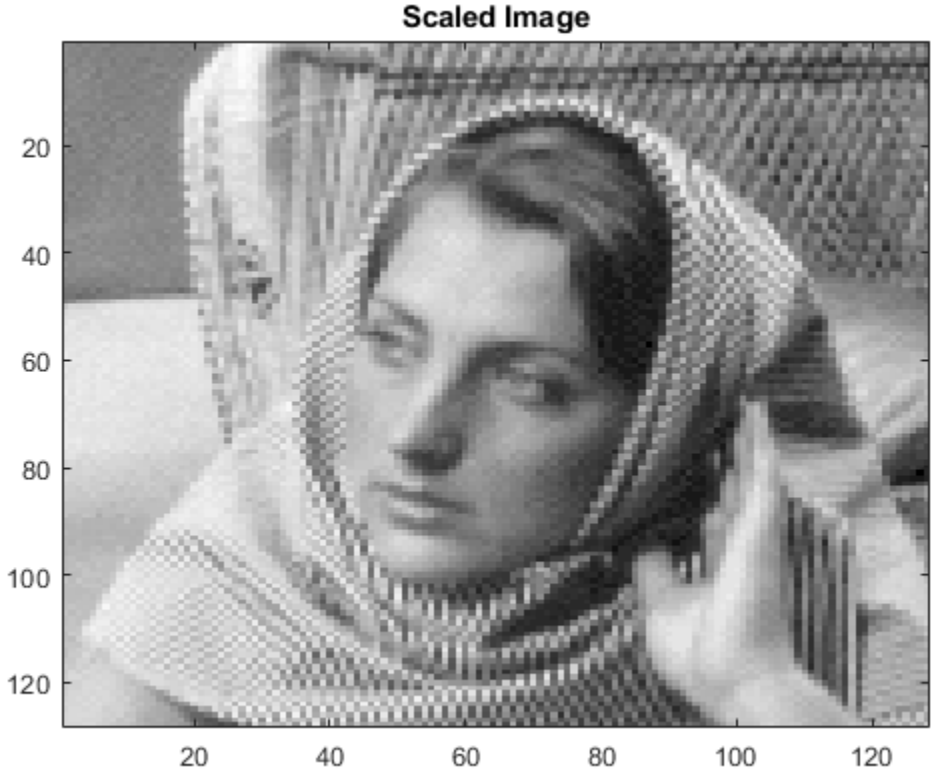
```
[cA1,cH1,cV1,cD1] = dwt2(X,'db1');
```

Display without scaling and with scaling.

```
image(cA1);
colormap(map);
title('Unscaled Image');
```




```
figure
image(wcodemat(cA1,NBCOL));
colormap(map);
title('Scaled Image');
```



Introduced before R2006a

wcoher

(Not recommended) Wavelet coherence

Note `wcoher` in not recommended. Use `wcoherence` instead.

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 character vectors:

- `'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.

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);
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');
```

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:

$$\frac{\left|S(C_x^*(a,b)C_y(a,b))\right|^2}{S(|C_x(a,b)|^2)\ S(|C_y(a,b)|^2)}$$

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.

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` | `wcoherence`

Topics

“1-D Continuous Wavelet Analysis”

"Continuous Wavelet Analysis of Cusp Signal"
Wavelet Coherence

Introduced in R2010b

wcoherence

Wavelet coherence and cross-spectrum

Syntax

```
wcoh = wcoherence(x,y)
[wcoh,wcs] = wcoherence(x,y)
[wcoh,wcs,period] = wcoherence(x,y,ts)
[wcoh,wcs,f] = wcoherence(x,y,fs)
[wcoh,wcs,f,coi] = wcoherence(___)
[wcoh,wcs,period,coi] = wcoherence(___,ts)
wcoherence(___)
```

Description

`wcoh = wcoherence(x,y)` returns the magnitude-squared wavelet coherence, which is a measure of the correlation between signals `x` and `y` in the time-frequency plane. Wavelet coherence is useful for analyzing nonstationary signals. The inputs `x` and `y` must be equal length, 1-D, real-valued signals. The coherence is computed using the analytic Morlet wavelet.

`[wcoh,wcs] = wcoherence(x,y)` returns the wavelet cross-spectrum of `x` and `y`. You can use the phase of the wavelet cross-spectrum values to identify the relative lag between the input signals.

`[wcoh,wcs,period] = wcoherence(x,y,ts)` uses a `duration` `ts` with a positive, scalar input, as the sampling interval. The duration can be in years, days, hours, minutes, or seconds. `ts` is used to compute the scale-to period conversion, `period`. The `period` in an array of durations with the same time unit as specified in `ts`

`[wcoh,wcs,f] = wcoherence(x,y,fs)` uses the positive sampling frequency, `fs`, to compute the scale-to-frequency conversion, `f`. The `fs` input is in Hz.

`[wcoh,wcs,f,coi] = wcoherence(___)` returns the cone of influence, `coi`, for the wavelet coherence in cycles per sample. If you specify the sampling frequency, `fs`, the function returns both `f` and the cone of influence in Hz.

`[wcoh,wcs,period,coi] = wcoherence(___,ts)` returns the cone of influence, `coi`, in cycles per unit time.

`wcoherence(___)` with no output arguments plots the wavelet coherence and cone of influence in the current figure. Due to the inverse relationship between frequency and period, a plot that uses the sampling interval is the inverse of a plot the uses the sampling frequency. For areas where the coherence exceeds 0.5, plots that use the sampling frequency display arrows to show the phase lag of `y` with respect to `x`. The arrows are spaced in time and scale. The direction of the arrows corresponds to the phase lag on the unit circle. For example, a vertical arrow indicates a π/2 or quarter-cycle phase lag. The corresponding lag in time depends on the duration of the cycle.

Examples

Wavelet Coherence of Two Sine Waves

Use default `wcoherence` settings to obtain the wavelet coherence between a sine wave with random noise and a frequency-modulated signal with decreasing frequency over time.

```
t  = linspace(0,1,1024);
x = -sin(8*pi*t) + 0.4*randn(1,1024);
x = x/max(abs(x));
y = wnoise('doppler',10);
wcoh = wcoherence(x,y);
```

The default coherence computation uses the Morlet wavelet, 12 voices per octave and smooths 12 scales. The default number of octaves is equal to `floor(log2(numel(x)))-1`, which in this case is 9.

Effect of Sampling Interval on Wavelet Coherence

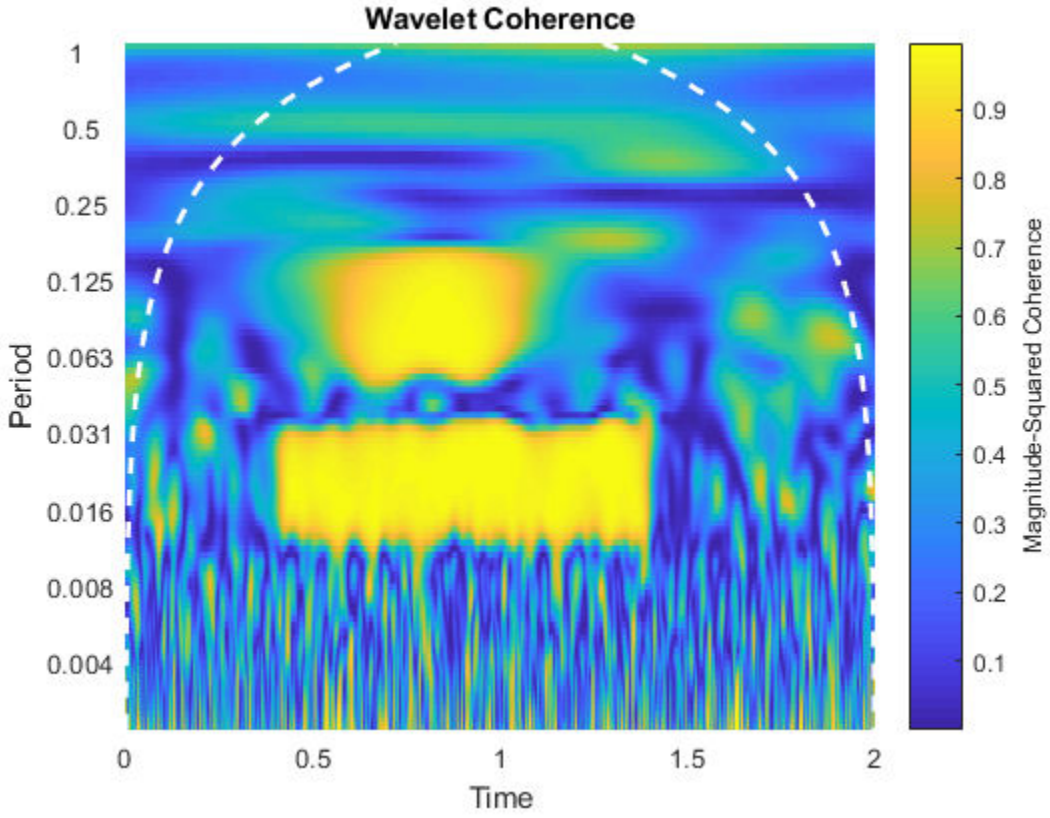
Obtain the wavelet coherence data for two signals, specifying a sampling interval of 0.001 seconds. Both signals consist of two sine waves (10 Hz and 50 Hz) in white noise. The sine waves have different time supports.

Set the random number generator to its default settings for reproducibility. Then, create the two signals and obtain the coherence.

```
rng default;
t = 0:0.001:2;
x = cos(2*pi*10*t).*(t>=0.5 & t<1.1)+ ...
cos(2*pi*50*t).*(t>= 0.2 & t< 1.4)+0.25*randn(size(t));
y = sin(2*pi*10*t).*(t>=0.6 & t<1.2)+...
sin(2*pi*50*t).*(t>= 0.4 & t<1.6)+ 0.35*randn(size(t));
[wcoh,~,period,coi] = wcoherence(x,y,seconds(0.001));
```

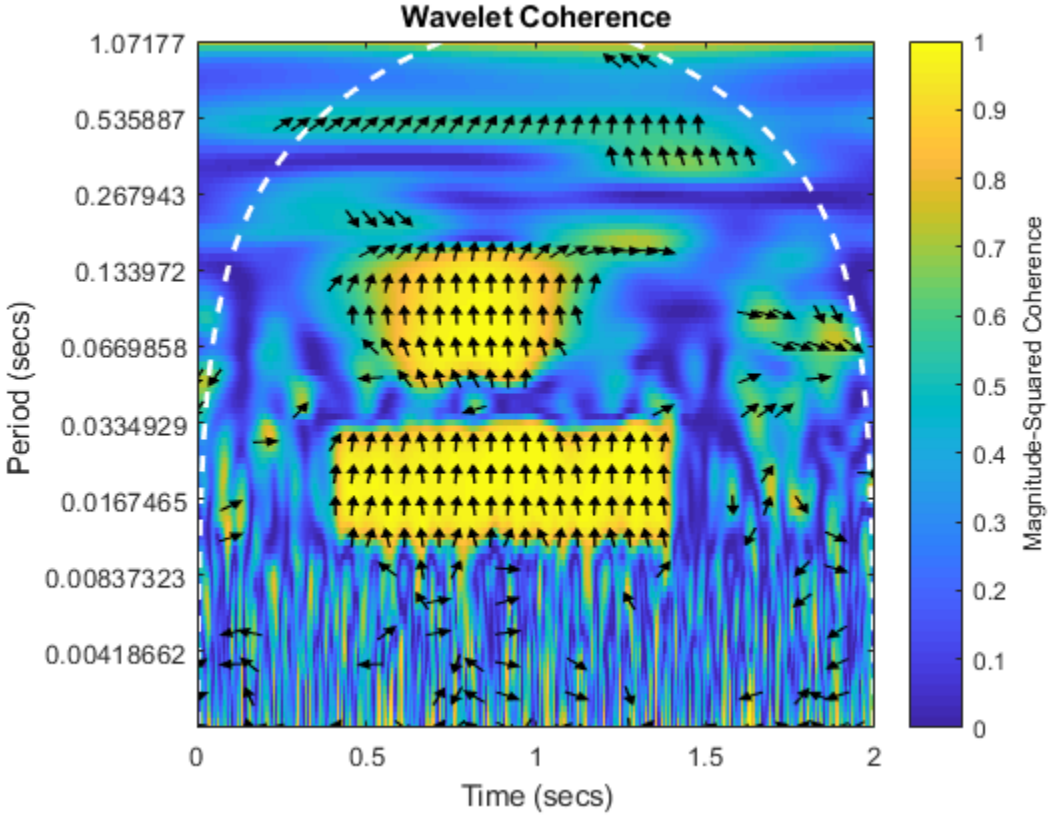
Use the `pcolor` command to plot the coherence and cone of influence.

```
period = seconds(period);
coi = seconds(coi);
h = pcolor(t,log2(period),wcoh);
h.EdgeColor = 'none';
ax = gca;
ytick=round(pow2(ax.YTick),3);
ax.YTickLabel=ytick;
ax.XLabel.String='Time';
ax.YLabel.String='Period';
ax.Title.String = 'Wavelet Coherence';
hcol = colorbar;
hcol.Label.String = 'Magnitude-Squared Coherence';
hold on;
plot(ax,t,log2(coi),'w--','linewidth',2);
```



Use `wcoherence(x,y,seconds(0.001))` without any outputs arguments. This plot includes the phase arrows and the cone of influence.

```
wcoherence(x,y,seconds(0.001));
```



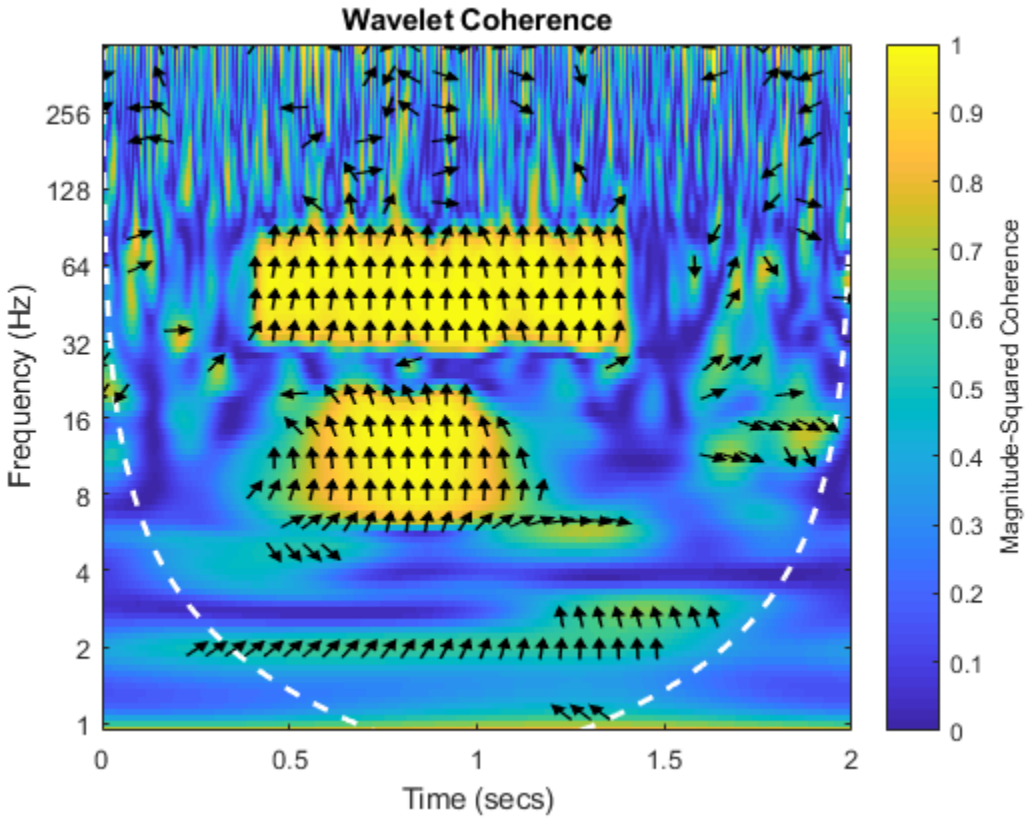
Effect of Sampling Frequency on Wavelet Coherence

Obtain the wavelet coherence for two signals, specifying a sampling frequency of 1000 Hz. Both signals consist of two sine waves (10 Hz and 50 Hz) in white noise. The sine waves have different time supports.

Set the random number generator to its default settings for reproducibility. Then, create the two signals and obtain the coherence.

```
rng default;
t = 0:0.001:2;
```

```
x = cos(2*pi*10*t).*(t>=0.5 & t<1.1)+...
cos(2*pi*50*t).*(t>= 0.2 & t< 1.4)+0.25*randn(size(t));
y = sin(2*pi*10*t).*(t>=0.6 & t<1.2)+...
sin(2*pi*50*t).*(t>= 0.4 & t<1.6)+ 0.35*randn(size(t));
wcoherence(x,y,1000);
```



This coherence plot is flipped with respect to the coherence plot in the previous example, which specifies a sampling interval instead of a sampling frequency.

Obtain the scale-to-freqeuncy conversion output in `f`.

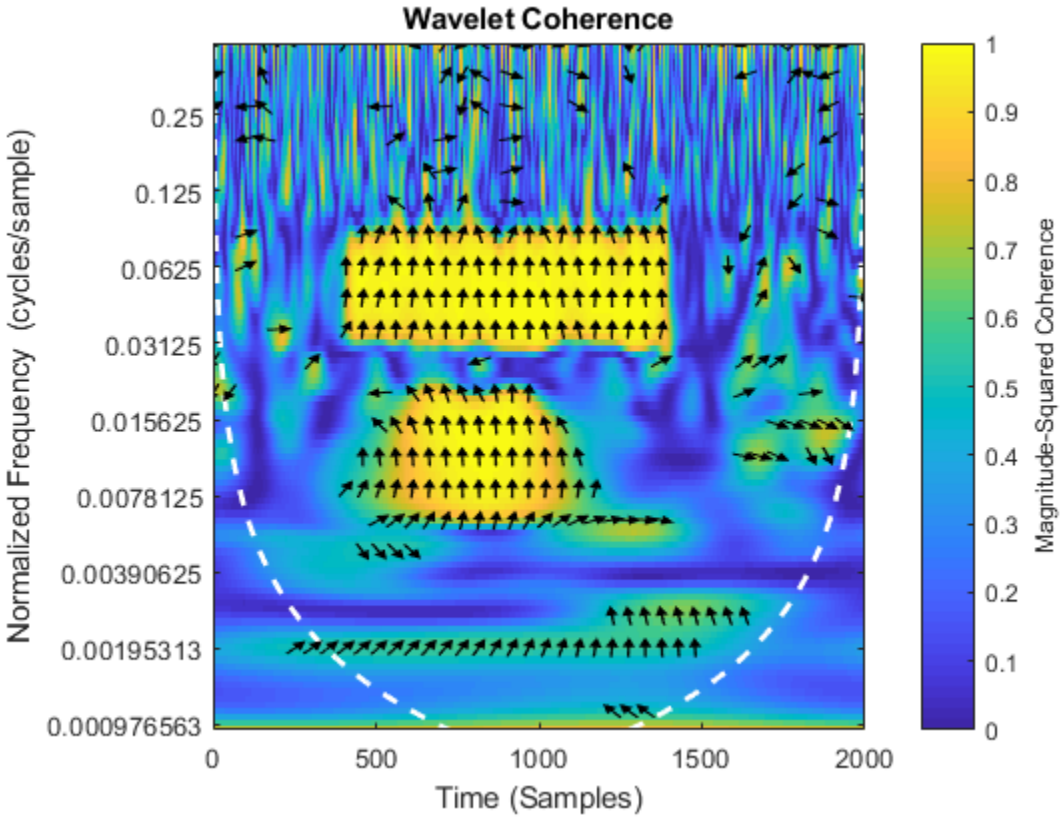
```
[wcoh,wcs,f] = wcoherence(x,y,1000);
```

Effect of Number of Smoothed Scales on Wavelet Coherence

Obtain the wavelet coherence for two signals. Both signals consist of two sine waves (10 Hz and 50 Hz) in white noise. Use the default number of scales to smooth. This value is equivalent to the number of voices per octave. Both values default to 12.

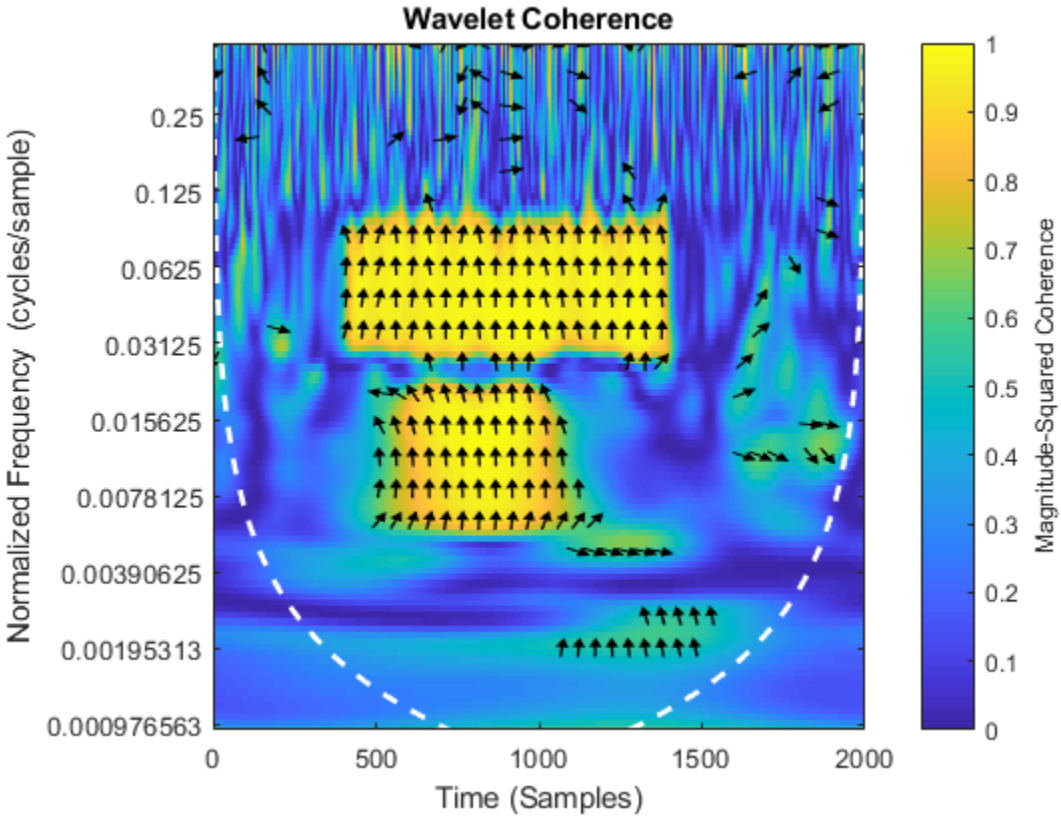
Set the random number generator to its default settings for reproducibility. Then, create the two signals and obtain the coherence.

```
rng default;
t = 0:0.001:2;
x = cos(2*pi*10*t).*(t>=0.5 & t<1.1)+ ...
cos(2*pi*50*t).*(t>= 0.2 & t< 1.4)+0.25*randn(size(t));
y = sin(2*pi*10*t).*(t>=0.6 & t<1.2)+...
sin(2*pi*50*t).*(t>= 0.4 & t<1.6)+ 0.35*randn(size(t));
wcoherence(x,y);
```



Set the number of scales to smooth to 16. The increased smoothing causes reduced low frequency resolution.

```
wcoherence(x,y,'NumScalesToSmooth',16);
```

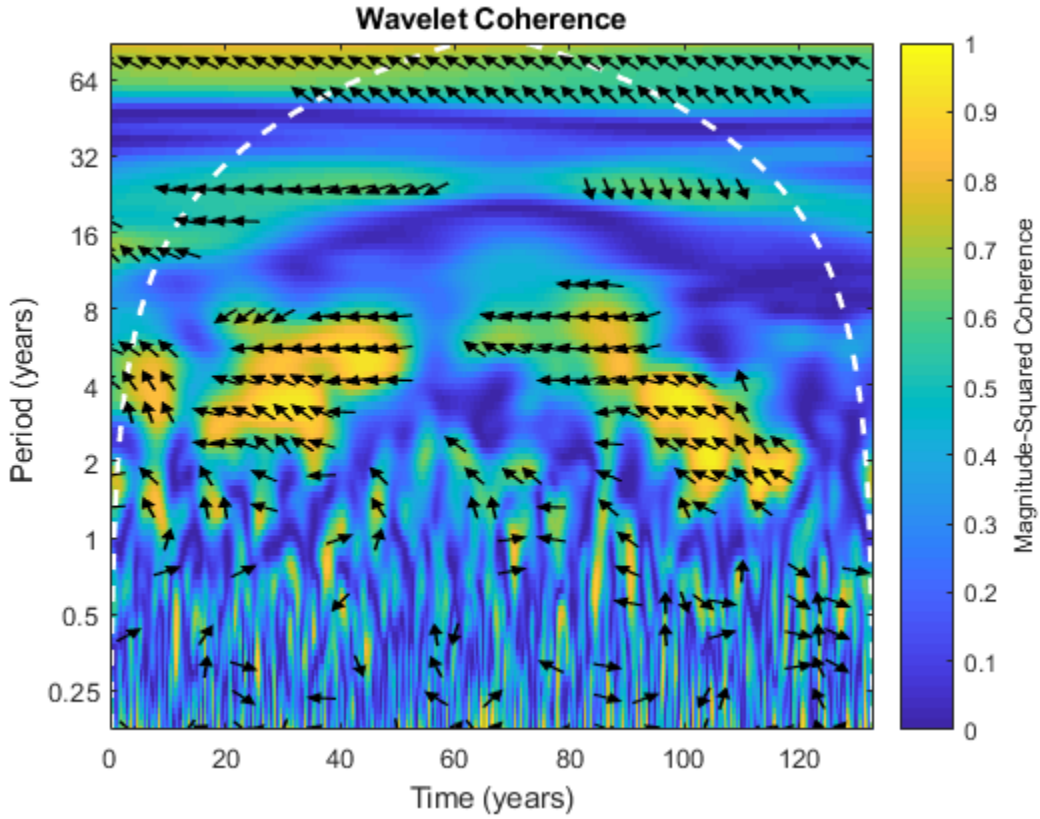


Effect of Phase Display Threshold on Wavelet Coherence of Weather Data

Compare the effects of using different phase display thresholds on the wavelet coherence.

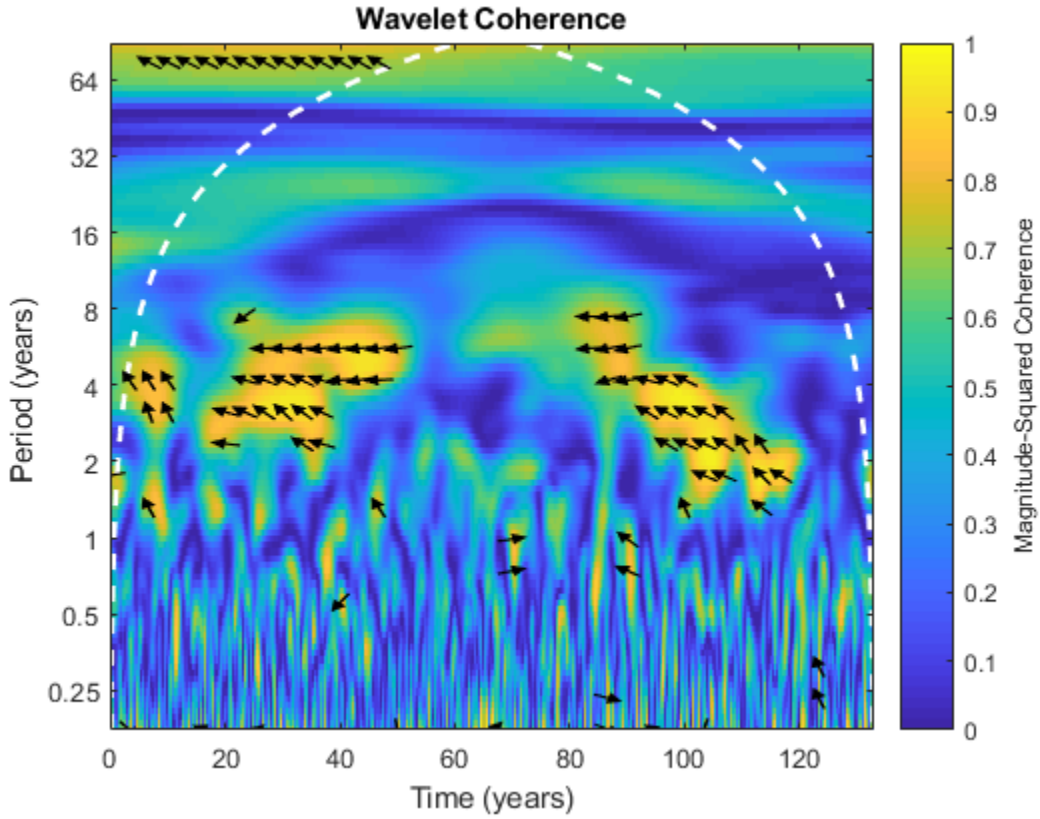
Plot the wavelet coherence between the El Nino time series and the All India Average Rainfall Index. The data are sampled monthly. Specify the sampling interval as 1/12 of a year to display the periods in years. Use the default phase display threshold of 0.5, which shows phase arrows only where the coherence is greater than or equal to 0.5.

```
load ninoairdata;
wcoherence(nino,air,years(1/12));
```



Set the phase display threshold to 0.7. The number of phase arrows decreases.

```
wcoherence(nino,air,years(1/12),'PhaseDisplayThreshold',0.7);
```



Input Arguments

`x` — Input signal

vector of real values

Input signal, specified as a vector of real values. `x` must be a 1-D, real-valued signal. The two input signals, `x` and `y`, must be the same length and must have at least four samples.

`y` — Input signal

vector of real values

Input signal, specified as vector of real values. `y` must be a 1-D, real-valued signal. The two input signals, `x` and `y`, must be the same length and must have at least four samples.

`ts` — Sampling interval
duration with positive scalar input

Sampling interval, also known as the sampling period, specified as a `duration` with positive scalar input. Valid durations are `years`, `days`, `hours`, `seconds`, and `minutes`. You can also use the `duration` function to specify `ts`. You cannot use calendar durations (`caldays`, `calweeks`, `calmonths`, `calquarters`, or `calyears`). You cannot specify both `ts` and `fs`.

`fs` — Sampling frequency
positive scalar

Sampling frequency, specified as a positive scalar.

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: `'PhaseDisplayThreshold',0.7;` specifies the threshold for displaying phase vectors.

`VoicesPerOctave` — Number of voices per octave
12 (default) | integer from 10 to 32

Number of voices per octave to use in the wavelet coherence, specified as an integer from 10 to 32.

`NumScalesToSmooth` — Number of scales to smooth
number of voices per octave (default) | positive integer

Number of scales to smooth in time and scale, specified as a positive integer. This value must be less than one half the total number of scales. The function uses a moving average filter to smooth across scale. If you do not specify the number of scales to smooth, the value of `NumScalesToSmooth` defaults to the number of voices per octave. If your

coherence is noisy, you can specify a larger `NumScalesToSmooth` value to smooth the coherence more.

`NumOctaves` — Number of octaves
positive integer

Number of octaves to use in the wavelet coherence, specified as a positive integer between 1 and `floor(log2(numel(x)))-1`. If you do not need to examine lower frequency values, use a smaller `NumOctaves` value.

`PhaseDisplayThreshold` — Threshold for displaying phase vectors
0.5 (default) | real scalar between 0 and 1

Threshold for displaying phase vectors, specified as a real scalar between 0 and 1. This function displays phase vectors for regions with coherence greater than or equal to the specified threshold value. Lowering the threshold value displays more phase vectors. If you use `wcoherence` with any output arguments, the `PhaseDisplayThreshold` value is ignored.

Output Arguments

`wcoh` — Wavelet coherence
matrix

Wavelet coherence, returned as a matrix. The coherence is computed using the analytic Morlet wavelet over logarithmic scales, with a default value of 12 voices per octave. The default number of octaves is equal to `floor(log2(numel(x)))-1`. If you do not specify a sampling interval, sampling frequency is assumed.

`wcs` — Wavelet cross spectrum
matrix of complex values

Wavelet cross-spectrum, returned as a matrix of complex values. You can use the phase of the wavelet cross-spectrum values to identify the relative lag between the input signals.

Data Types: `double`
Complex Number Support: Yes

`period` — Scale-to-period conversion
array of durations

Scale-to-period conversion, returned as an array of durations. The conversion values are computed from the sampling period specified in `ts`. Each `period` element has the same format as `ts`.

`f` — Scale-to-frequency conversion
vector

Scale-to-frequency conversion, returned as a vector. The vector contains the peak frequency values for the wavelets used to compute the coherence. If you want to output `f`, but do not specify a sampling frequency input, `fs`, the returned wavelet coherence is in cycles per sample.

`coi` — Cone of influence
array of doubles or array of durations

Cone of influence for the wavelet coherence, returned as either an array of doubles or array of durations. The cone of influence indicates where edge effects occur in the coherence data. If you specify a sampling frequency, `fs`, the cone of influence is in Hz. If you specify a sampling interval or period, `ts`, the cone of influence is in periods. Due to the edge effects, give less credence to areas of apparent high coherence that are outside or overlap the cone of influence. The cone of influence is indicated by a dashed line.

Definitions

Wavelet Cross Spectrum

The wavelet cross- spectrum is a measure of the distribution of power of two signals.

The wavelet cross spectrum of two time series, x and y, is:

$$C_{xy}(a,b) = S(C_x^*(a,b)C_y(a,b))$$

$C_x(a,b)$ 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

Wavelet coherence is a measure of the correlation between two signals.

The wavelet coherence of two time series x and y is:

$$\frac{\left|S(C_x^*(a,b)C_y(a,b))\right|^2}{S(|C_x(a,b)|^2)\ S(|C_y(a,b)|^2)}$$

$C_x(a,b)$ 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.

References

[1] 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*. Vol. 11, Issue 5/6, 2004, pp. 561–566.

[2] Maraun, D., J. Kurths, and M. Holschneider. "Nonstationary Gaussian processes in wavelet domain: Synthesis, estimation and significance testing.” *Physical Review E 75*. 2007, pp. 016707-1–016707-14.

[3] Torrence, C., and P. Webster. "Interdecadal changes in the ESNO-Monsoon System." *Journal of Climate*. Vol. 12, 1999, pp. 2679–2690.

See Also

`cwt` | `days` | `duration` | `hours` | `minutes` | `seconds` | `years`

Introduced in R2016a

wcompress

True compression of images using wavelets

Syntax

```
wcompress('c',X,SAV_FILENAME,COMP_METHOD)
wcompress(...,'ParName1',ParVal1,'ParName2',ParVal2,...)
[COMPRAT,BPP] = wcompress('c',...)
XC = wcompress('u',SAV_FILENAME)
XC = wcompress('u',SAV_FILENAME,'plot')
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 “Wavelet 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`. You must have write permission in the current working directory or MATLAB will change directory to `tempdir` and write the `.wtc` file in that directory. `X` can be either a 2-D array containing an indexed image or a 3-D array of `uint8` containing a truecolor image. Both the row and column size of the image must be powers of two.

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.

Note Data written to .wtc files uses uint64 precision. In releases previous to R2016b, data was written using uint32 . If your code is affected adversely by this change, use the legacy option to compress and uncompress your data using the previous behavior.

```
wcompress('c',X,SAV_FILENAME,COMP_METHOD,'legacy')
```

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
'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.

1 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.

1 Coefficients Thresholding Methods (**CTM-2**):

MATLAB Name	Compression Method Name
'gbl_mmc_f'	Global thresholding of coefficients and fixed encoding

MATLAB Name	Compression Method Name
'gbl_mmc_h'	Global thresholding of coefficients and Huffman encoding

Note The Discrete Wavelet Transform uses the periodized extension mode.

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 character vector (see `waveletfamilies`). The default for is *bior4.4*

- `'ParName'` = `'level'` or `'LEVEL'` sets the level of decomposition.

  `ParVal` is an integer such that: `1 ≤ level ≤ 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 character vectors:

  `'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 character vectors:

  `'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 $0 \leq$ `ParVal` ≤ 8 (grayscale) or `24` (truecolor).
- `'ParName' = 'comprat'` or `'COMPRAT'` sets the compression ratio.

  `ParVal` must be such that $0 \leq$ `ParVal` ≤ 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` ≤ 100.

- `'ParName'` = `'bpp'` or `'BPP'` sets the bit-per-pixel ratio.

  `ParVal` must be such that: $0 \leq$ `ParVal` ≤ 8 (grayscale) or `24` (truecolor)

- `'ParName'` = `'comprat'` or `'COMPRAT'` sets the compression ratio.

  `ParVal` must be such that: $0 \leq$ `ParVal` ≤ 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` ≤ 200.

Display parameter

- `'ParName'` = `'plotpar'` or `'PLOTPAR'` sets the plot parameter.

  `ParVal` must be one of the following character vectors 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`, which contains the compressed image, 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

Image Compression Using Basic Parameters

This example shows how to compress and uncompress the jpeg image arms.jpg.

Use the spatial orientation tree wavelet (`'stw'`) compression method and save the compressed image to a file.

```
wcompress('c','arms.jpg','comp_arms.wtc','stw');
```

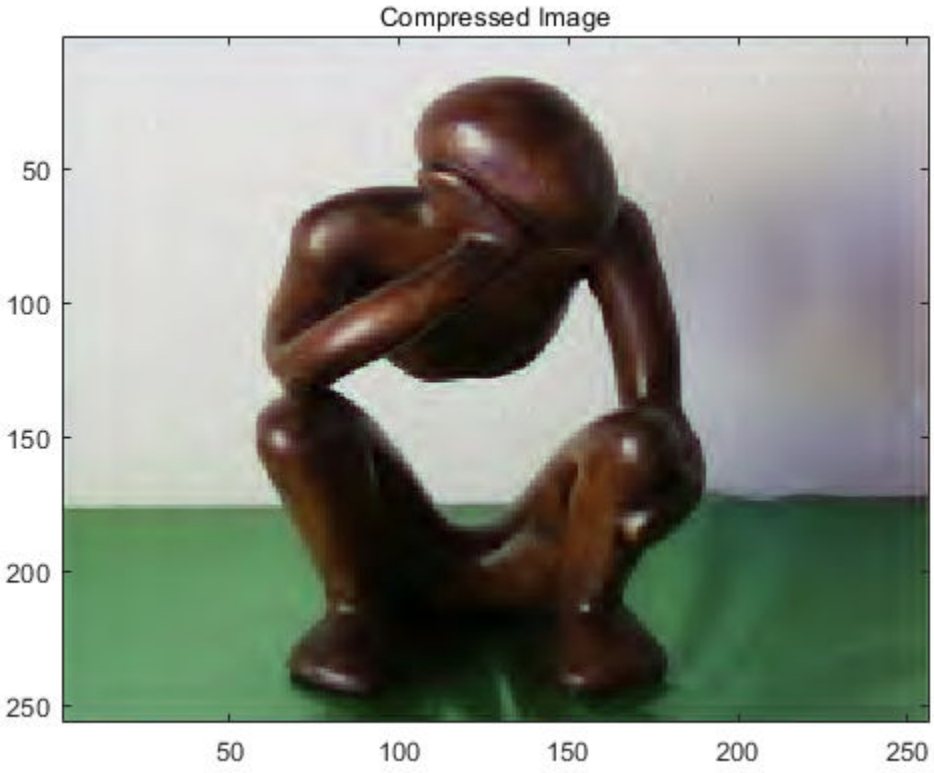
Load the stored image and display the step-by-step uncompression to produce the uncompressed image.

```
wcompress('u','comp_arms.wtc','step');
```

Image Compression and Uncompression Using Advanced Parameters.

This example show how to compress a jpeg image using the adaptively scanned wavelet difference reduction compression method (`'aswdr'`). The conversion color (`'cc'`) uses the Karhunen-Loeve transform (`'kit'`). The maximum number of loops (`'maxloop'`) is set to 11 and the plot type (`'plotpar'`) is set to step through the compression. Show the compression ratio (`cratio`) and the bit-per-pixel ratio (`bpp`), which indicate the quality of the compression.

```
[cratio,bpp] = wcompress('c','woodstatue.jpg','woodstatue.wtc', ...
               'aswdr','cc','klt','maxloop',11,'plotpar','step');
```




```
cratio
```

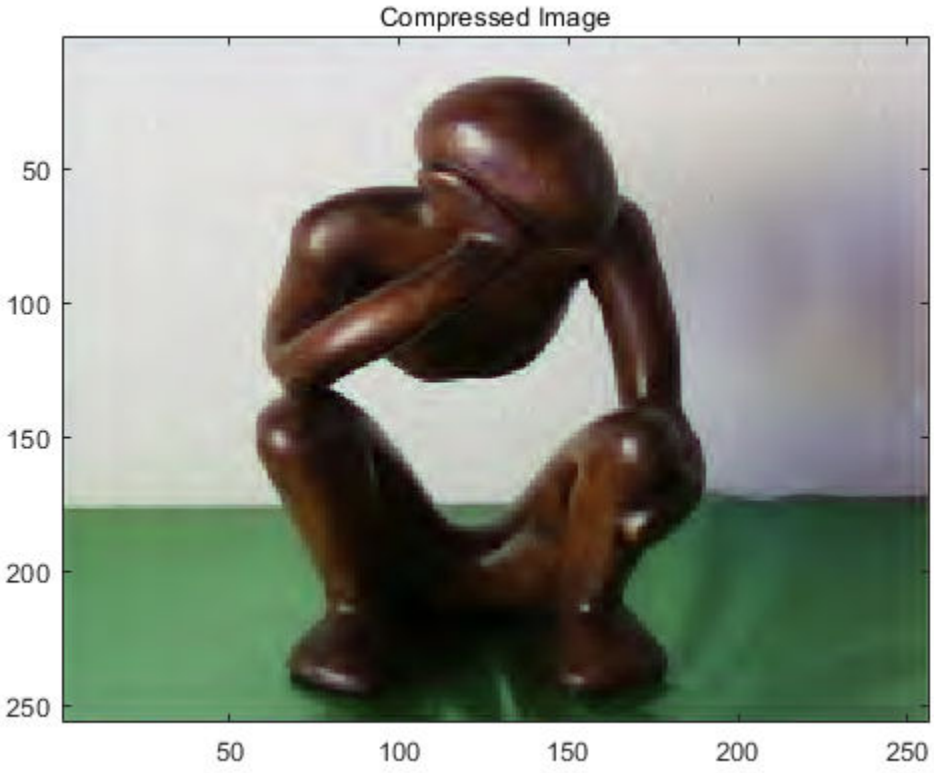
```
cratio = 3.0792
```

```
bpp
```

```
bpp = 0.7390
```

Load the compressed image and step through the uncompression process.

```
wcompress('u','woodstatue.wtc','step');
```



Compression and Uncompression of a Grayscale Image

This example shows how to compress a grayscale image using the set partitioning in hierarchical trees (`'spiht'`) compression method. It also computes the mean square error (MSE) and the peak signal to noise ratio (PSNR) error values. You use these two measures to quantify the error between two images. The PSNR is expressed in decibels.

Load the image and store it in a file.

```
load mask;
[cr,bpp] = wcompress('c',X,'mask.wtc','spiht','maxloop',12)
```

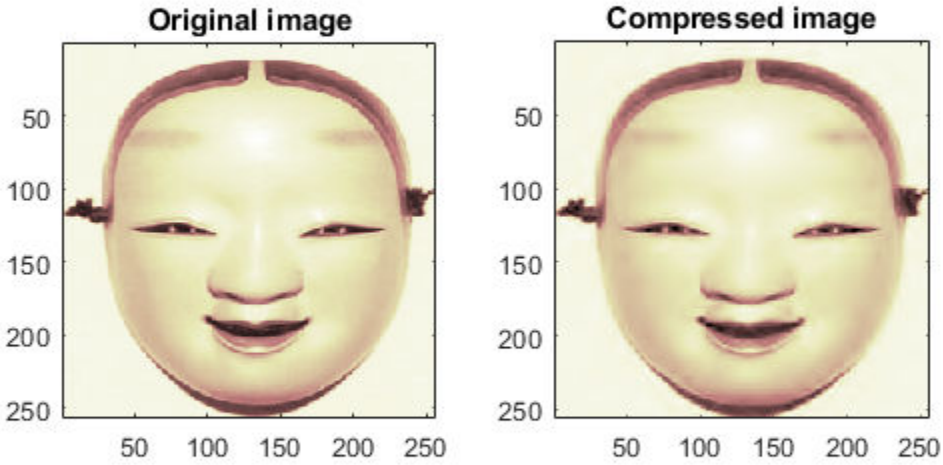
```
cr = 2.8610

bpp = 0.2289
```

Load the stored image from the file, uncompress it, and delete the file.

```
Xc = wcompress('u','mask.wtc');
delete('mask.wtc')
```

Display the original and compressed images.

```
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
```



Compute the MSE and PSNR.

```
D = abs(X-Xc).^2;
mse  = sum(D(:))/numel(X)
```

```
mse = 33.6564
```

```
psnr = 10*log10(255*255/mse)
```

```
psnr = 32.8601
```

Compression and Uncompression of a Truecolor Image

is example shows how to compress a trucolor image using the set partitioning in hierarchical trees - 3D (`'spiht_3D'`) compression method.

Load, compress, and store the image in a file. Plot the original and compressed images. Display the compression ratio (`'cratio'`) and the bits-per-pixel (`'bpp'`), which indicate the quality of the compression.

```
load mask;
X = imread('wpeppers.jpg');
[cratio,bpp] = wcompress('c',X,'wpeppers.wtc','spiht','maxloop',12)
```

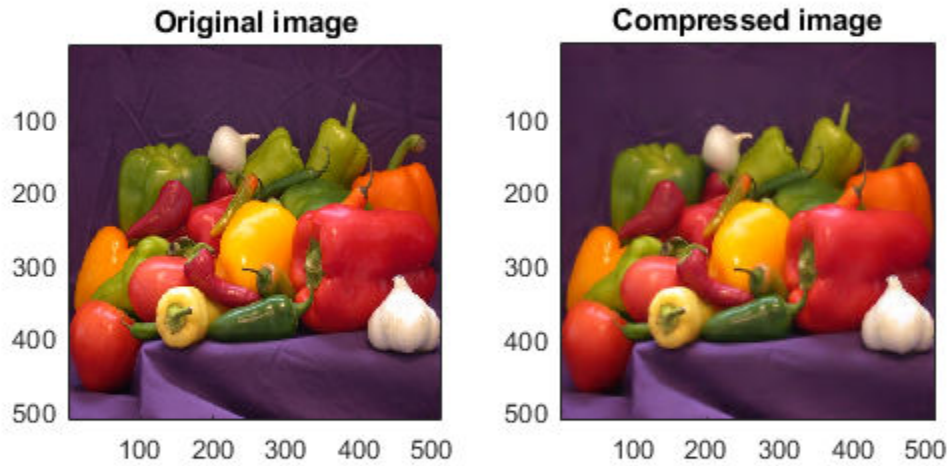
```
cratio = 1.6549
```

```
bpp = 0.3972
```

```
Xc = wcompress('u','wpeppers.wtc');
delete('wpeppers.wtc')
```

Display the original and compressed images.

```
subplot(1,2,1); image(X);  title('Original image'), axis square
subplot(1,2,2); image(Xc); title('Compressed image'), axis square
```



Compute the mean square error (MSE) and the peak signal-to-noise ratio (PSNR) error values. You use these two measures to quantify the error between two images. The PSNR is expressed in decibels.

```
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` | `path` | `tempdir` | `wmaxlev`

Introduced in R2008b

wdcbm

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` / $(j+2-i)^{ALPHA}$.

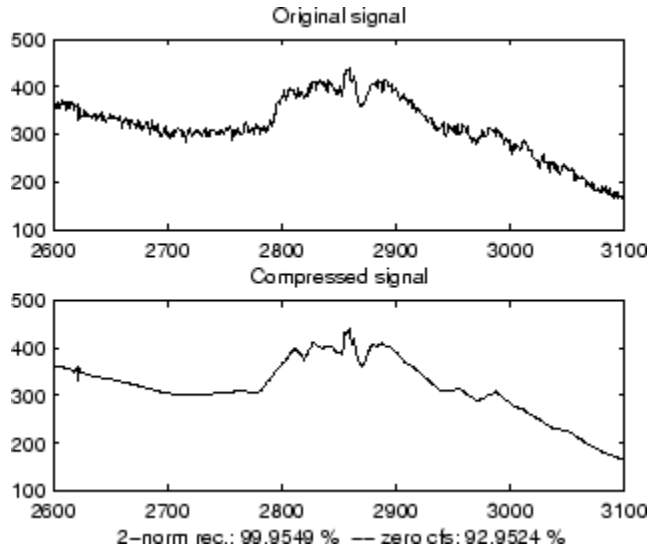
Typically `ALPHA` = 1.5 for compression and `ALPHA` = 3 for de-noising.

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]);
```



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`

Introduced before R2006a

wdcbm2

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` $(j+2-i)^{ALPHA}$.

Typically `ALPHA` = 1.5 for compression and `ALPHA` = 3 for de-noising.

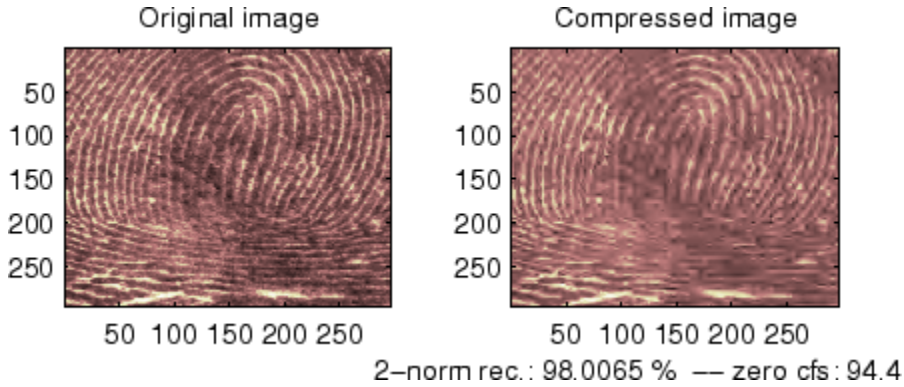
A default value for `M` is `M` = `prod(S(1,:))`, the length of the coarsest approximation coefficients, since the previous formula leads for i = j+1, to n_{j+1} = `M` = `prod(S(1,:))`.

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]);
```



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`

Introduced before R2006a

wdecenergy

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.

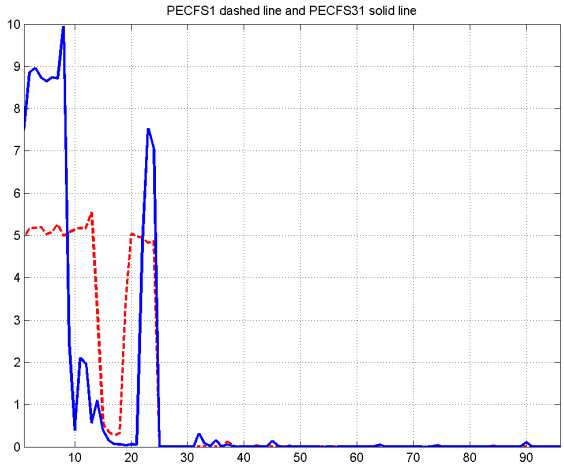
LONGS is a vector containing the lengths of each family of coefficients.

`[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')
```



See Also

`mdwtdec` | `mdwtrec`

Introduced in R2012a

wden

Automatic 1-D de-noising

Syntax

```
XD = wden(X,TPTR,SORH,SCAL,N,'wname')
XD = wden(C,L,TPTR,SORH,SCAL,N,'wname')
XD = wden(W,'modwtsqtwolog',SORH,'mln',N,WNAME)
[XD,CXD] = wden(...)
[XD,CXD,LXD] = wden(...)
[XD,CXD,LXD,THR] = wden(...)
[XD,CXD,THR] = wden(...)
```

Description

`wden` is a one-dimensional de-noising function.

`wden` performs an automatic de-noising process of a one-dimensional signal using wavelets.

`XD = wden(X,TPTR,SORH,SCAL,N,'wname')` returns a de-noised version `XD` of input signal `X` obtained by thresholding the wavelet coefficients.

`TPTR` character vector 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 character vector containing the name of the desired orthogonal wavelet (see `wmaxlev` and `wfilters` for more information).

`XD = 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.

`XD = wden(W,'modwtsqtwolog',SORH,'mln',N,WNAME)` returns the denoised signal obtained by operating on the MODWT transform matrix `W`, where `W` is the output of MODWT. You must use the same wavelet in both `modwt` and `wden`.

`[XD,CXD] = wden(...)` returns the denoised wavelet coefficients. For DWT denoising, `CXD` is a vector (see `wavedec`). For MODWT denoising, `CXD` is a matrix with N+1 rows (see `modwt`). The number of columns is equal to the length of the input signal `X`.

`[XD,CXD,LXD] = wden(...)` returns the number of coefficients by level for DWT denoising. See `wavedec` for details. The `LXD` output is not supported for MODWT denoising. The additional output arguments `[CXD,LXD]` are the wavelet decomposition structure (see `wavedec` for more information) of the de-noised signal `XD`.

`[XD,CXD,LXD,THR] = wden(...)` returns the denoising thresholds by level for DWT denoising.

`[XD,CXD,THR] = wden(...)` returns the denoising thresholds by level for MODWT denoising when you specify `'modwtsqtwolog'`.

Examples

Automatic 1-D Denoising Using Wavelets

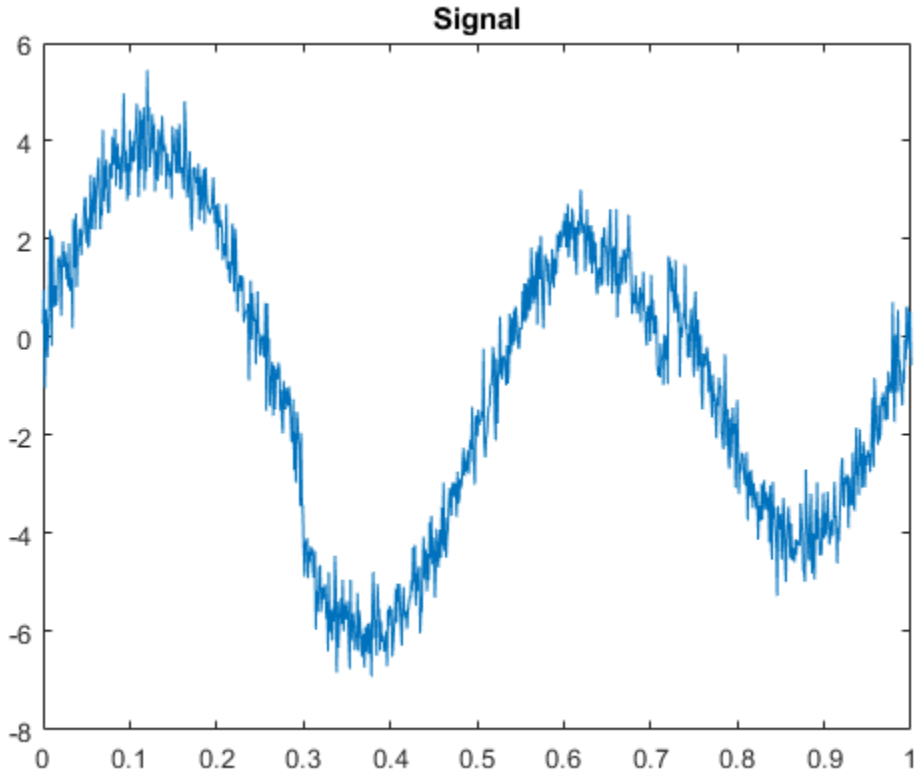
In this example, three different denoising techniques are applied to a noisy signal. Plots comparing results are generated. The thresholds produced by each technique are also shown.

First, to ensure reproducibility of results, set a seed that will be used to generate the random noise.

```
rng('default')
```

Create a signal consisting of a 2-Hz sine wave with transients at 0.3 and 0.72 seconds. Add randomly generated noise to the signal and plot the result.

```
N = 1000;
t = linspace(0,1,N);
x = 4*sin(4*pi*t);
x = x - sign(t-0.3) - sign(0.72-t);
sig = x + 0.5*randn(size(t));
plot(t,sig)
title('Signal')
```

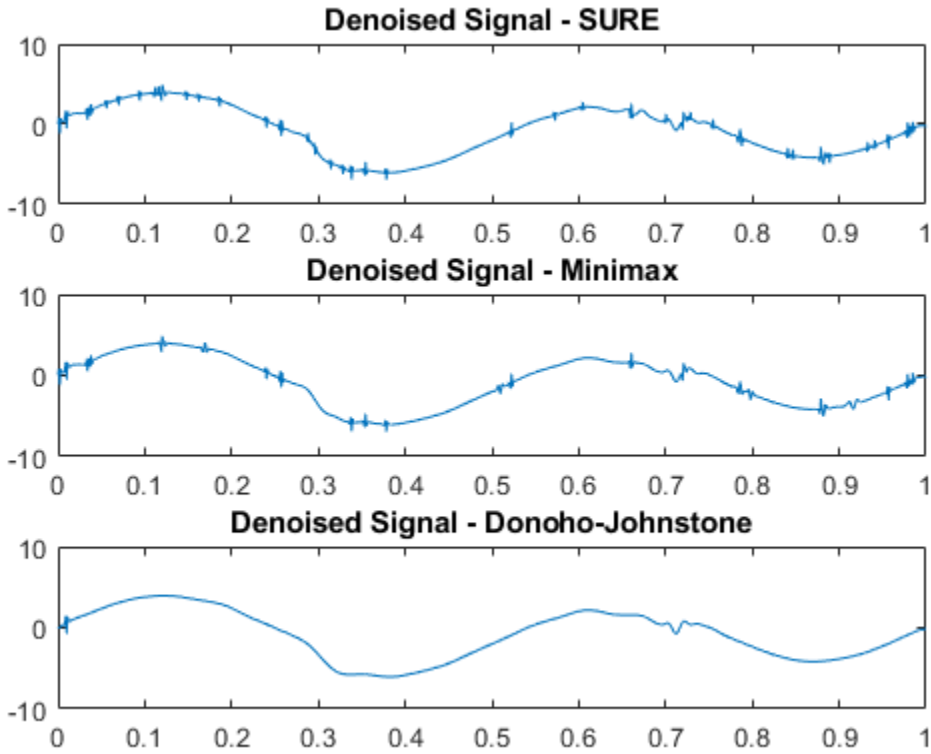


Using the `sym8` wavelet, perform a level 5 wavelet decomposition of the signal and denoise it by applying three different threshold selection rules to the wavelet coefficients: SURE; minimax; Donoho and Johnstone's universal threshold with level-dependent estimation of the noise. In each case, apply hard thresholding.

```
lev = 5;
wname = 'sym8';
[dnsig1,c1,l1,threshold_SURE] = wden(sig,'rigrsure','h','mln',lev,wname);
[dnsig2,c2,l2,threshold_Minimax] = wden(sig,'minimaxi','h','mln',lev,wname);
[dnsig3,c3,l3,threshold_DJ] = wden(sig,'sqtwolog','h','mln',lev,wname);
```

Plot and compare the three denoised signals.

```
subplot(3,1,1)
plot(t,dnsig1)
title('Denoised Signal - SURE')
subplot(3,1,2)
plot(t,dnsig2)
title('Denoised Signal - Minimax')
subplot(3,1,3)
plot(t,dnsig3)
title('Denoised Signal - Donoho-Johnstone')
```



Compare the thresholds applied at each detail level for the three denoising methods.

```
threshold_SURE
```

```
threshold_SURE =

    0.8079    1.1448    1.3915    0.9816    0.8667
```

```
threshold_Minimax
```

```
threshold_Minimax =

    1.0522    1.0881    1.2237    1.3366    1.1985
```

```
threshold_DJ
```

```
threshold_DJ =

    1.7644    1.8247    2.0520    2.2413    2.0097
```

Compare DWT and MODWT denoising of a sinusoid with two jumps

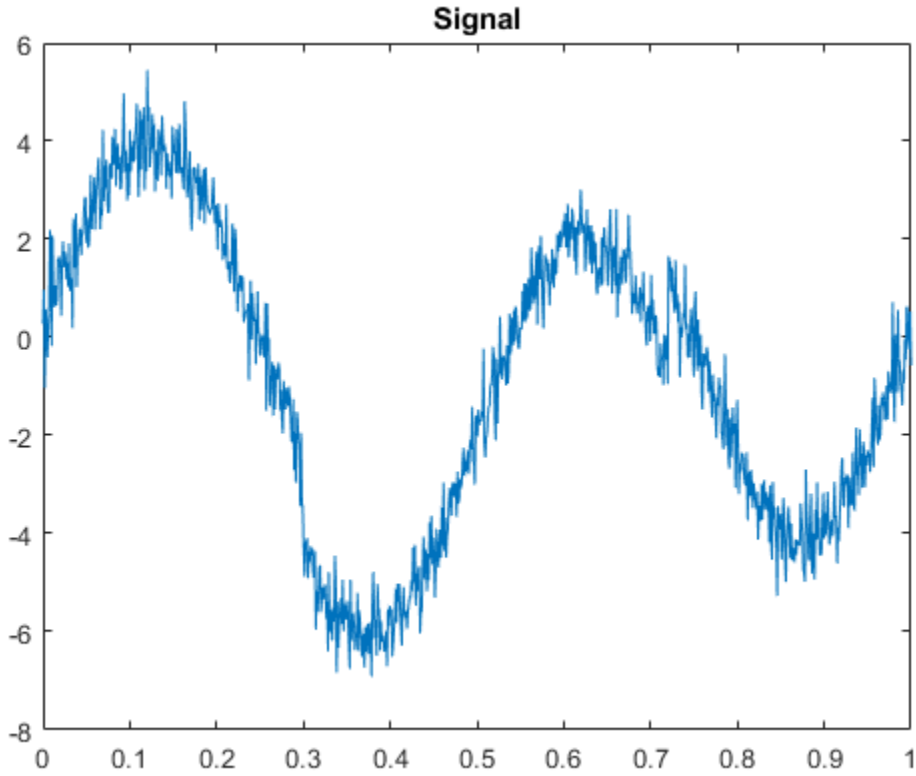
In this example, a signal is denoised using the DWT and MODWT. Plots comparing results are generated. The thresholds used in each case are also shown.

First, to ensure reproducibility of results, set a seed that will be used to generate random noise.

```
rng('default')
```

Create a signal consisting of a 2-Hz sine wave with transients at 0.3 and 0.72 seconds. Add randomly generated noise to the signal and plot the result.

```
N = 1000;
t = linspace(0,1,N);
x = 4*sin(4*pi*t);
x = x - sign(t-0.3) - sign(0.72-t);
sig = x + 0.5*randn(size(t));
plot(t,sig)
title('Signal')
```



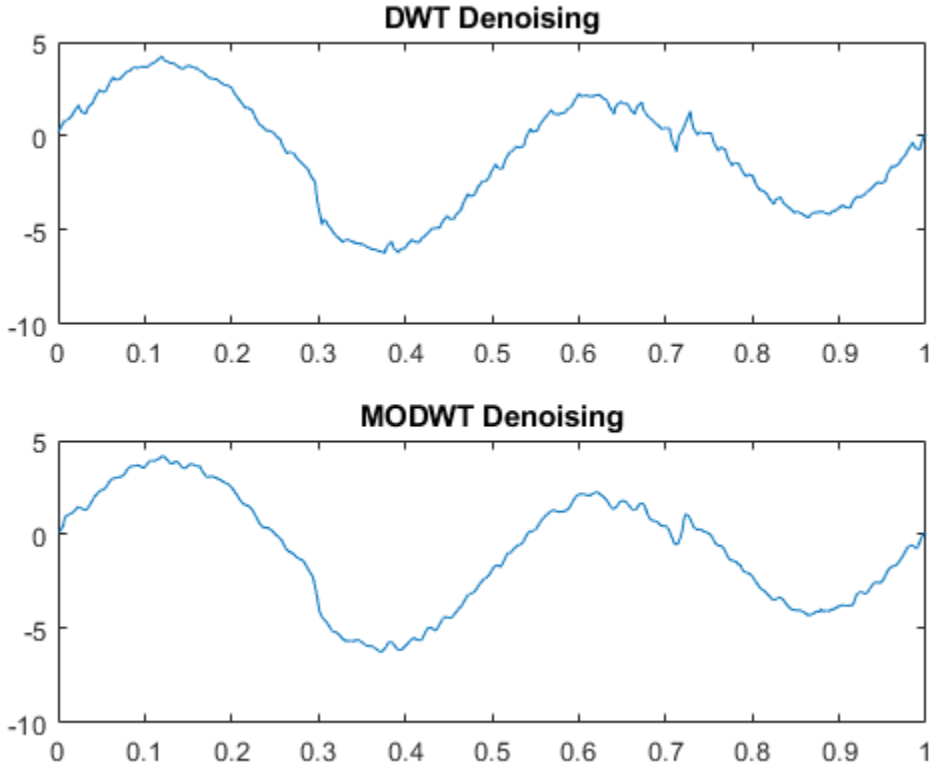
Using the `db2` wavelet, perform a level 3 wavelet decomposition of the signal and denoise it using Donoho and Johnstone's universal threshold with level-dependent estimation of the noise. Obtain denoised versions using DWT and MODWT and soft thresholding.

```
wname = 'db2';
lev = 3;
[xdDWT,c1,l1,threshold_DWT] = wden(sig,'sqtwolog','s','mln',lev,wname);
[xdMODWT,c2,threshold_MODWT] = wden(sig,'modwtsqtwolog','s','mln',lev,wname);
```

Plot and compare the results.

```
subplot(2,1,1)
plot(t,xdDWT)
title('DWT Denoising')
```

```
subplot(2,1,2)
plot(t,xdMODWT)
title('MODWT Denoising')
```



Compare the thresholds applied in each case.

```
threshold_DWT
```

```
threshold_DWT =

    1.7700    1.7975    2.3619
```

```
threshold_MODWT
```

```
threshold_MODWT =

    1.2760    0.6405    0.3787
```

Compare DWT and MODWT denoising of a blocky signal

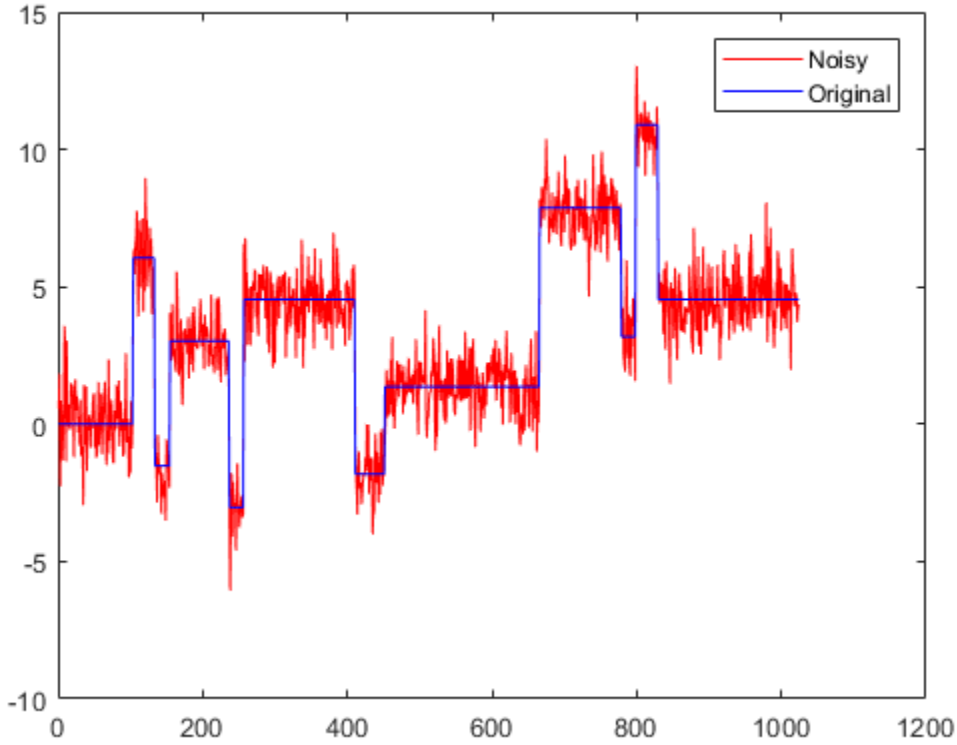
In this example, a blocky signal is denoised using the Haar wavelet with DWT and MODWT denoising. Plots are generated comparing results. Metrics comparing the original with denoised versions are also produced.

First, to ensure reproducibility of results, set a seed that will be used to generate random noise.

```
rng('default')
```

Generate a signal and a noisy version with the square root of the signal-to-noise ratio equal to 3. Plot and compare each.

```
[osig,nsig] = wnoise('blocks',10,3);
plot(nsig,'r')
hold on
plot(osig,'b')
legend('Noisy','Original')
```

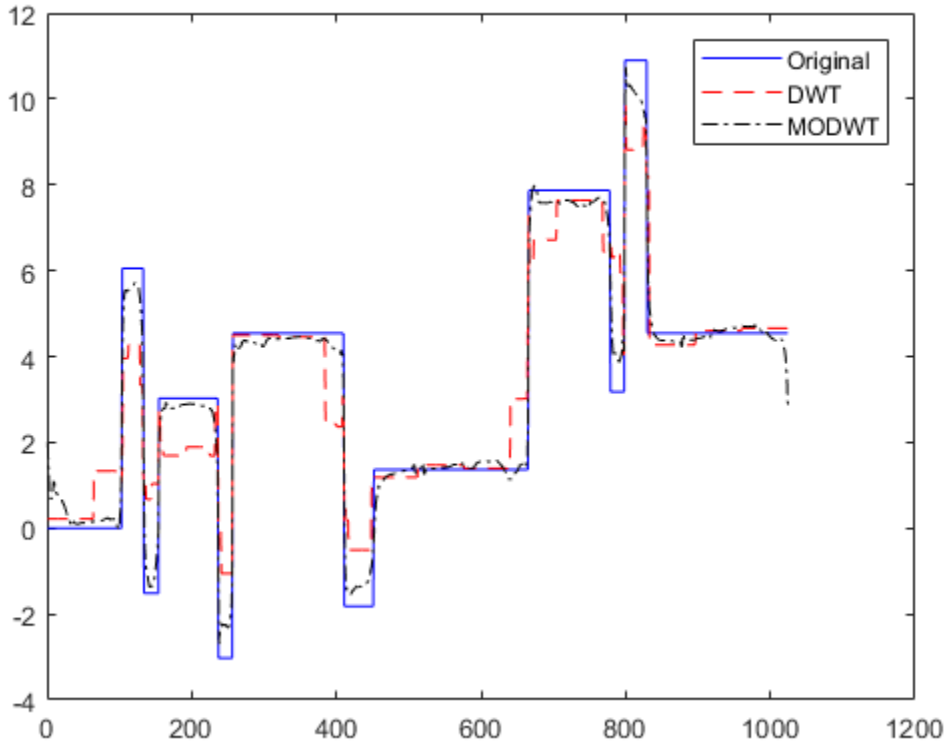


Using the Haar wavelet, perform a level 6 wavelet decomposition of the noisy signal and denoise it using Donoho and Johnstone's universal threshold with level-dependent estimation of the noise. Obtain denoised versions using DWT and MODWT and soft thresholding.

```
wname = 'haar';
lev = 6 ;
[xdDWT,c1,l1] = wden(nsig,'sqtwolog','s','mln',lev,wname);
[xdMODWT,c2] = wden(nsig,'modwtsqtwolog','s','mln',lev,wname);
```

Plot and compare the original, noise-free version of the signal with the two denoised versions.

```
figure
plot(osig,'b')
hold on
plot(xdDWT,'r--')
plot(xdMODWT,'k-.')
legend('Original','DWT','MODWT')
hold off
```



Calculate the L2 and L-infinity norms of the difference between the original signal and the two denoised versions.

```
L2norm_original_DWT = norm(abs(osig-xdDWT),2)
```

```
L2norm_original_DWT = 36.1194
```

```
L2norm_original_MODWT = norm(abs(osig-xdMODWT),2)

L2norm_original_MODWT = 14.5987

LInfinity_original_DWT = norm(abs(osig-xdDWT),Inf)

LInfinity_original_DWT = 4.7181

LInfinity_original_MODWT = norm(abs(osig-xdMODWT),Inf)

LInfinity_original_MODWT = 2.9655
```

Algorithms

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 σ 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 “Wavelet 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.

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Variable-size data support must be enabled.

See Also

`thselect` | `wavedec` | `wdencmp` | `wfilters` | `wthresh`

Introduced before R2006a

wdencmp

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 character vector 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 1-D Signal Using Default Global Threshold

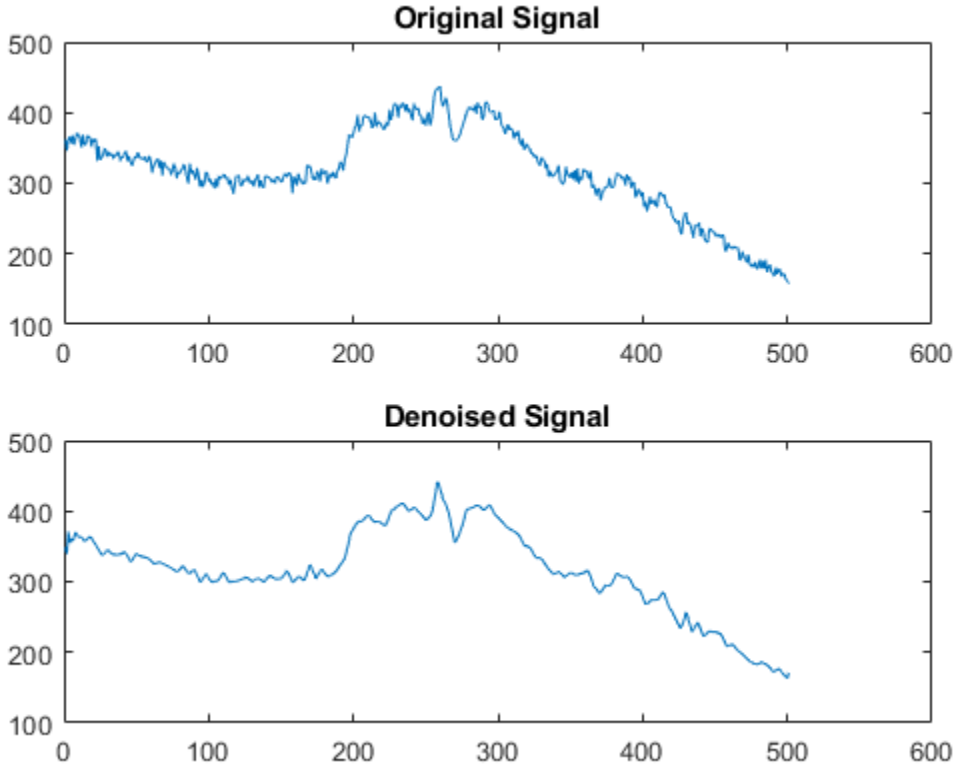
Denoise 1-D electricity consumption data using the Donoho-Johnstone global threshold.

Load the signal and select a segment for denoising.

```
load leleccum; indx = 2600:3100;
x = leleccum(indx);
```

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');
```



Denoise 1-D Signal Using Default Global Threshold

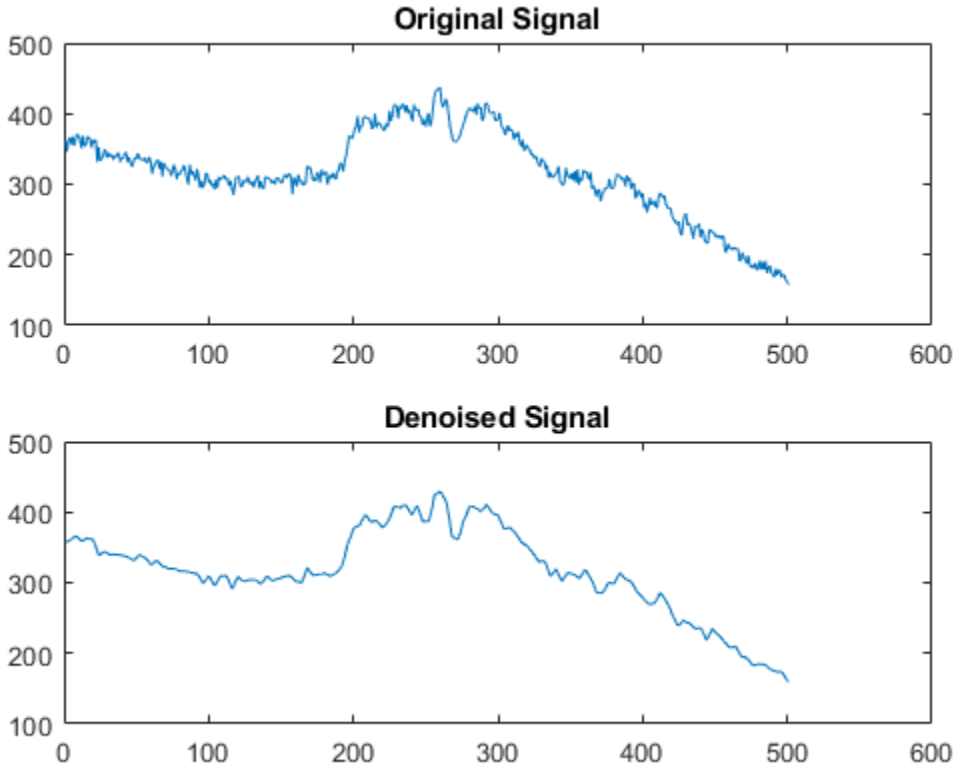
Denoise 1-D electricity consumption data using the Donoho-Johnstone global threshold.

Load the signal and select a segment for denoising.

```
load leleccum; indx = 2600:3100;
x = leleccum(indx);
```

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Variable-size data support must be enabled.

See Also

`ddencmp` | `wavedec` | `wavedec2` | `wbmpen` | `wcompress` | `wdcbm2` | `wden` | `wpdencmp` | `wthresh`

Introduced before R2006a

wdenoise

Denoise data using an empirical Bayesian method with a Cauchy prior

Syntax

```
XDEN = wdenoise(X)
XDEN = wdenoise(X,LEVEL)

XDEN = wdenoise(___,Name,Value)

[XDEN,DENOISEDCFS] = wdenoise(___)
[XDEN,DENOISEDCFS,ORIGCFS] = wdenoise(___)
```

Description

`XDEN = wdenoise(X)` denoises the data in `X` using an empirical Bayesian method with a Cauchy prior. By default, the `sym4` wavelet is used with a posterior median threshold rule. Denoising is down to the minimum of `floor(log2N)` and `wmaxlev(N,'sym4')` where N is the number of samples in the data. (For more information, see `wmaxlev`.) `X` is a real-valued vector, matrix, or timetable.

- If `X` is a matrix, `wdenoise` denoises each column of `X`.
- If `X` is a timetable, `wdenoise` must contain real-valued vectors in separate variables, or one real-valued matrix of data.
- `X` is assumed to be uniformly sampled.
- If `X` is a timetable and the timestamps are not linearly spaced, `wdenoise` issues a warning.

`XDEN = wdenoise(X,LEVEL)` denoises `X` down to `LEVEL`. `LEVEL` is a positive integer less than or equal to `floor(log2N)` where N is the number of samples in the data. If unspecified, `LEVEL` defaults to the minimum of `floor(log2N)` and `wmaxlev(N,'sym4')`.

`XDEN = wdenoise(___,Name,Value)` specifies options using name-value pair arguments in addition to any of the input arguments in previous syntaxes.

`[XDEN,DENOISEDCFS] = wdenoise(___)` returns the denoised wavelet and scaling coefficients in the cell array `DENOISEDCFS`. The elements of `DENOISEDCFS` are in order of decreasing resolution. The final element of `DENOISEDCFS` contains the approximation (scaling) coefficients.

`[XDEN,DENOISEDCFS,ORIGCFS] = wdenoise(___)` returns the original wavelet and scaling coefficients in the cell array `ORIGCFS`. The elements of `ORIGCFS` are in order of decreasing resolution. The final element of `ORIGCFS` contains the approximation (scaling) coefficients.

Examples

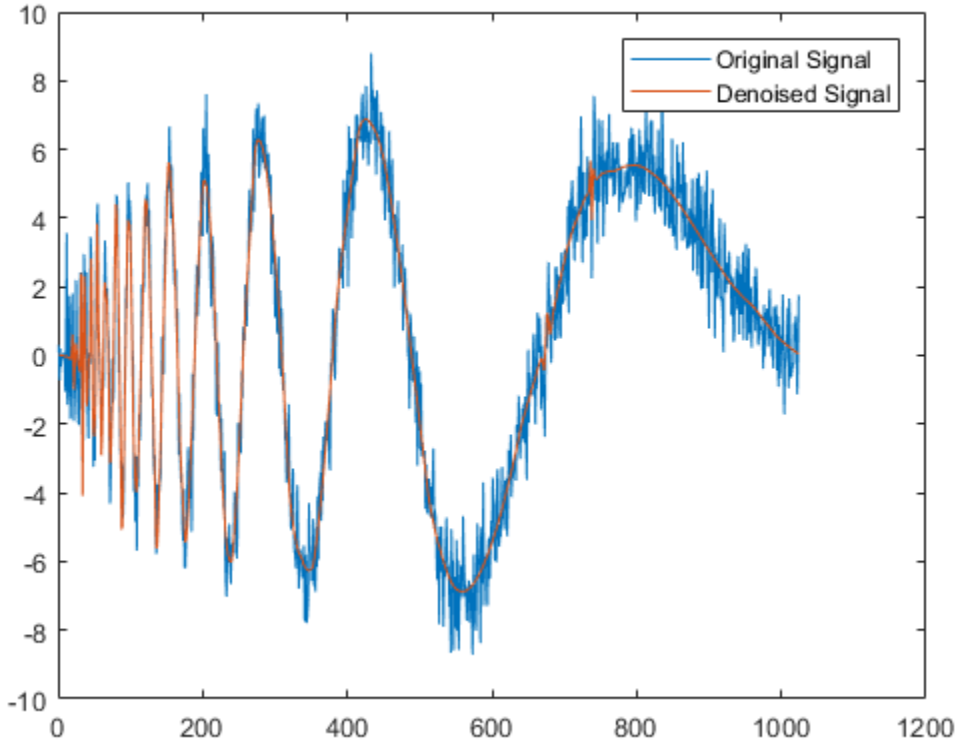
Denoise A Signal Using Default Values

Obtain the denoised version of a noisy signal using default values.

```
load noisdopp
xden = wdenoise(noisdopp);
```

Plot the original and denoised signals.

```
plot([noisdopp' xden'])
legend('Original Signal','Denoised Signal')
```



Denoise a Timetable Using Block Thresholding

Denoise a timetable of noisy data down to level 5 using block thresholding.

Load a noisy dataset.

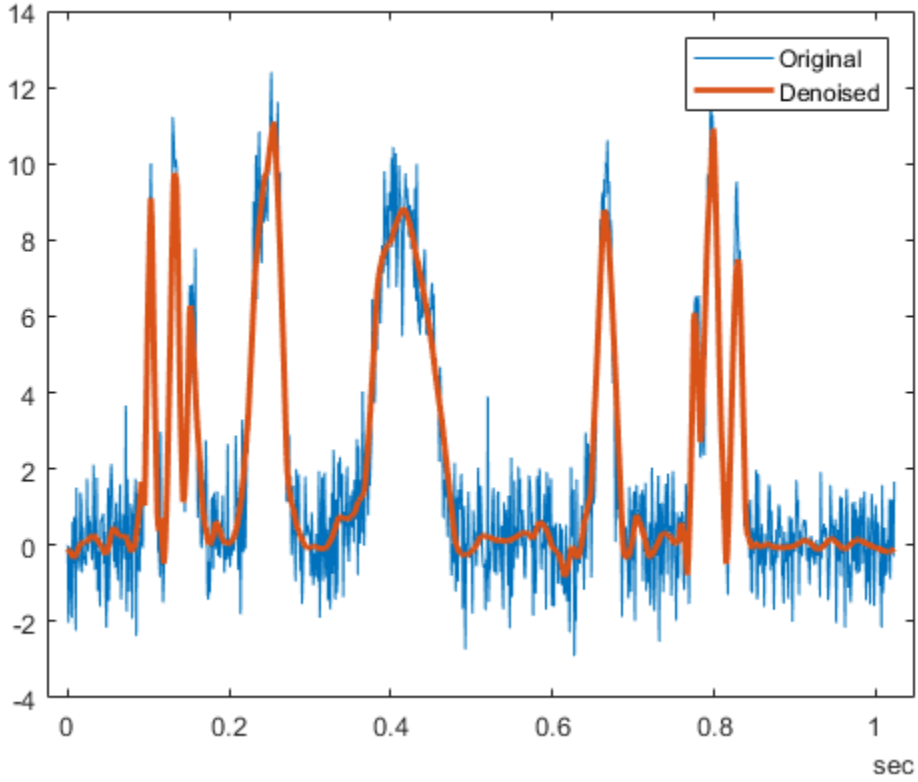
```
load wnoisydata
```

Denoise the data down to level 5 using block thresholding by setting the name-value pair `'DenoisingMethod','BlockJS'`.

```
xden = wdenoise(wnoisydata,5,'DenoisingMethod','BlockJS');
```

Plot the original data and the denoised data.

```
h1 = plot(wnoisydata.t,[wnoisydata.noisydata(:,1) xden.noisydata(:,1)]);
h1(2).LineWidth = 2;
legend('Original','Denoised')
```

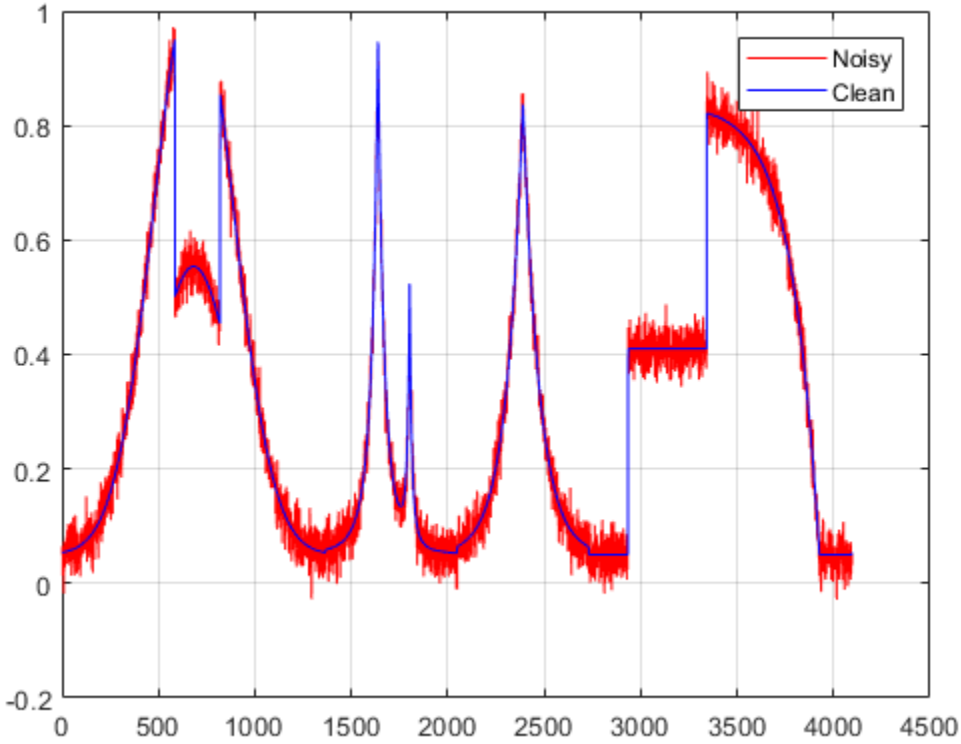


Compare Denoised Signals

Denoise a signal in different ways and compare results.

Load a datafile that contains clean and noisy versions of a signal. Plot the signals.

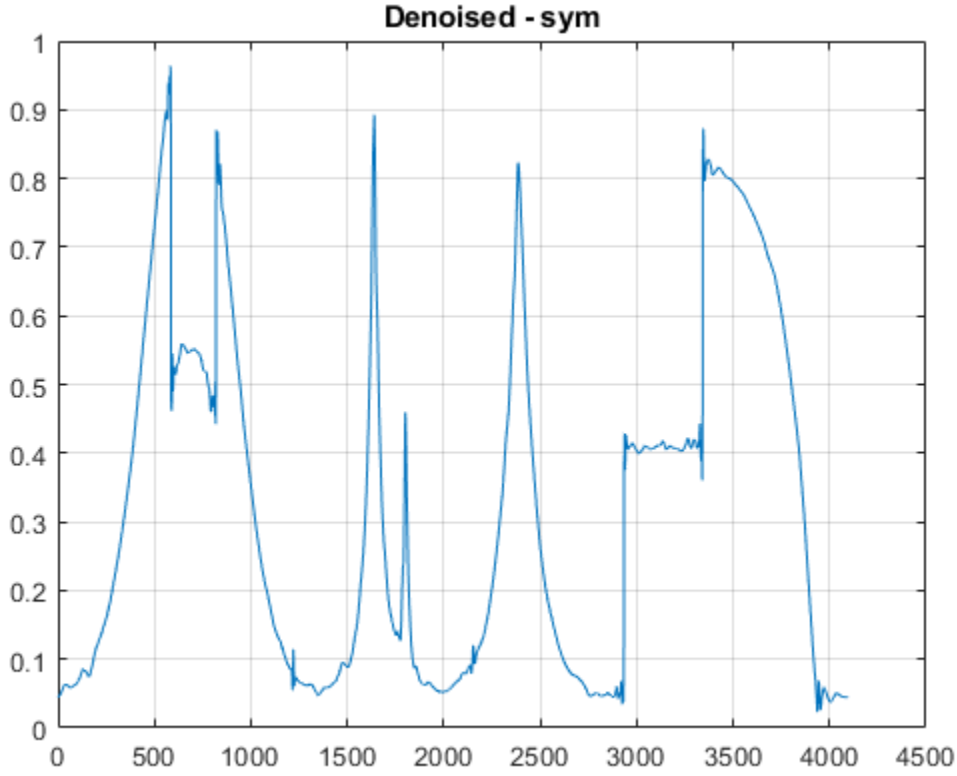
```
load fdata.mat
plot(fNoisy,'r-')
hold on
plot(fClean,'b-')
grid on
legend('Noisy','Clean');
```



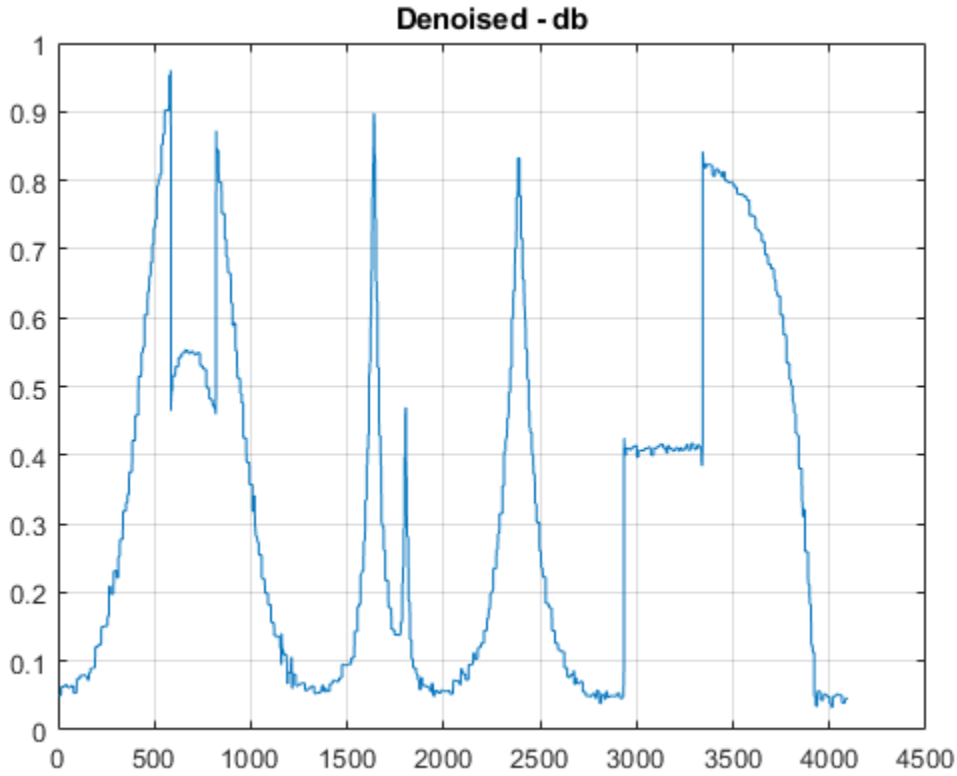
Denoise the signal using the `sym4` and `db1` wavelets, with a nine-level wavelet decomposition. Plot the results.

```
cleansym = wdenoise(fNoisy,9,'Wavelet','sym4');
cleandb = wdenoise(fNoisy,9,'Wavelet','db1');
figure
plot(cleansym)
```

```
title('Denoised - sym')
grid on
```




```
figure
plot(cleandb)
title('Denoised - db')
grid on
```



Compute the SNR of each denoised signal. Confirm that using the `sym4` wavelet produces a better result.

```
snrsym = -20*log10(norm(abs(fClean-cleansym))/norm(fClean))
```

```
snrsym = 35.3294
```

```
snrdb = -20*log10(norm(abs(fClean-cleandb))/norm(fClean))
```

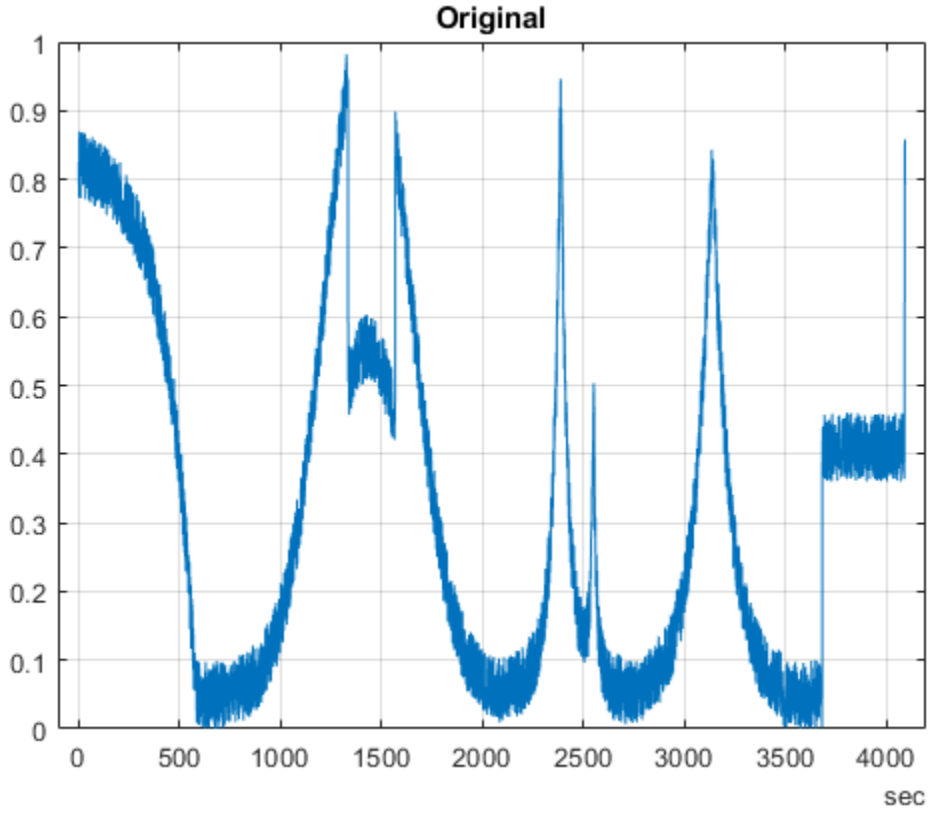
```
snrdb = 32.2672
```

Load in a file which contains noisy data of 100 time series. Every time series is a noisy version of `fClean`. Denoise the time series twice, estimating the noise variance differently in each case.

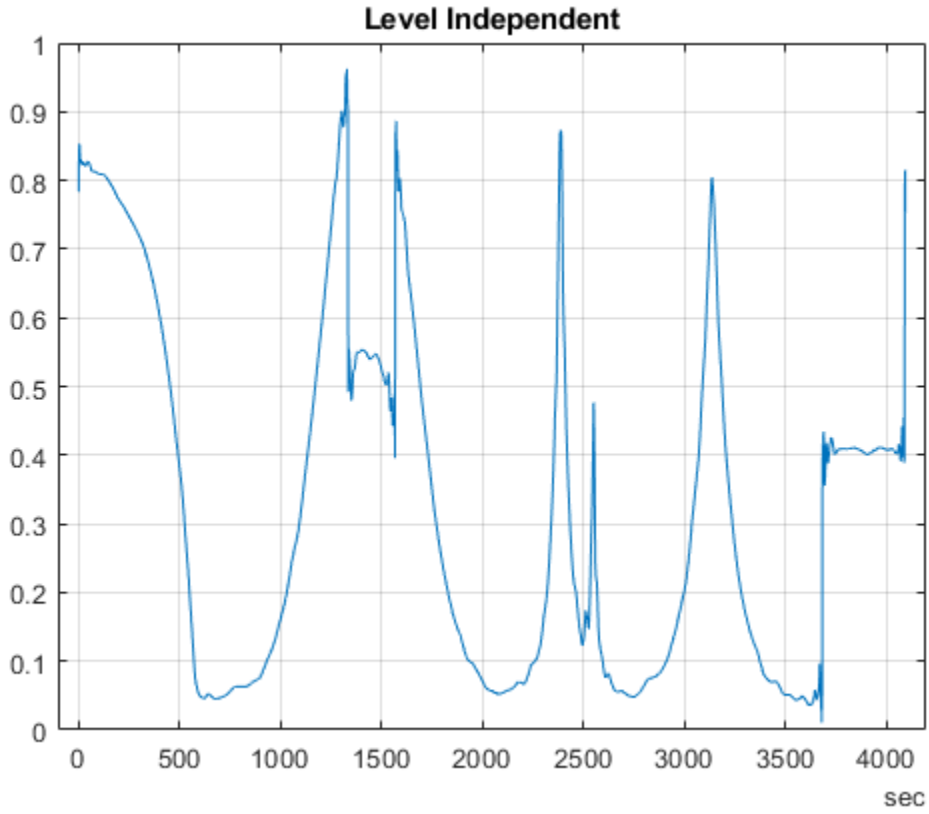
```
load fdataTS.mat
cleanTSld = wdenoise(fdataTS,9,'NoiseEstimate','LevelDependent');
cleanTSli = wdenoise(fdataTS,9,'NoiseEstimate','LevelIndependent');
```

Compare one of the noisy time series with its two denoised versions.

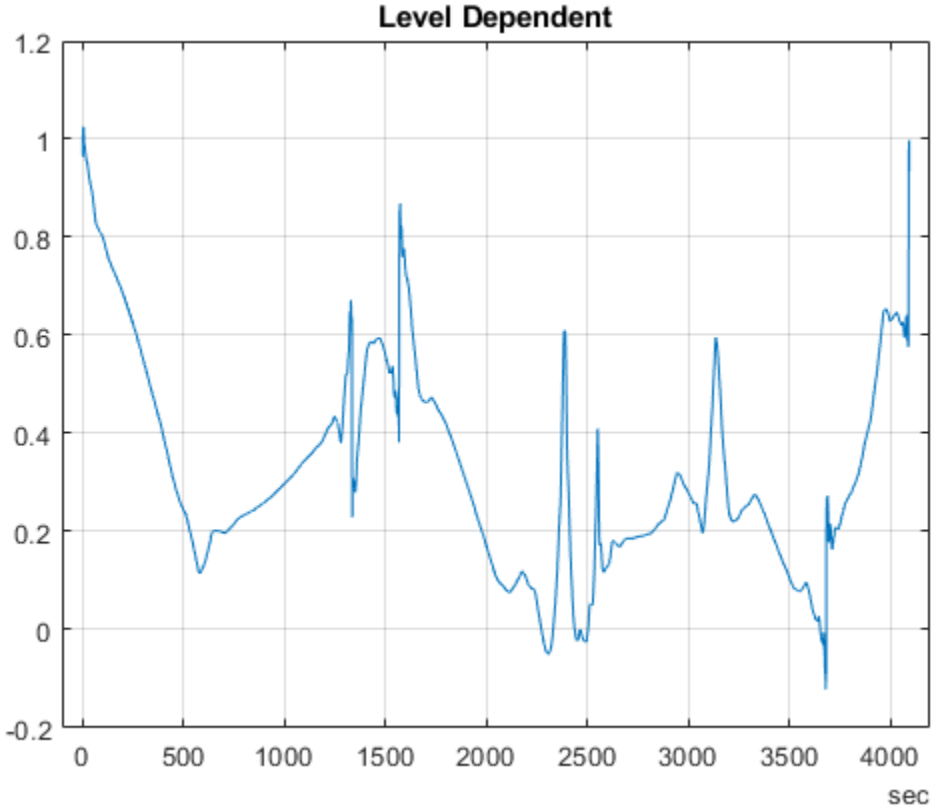
```
figure
plot(fdataTS.Time,fdataTS.fTS15)
title('Original')
grid on
```




```
figure
plot(cleanTSli.Time,cleanTSli.fTS15)
title('Level Independent')
grid on
```




```
figure
plot(cleanTSld.Time,cleanTSld.fTS15)
title('Level Dependent')
grid on
```



Input Arguments

x — Input data

vector | matrix | timetable

Input data, specified as a matrix, vector, or timetable of real values. If `X` is a vector, it must have at least two samples. If `X` is a matrix or timetable, it must have at least two rows.

Data Types: `double`

`LEVEL` — Level of wavelet decomposition

positive integer

Level of wavelet decomposition, specified as a positive integer. `LEVEL` is a positive integer less than or equal to `floor(log2N)` where N is the number of samples in the data.

- If unspecified, `LEVEL` defaults to the minimum of `floor(log2`N`)` and `wmaxlev(N,'sym4')`.
- For James-Stein block thresholding, `'BlockJS'`, there must be `floor(log2N)` coefficients at the coarsest resolution level, `LEVEL`.

Data Types: `double`

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','db6','DenoisingMethod','Bayes'` denoises using the Daubechies `db6` wavelet and the empirical Bayesian method.

`Wavelet` — Name of wavelet

'sym4' (default) | character array

Name of wavelet, specified as a character array, to use for denoising. The wavelet must be orthogonal or biorthogonal. Orthogonal and biorthogonal wavelets are designated as type 1 and type 2 wavelets respectively in the wavelet manager, `wavemngr`.

- Valid built-in orthogonal wavelet families begin with `haar`, `dbN`, `fkN`, `coifN`, or `symN` where `N` is the number of vanishing moments for all families except `fk`. For `fk`, `N` is the number of filter coefficients.
- Valid biorthgonal wavelet families begin with `'biorNr.Nd'` or `'rbioNd.Nr'`, where `Nr` and `Nd` are the number of vanishing moments in the reconstruction (synthesis) and decomposition (analysis) wavelet.

Determine valid values for the vanishing moments by using `waveinfo` with the wavelet family short name. For example, enter `waveinfo('db')` or `waveinfo('bior')`. Use

`wavemngr('type',WNAME)` to determine if a wavelet is orthogonal (returns 1) or biorthogonal (returns 2).

`DenoisingMethod` — Denoising method

'Bayes' (default) | 'BlockJS' | 'FDR' | 'Minimax' | 'SURE' | 'UniversalThreshold'

Denoising method, specified as a character array, used to determine the denoising thresholds for the data `X`.

- `Bayes` - Empirical Bayes

  This method uses a threshold rule based on assuming measurements have independent prior distributions given by a mixture model. Because measurements are used to estimate the weight in the mixture model, the method tends to work better with more samples. By default, the posterior median rule is used to measure risk [8].

- `BlockJS` - Block James-Stein

  This method is based on determining an `optimal block size and threshold. The resulting block thresholding estimator yields simultaneously optimal global and local adaptivity [3].

- `FDR` - False Discovery Rate

  This method uses a threshold rule based on controlling the expected ratio of false positive detections to all positive detections. The `FDR` method works best with sparse data. Choosing a ratio, or Q-value, less than `1/2` yields an asymptotically minimax estimator [1].

- `Minimax` - Minimax Estimation

  This method uses a fixed threshold chosen to yield minimax performance for mean square error against an ideal procedure. The minimax principle is used in statistics to design estimators. See `thselect` for more information.

- `SURE` - Stein's Unbiased Risk Estimate

  This method uses a threshold selection rule based on Stein's 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.

- `UniversalThreshold` - Universal Threshold $\sqrt{2\ln(\bullet)}$

This method uses a fixed-form threshold yielding minimax performance multiplied by a small factor proportional to `log(length(X))`.

Note For `'FDR'`, there is an optional argument for the Q-value, which is the proportion of false positives. Q is a real-valued scalar between `0` and `1/2`, `0 < Q <= 1/2`. To specify `'FDR'` with a Q-value, use a cell array where the second element is the Q-value. For example, `'DenoisingMethod',{'FDR',0.01}`. If unspecified, Q defaults to `0.05`.

`ThresholdRule` — Threshold rule

character array

Threshold rule, specified as a character array, to use to shrink the wavelet coefficients. `'ThresholdRule'` is valid for all denoising methods, but the valid options and defaults depend on the denoising method. Rules possible for different denoising methods are specified as follows:

- `'BlockJS'`: The only supported option is `'James-Stein'`. You do not need to specify `'ThresholdRule'` for `'BlockJS'`.
- `'SURE'`, `'Minimax'`, `'UniversalThreshold'`: Valid options are `'Soft'` or `'Hard'`. The default is `'Soft'`.
- `'Bayes'`: Valid options are `'Median'`, `'Mean'`, `'Soft'`, or `'Hard'`. The default is `'Median'`.
- `'FDR'`: The only supported option is `'Hard'`. You do not need to define `'ThresholdRule'` for `'FDR'`

`NoiseEstimate` — Method of estimating variance of noise

`'LevelIndependent'` (default) | `'LevelDependent'`

Method of estimating variance of noise in the data, specified as a character array. Valid options are `'LevelIndependent'` and `'LevelDependent'`.

- `'LevelIndependent'` estimates the variance of the noise based on the finest-scale (highest-resolution) wavelet coefficients.
- `'LevelDependent'` estimates the variance of the noise based on the wavelet coefficients at each resolution level.
- For the block James-Stein estimator (`'BlockJS'`), `'LevelIndependent'` is the only supported option.

Output Arguments

`XDEN` — Denoised data
vector | matrix | timetable

Denoised vector, matrix, or timetable version of `X`. For timetable input, `XDEN` has the same variable names and timestamps as the original timetable.

Data Types: `double`

`DENOISEDCFS` — Denoised wavelet and scaling coefficients
cell array

Denoised wavelet and scaling coefficients of the denoised data `XDEN`, returned in a cell array. The elements of `DENOISEDCFS` are in order of decreasing resolution. The final element of `DENOISEDCFS` contains the approximation (scaling) coefficients.

Data Types: `double`

`ORIGCFS` — Original wavelet and scaling coefficients
cell array

Original wavelet and scaling coefficients of the data `X`, returned in a cell array. The elements of `ORIGCFS` are in order of decreasing resolution. The final element of `ORIGCFS` contains the approximation (scaling) coefficients.

Data Types: `double`

Algorithms

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 σ is 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 "Wavelet Denoising and Nonparametric Function Estimation", in the User's Guide, and in the help of the `thselect` function.

References

[1] Abramovich, F., Y. Benjamini, D.L. Donoho, I.M. Johnstone (2006), "Adapting to Unknown Sparsity by Controlling the False Discovery Rate," *Ann. Statist.*, 34(2), pp. 584–653.

[2] Antoniadis, A., G. Oppenheim, Eds. (1995), *Wavelets and Statistics*, 103, Lecture Notes in Statistics, Springer Verlag.

[3] Cai, T.T. (2002), "On Block Thresholding in Wavelet Regression: Adaptivity, Block size, and Threshold Level," *Statistica Sinica*, 12, pp. 1241–1273.

[4] 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.

[5] Donoho, D.L., I.M. Johnstone (1994), "Ideal Spatial Adaptation by Wavelet Shrinkage," *Biometrika*, Vol. 81, pp. 425–455.

[6] Donoho, D.L. (1995), "De-noising by soft-thresholding," *IEEE Trans. on Inf. Theory*, 42 3, pp. 613– 627.

[7] 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.

[8] Johnstone, I.M., B.W. Silverman (2004), "Needles and Straw in Haystacks: Empirical Bayes Estimates of Possibly Sparse Sequences," *Ann. Statist.* 32(4), pp. 1594-1649.

See Also

Wavelet Signal Denoiser | `thselect` | `waveinfo` | `wavemngr` | `wmaxlev`

Introduced in R2017b

wenergy

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
```

Introduced before R2006a

wenergy2

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
```

Introduced before R2006a

wentropy

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 character vector 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'`	Character vector	`P` is a character vector containing the file name of your own entropy function, with a single input `X`.

Entropy Type Name (*T*)	Parameter (P)	Comments
`FunName`	No constraints on `P`	`FunName` is any other character vector 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.

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.

  $$E1(s_i) = s_i^2 \log(s_i^2)$$

  so,

  $$E1(s) = -\sum_i s_i^2 \log(s_i^2)$$

  ,

with the convention $0log(0) = 0$.

- The concentration in l^p norm entropy with $1 \le p$.

  $E2(s_i) = |s_i|^p$ so $E2(s) = \sum_i |s_i|^p = \|s\|_p^p$

- The "log energy" entropy.

  $E3(s_i) = \log(s_i^2)$

  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 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 Denoising" 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.

Introduced before R2006a

wextend

Extend vector or matrix

Syntax

```
YEXT= wextend(TYPE,MODE,X,LEN)
YEXT = wextend(___,LOC)
```

Description

`YEXT= wextend(TYPE,MODE,X,LEN)` extends real-valued input vector or matrix `X` by length `LEN`, using the `TYPE` method and `MODE` extension. The `TYPE` specifies the dimension of the extension. The `MODE` specifies the rule to apply to fill in values in the extension.

`YEXT = wextend(___,LOC)` also specifies the location of the extension.

Examples

Extending Vectors and Matrices

Extend Vector

Extend a vector using a number of different methods.

Create a vector and set the extension length to 2.

```
len = 2;
x = [1 2 3]
```

```
x =

     1     2     3
```

Perform a zero-pad extension. To verify that different forms of the input arguments are possible, perform this extension twice. The result is the same both times.

```
xextzpd1 = wextend('1','zpd',x,len)
```

```
xextzpd1 =

     0     0     1     2     3     0     0
```

```
xextzpd2 = wextend('1D','zpd',x,len,'b')
```

```
xextzpd2 =

     0     0     1     2     3     0     0
```

Perform a half-point symmetric extension.

```
xextsym = wextend('1D','sym',x,len)
```

```
xextsym =

     2     1     1     2     3     3     2
```

Perform a periodic extension. Since the input vector is of odd length, wextend appends an extra example to the end before extending using the 'ppd' mode. This sample is equal to the last value on the right.

```
xextper = wextend('1D','per',x,len)
```

```
xextper =

     3     3     1     2     3     3     1     2
```

Extend Matrix

Extend a small matrix using a number of different methods.

Create a matrix and set the extension length to 2.

```
len = 2;
X = [1 2 3; 4 5 6]
```

```
X =

     1     2     3
     4     5     6
```

Perform a zero-pad extension of the array.

```
Xextzpd = wextend(2,'zpd',X,len)

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
```

Perform a half-point symmetric extension of the array.

```
Xextsym = wextend('2D','sym',X,len)

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
```

Extend `uint8` Data Beyond Range Limits

Observe the effects of symmetric, antisymmetric, and smooth extensions on a `uint8` vector when values are at or near the limits of the data type's range.

Symmetric Extensions

The smallest `uint8` integer is 0, and the largest is 255. Create a vector of `uint8` integers that includes those limits.

```
dataVector = uint8([0 1 2 253 254 255])
```

```
dataVector = 1x6 uint8 row vector

     0     1     2   253   254   255
```

Obtain whole-point and half-point symmetric extensions of the vector. Extend the vector by two values on the left and right.

```
wholePointSym = wextend('1','symw',dataVector,2)
```

```
wholePointSym = 1x10 uint8 row vector

     2     1     0     1     2   253   254   255   254   253
```

```
halfPointSym = wextend('1','symh',dataVector,2)
```

```
halfPointSym = 1x10 uint8 row vector

     1     0     0     1     2   253   254   255   255   254
```

Extending symmetrically never results in values outside the `uint8` range.

Antisymmetric Extensions

Create a type `double` copy of the vector, and then obtain a whole-point antisymmetric extension of the copy. The extension includes negative values and values greater than 255.

```
dataVectorDouble = double(dataVector);
wholePointAsymDouble = wextend('1','asymw',dataVectorDouble,2)
```

```
wholePointAsymDouble =

    -2    -1     0     1     2   253   254   255   256   257
```

Obtain a whole-point antisymmetric extension of the original `uint8` vector. Values outside the `uint8` range are mapped to the closest `uint8` integer, which is 0 for negative values and 255 for values greater than 255.

```
wholePointAsym = wextend('1','asymw',dataVector,2)
```

```
wholePointAsym = 1x10 uint8 row vector

     0     0     0     1     2   253   254   255   255   255
```

Now obtain half-point antisymmetric extensions of the `double` copy and the original `uint8` vector.

```
halfPointAsymDouble = wextend('1','asymh',dataVectorDouble,2)

halfPointAsymDouble =

    -1     0     0     1     2   253   254   255  -255  -254
```

```
halfPointAsym = wextend('1','asymh',dataVector,2)

halfPointAsym = 1x10 uint8 row vector

     0     0     0     1     2   253   254   255     0     0
```

As with the whole-point antisymmetric extension, negative values in the extended `uint8` data are mapped to 0.

Smooth Extensions

Obtain order-0 smooth extensions of the `double` copy and the original `uint8` vector.

```
smooth0Double = wextend('1','sp0',dataVectorDouble,2)

smooth0Double =

     0     0     0     1     2   253   254   255   255   255
```

```
smooth0 = wextend('1','sp0',dataVector,2)

smooth0 = 1x10 uint8 row vector

     0     0     0     1     2   253   254   255   255   255
```

Results are identical. Next, obtain an order-1 smooth extension of each vector.

```
smooth1Double = wextend('1','sp1',dataVectorDouble,2)
```

```
smooth1Double =

    -2    -1     0     1     2   253   254   255   256   257

smooth1 = wextend('1','sp1',dataVector,2)

smooth1 = 1x10 uint8 row vector

     0     0     0     1     2   253   254   255   255   255
```

The values in the `double` result that are outside the `uint8` range are mapped to the closest `uint8` values in the `uint8` extension.

Extend `int8` Data Beyond Range Limits

Observe the effects of symmetric, antisymmetric, and smooth extensions of `int8` data when values are at or near the limits of the data type's range.

Symmetric Extensions

The smallest `int8` integer is -128, and the largest is 127. Create a vector of `int8` integers that includes those limits.

```
dataVector = int8([-128 -127 -126 125 126 127])

dataVector = 1x6 int8 row vector

   -128   -127   -126    125    126    127
```

Obtain whole-point and half-point symmetric extensions of the data. Extend the vector by two values on the left and right.

```
wholePointSym = wextend('1','symw',dataVector,2)

wholePointSym = 1x10 int8 row vector

   -126   -127   -128   -127   -126    125    126    127    126    125

halfPointSym = wextend('1','symh',dataVector,2)
```

```
halfPointSym = 1x10 int8 row vector

   -127   -128   -128   -127   -126    125    126    127    127    126
```

Extending symmetrically never results in values outside the `int8` range.

Antisymmetric Extensions

Create a type `double` copy of the vector, and then obtain a whole-point antisymmetric extension of the copy. The extension includes negative values less than -128 and values greater than 127.

```
dataVectorDouble = double(dataVector);
wholePointsAsymDouble = wextend('1','asymw',dataVectorDouble,2)

wholePointsAsymDouble =

  -130  -129  -128  -127  -126   125   126   127   128   129
```

Obtain a whole-point antisymmetric extension of the original `int8` vector. Values outside the `int8` range are mapped to the closest `int8` integer, which is -128 for values less than -128 and 127 for values greater than 127.

```
wholePointAsym = wextend('1','asymw',dataVector,2)

wholePointAsym = 1x10 int8 row vector

   -128   -128   -128   -127   -126    125    126    127    127    127
```

Now obtain half-point antisymmetric extensions of the `double` copy and the original `int8` vector.

```
halfPointAsymDouble = wextend('1','asymh',dataVectorDouble,2)

halfPointAsymDouble =

   127   128  -128  -127  -126   125   126   127  -127  -126


halfPointAsym = wextend('1','asymh',dataVector,2)
```

```
halfPointAsym = 1x10 int8 row vector

    127    127   -128   -127   -126    125    126    127   -127   -126
```

In the `double` result, the first value is 127, which can be represented as an `int8` integer. The second value is 128, which cannot be represented as an `int8` integer. Therefore, in the `int8` result, it is being mapped to 127. The remaining values in the type `double` result can all be represented as `int8` integers.

Smooth Extensions

Obtain order-0 smooth extensions of the `double` copy and the original `int8` vector.

```
smooth0Double = wextend('1','sp0',dataVectorDouble,2)

smooth0Double =

  -128  -128  -128  -127  -126   125   126   127   127   127


smooth0 = wextend('1','sp0',dataVector,2)

smooth0 = 1x10 int8 row vector

   -128   -128   -128   -127   -126    125    126    127    127    127
```

The results are identical. Now obtain an order-1 smooth extension of each vector.

```
smooth1Double = wextend('1','sp1',dataVectorDouble,2)

smooth1Double =

  -130  -129  -128  -127  -126   125   126   127   128   129


smooth1 = wextend('1','sp1',dataVector,2)

smooth1 = 1x10 int8 row vector

   -128   -128   -128   -127   -126    125    126    127    127    127
```

The values in the `double` result outside the `int8` range are mapped to the closest `int8` values in the `int8` extension.

Input Arguments

`TYPE` — Extension method

`1 | '1' | '1d' | '1D' | 2 | '2' | '2d' | '2D' | 'ar' | 'addrow' | 'ac' | 'addcol'`

Extension method used on the input, specified as one of the values listed here.

`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

Data Types: `double` | `char`

`MODE` — Specific extension

`'zpd' | 'sp0' | 'spd' | 'sp1' | 'sym' | 'symh' | 'symw' | 'asym' | 'asymh' | 'asymw' | 'ppd' | 'per'`

Specific extension method to use to extend the input, specified as one of the values listed here.

`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

MODE	Description
'asymw'	Antisymmetric padding (whole point): boundary value antisymmetric replication
'ppd'	Periodized extension (1)
'per'	Periodized extension (2) If the signal length is odd, wextend adds to the right an extra sample that is equal to the last value, and performs the extension using the 'ppd' mode. Otherwise, 'per' reduces to 'ppd'. This rule also applies to images.

For more information on symmetric extension modes, see [1].

Note The extension modes 'sp0' and 'spd' (or 'sp1') cast the data internally to double precision before performing the extension. For integer data types, wextend warns if the conversion to double causes a loss of precision or the requested extension results in integers beyond the range where double precision numbers can represent consecutive integers exactly.

Data Types: char

x — Input data

real-valued vector or matrix

Input data, specified as a real-valued vector or matrix.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

LEN — Amount by which to extend the input

positive integer | two-element vector

Amount by which to extend the input, specified as a positive integer or two-element vector of positive integers. You can extend a matrix by expressing LEN as [LROW,LCOL], where LROW is the number of rows to add and LCOL is the number of columns to add. You can perform a 2-D extension of a matrix by the same amount in both directions by specifying LEN as single integer.

Data Types: `single` | `double`

LOC — Location of extension

`'l'` | `'u'` | `'r'` | `'d'` | `'b'` | two-character array

Location of the extension, specified as `'l'`, `'u'`, `'r'`, `'d'`. `'b'`, or a two-character array. You can extend a vector on one side or both sides. You can extend a matrix by adding rows or columns to any combinations of sides. The options for `LOC` are as follows:

- `'l'`—Extension left
- `'u'`—Extension up
- `'r'`—Extension right
- `'d'`—Extension down
- `'b'`—Extension on both sides

The valid and default values for `LOC`, and the behavior of `LEN`, depend on the specified `TYPE`.

TYPE	LOC
`1`, `'1'`, `1d'` or `'1D'`	`'l'`, `'u'`, `'r'`, `'d'`, or `'b'` Example: `wextend('1D','zpd',X,3,'r')` extends input vector `X` three elements to the right. Default: `'b'` `LEN` is the length of the extension.
`2`, `'2'`, `'2d'` or `'2D'`	`[LOCROW,LOCCOL]`, where `LOCROW` and `LOCCOL` are 1-D extension locations. Example: `wextend('2D','zpd',X,[2 3],'ub')` extends input vector or matrix `X` two rows up and three columns on both sides. Default: `'bb'` `LEN`, specified as `[LROW,LCOL]`, is the number of rows and columns to add.
`'ar'` or `'addrow'`	`'l'`, `'u'`, `'r'`, `'d'`, or `'b'` Example: `wextend('addrow','zpd',X,4,'d')` extends input vector or matrix `X` four rows down. Default: `'b'` `LEN` is the number of rows to add.

TYPE	LOC
`'ac'` or `'addcol'`	`'l'`, `'u'`, `'r'`, `'d'`, or `'b'` Example: `wextend('addcol','zpd',X,1,'l')` extends input vector or matrix `X` one column to the left. Default: `'b'` `LEN` is the number of columns to add.

Data Types: `char`

Tips

- For most wavelet applications, either a periodic extension or symmetric extension works fine.

Algorithms

When a value is outside the input data type's range, `wextend` maps it to the closest value of the input data type. For examples of data being extended beyond a data type's range, see “Extend uint8 Data Beyond Range Limits” on page 1-946 and “Extend int8 Data Beyond Range Limits” on page 1-949.

References

[1] Strang, G., and T. Nguyen. *Wavelets and Filter Banks*. Wellesley, MA: Wellesley-Cambridge Press, 1996.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The generated code can return a column vector when MATLAB returns a row vector if all of the following conditions are true:

  - `TYPE` specifies a 1-D extension.
  - Input `X` is a variable-size vector.
  - Input `X` is not a variable-length row vector (1-by-:).

  Code generation does not produce a warning or error message about the shape mismatch. In the output vector that the generated code returns, the values match the values in the output vector that MATLAB returns.

  In this case, to generate code that returns a row vector, pass `X(:).'` instead of `X`.
- Input `X` must be of type `double`.

See Also

`dwtmode`

Introduced before R2006a

wfbm

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`

Introduced before R2006a

wfbmesti

Parameter estimation of fractional Brownian motion

Syntax

```
HEST = wfbmesti(X)
```

Description

`HEST = wfbmesti(X)` returns a one-by-three vector `HEST` which contains three estimates of the fractal index `H` of the input signal `X`. The signal `X` is assumed to be a realization of fractional Brownian motion with Hurst index `H`.

The first two elements of the vector are estimates based on the second derivative with the second computed in the wavelet domain.

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

```
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

Hurst Parameter Estimation

This example shows how to estimate the Hurst index of a fractional Brownian motion. The example simulates 1,000 realizations of fractional Brownian motion with H=0.6. Each realization consists of 10,000 samples. At the end of the simulation, the three estimates of the Hurst index are compared.

Initialize the random number generator for repeatable results. Set the Hurst index equal to 0.6 and the length of the realizations to be 10,000.

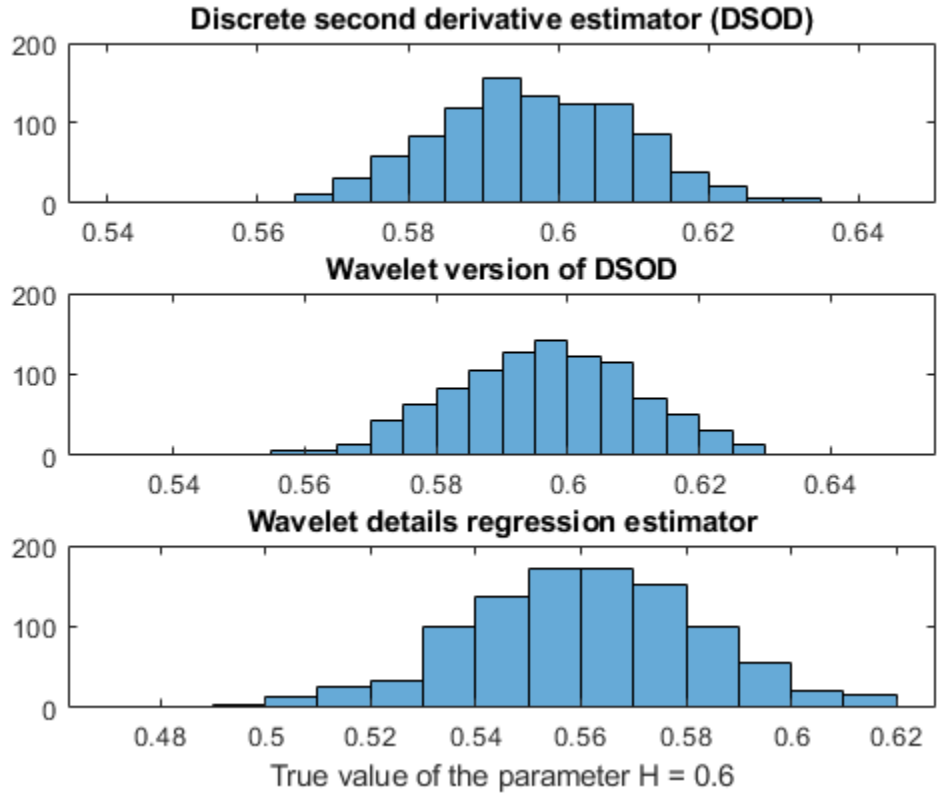
```
rng default;
H = 0.6;
len = 10000;
```

Generate 1,000 realizations of fractional Brownian motion and compute the estimates of the Hurst parameter.

```
n = 1000;
Hest = zeros(n,3);
for ii = 1:n
    fBm06 = wfbm(H,len);
    Hest(ii,:) = wfbmesti(fBm06);
end
```

Compare the estimates.

```
subplot(311), histogram(Hest(:,1));
title('Discrete second derivative estimator (DSOD)')
subplot(312), histogram(Hest(:,2));
title('Wavelet version of DSOD')
subplot(313), histogram(Hest(:,3));
title('Wavelet details regression estimator')
xlabel('True value of the parameter H = 0.6')
```



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`

Introduced before R2006a

wfilters

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 character vector `'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'`
Fejer-Korovkin filters	`'fk4'`, `'fk6'`, `'fk8'`, `'fk14'`, `'fk22'`
Discrete Meyer	`'dmey'`

Wavelet Families	Wavelets
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

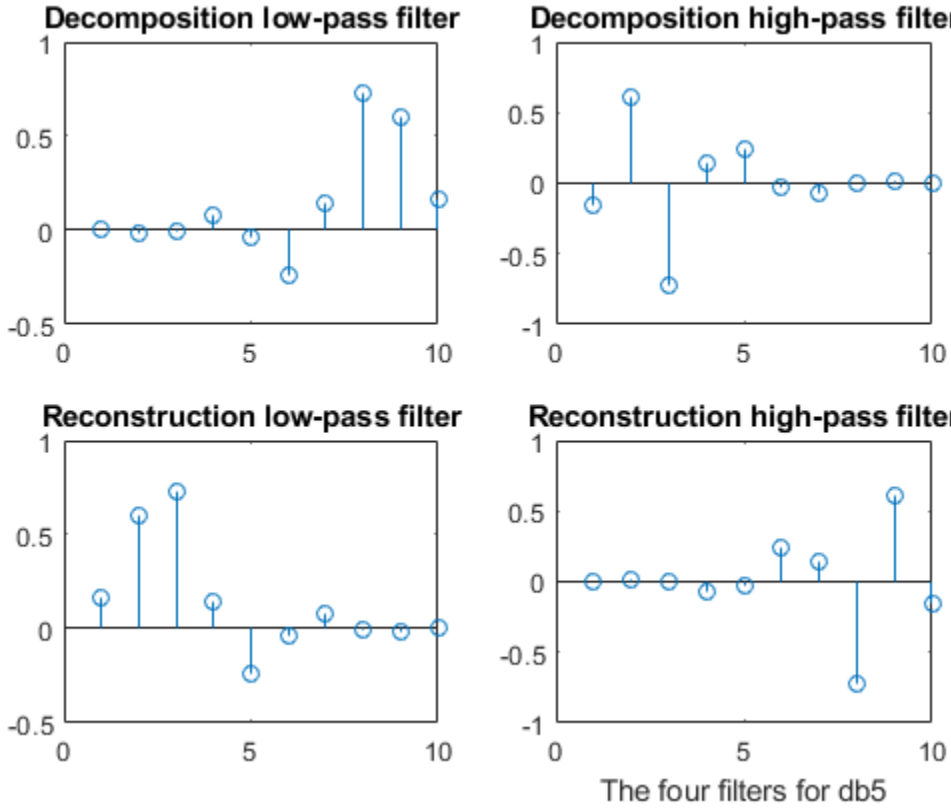
Compute Four Filters

Set the wavelet name.

```
wname = 'db5';
```

Compute the four filters associated with wavelet name given by the input character vector `wname` and plot the results.

```
[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')
```



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`

Introduced before R2006a

wfusimg

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`.

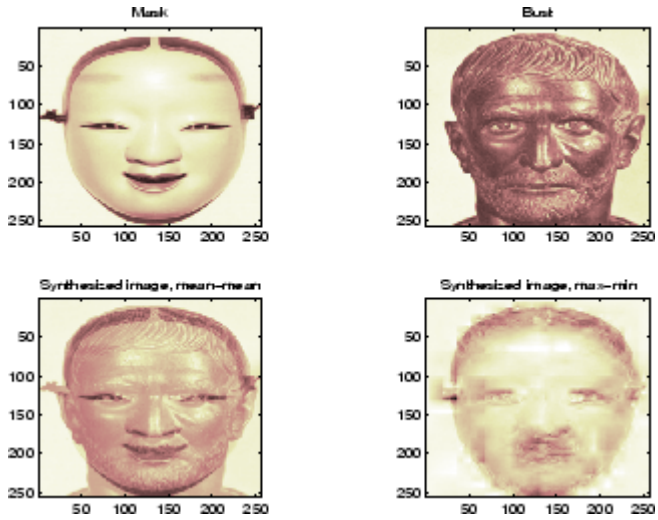
Examples

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.

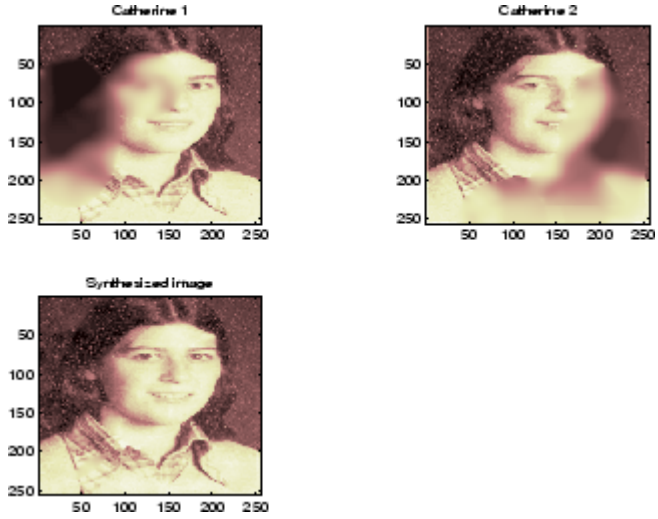
```
% Example 1: Fusion of two different images

% 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')
```

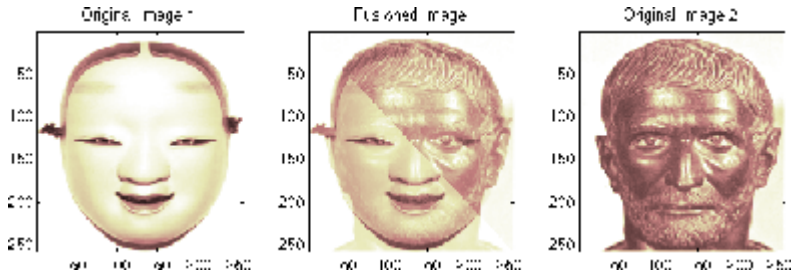



```
% Example 2: Restoration by fusion of fuzzy images

% 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);
```

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.

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

`wextend` | `wfusmat`

Introduced before R2006a

wfusmat

Fusion of two matrices or arrays

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,:) = B(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 character vector `'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.

Introduced before R2006a

wkeep

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` is equal to `'c'`, `'l'`, or `'r'`, `Y` is the central, left, or right 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
```

Introduced before R2006a

wmaxlev

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 character vector `'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

Introduced before R2006a

wmpalg

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 ℓ^2 squared norm of the expansion coefficient vector, `COEFF`, to the ℓ^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 ℓ^2 norm corresponding to the columns of `MPDICT`.

`[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 character vector. 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:

$$\phi_k(n) = \begin{cases} \dfrac{1}{\sqrt{N}} & k = 0 \\ \sqrt{\dfrac{2}{N}} \cos\left(\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right) & k = 1, 2, \ldots, N-1 \end{cases}$$

- `'sin'` Sine subdictionary. The sine subdictionary is:

$$\phi_k(t) = \sin(2\pi k t) \quad k = 1, 2, \ldots \left\lceil \frac{N}{2} \right\rceil \quad 0 \le t \le 1$$

- `'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 α_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.

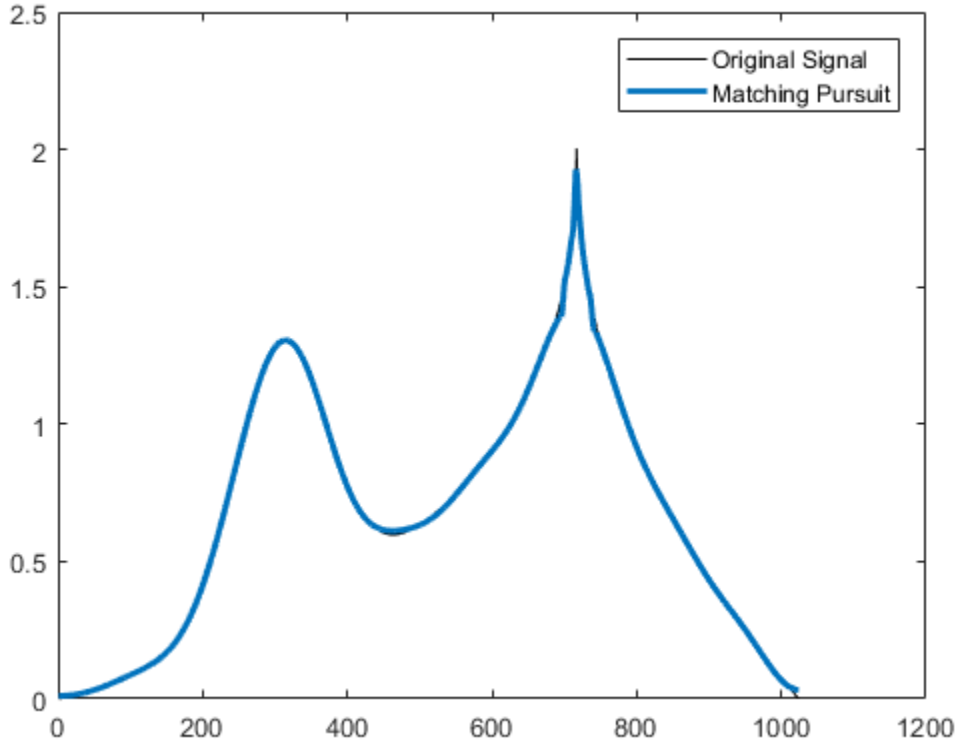
Examples

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',3);
```

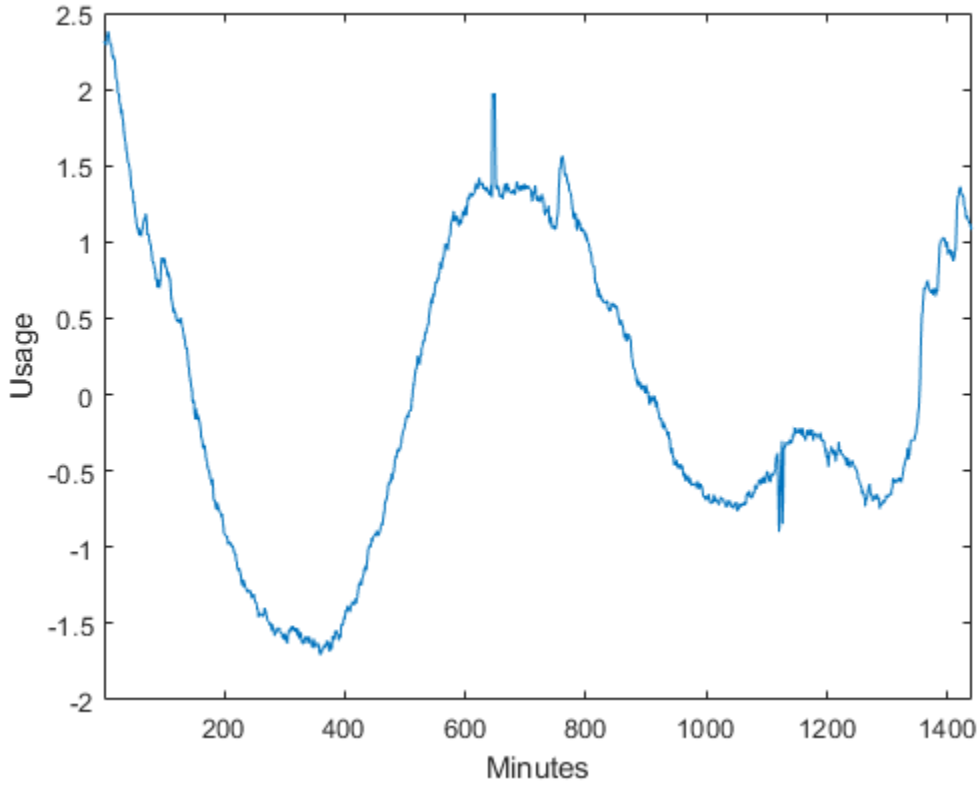
Matching Pursuit of Electricity Consumption Data

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]);
```

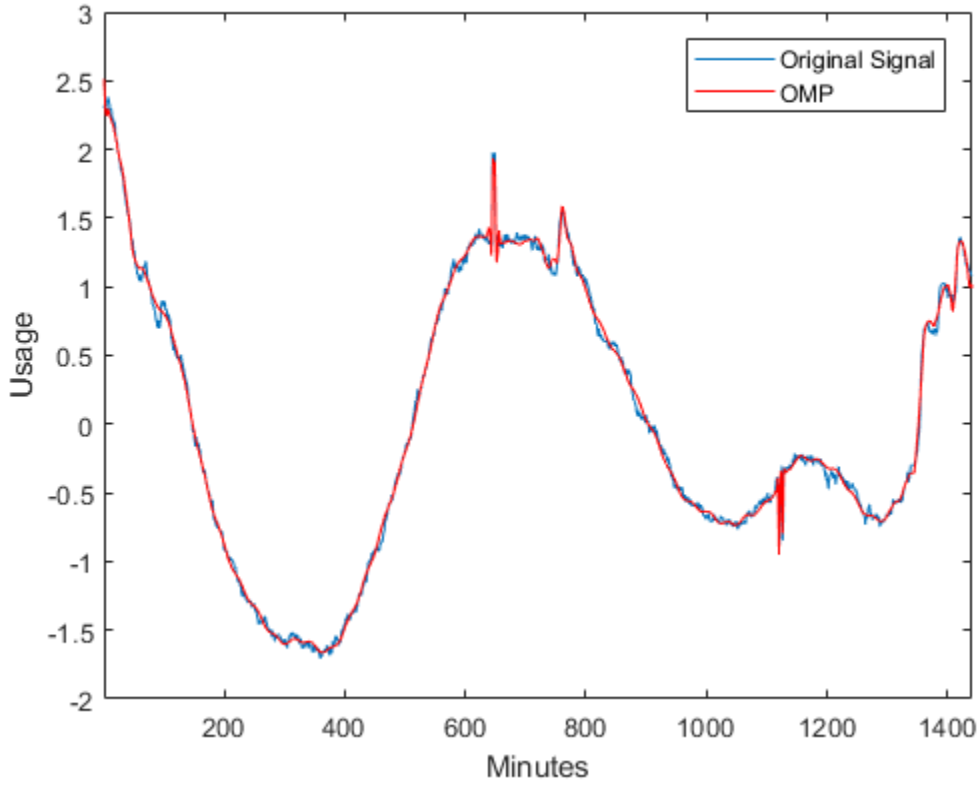


Construct a dictionary for matching pursuit consisting of the Daubechies' extremal-phase wavelet with 2 vanishing moments at level 2, the Daubechies' least-asymmetric wavelet with 4 vanishing moments at levels 1 and 4, the discrete cosine transform-II basis, and the sine basis.

```
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))
```

```
ans = 2.2204e-15
```

- “Matching Pursuit”
- “Matching Pursuit Using Wavelet Analyzer App”

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

Wavelet Analyzer | `wmpdictionary`

Topics

“Matching Pursuit”
“Matching Pursuit Using Wavelet Analyzer App”
“Matching Pursuit Algorithms”

Introduced in R2012a

wmpdictionary

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 1-995 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:

$$\phi_k(n) = \begin{cases} \dfrac{1}{\sqrt{N}} & k = 0 \\ \sqrt{\dfrac{2}{N}} \cos\left(\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right) & k = 1, 2, \ldots, N-1 \end{cases}$$

- `'sin'` Sine subdictionary. The sine subdictionary is

  $\phi_k(t) = \sin(2\pi kt) \quad k = 1, 2, \ldots \left\lceil \frac{N}{2} \right\rceil \quad 0 \le t \le 1$

  where t is a linearly-spaced N-point vector.
- `'cos'` Cosine subdictionary. The cosine subdictionary is

  $\phi_k(t) = \cos(2\pi kt) \quad k = 1, 2, \ldots \left\lceil \frac{N}{2} \right\rceil \quad 0 \le t \le 1$

  where t is a linearly-spaced N-point vector.
- `'poly'` Polynomial subdictionary. The polynomial subdictionary is:

  $p_n(t) = t^{n-1} \quad n = 1, 2, \ldots 20 \qquad 0 \le t \le 1$

  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 1-995 for an example using `LONGS`.

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.

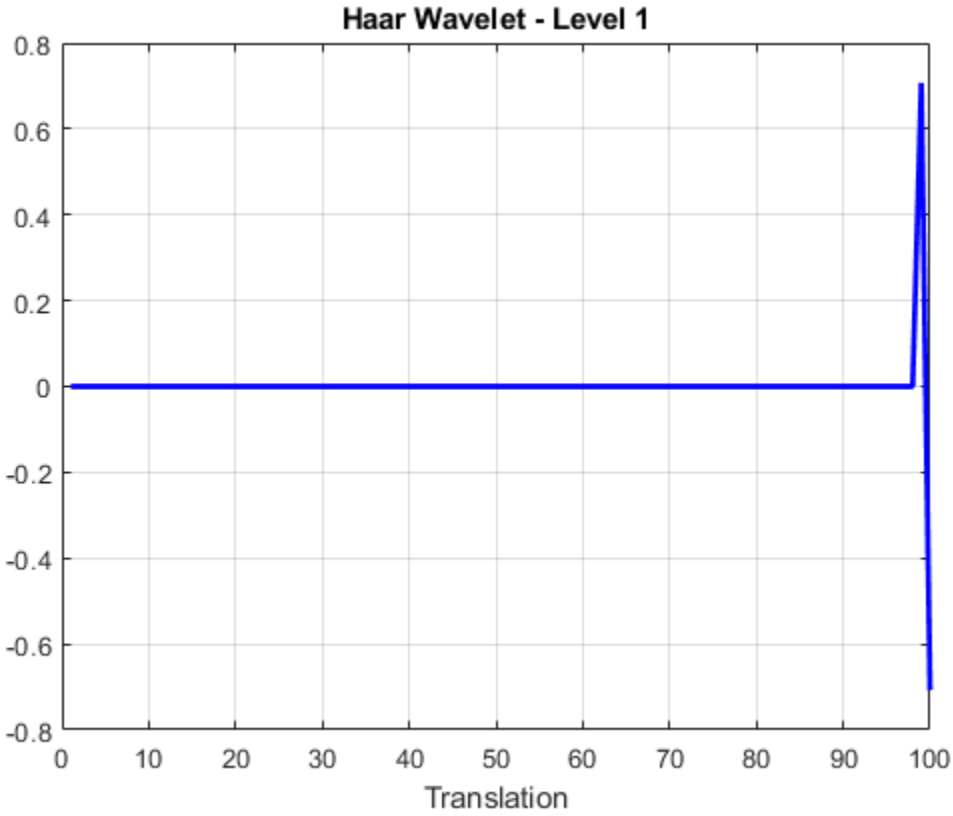
```
[mpdict,nbvect] = wmpdictionary(100,'lstcpt',{{'wphaar',2},'dct'});
```

Visualize Haar Wavelet Dictionary

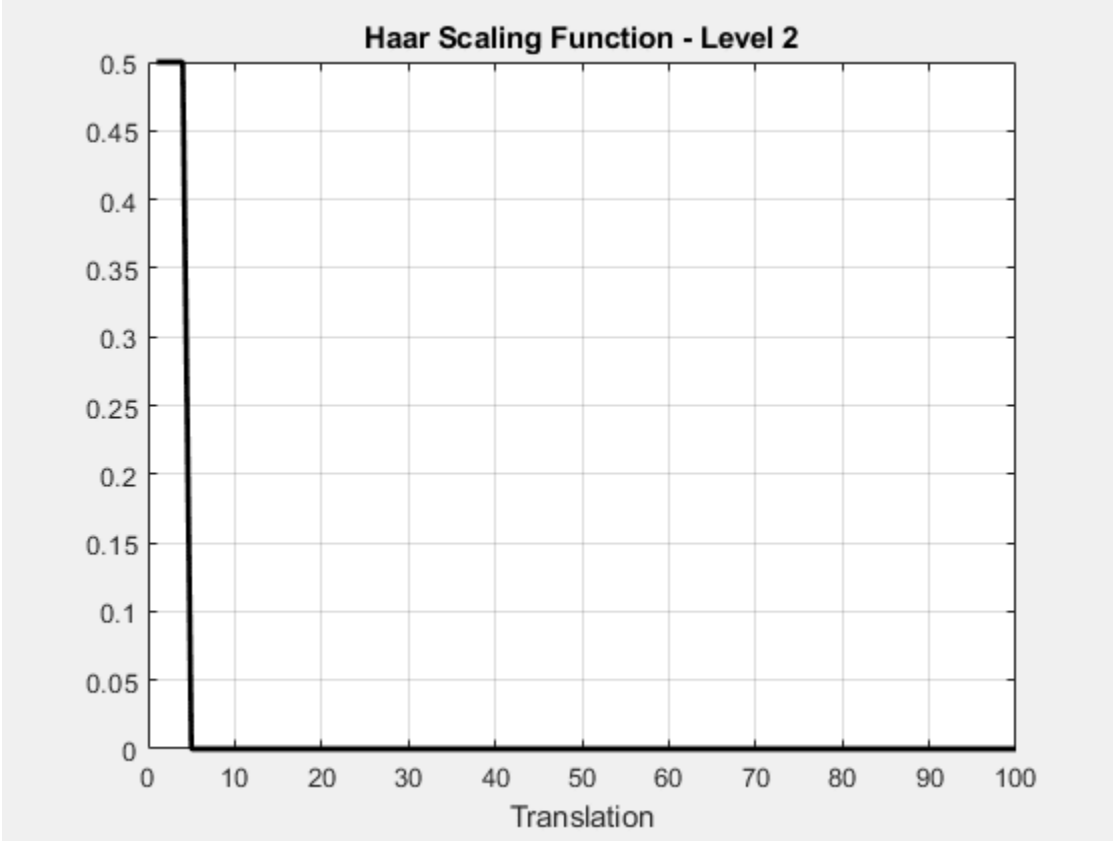
Use the output argument, `LONGS`, to visualize a dictionary. Create a Haar wavelet dictionary consisting of level-2 scaling functions, and level-1 and level-2 wavelet functions.

```
[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
        plot(mpdict(:,nn),'b','linewidth',2)
        grid on
        xlabel('Translation')
```

```
            title('Haar Wavelet - Level 1')
        end

        pause(0.2)

    end
```



Evaluating the code in the Command Window will plot each member of the dictionary once. The animation below infinitely loops through all the plots.



- “Matching Pursuit”
- “Matching Pursuit Using Wavelet Analyzer App”

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.

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

Wavelet Analyzer | `wmpalg`

Topics

“Matching Pursuit”
“Matching Pursuit Using Wavelet Analyzer App”
“Matching Pursuit Algorithms”

Introduced in R2012a

wmspca

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 <= 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`

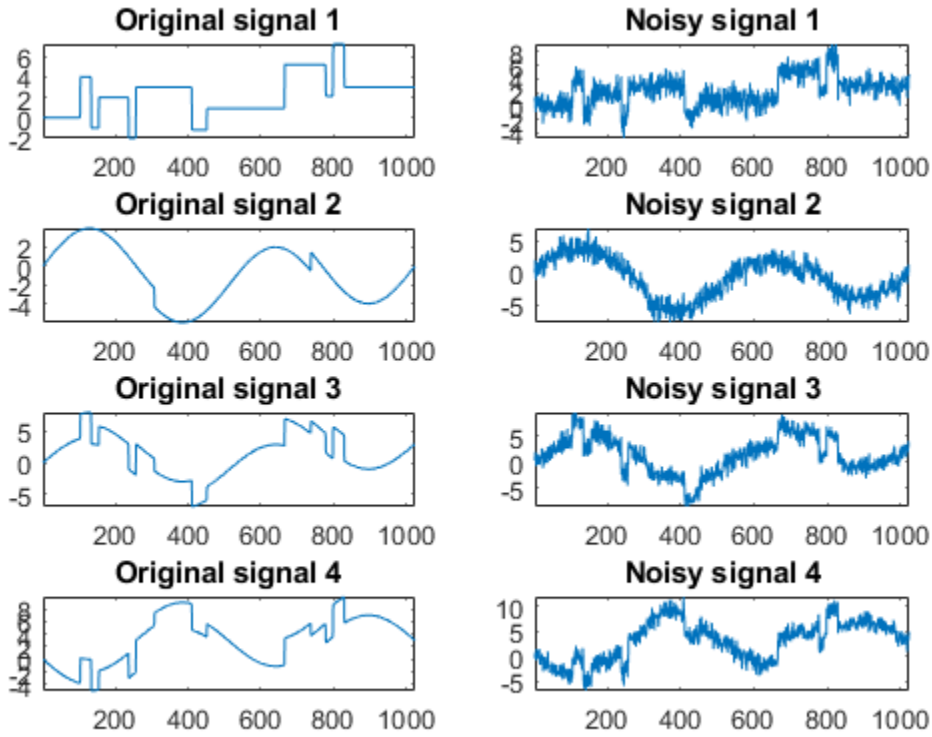
Examples

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.

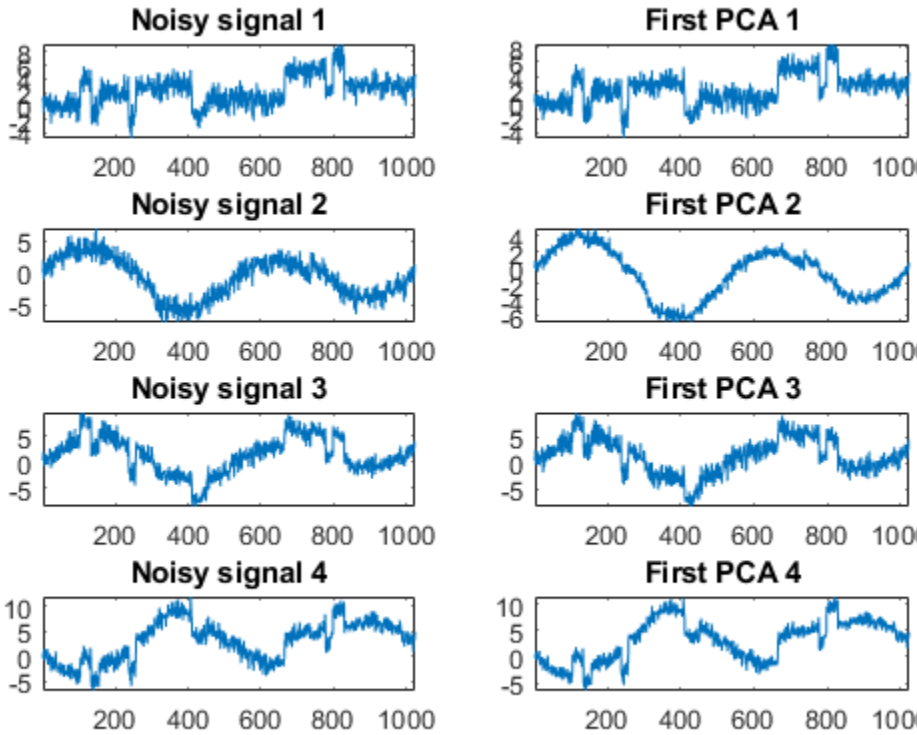
```
level = 5;
wname = 'sym4';
npc = 'heur';
[x_sim, qual, npc] = wmspca(x,level,wname,npc);
```

Plot the result and examine the quality of the approximation.

```
qual
```

```
qual =

   97.4064   94.6863   97.8333   99.5441
```

```
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
```



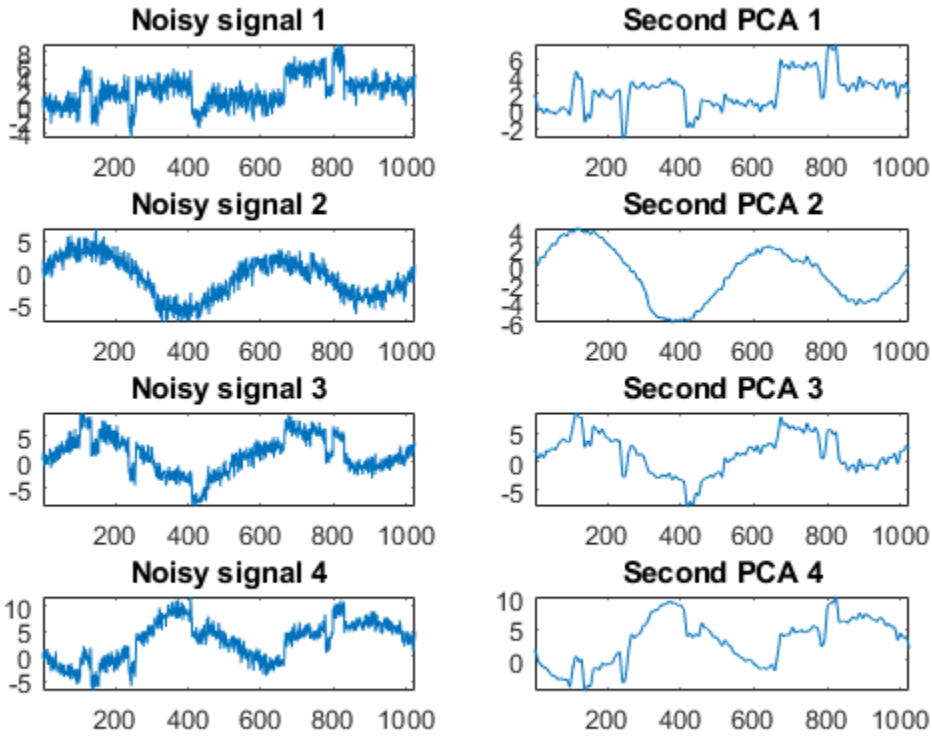
The quality results are all close to 100%. The `npc` vector gives the number of principal components retained at each level.

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`

Introduced in R2006b

wmulden

Wavelet multivariate de-noising

Syntax

```
[X_DEN,NPC,NESTCOV,DEC_DEN,PCA_Params,DEN_Params] = ...
wmulden(X,LEVEL,WNAME,NPC_APP,NPC_FIN,TPTR,SORH)
[...] = wmulden(X,LEVEL,WNAME,'mode',EXTMODE,NPC_APP,...)
[...] = wmulden(DEC,NPC_APP)
[...] = wmulden(X,LEVEL,WNAME,'mode',EXTMODE,NPC_APP)
[DEC,PCA_Params] = wmulden('estimate',DEC,NPC_APP,NPC_FIN)
[X_DEN,NPC,DEC_DEN,PCA_Params] = 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 <= `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.
- The 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 character vector 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

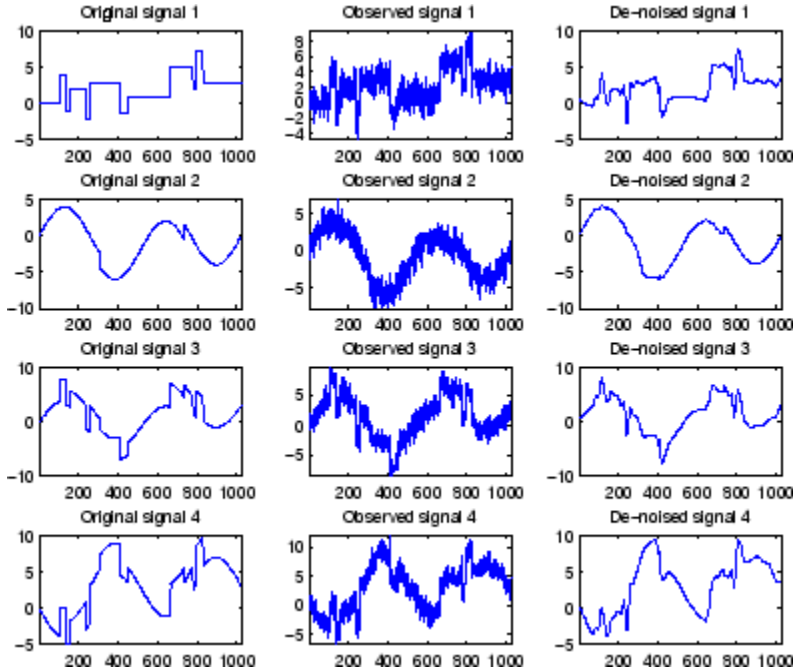
`[DEC,PCA_Params] = wmulden('estimate',DEC,NPC_APP,NPC_FIN)` returns the wavelet decomposition `DEC` and the Principal Components Estimates `PCA_Params`.

`[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`.

Examples

```
%  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`

Introduced in R2006b

wnoise

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.

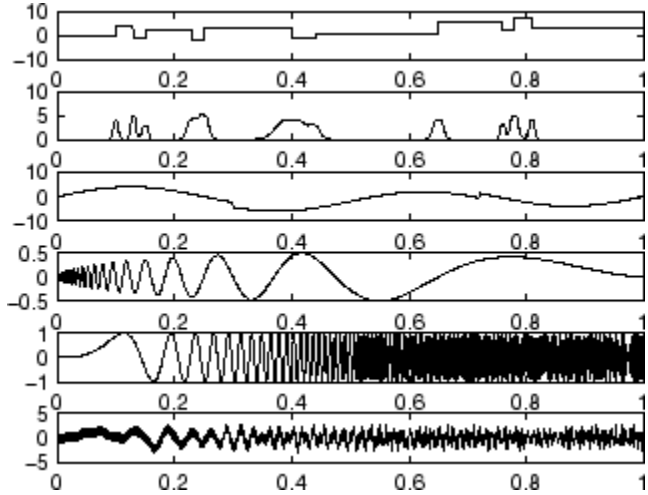
The six functions below are due to Donoho and Johnstone (See “References”).

`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`

Introduced before R2006a

wnoisest

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)
```

```
stdc =

    0.9559    1.0556
```

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`thselect` | `wavedec` | `wden`

Introduced before R2006a

wp2wtree

Extract wavelet tree from wavelet packet tree

Syntax

```
T = wp2wtree(T)
```

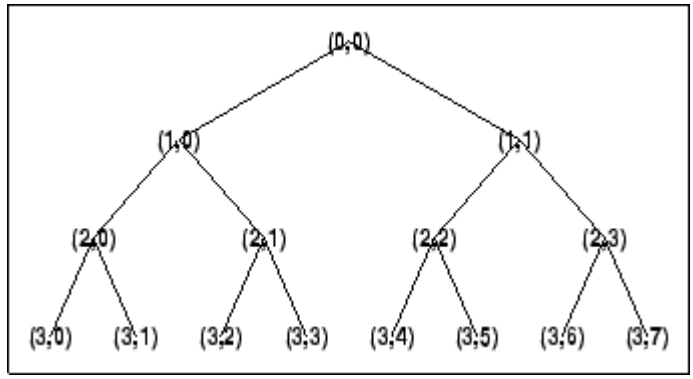
Description

`wp2wtree` is a one- or two-dimensional wavelet packet analysis function.

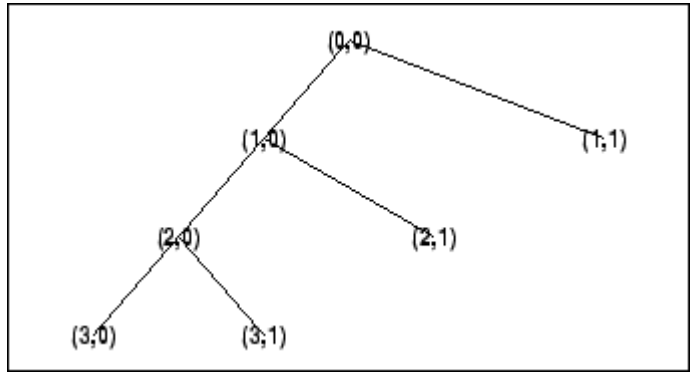
`T = wp2wtree(T)` computes the modified wavelet packet tree *T* corresponding to the wavelet decomposition tree.

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`

Introduced before R2006a

wpbmpen

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=$|c(t^*)|$.

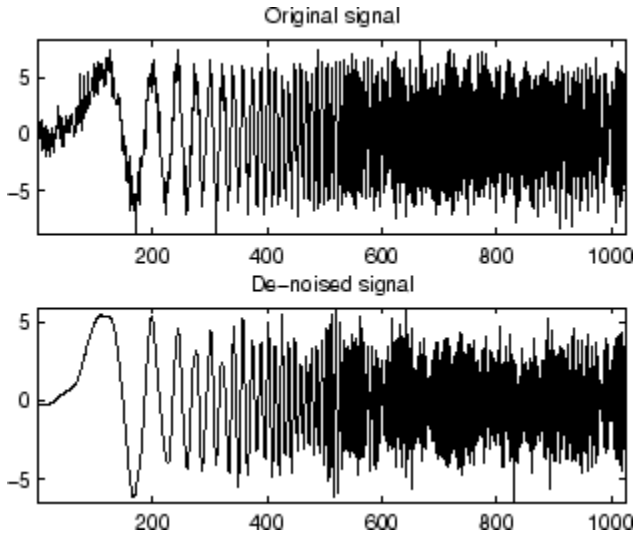
`wpbmpen(T,SIGMA,ALPHA,ARG)` computes the global threshold and, in addition, plots three curves:

- `2*SIGMA^2*t*(ALPHA + log(n/t))`
- `sum(c(k)^2,k£t)`
- `crit(t)`

Examples

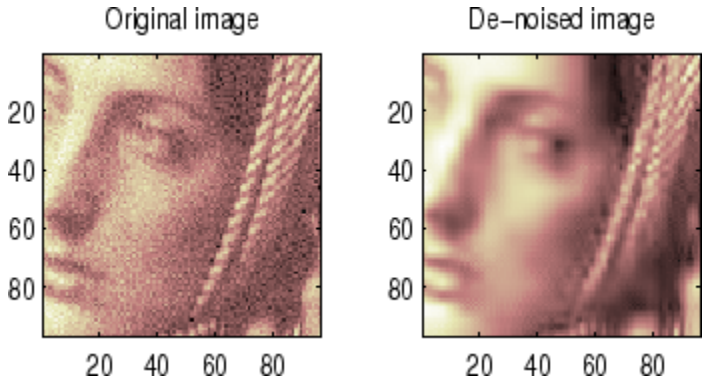
```
% Example 1: Signal de-noising.
% Load noisy chirp signal.
load noischir; x = noischir;

% 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')
```




```
% 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

Introduced before R2006a

wpcoef

Wavelet packet coefficients

Syntax

```
X = wpcoef(T,N)
X = wpcoef(T)
```

Description

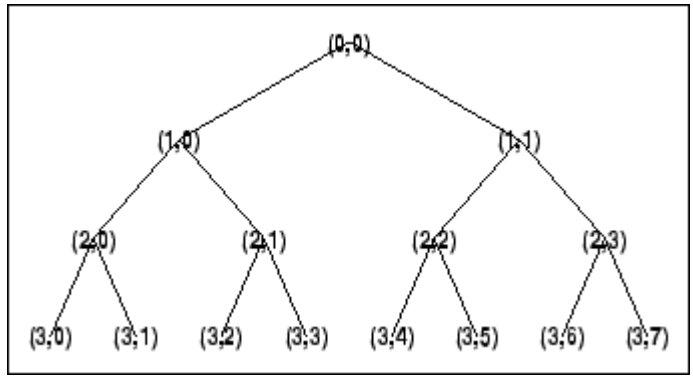
`wpcoef` is a one- or two-dimensional wavelet packet analysis function.

`X = wpcoef(T,N)` returns the coefficients associated with the node `N` of the wavelet packet tree `T`. If `N` doesn't exist, `X` = [ ];

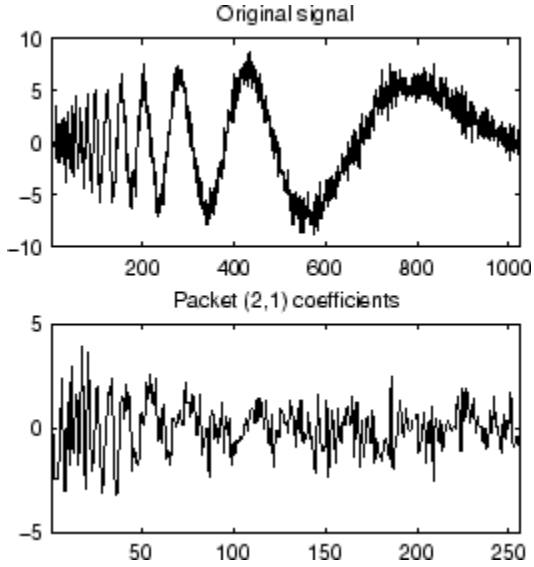
`X = wpcoef(T)` is equivalent to `X = wpcoef(T,0)`.

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

Topics

"Reconstructing a Signal Approximation from a Node"

Introduced before R2006a

wpcutree

Cut wavelet packet tree

Syntax

```
T = wpcutree(T,L)
T
[T,RN] = wpcutree(T,L)
```

Description

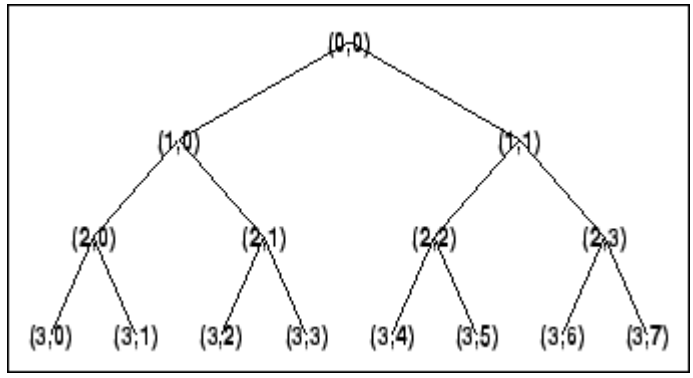
`wpcutree` is a one- or two-dimensional wavelet packet analysis function.

`T = wpcutree(T,L)` cuts the tree `T` at level `L`.

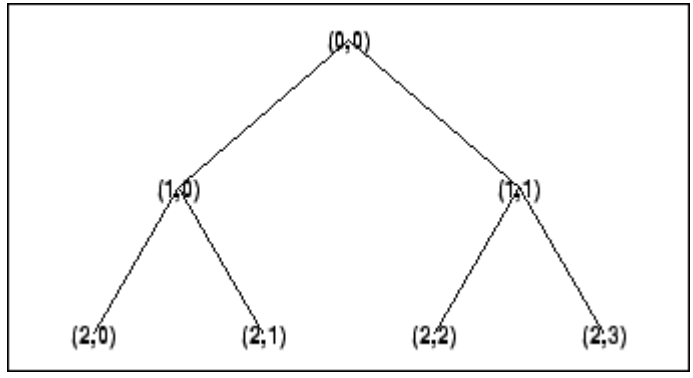
`[T,RN] = wpcutree(T,L)` returns the same arguments as above and, in addition, the vector `RN` contains the indices of the reconstructed nodes.

Examples

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

% 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)
```




```
% Cut wavelet packet tree at level 2.
nwpt = wpcutree(wpt,2);

% Plot new wavelet packet tree nwpt.
plot(nwpt)
```



See Also

`wpdec` | `wpdec2`

Introduced before R2006a

wpdec

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 character vector 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'`	$0 \le$ `P`	`P` is the threshold.
`'sure'`	$0 \le$ `P`	`P` is the threshold.
`'norm'`	$1 \le$ `P`	`P` is the power.
`'user'`	character vector	`P` is a character vector containing the file name of your own entropy function, with a single input `X`.

Entropy Type Name (E)	Parameter (P)	Comments
`FunName`	No constraints on `P`	`FunName` is any other character vector 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.

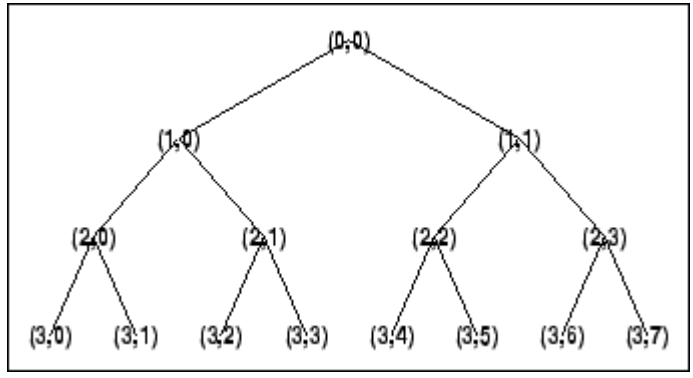
Simple and efficient algorithms exist for both wavelet packets decomposition and optimal decomposition selection. Adaptive filtering algorithms with direct applications in optimal signal coding and data compression can then be produced.

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`

Topics

"Build Wavelet Tree Objects"
"Examples Using Wavelet Packet Tree Objects"
"Objects in the Wavelet Toolbox Software"

Introduced before R2006a

wpdec2

Wavelet packet decomposition 2-D

Syntax

```
T = wpdec2(X,N,'wname',E,P)
T = wpdec2(X,N,'wname')
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 character vector 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'`	$0 \le P$	`P` is the threshold.
`'sure'`	$0 \le P$	`P` is the threshold.
`'norm'`	$1 \le P$	`P` is the power.
`'user'`	Character vector	`P` is a character vector containing the file name of your own entropy function, with a single input `X`.

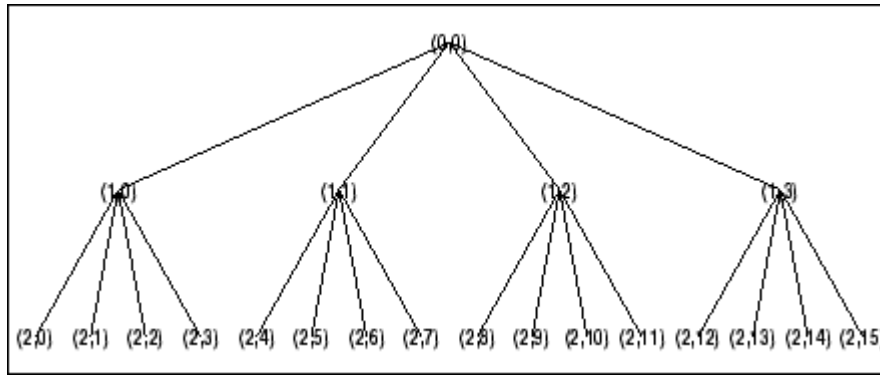
Entropy Type Name (E)	Parameter (P)	Comments
`STR`	No constraints on `P`	`STR` is any other character vector except those used for the previous Entropy Type Names listed above. `STR` contains the file name of your own entropy function, with `X` as input and `P` as additional parameter to your entropy 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.

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)
```



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.

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`

Topics

"Build Wavelet Tree Objects"
"Examples Using Wavelet Packet Tree Objects"
"Objects in the Wavelet Toolbox Software"

Introduced before R2006a

wpdencmp

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` equal to `'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 character vector containing the wavelet name. Best decomposition is performed using entropy criterion defined by character vector `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.

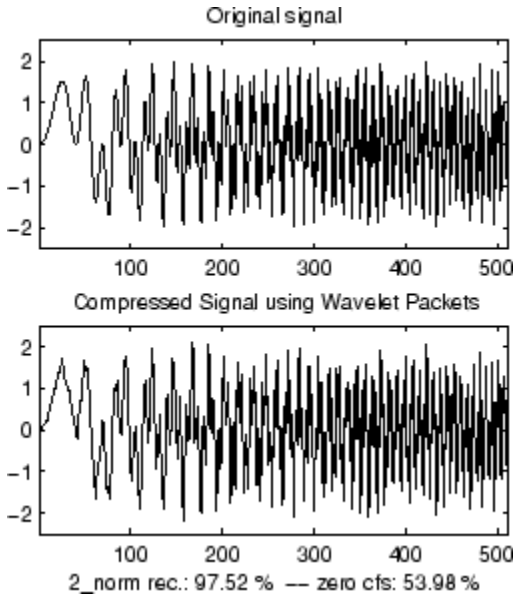
`[XD,TREED,PERF0,PERFL2] = wpdencmp(TREE,SORH,CRIT,PAR,KEEPAPP)` has the same output arguments, using the same options as above, but obtained directly from the input wavelet packet tree decomposition `TREE` (see `wpdec` for more information) of the signal to be de-noised or compressed.

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.
```




```
% 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`

Introduced before R2006a

wpfun

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 $W_0(x) = \phi(x)$ 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

$$(W_{2n+1}(x) = W_n(2x) - W_n(2x-1))$$

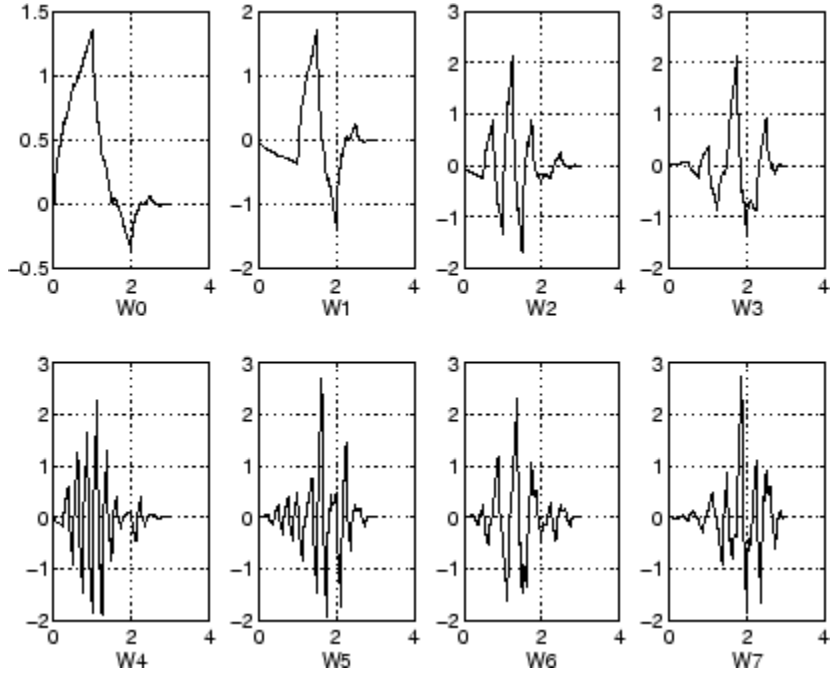
$W_0(x) = \phi(x)$ is the `haar` scaling function and $W_1(x) = \psi(x)$ is the `haar` wavelet, both supported in [0,1].

Then we can obtain W_{2n} 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`

Introduced before R2006a

wpjoin

Recompose wavelet packet

Syntax

```
T = wpjoin(T,N)
[T,X] = wpjoin(T,N)
T = wpjoin(T)
T = wpjoin(T,0)
[T,X] = wpjoin(T)
[T,X] = wpjoin(T,0)
```

Description

`wpjoin` is a one- or two-dimensional wavelet packet analysis function.

`wpjoin` updates the wavelet packet tree after the recomposition of a node.

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

`T = wpjoin(T,N)` returns the modified wavelet packet tree `T` corresponding to a recomposition of the node `N`.

`[T,X] = wpjoin(T,N)` also returns the coefficients of the node.

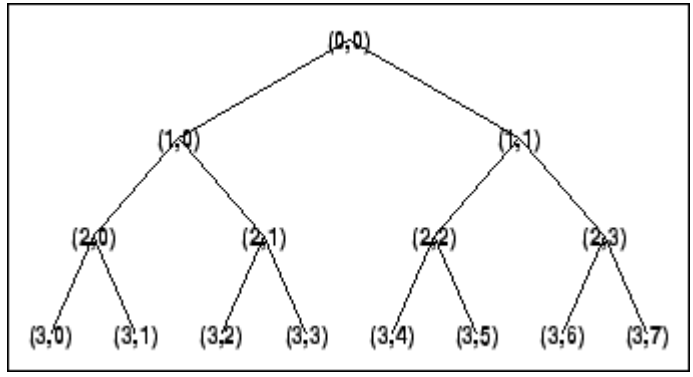
`T = wpjoin(T)` is equivalent to `T = wpjoin(T,0)`.

`[T,X] = wpjoin(T)` is equivalent to `[T,X] = wpjoin(T,0)`.

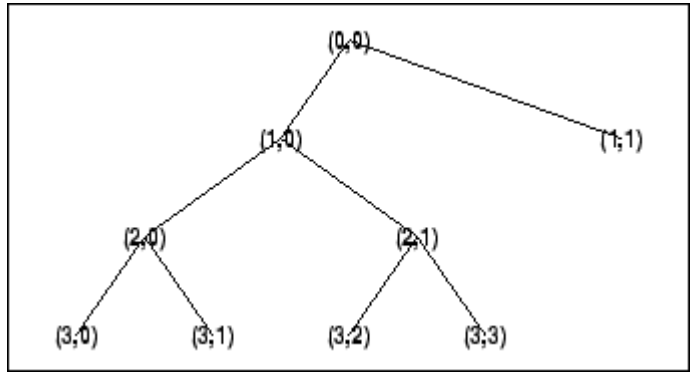
Examples

```
% 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)
```




```
% Recompose packet (1,1) or 2
wpt = wpjoin(wpt,[1 1]);

% Plot wavelet packet tree wpt.
plot(wpt)
```



See Also

wpdec | wpdec2 | wpsplt

Introduced before R2006a

wprcoef

Reconstruct wavelet packet coefficients

Syntax

```
X = wprcoef(T,N)
X = wprcoef(T)
X = wprcoef(T,0)
```

Description

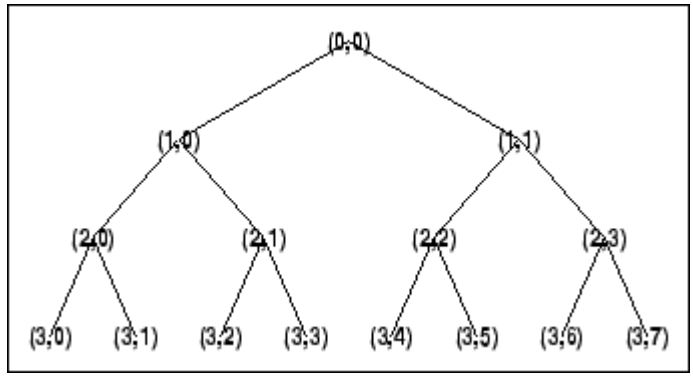
`wprcoef` is a one- or two-dimensional wavelet packet analysis function.

`X = wprcoef(T,N)` computes reconstructed coefficients of the node `N` of the wavelet packet tree *T*.

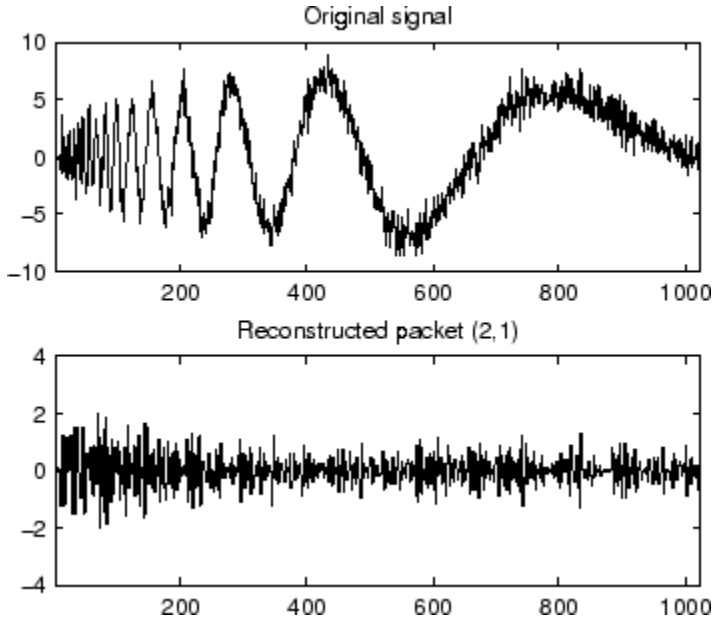
`X = wprcoef(T)` is equivalent to `X = wprcoef(T,0)`.

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)');
```



See Also

`wpdec` | `wpdec2` | `wprec` | `wprec2`

Topics

"Reconstructing a Signal Approximation from a Node"

Introduced before R2006a

wprec

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`

Introduced before R2006a

wprec2

Wavelet packet reconstruction 2-D

Syntax

```
X = wprec2(T)
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`

Introduced before R2006a

wpspectrum

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 character vector `'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.

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.

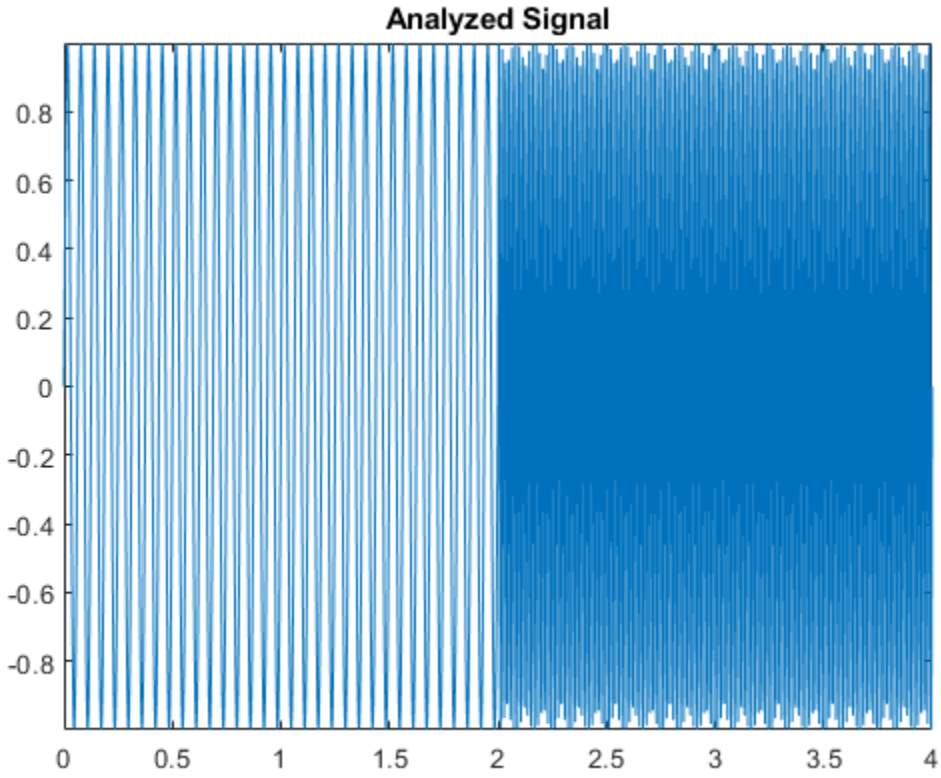
Examples

Wavelet Packet Spectrum for Sinusoids

This example shows wavelet packet spectrum for signal consisting of two sinusoids with disjoint support.

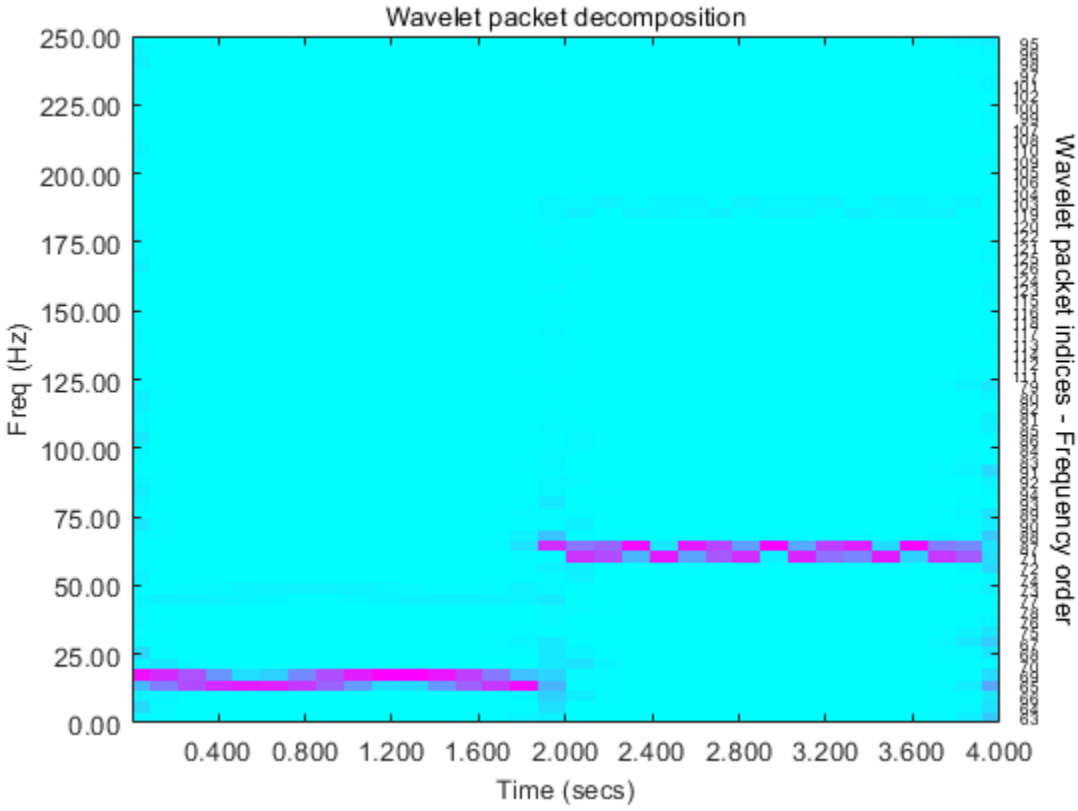
Define wavelet.

```
fs = 500;
t = 0:1/fs:4;
y = sin(32*pi*t).*(t<2) + sin(128*pi*t).*(t>=2);
plot(t,y);
axis tight
title('Analyzed Signal');
```



Define wavelet packet spectrum.

```
level = 6;
wpt = wpdec(y,level,'sym6');
figure;
[S,T,F] = wpspectrum(wpt,fs,'plot');
```



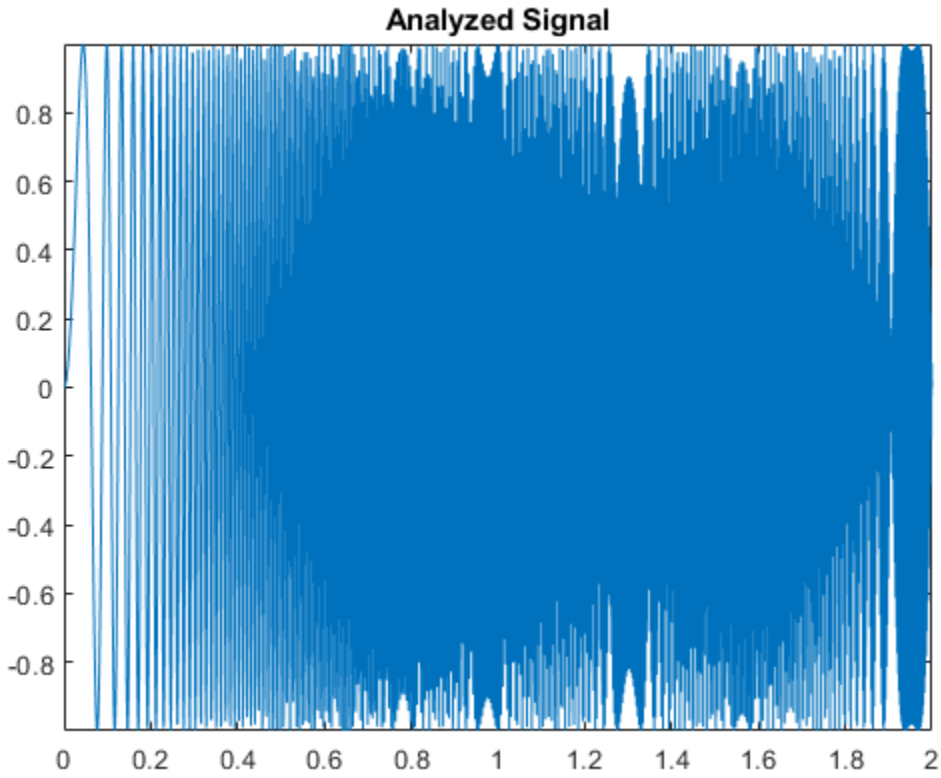
Wavelet Packet Spectrum of Chirp Signal

Create the chirp signal.

```
fs = 1000;
t = 0:1/fs:2;
% create chirp signal
y = sin(256*pi*t.^2);
```

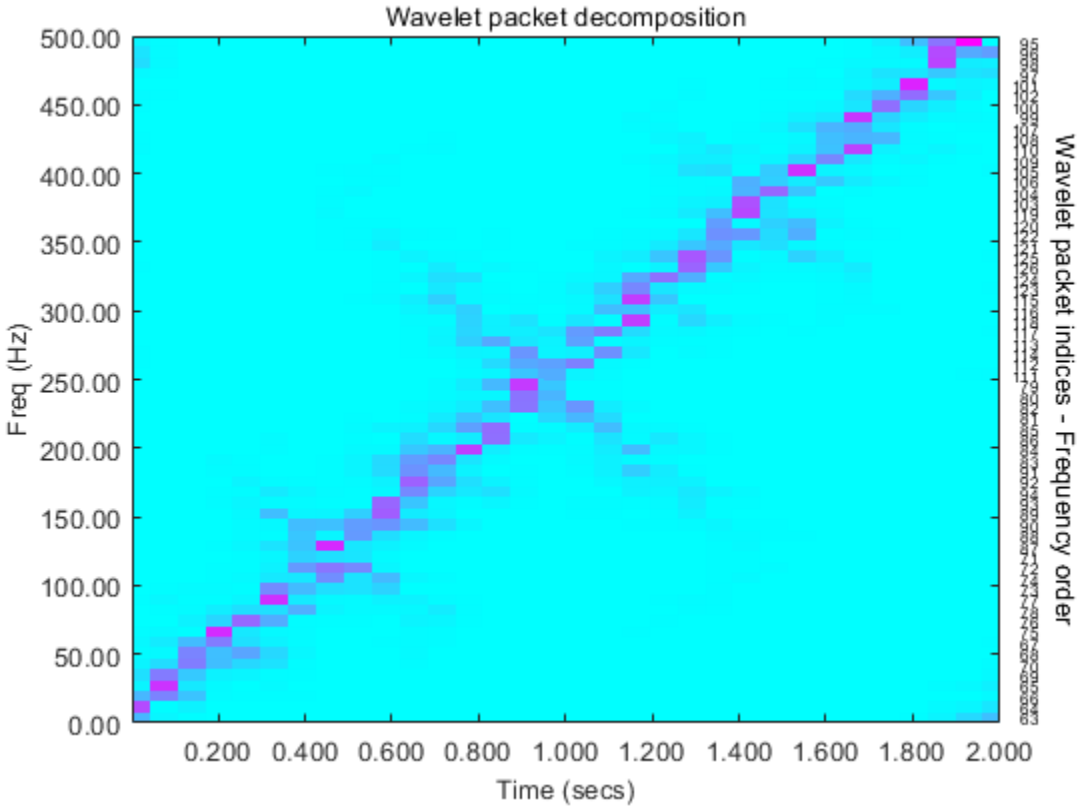
Plot the analyzed signal.

```
plot(t,y);
axis tight
title('Analyzed Signal');
```



Get the wavelet packet spectrum estimates.

```
level = 6;
wpt = wpdec(y,level,'sym8');
figure;
[S,T,F] = wpspectrum(wpt,fs,'plot');
```



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 $Fs / 2^{J+1}$.

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`

Topics

"Wavelet Packet Spectrum"

Introduced in R2010b

wpsplt

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`.

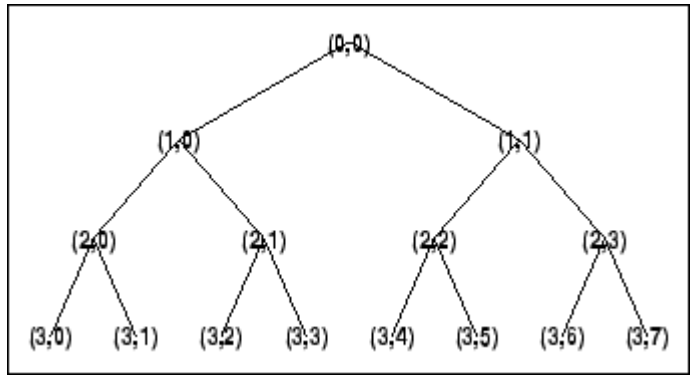
For a two-dimensional decomposition,

`[T,cA,cH,cV,cD] = wpsplt(T,N)` with `cA` = approximation and `cH`, `cV`, `c` = horizontal, vertical, and diagonal details of node `N`.

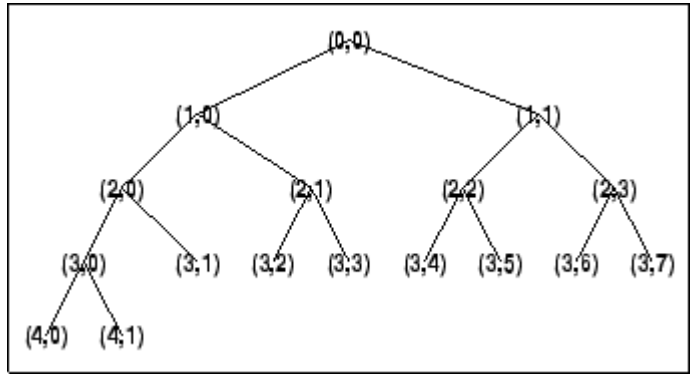
Examples

```
% 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

Introduced before R2006a

wpthcoef

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`

Introduced before R2006a

wptree

WPTREE constructor

Syntax

```
T = wptree(ORDER,DEPTH,X,WNAME,ENT_TYPE,PARAMETER)
T = wptree(ORDER,DEPTH,X,WNAME)
T = wptree(ORDER,DEPTH,X,WNAME,'shannon')
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 character vector 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

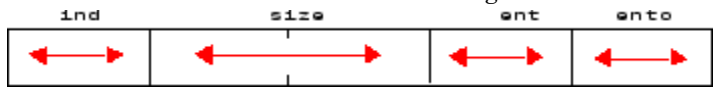
Fields from the DTREE parent object:

'allNI'	All nodes information

`'allNI'` is an array of size `nbnode` by `5`, which contains

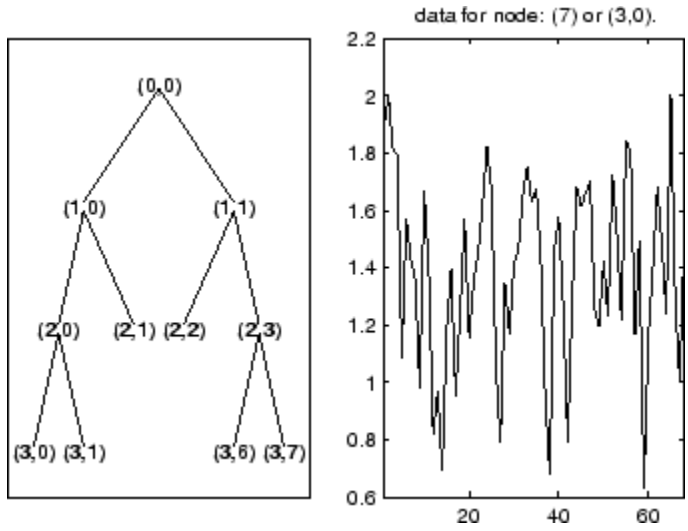
`ind`	Index
`size`	Size of data
`ent`	Entropy
`ento`	Optimal entropy

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`

Introduced before R2006a

wpviewcf

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.

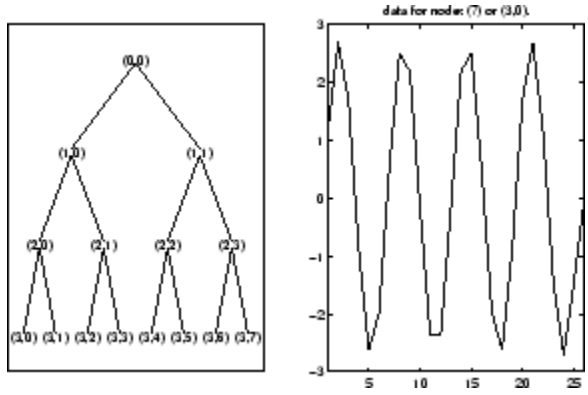
Color Mode	Description
1	Frequency order – Global coloration – Absolute values
2	Frequency order – By level – Absolute values
3	Frequency order – Global coloration – Values
4	Frequency order – By level coloration – Values
5	Natural order – Global coloration – Absolute values
6	Natural order – By level – Absolute values
7	Natural order – Global coloration – Values
8	Natural order – By level coloration – Values

`wpviewcf(T,CMODE,NBCOL)` uses `NBCOL` colors.

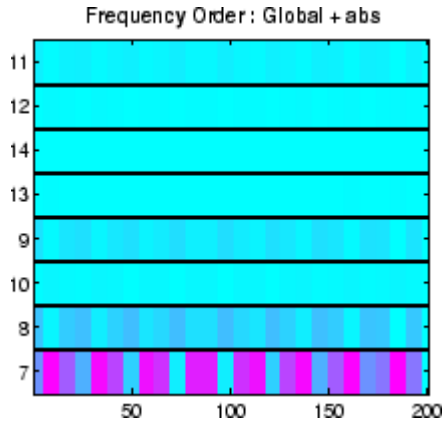
Examples

```
% 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);
```




```
% Plot the colored wavelet packet coefficients.
wpviewcf(t,1);
```



See Also

wpdec

Introduced before R2006a

wrcoef

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 character vector 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 $N \leq$ `length(L)-2`.

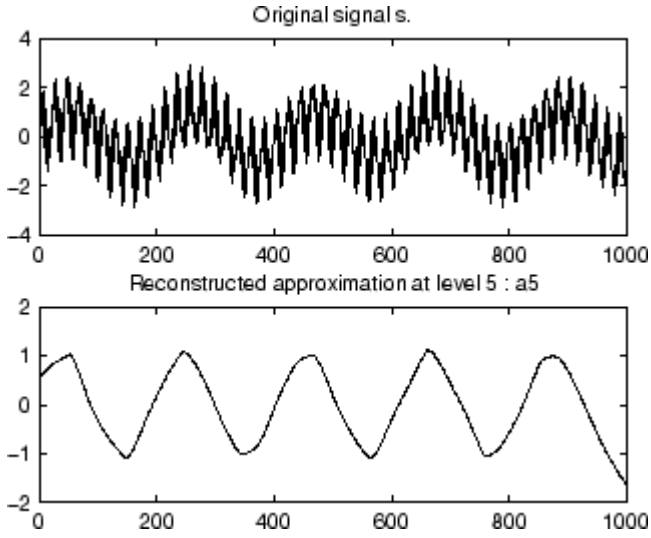
`X = wrcoef('type',C,L,Lo_R,Hi_R,N)` computes coefficients as above, given the reconstruction filters you specify.

`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

Introduced before R2006a

wrcoef2

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 character vector 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 ≤ N ≤ size(S,1)-2` if *`'type'`*`= 'a'` and such that `1 ≤ N ≤ 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`.

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
```

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.

See Also

`appcoef2` | `detcoef2` | `wavedec2`

Introduced before R2006a

wrev

Flip vector

Syntax

```
Y = wrev(X)
```

Description

`wrev` is a general utility.

`Y = wrev(X)` reverses the vector `X`.

Examples

```
v = [1 2 3];
wrev(v)
wrev(v')
```

See Also

`fliplr` | `flipud`

Introduced before R2006a

write

Write values in WPTREE fields

Syntax

```
T = write(T,'cfs',NODE,COEFS)
T = write(T,'cfs',N1,CFS1,'cfs',N2,CFS2, ...)
```

Description

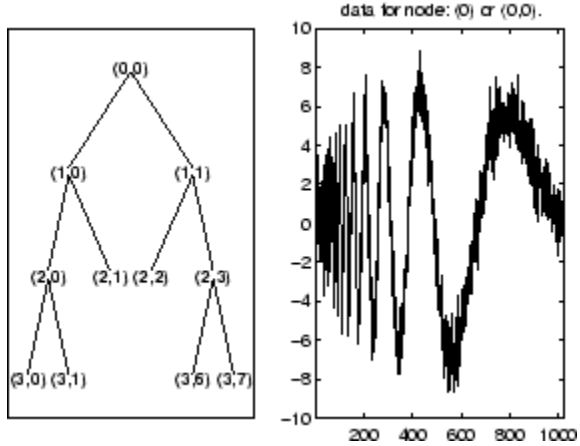
`T = write(T,'cfs',NODE,COEFS)` writes coefficients for the terminal node `NODE`.

`T = write(T,'cfs',N1,CFS1,'cfs',N2,CFS2, ...)` writes coefficients `CFS1, CFS2, ...` for the terminal nodes `N1, N2, ....`

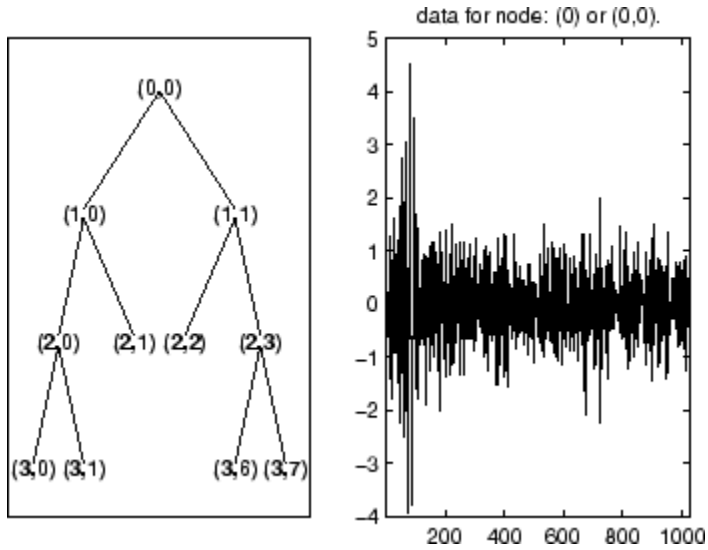
Caution The coefficients values must have the suitable size. You can use `S = read(T,'sizes',NODE)` or `S = read(T,'sizes',[N1;N2; ...])` in order to get those sizes.

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);
```




```
% 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)
```



See Also

`disp` | `get` | `read` | `set`

Introduced before R2006a

wscalogram

Scalogram for continuous wavelet transform

Note This function is no longer recommended. Use `cwt` instead.

Syntax

```
SC = wscalogram(TYPEPLOT,COEFS)
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.

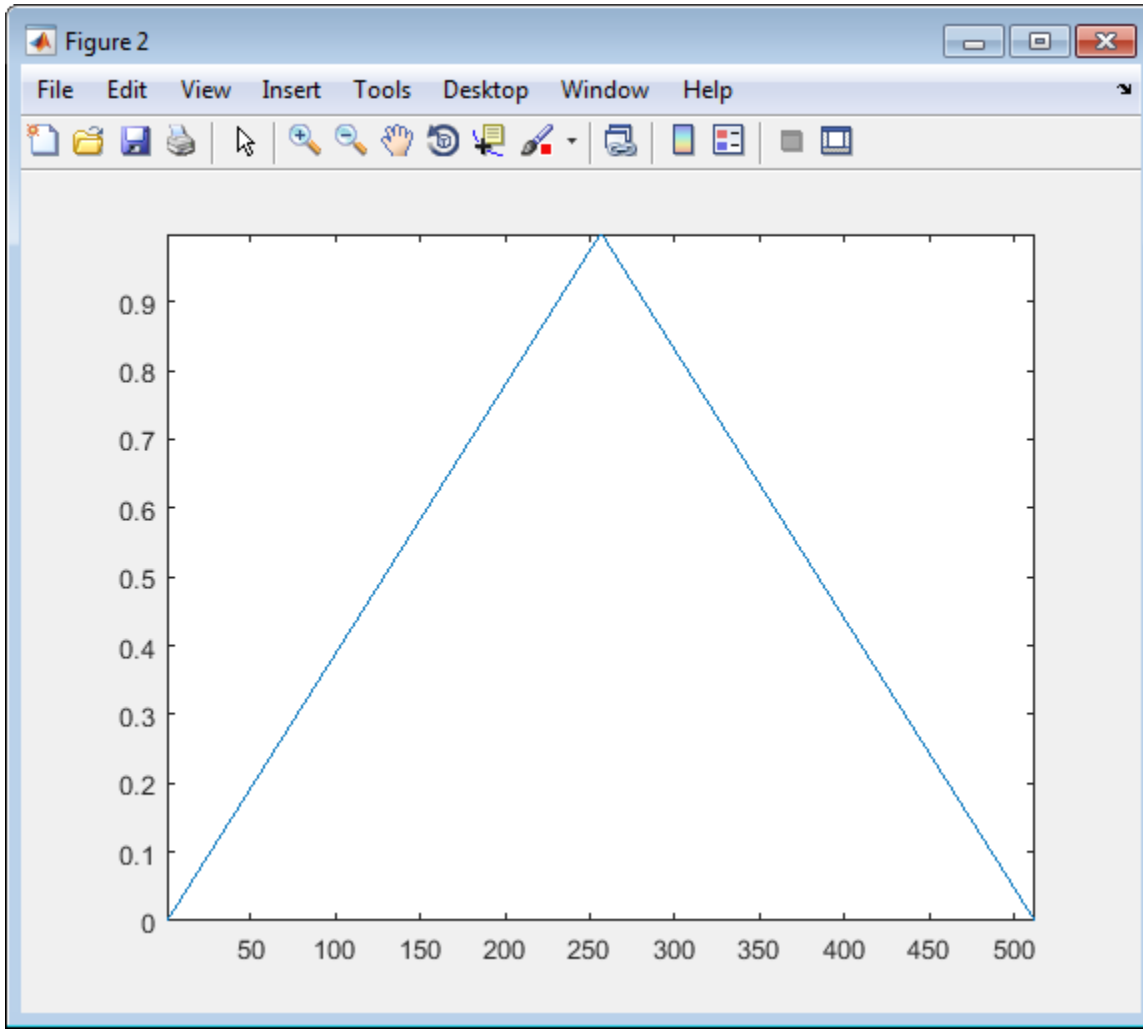
If `power > 0`, coefficients are first normalized

```
coefs(k,:) = coefs(k,:)/(scales(k)^power)
```

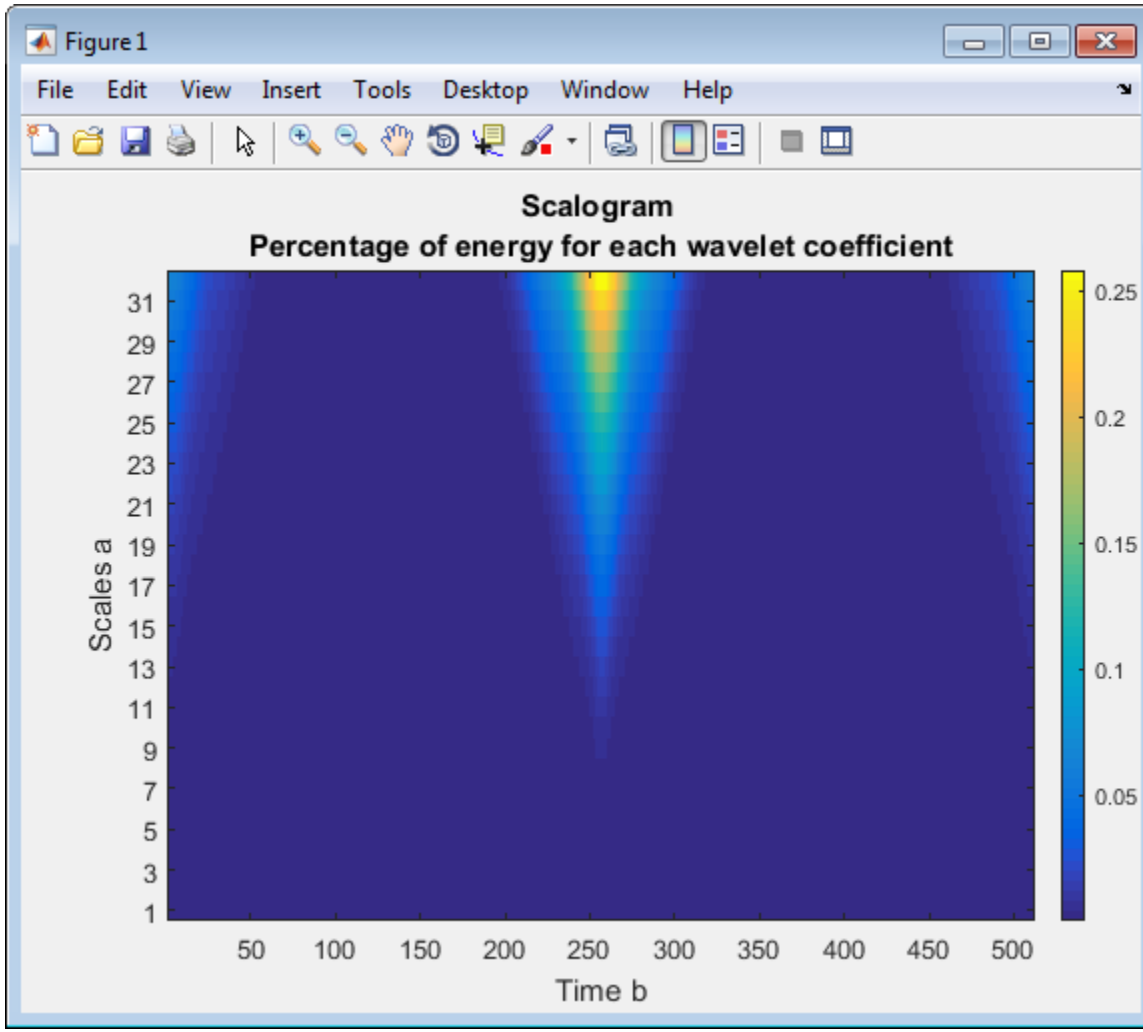
and then the scalogram is computed as explained above.

Examples

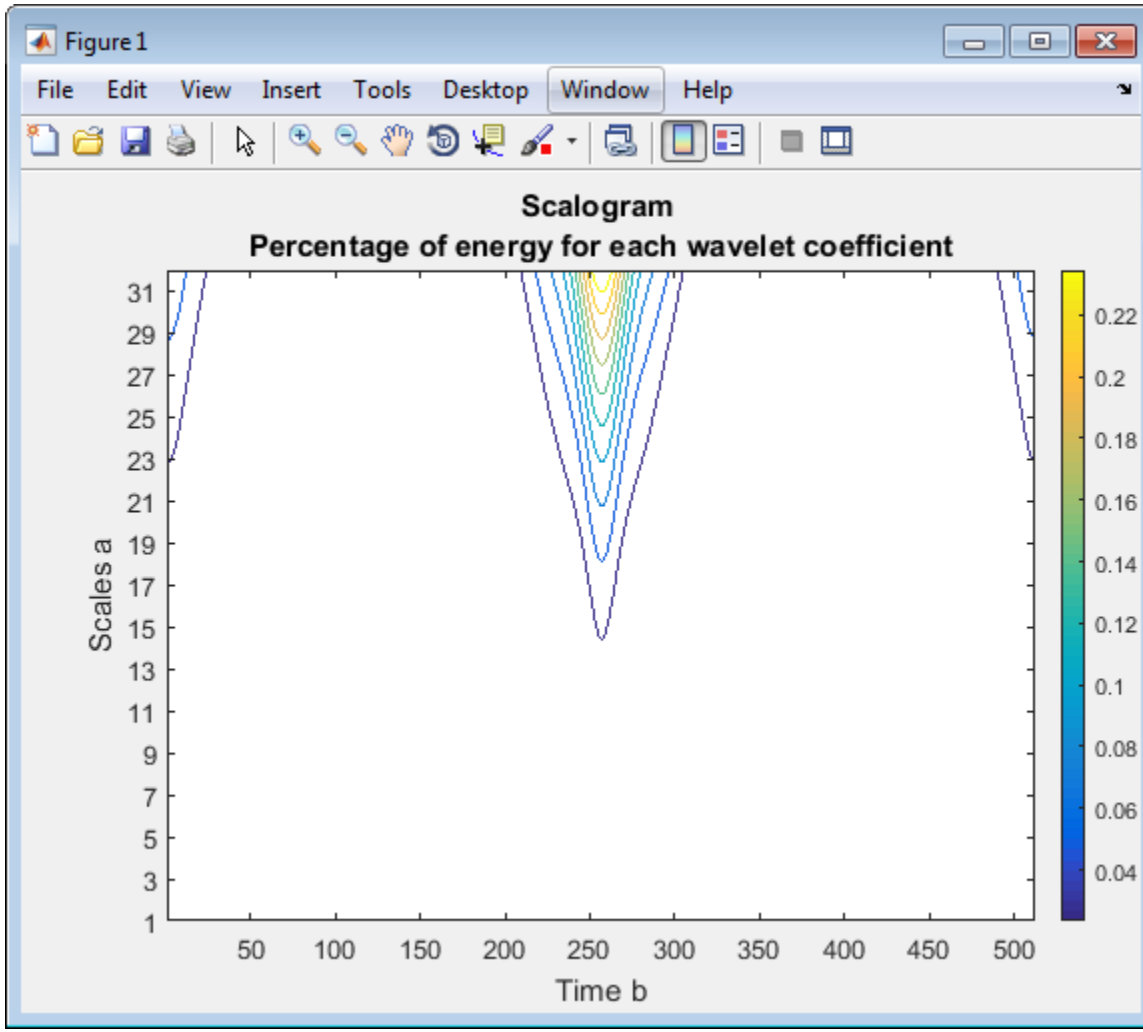
```
% Compute signal s
t = linspace(-1,1,512);
s = 1-abs(t);

% 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);
```




```
% Compute and plot the scalogram (contour option)
figure;
SC = wscalogram('contour',COEFS);
```



See Also

`cwt`

Introduced in R2008a

wsst

Wavelet synchrosqueezed transform

Syntax

```
sst = wsst(x)
[sst,f] = wsst(x)
[___] = wsst(x,fs)
[___] = wsst(x,ts)
[___] = wsst(___,wav)
wsst(___)
[___] = wsst(___,Name,Value)
```

Description

`sst = wsst(x)` returns the wavelet synchrosqueezed transform, `sst`, which you use to examine data in the time-frequency plane. The synchrosqueezed transform has reduced energy smearing when compared to the continuous wavelet transform. The input, `x`, must be a 1-D real-valued signal with at least four samples. `wsst` computes the synchrosqueezed transform using the analytic Morlet wavelet.

`[sst,f] = wsst(x)` returns a vector of frequencies, `f`, in cycles per sample. The frequencies correspond to the rows of `sst`.

`[___] = wsst(x,fs)` computes the sychrosqueezed transform using the specified sampling frequency, `fs`, in Hz, to compute the synchrosqueezed transform. If you specify an `f` output, `wsst` returns the frequencies in Hz. You can use any previous combination of output values.

`[___] = wsst(x,ts)` uses a `duration` `ts` with a positive, scalar input, as the sampling interval. The duration can be in years, days, hours, minutes, or seconds. If you specify `ts` and the `f` output, `wsst` returns the frequencies in `f` in cycles per unit time, where the time unit is derived from specified duration.

`[___] = wsst(___,wav)` uses the analytic wavelet specified by `wav` to compute the synchrosqueezed transform. Valid values are `'amor'` and `'bump'`, which specify the analytic Morlet and bump wavelet, respectively.

`wsst(___)` with no output arguments plots the synchrosqueezed transform as a function of time and frequency. If you do not specify a sampling frequency, `fs`, or interval, `ts`, the synchrosqueezed transform is plotted in cycles per sample. If you specify a sampling frequency, the synchrosqueezed transform is plotted in Hz. If you specify a sampling interval using a duration , the plot is in cycles per unit time. The time units are derived from the duration.

`[___] = wsst(___,Name,Value)` returns the synchrosqueezed transform with additional options specified by one or more `Name,Value` pair arguments.

Examples

Synchrosqueezed Transform of Speech Signal

Obtain the wavelet synchrosqueezed transform of a speech sample using default values.

```
load mtlb;
sst = wsst(mtlb);
```

Synchrosqueezed Transform and Reconstruction of Speech Signal

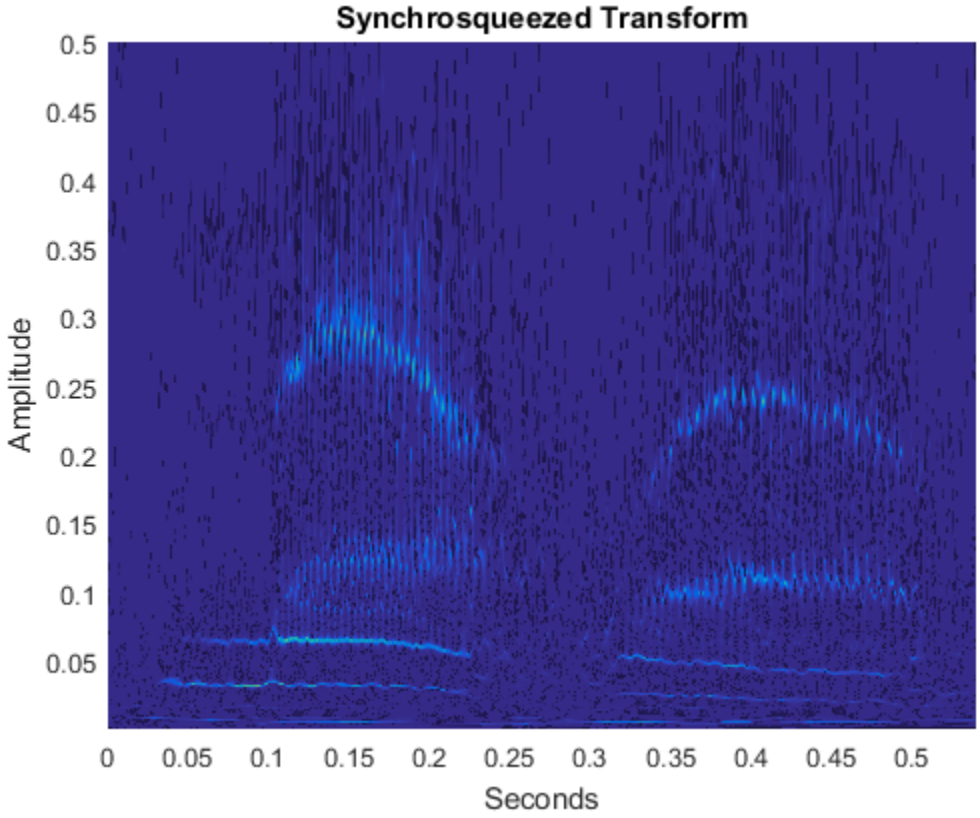
Obtain the wavelet synchrosqueezed transform of a speech signal and compare the original and reconstructed signals.

Load the speech signal and obtain its synchrosqueezed transform.

```
load mtlb;
soundsc(mtlb);
dt = 1/Fs;
t = 0:dt:numel(mtlb)*dt-dt;
[sst,f] = wsst(mtlb);
```

Plot the synchroqueezed transform.

```
pcolor(t,f,abs(sst));
shading interp
xlabel('Seconds'); ylabel('Amplitude');
title('Synchrosqueezed Transform');
```



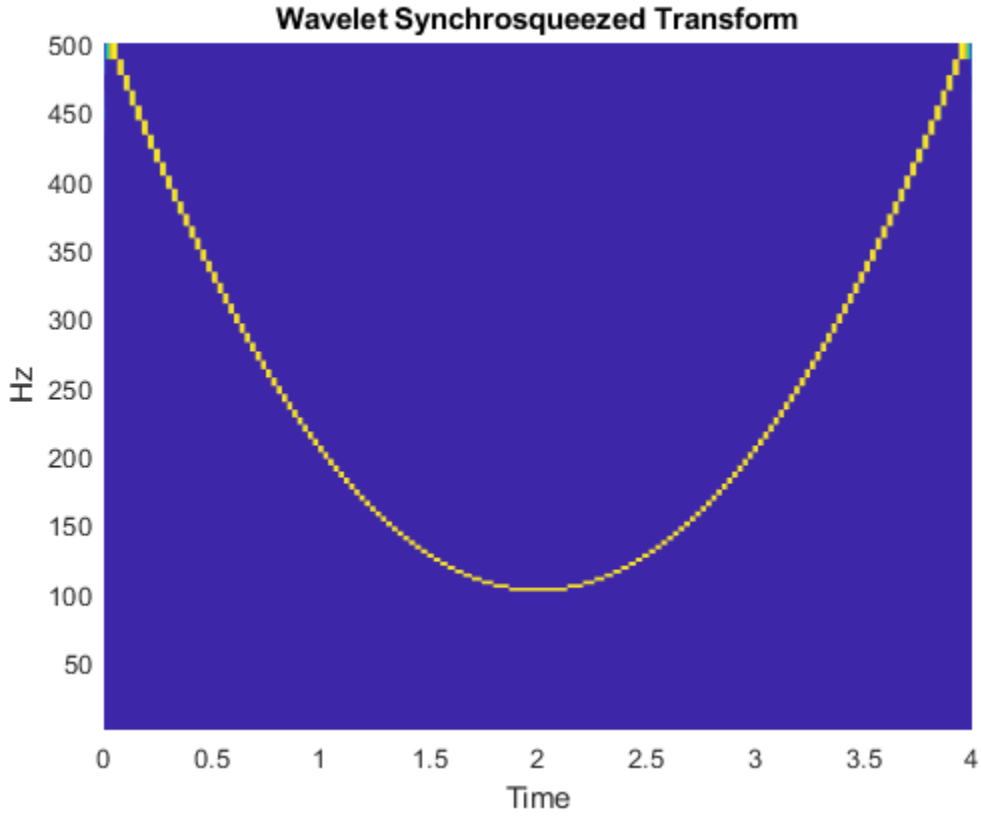
Obtain the inverse synchroqueezed transform and play the reconstructed speech signal.

```
xrec = iwsst(sst);
soundsc(xrec);
```

Synchrosqueezed Transform of Quadratic Chirp

Obtain and plot the wavelet synchrosqueezed transform of a quadratic chirp. The chirp is sampled at 1000 Hz.

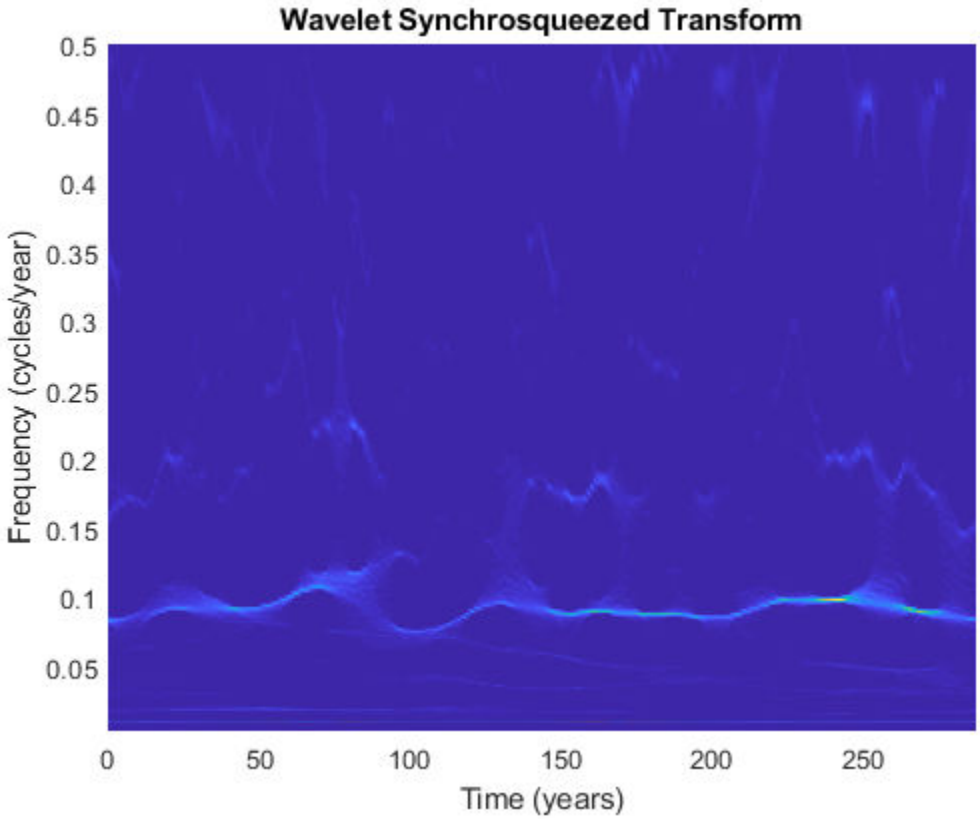
```
load quadchirp;
[sst,f] = wsst(quadchirp,1000);
hp = pcolor(tquad,f,abs(sst));
hp.EdgeColor = 'none';
title('Wavelet Synchrosqueezed Transform');
xlabel('Time'); ylabel('Hz');
```



Synchrosqueezed Transform of Sunspot Data

Obtain the wavelet synchrosqueezed transform of sunspot data using the default Morlet wavelet. Specify the sampling interval to be one year.

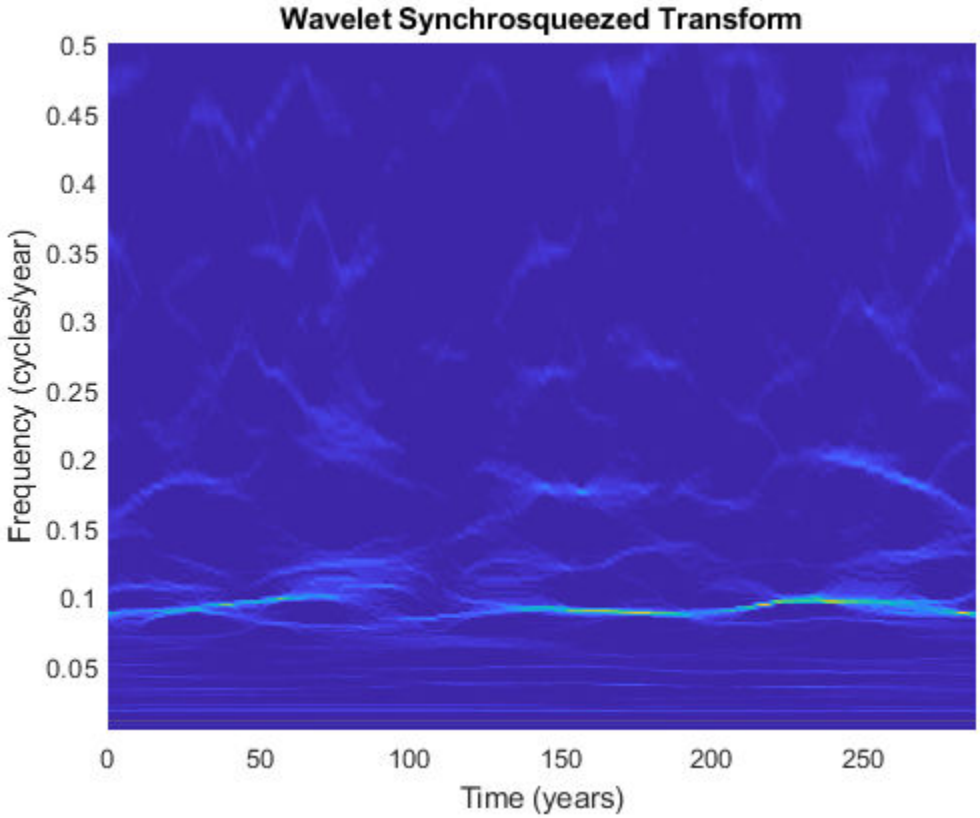
```
load sunspot.dat;
wsst(sunspot(:,2),years(1))
```



Synchrosqueezed Transform of Sunspot Data Using Bump Wavelet

Obtain and plot the wavelet synchrosqueezed transform of sunspot data using the bump wavelet. Specify the sampling interval to be 1 for one sample per year.

```
load sunspot.dat;
wsst(sunspot(:,2),years(1),'bump')
```



- “Time-Frequency Reassignment and Mode Extraction with Synchrosqueezing”

Input Arguments

x — Input signal

row or column vector of real values

Input signal, specified as a row or column vector. `x` must be a 1-D, real-valued signal with at least four samples.

`fs` — Sampling frequency
positive scalar

Sampling frequency, specified as a positive scalar.

`ts` — Sampling interval
duration with positive scalar input

Sampling interval, also known as the sampling period, specified as a `duration` with positive scalar input. Valid durations are `years`, `days`, `hours`, `seconds`, and `minutes`. You cannot use calendar durations (`caldays`, `calweeks`, `calmonths`, `calquarters`, or `calyears`). You cannot specify both `ts` and `fs`.

Example: `sst = wsst(x,hours(12))`

`wav` — Analytic wavelet
`'amor'` (default) | `'bump'`

Analytic wavelet used to compute the synchrosqueezed transform, specified as `'amor'` or `'bump'`. These character vectors specify the analytic Morlet wavelet and bump wavelet, respectively.

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: `'VoicesPerOctave',26`

`VoicesPerOctave` — Number of voices per octave
`32` (default) | integer from 10 to 48

Number of voices per octave to use in the synchrosqueezed transform, specified as the comma-separated pair consisting of `'VoicesPerOctave'` and an integer from 10 to 48. The product of the number of voices per octave and the number of octaves is the number of scales. The number of octaves depends on the size of the input `x` and is `floor(log2(numel(x)))-1`.

`ExtendSignal` — Extend input signal symmetrically

`false` (default) | `true`

Option to extend the input signal symmetrically, specified as the comma-separated pair consisting of `'ExtendSignal'` and either `false` or `true`. Extending the signal symmetrically can mitigate boundary effects. If you specify `false`, then the signal is not extended. If you specify `true`, then the signal is extended.

Output Arguments

`sst` — Synchrosqueezed transform

matrix

Synchrosqueezed transform, returned as a matrix. By default, the synchrosqueezed transform uses `floor(log2(numel(x)))-1` octaves, 32 voices per octave, and the analytic Morlet wavelet. `sst` is an *Na*-by-*N* matrix where *Na* is the number of scales, and *N* is the number of samples in `x`. The default number of scales is `32*(floor(log2(numel(x)))-1)`.

`f` — Frequencies

vector

Frequencies of the synchrosqueezed transform, returned as a vector. The frequencies correspond to the rows of the `sst`. If you do not specify `fs` or `ts`, the frequencies are in cycles per sample. If you specify `fs`, the frequencies are in Hz. If you specify `ts`, the frequencies are in cycles per unit time. The length of the frequency vector is the same as the number of `sst` rows. If you specify `ts` as the sampling interval, `ts` is used to compute the scale-to-frequency conversion for `f`.

References

[1] I. Daubechies, I., J. Lu, and H. T. Wu. "Synchrosqueezed Wavelet Transforms: an Empricial Mode Decomposition-like Tool", *Applied and Computational Harmonic Analysis*. Vol. 30(2), pp. 243–261.

[2] Thakur, G., E. Brevdo, N. S. Fučkar, and H. T. Wu. "The Synchrosqueezing algorithm for time-varying spectral analysis: robustness properties and new paleoclimate applications." *Signal Processing*. Vol. 93, pp. 1079–1094.

See Also

`days` | `duration` | `hours` | `iwsst` | `minutes` | `seconds` | `wsstridge` | `years`

Topics

“Time-Frequency Reassignment and Mode Extraction with Synchrosqueezing”
“Wavelet Synchrosqueezing”

Introduced in R2016a

wsstridge

Time-frequency ridges from wavelet synchrosqueezing

Syntax

```
fridge = wsstridge(sst)
[fridge,iridge] = wsstridge(sst)
[___] = wsstridge(sst,penalty)
[___] = wsstridge(___,f)
[___]= wsstridge(___,Name,Value)
```

Description

`fridge = wsstridge(sst)` extracts the maximum energy time-frequency ridge in cycles per sample from the wavelet synchrosqueezed transform, `sst`. The `sst` input is the output of `wsst`. Each ridge is a separate signal mode.

`[fridge,iridge] = wsstridge(sst)` returns in `iridge` the row indices of `sst`. The row indices are the maximum time-frequency ridge at each sample. Use `iridge` to reconstruct the signal mode along a time-frequency ridge using `iwsst`.

`[___] = wsstridge(sst,penalty)` multiplies the squared distance between frequency bins by the `penalty` value. You can include any of the ouitput arguments from previous syntaxes.

`[___] = wsstridge(___,f)` returns the maximum energy time-frequency ridge in cycles per unit time based on the `f` input frequency vector. `f` is the frequency output of `wsst`. The `f` input and `fridge` output have the same units.

`[___]= wsstridge(___,Name,Value)` returns the time-frequency ridge with additional options specified by one or more `Name,Value` pair arguments.

Examples

Extract Time-Frequency Ridge from Chirp Signal

Obtain the wavelet synchrosqueezed transform of a quadratic chirp and extract the maximum time-frequency ridge, in `fridge`, and the associated row indices, in `iridge`.

Load the chirp signal and obtain its synchrosqueezed transform.

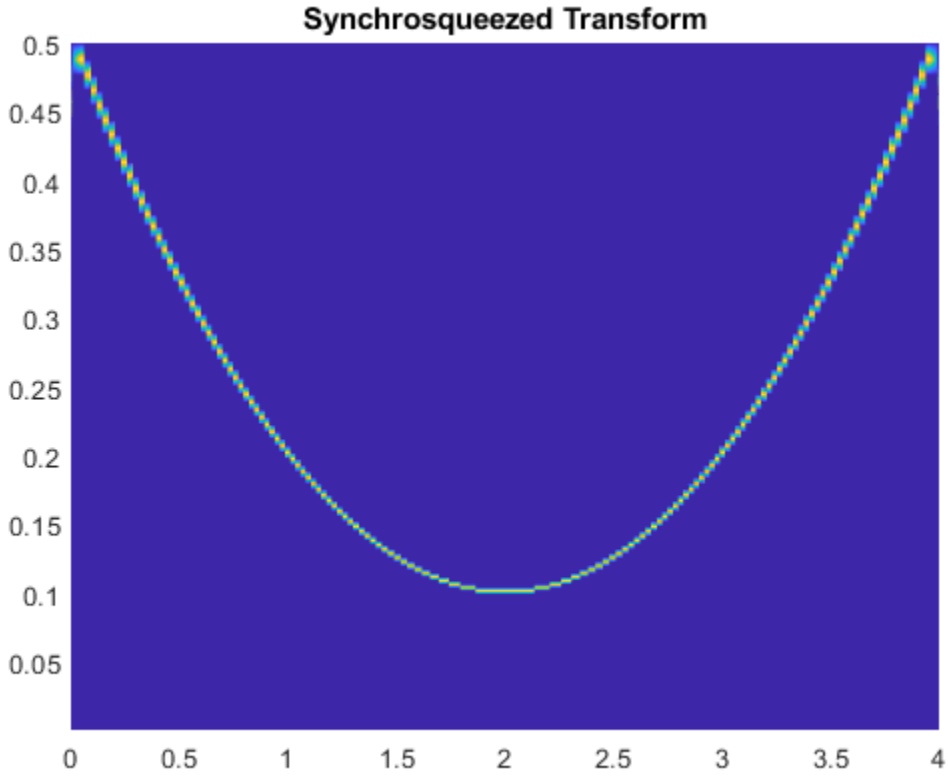
```
load quadchirp;
[sst,f] = wsst(quadchirp);
```

Extract the maximum time-frequency ridge.

```
[fridge,iridge] = wsstridge(sst);
```

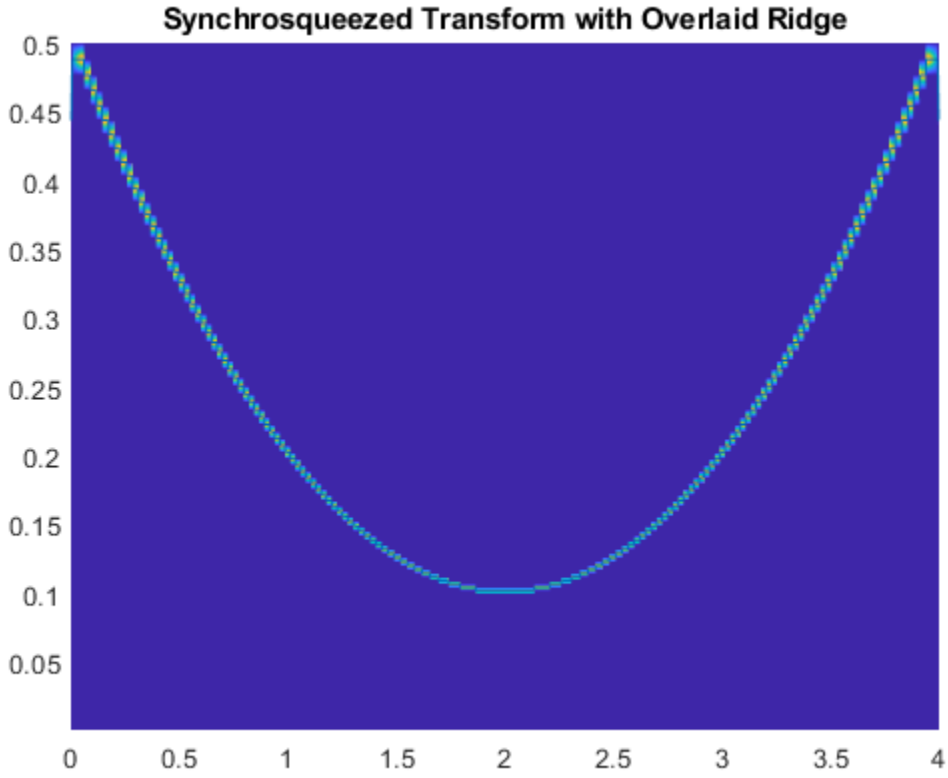
Plot the synchrosqueezed transform.

```
pcolor(tquad,f,abs(sst))
shading interp
title('Synchrosqueezed Transform')
```



Overlay the plot of the maximum energy frequency ridge.

```
hold on
plot(tquad,fridge)
title('Synchrosqueezed Transform with Overlaid Ridge')
```



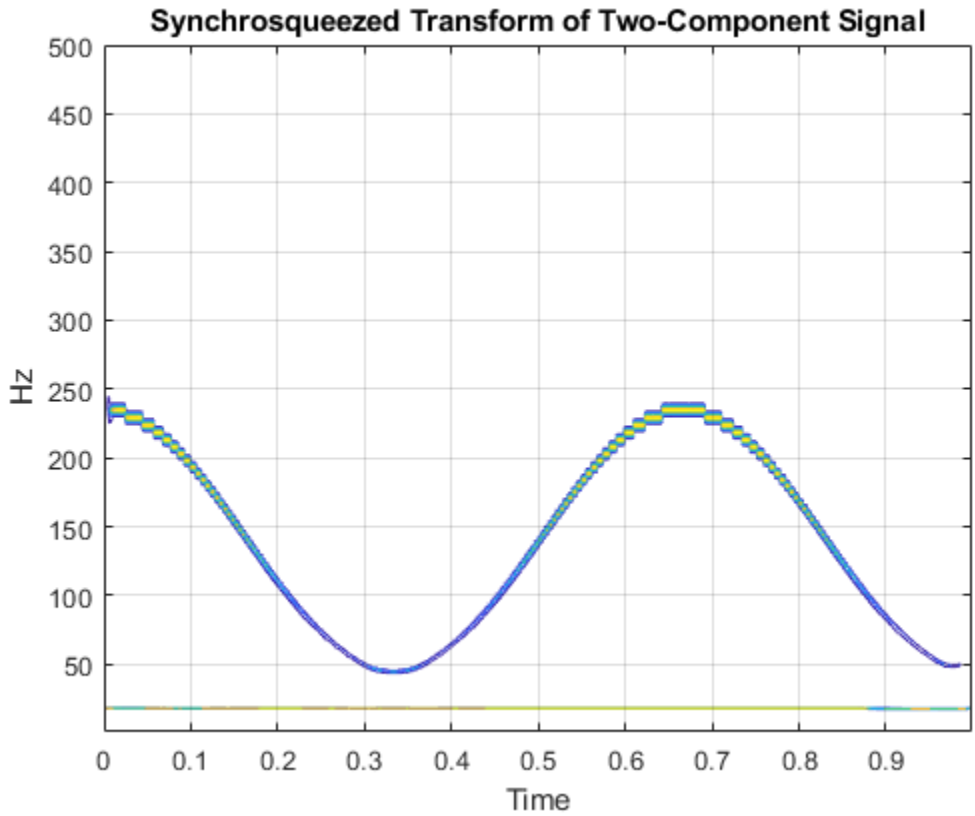
Extract Time-Frequency Ridge from Multicomponent Signal

Extract the two highest energy modes from a multicomponent signal.

Obtain and plot the wavelet synchrosqueezed transform.

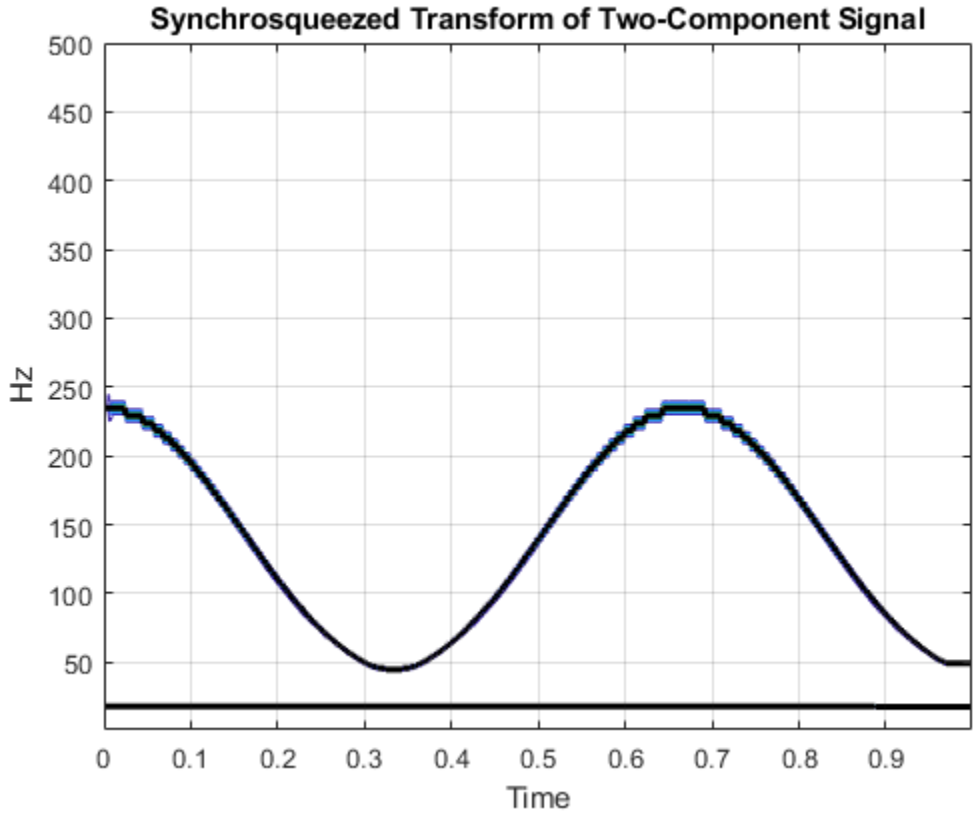
```
load multicompsig;
sig = sig1+sig2;
[sst,F] = wsst(sig,sampfreq);
contour(t,F,abs(sst));
xlabel('Time'); ylabel('Hz');
```

```
grid on;
title('Synchrosqueezed Transform of Two-Component Signal');
```



Using a penalty of 10, extract the two highest energy modes and plot the result.

```
[fridge,iridge] = wsstridge(sst,10,F,'NumRidges',2);
hold on;
plot(t,fridge,'k','linewidth',2);
```



- “Time-Frequency Reassignment and Mode Extraction with Synchrosqueezing”

Input Arguments

`sst` — Synchrosqueezed transform
matrix

Synchrosqueezed transform, specified as a matrix.

`penalty` — Frequency bins scaling penalty
0 (default) | nonnegative scalar

Frequency bins scaling penalty, specified as a nonnegative scalar. This input penalizes changes in frequency by multiplying the penalty value by the squared distance between frequency bins. Use a penalty term when you extract multiple ridges, or when you have a single modulated component in additive noise. The penalty term prevents jumps in frequency that occur when the region of highest energy in the time-frequency plane changes abruptly.

`f` — Synchrosqueezed transform frequencies

vector

Synchrosqueezed transform frequencies corresponding to the rows of the synchrosqueezed transform, which is the vector output of `wsst`. The number of elements in the frequency vector is equal to the number of rows in the `sst` input.

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: `'NumRidges',3`

`NumRidges` — Number of highest energy time-frequency ridges

1 (default) | positive integer

Number of highest energy time-frequency ridges to extract, specified as the comma-separated pair consisting of `'NumRidges'` and a positive integer. If this integer is greater than 1, `wsstridge` iteratively determines the maximum energy time-frequency ridge by removing the previously computed ridges and the default or specified `'NumFrequencyBins'` on either side of each ridge bin.

`NumFrequencyBins` — Number of frequency bins to remove

4 (default) | positive integer

Number of frequency bins to remove from synchrosqueezed transform `sst` when extracting multiple ridges, specified as the comma-separated pair consisting of `'NumFrequencyBins'` and a positive integer. This integer must be less than or equal to `round(size(sst,1)/4)`. You can specify the nunber of frequency bins to remove only if you extract more than one ridge. After extracting the highest energy time-frequency

ridge, `wsstridge` removes the `sst` values corresponding to the `iridge` indices at each time step. The energy is removed along the time-frequency ridge extended on both sides of the `iridge` index by the specified number of frequency bins. If the index of the extended time-frequency ridge exceeds the number of frequency bins at any time step, `wsstridge` truncates the removal region at the first or last frequency bin. To specify `'NumFrequencyBins'`, you must specify `'NumRidges'`.

Output Arguments

`fridge` — Time-frequency ridge frequencies

vector or matrix

Time-frequency ridge frequencies, returned as a vector or matrix. The frequencies correspond to the time-frequency ridge at each time step. `fridge` is an *N*-by-`nr` matrix where *N* is the number of time samples (columns) in `sst` and `nr` is the number of ridges. The first column of the matrix contains the frequencies for the maximum energy time-frequency ridge in `sst`. Subsequent columns contain the frequencies for the time-frequency ridges in decreasing energy order. By default, `fridge` contains frequencies in cycles per sample.

`iridge` — Time-frequency ridge indices

vector or matrix

Time-frequency ridge row indices of `sst`, returned as a vector or matrix. The row indices in `iridge` correspond to the row index of the maximum time-frequency ridge for each `sst` column. `iridge` is an *N*-by-`nr` matrix where *N* is the number of time samples (columns) in `sst`, and `nr` is the number of ridges. The first column of the matrix contains the indices for the maximum energy time-frequency ridge in `sst`. Subsequent columns contain the indices for the time-frequency ridges in decreasing energy order.

Algorithms

`wsstridge` uses a penalized forward-backward greedy algorithm to extract the maximum energy time-frequency ridges from the wavelet synchrosqueezed transform matrix. The algorithm finds the maximum time-frequency ridge by minimizing `-ln(E)` at each time point, where `E` is the absolute value of the synchrosqueezed transform. The is equivalent to maximizing the value of `E`. The algorithm optionally constrains jumps in

frequency with a penalty that is proportional to the square of the distance between frequency bins.

The following example illustrates the time-frequency ridge algorithm using a penalty that is 2 times the squared distance between frequency bins. This simple synchrosqueezed transform matrix has three frequency bins and three time steps. The second row represents a sine wave.

1 Obtain the complex matrix output, y, of the wavelet synchroqueezed transform using `wsst`. Compute the `(-ln(|y|))`. Suppose you have this `(-ln(|y|))` matrix output,

```
1   4   4
2   2   2
5   5   4
```

2 Update the value for the (1,2) element.

a Leave the values the at the first time point unaltered. Begin the algorithm with the (1,2) element of the matrix, which presents the first frequency bin at the 2nd time point. Penalize the values in the first column based on their distance from the (1,2) element. Applying the penalty to the first column produces

```
original value + penalty(distance squared)

1 + 2(0*0) =  1
2 + 2(1*1) =  4
5 + 2(2*2)=  13

 1   4
 4   2
13   5
```

The minimum value of the first column is 1, which is in bin 1.

b Add the minimum value in column 1 to the current bin value, 4. The updated value for (1,2) becomes 5, which came from bin 1.

3 Update the values for the remaining elements in column 2.

Recompute the original column 1 values with the penalty factor using the same process as in Step 2a. Obtain the remaining second column values using the same process as in Step 2b. Repeat Step 2 for the third column. The final matrix is

$$\begin{matrix} 1 & 5_{(1)} & 9_{(1)} \\ 2 & 4_{(2)} & 6_{(2)} \\ 5 & 7_{(2)} & 8_{(2)} \end{matrix}$$

The subscripts indicate the index of the bin in the previous column from which a value came.

4 Starting at the last column of the matrix, find the minimum value. Walk back in time through the matrix by going from the current bin to the origin of that bin at the previous time point. Keep track of the bin indices, which form the path composing the ridge. The algorithm smooths the transition by using the origin bin instead of the bin with the minimum value. For this example, the ridge indices are `2`, `2`, `2`, which matches the energy path of the sine wave in row 2 of the matrix shown in Step 1.

5 If you are extracting multiple ridges, the algorithm removes the first ridge from the synchrosqueezed transform and repeats the process.

References

[1] I. Daubechies, I., J. Lu, and H. T. Wu. "Synchrosqueezed Wavelet Transforms: an Empricial Mode Decomposition-like Tool", *Applied and Computational Harmonic Analysis*. Vol. 30(2), pp. 243–261.

[2] Thakur, G., E. Brevdo, N. S. Fučkar, and H. T. Wu. "The Synchrosqueezing algorithm for time-varying spectral analysis: robustness properties and new paleoclimate applications." *Signal Processing*. Vol. 93, pp. 1079–1094.

See Also

`iwsst` | `wsst`

Topics

“Time-Frequency Reassignment and Mode Extraction with Synchrosqueezing”
“Wavelet Synchrosqueezing”

Introduced in R2016a

wtbo

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

Introduced before R2006a

wtbxmngr

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

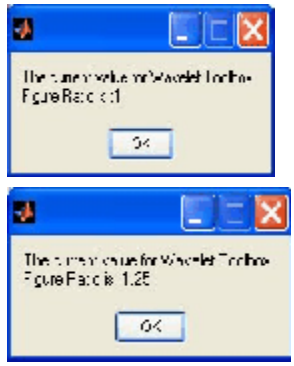
Option	Description
`'LargeFonts'`	Sets the size of future-created figures to use large fonts.
`'DefaultSize'`	Restores the default figure size for future- created figures.
`'FigRatio'`	Returns the current figure ratio value.
`'FigRatio',ratio`	Changes the size of future-created figures by multiplying 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
```



Introduced before R2006a

wthcoef

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`wavedec` | `wthresh`

Introduced before R2006a

wthcoef2

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.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`wavedec2` | `wthresh`

Introduced before R2006a

wthresh

Soft or hard thresholding

Syntax

```
Y = wthresh(X,SORH,T)
Y = wthresh(X,'s',T)
Y = wthresh(X,'h',T)
```

Description

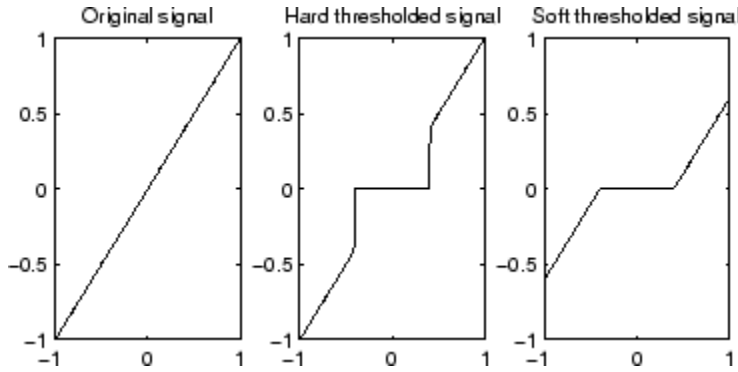
`Y = wthresh(X,SORH,T)` returns the soft (if `SORH = 's'`) or hard (if `SORH = 'h'`) thresholding of the input vector or matrix `X`. *T* is the threshold value.

`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 $Y = X.1_{(|X| > T)}$, 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.
```



Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`wden` | `wdencmp` | `wpdencmp`

Introduced before R2006a

wthrmngr

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 Wavelet Analyzer app 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`.

METHOD	Description
'heursure'	See 'heursure' option in thselect, and see also wden.
'minimaxi'	See 'minimaxi' option in thselect, and see also wden.
'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.

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)
```

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
```

`[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

`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`.

`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)
```

`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
```

`[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

`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

`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)
```

`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

`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)
```

`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.

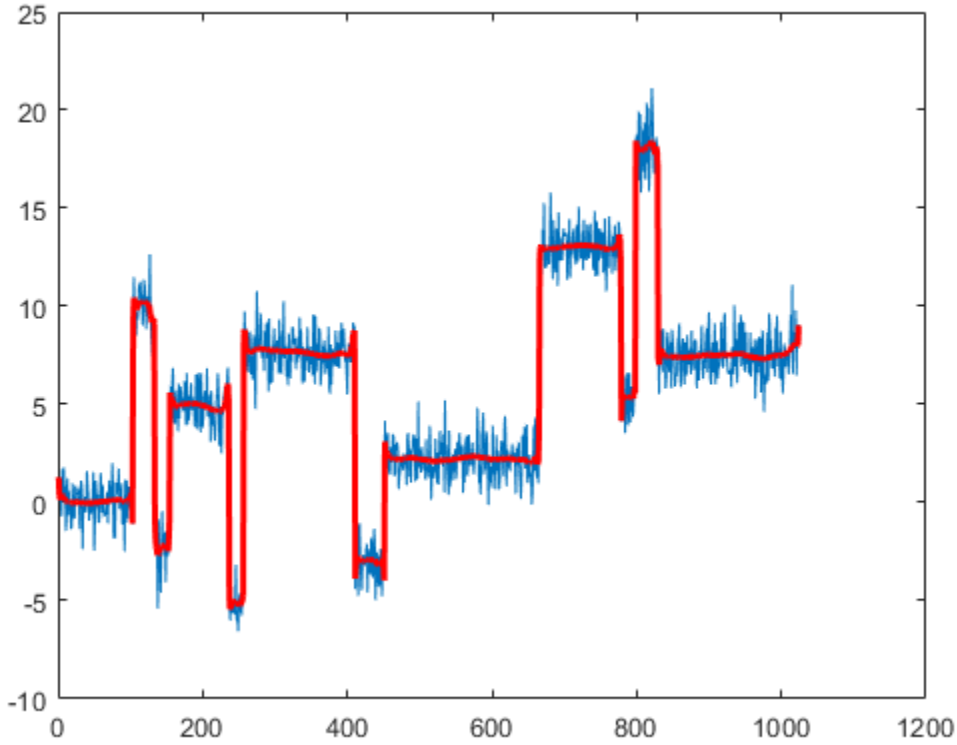
```
swcnew = swc;
ThreshSL = mean(wthrmngr('sw1ddenoLVL','sqtwolog',swc,'sln'));
```

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);
```

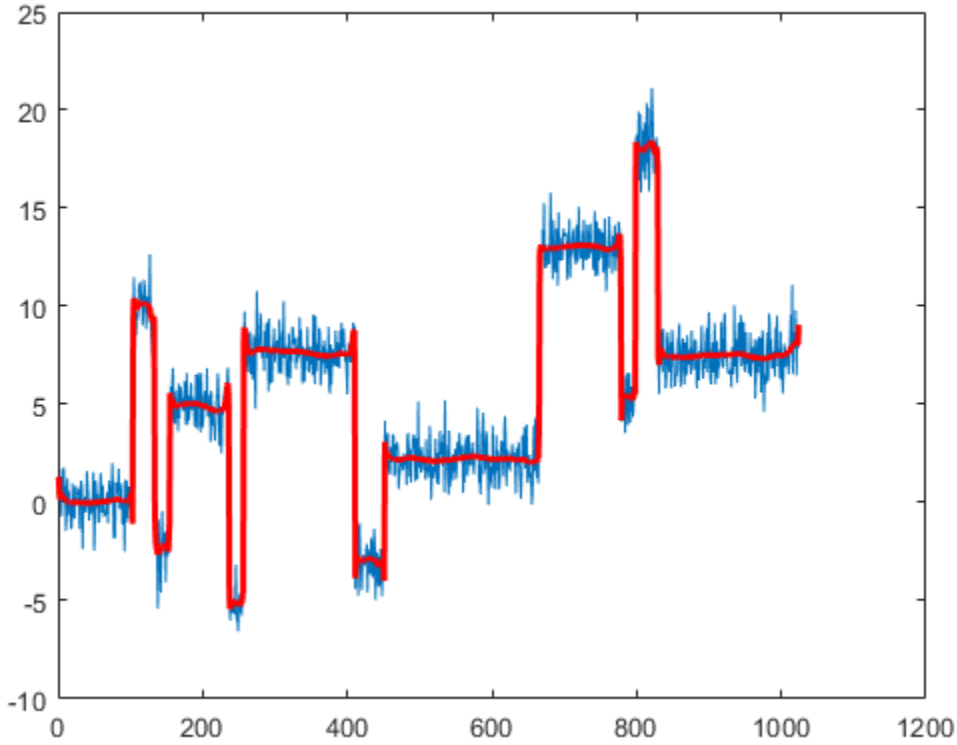


Image Compression— Birgé-Massart Thresholds

This example compresses an image using the Birgé-Massart strategy.

Load the image and add white Gaussian noise.

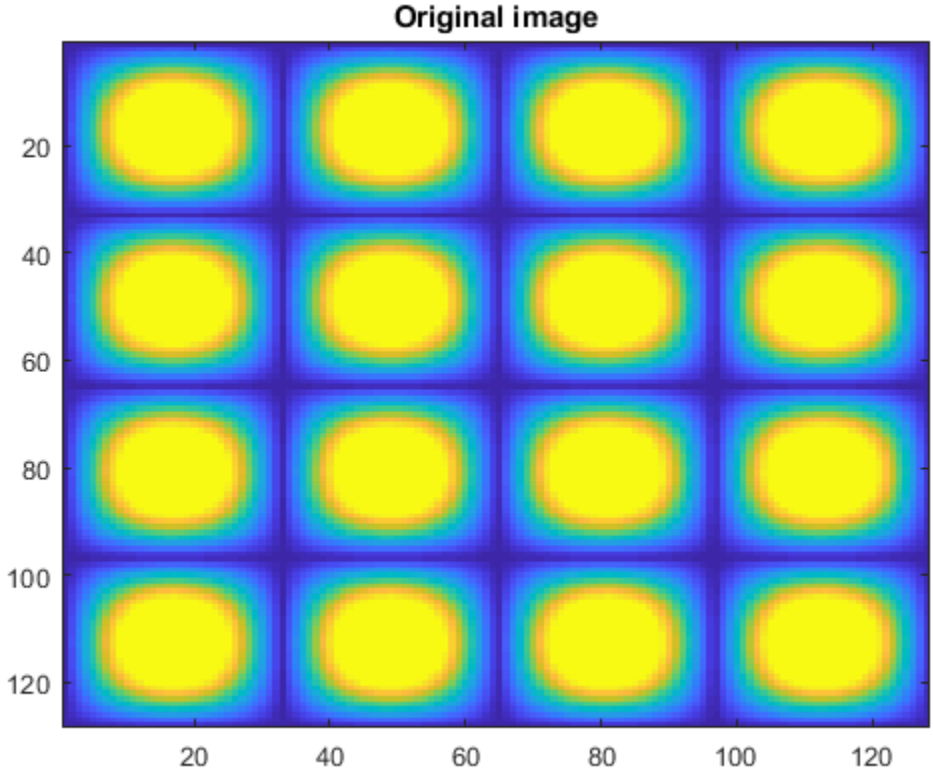
```
load sinsin
x = X+18*randn(size(X));
```

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.

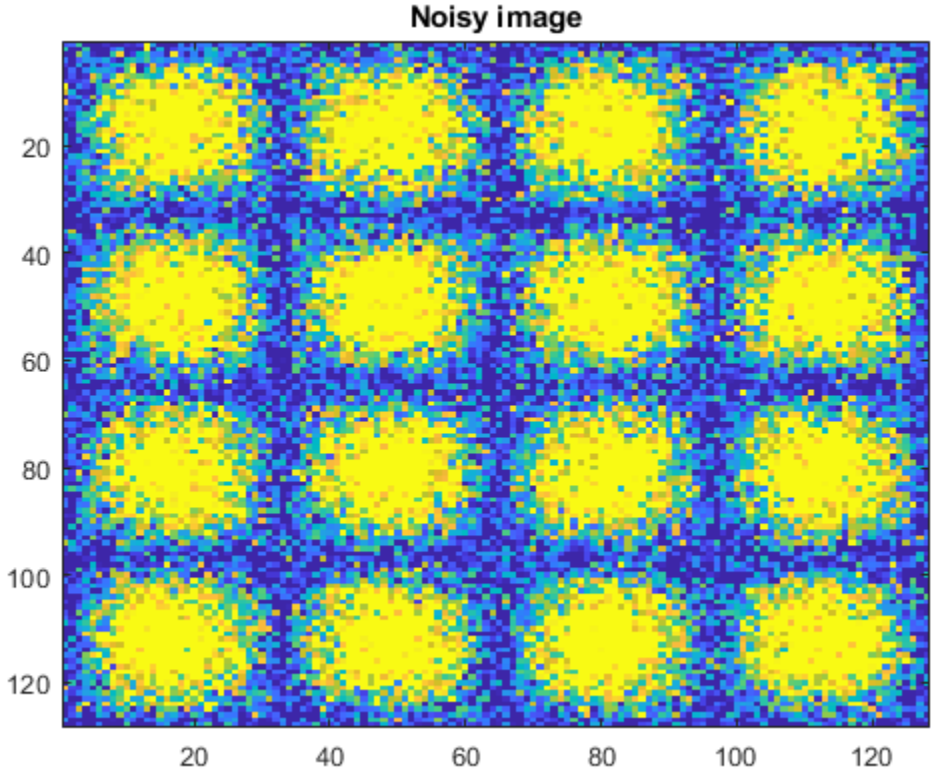
```
[C,L] = wavedec2(x,2,'sym8');
alpha = 2;
THR = wthrmngr('dw2dcompLVL','scarcehi',C,L,alpha);
```

Compress the image and display the result.

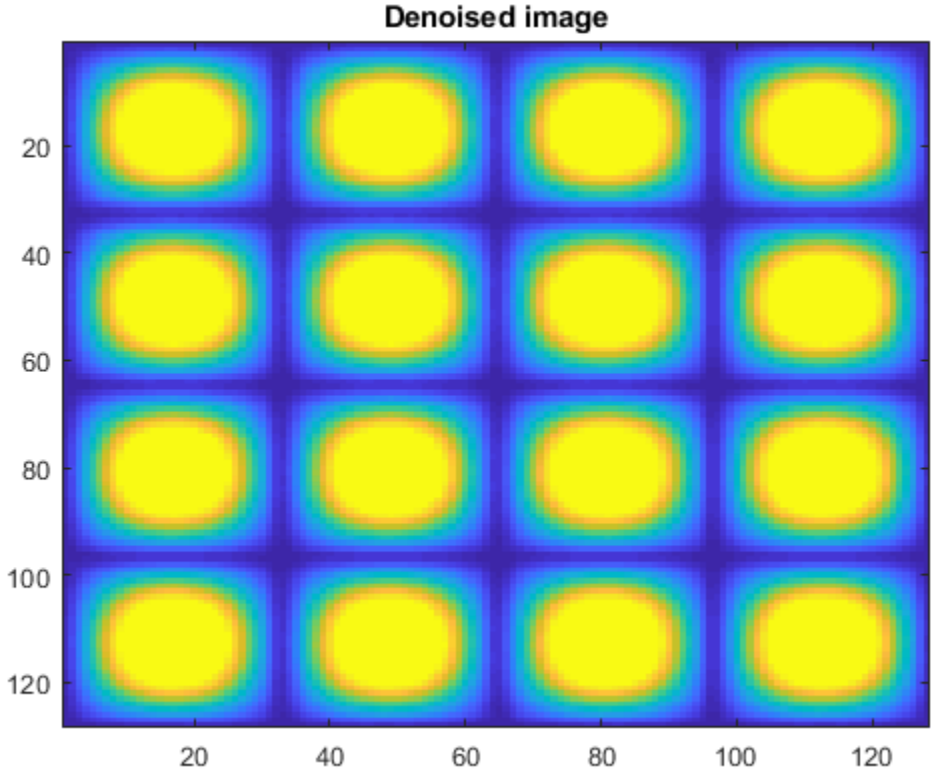
```
 Xd = wdencmp('lvd',X,'sym8',2,THR,'s');
image(X); title('Original image');
```




```
figure;
image(x); title('Noisy image');
```




```
figure;
image(Xd); title('Denoised image');
```



Introduced before R2006a

wtmm

Wavelet transform modulus maxima

Syntax

```
hexp = wtmm(x)
[hexp,tauq] = wtmm(x)
[___] = wtmm(x,'MinRegressionScale',scale)
[hexp,tauq,structfunc] = wtmm(___)

[localhexp,wt,wavscales] = wtmm(x,'ScalingExponent','local')

wtmm(___,'ScalingExponent','local')

[___] = wtmm(___,Name,Value)
```

Description

`hexp = wtmm(x)` returns an estimate of the global Holder exponent, `hexp`, for the real-valued, 1-D input signal, `x`. The global and local Holder exponents are estimated for the linearly-spaced moments of the structure functions from –2 to +2 in 0.1 increments.

`[hexp,tauq] = wtmm(x)` also returns an estimate of the partition function scaling exponents, `tauq`.

`[___] = wtmm(x,'MinRegressionScale',scale)` uses only scales greater than or equal to `scale` to estimate the global Holder exponent. This syntax can include any of the output arguments used in previous syntaxes.

`[hexp,tauq,structfunc] = wtmm(___)` also returns the multiresolution structure functions, `structfunc`, for the global Holder exponent estimate. This syntax can include any of the input arguments used in previous syntaxes.

`[localhexp,wt,wavscales] = wtmm(x,'ScalingExponent','local')` returns the local Holder exponent estimates, the continuous wavelet transform `wt`, and the

scales, `wavscales`, which are used to calculate the CWT used in the `wtmm` algorithm. The wavelet used in the CWT is the second derivative of a Gaussian.

`wtmm(___,'ScalingExponent','local')` with no output arguments plots the wavelet maxima lines in the current figure. Estimates of the local Holder exponents are displayed in a table to the right of the plot.

`[___] = wtmm(___,Name,Value)` returns the Holder exponent and other specified outputs with additional options specified by one or more `Name,Value` pair arguments.

Examples

Global Holder Exponent for Brownian Motion

Estimate the global Holder exponent for Brownian motion. This monofractal signal has a Holder exponent of approximately 0.5.

```
rng(100);
x = cumsum(randn(2^15,1));
hexp = wtmm(x)
```

```
hexp = 0.5010
```

Linearity of Scaling Exponents for Monofractal Signal

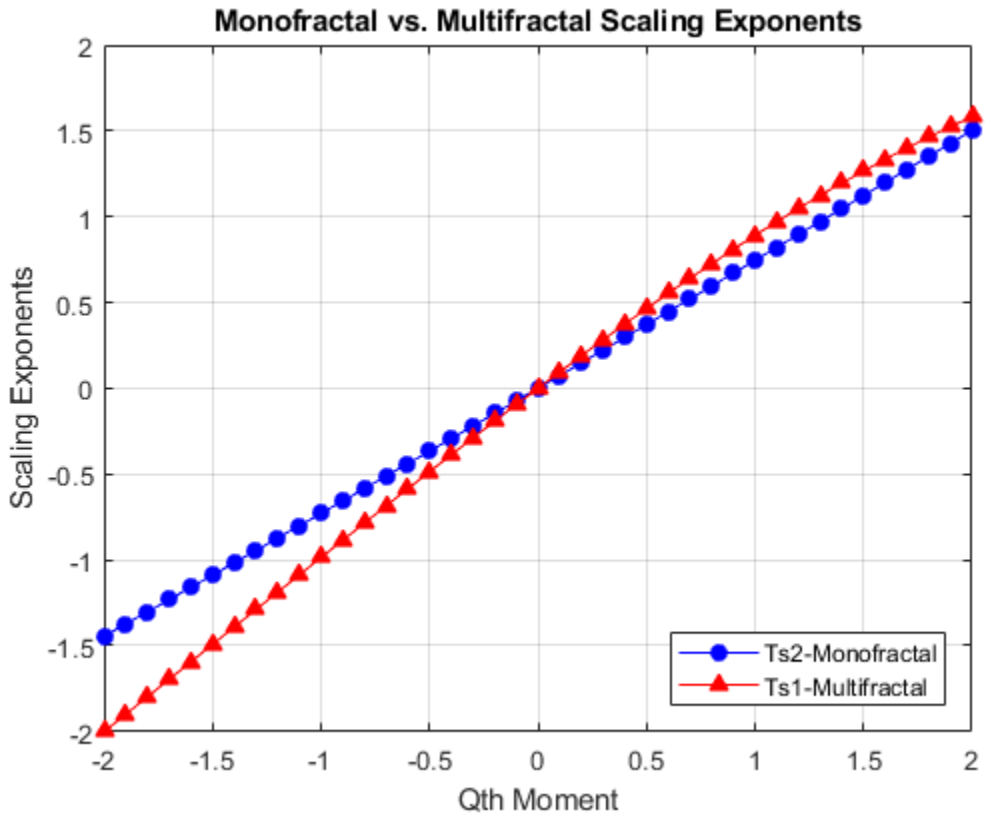
Confirm that for a monofractal signal, the scaling exponents are a linear function of the moments. For multifractal signals, the exponents are a nonlinear function of the moments.

Load a signal that contains two time series, each with 8000 samples. `Ts1` is a multifractal signal and `Ts2` is a monofractal fractional Brownian signal. Obtain the exponents using `wtmm`.

```
load RWdata;
[hexp1,tauq1] = wtmm(Ts1);
[hexp2,tauq2] = wtmm(Ts2);
```

Plot the scaling exponents.

```
expplot = plot(-2:0.1:2,tauq2,'b-o',-2:0.1:2,tauq1,'r-^');
grid on;
expplot(1).MarkerFaceColor = 'b';
expplot(2).MarkerFaceColor = 'r';
legend('Ts2-Monofractal','Ts1-Multifractal','Location','SouthEast');
title('Monofractal vs. Multifractal Scaling Exponents');
xlabel('Qth Moment');
ylabel('Scaling Exponents');
```



`Ts2`, which is the monofractal signal, is a linear function. `Ts1`, the multifractal signal, is not linear.

Structure Function of Wavelet Transform Modulus Maxima

Use the structure function output of `wtmm` to analyze a Brownian motion signal.

Create fractional Brownian motion with a Holder exponenet of 0.6.

```
Brn = wfbm(0.6,2^15);
[hexp,tauq,structfunc] = wtmm(Brn);
```

Compare the calculated Holder exponent with the theoretical value of 0.6.

```
hexp
```

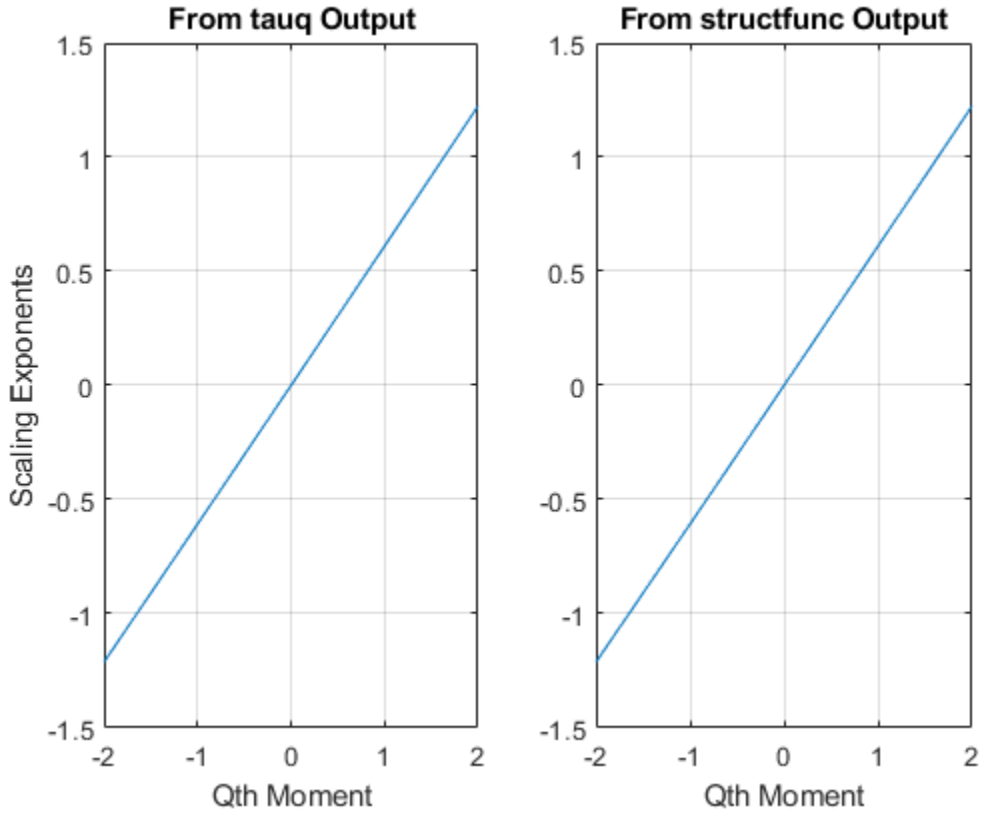
```
hexp = 0.6090
```

Use the data in the `structfunc` output and the `lscov` function to perform the regression on the data.

```
x = ones(length(structfunc.logscales),2);
x(:,2) = structfunc.logscales;
betahat = lscov(x,structfunc.Tq,structfunc.weights);
betahat = betahat(2,:);
```

Plot and compare the scaling exponents from the `tauq` output and from the regressed structure function output.

```
subplot(1,2,1)
plot(-2:.1:2,tauq)
grid on
title('From tauq Output')
xlabel('Qth Moment')
ylabel('Scaling Exponents')

subplot(1,2,2)
plot(-2:.1:2,betahat(1:41))
grid on
title('From structfunc Output')
xlabel('Qth Moment')
```



The plots are the same and show a linear relationship between the moments and the exponents. Therefore, the signal is monofractal. The Holder exponent returned in `hexp` is the slope of this line.

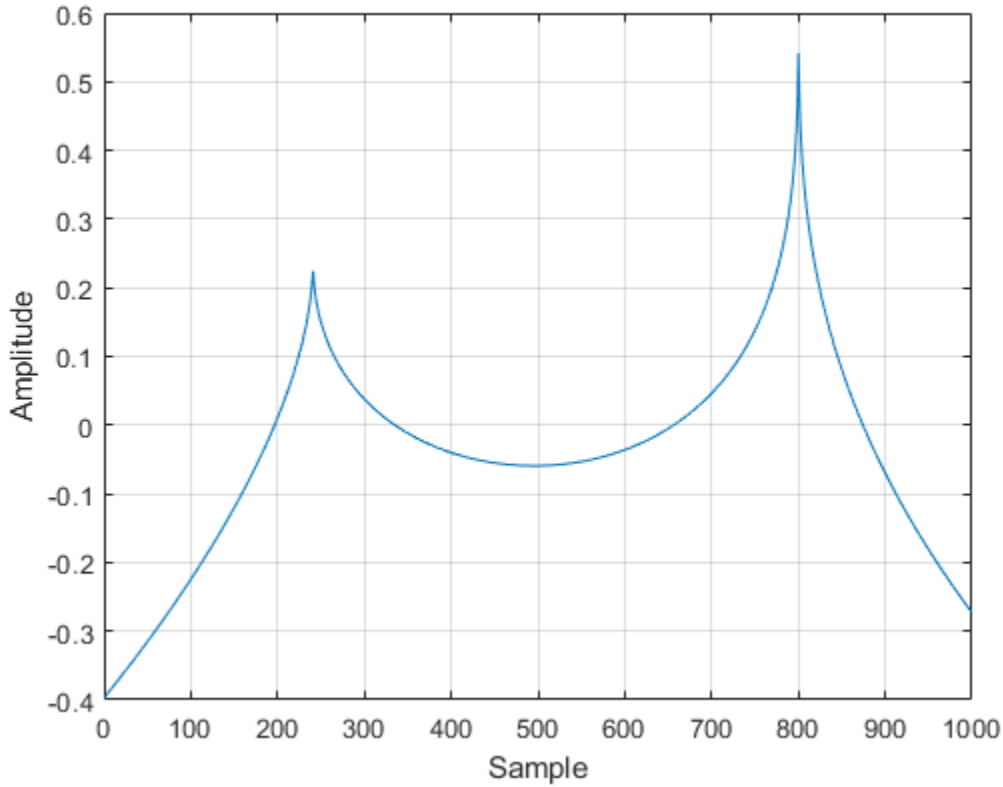
Local Holder Exponents for Cusp Signal and Delta Functions

Using a cusp signal and a signal containing delta functions, generate their local Holder exponents.

Cusp Signal

Load and plot a cusp signal. Note the difference between the two cusps.

```
load cusp;
plot(cusp)
grid on
xlabel('Sample')
ylabel('Amplitude')
```

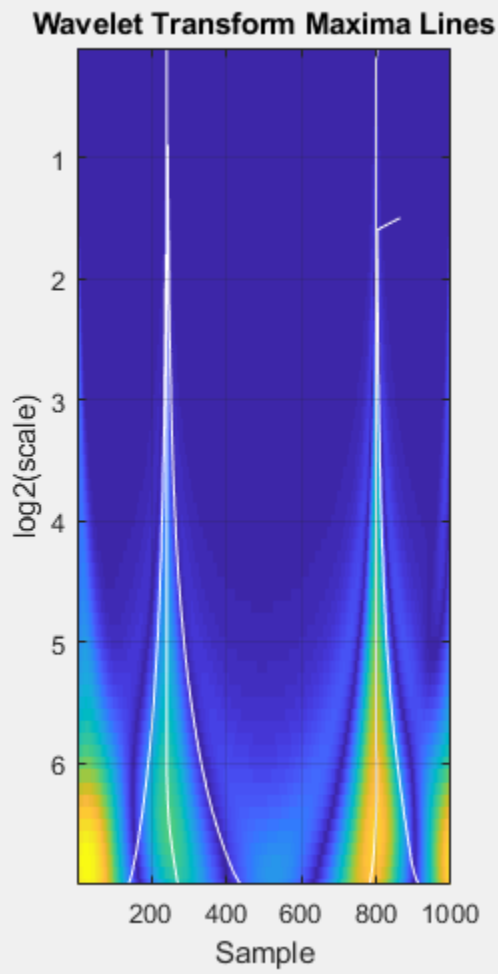


The equation for this cusp signal specifies a Holder exponent of 0.5 at sample 241 and a Holder exponent of 0.3 at sample 803.

```
-0.2*abs(x-241)^0.5 - 0.5*abs(x-803)^0.3 + 0.00346*x + 1.34
```

Obtain the local Holder exponents and plot the modulus maxima.

```
wtmm(cusp,'ScalingExponent','local');
```



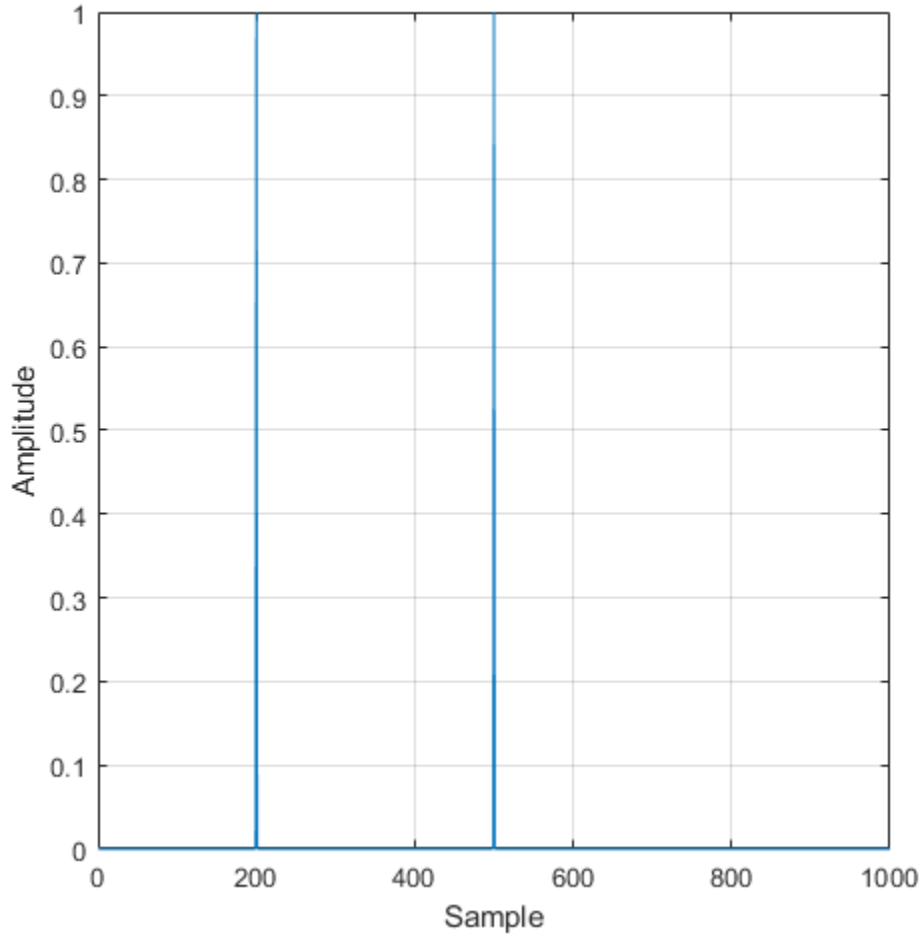
	Sample	Holder Exponent
1	238	0.5301
2	241	0.5020
3	244	0.5293
4	803	0.3006
5	864	0.3136

The Holder exponents at samples 241 and 803 are very close to the values specified in the cusp signal equation. The higher Holder value at sample 241 indicates that the signal at that point is closer to being differentiable than the signal at sample 803, which has a smaller Holder value.

Delta Functions

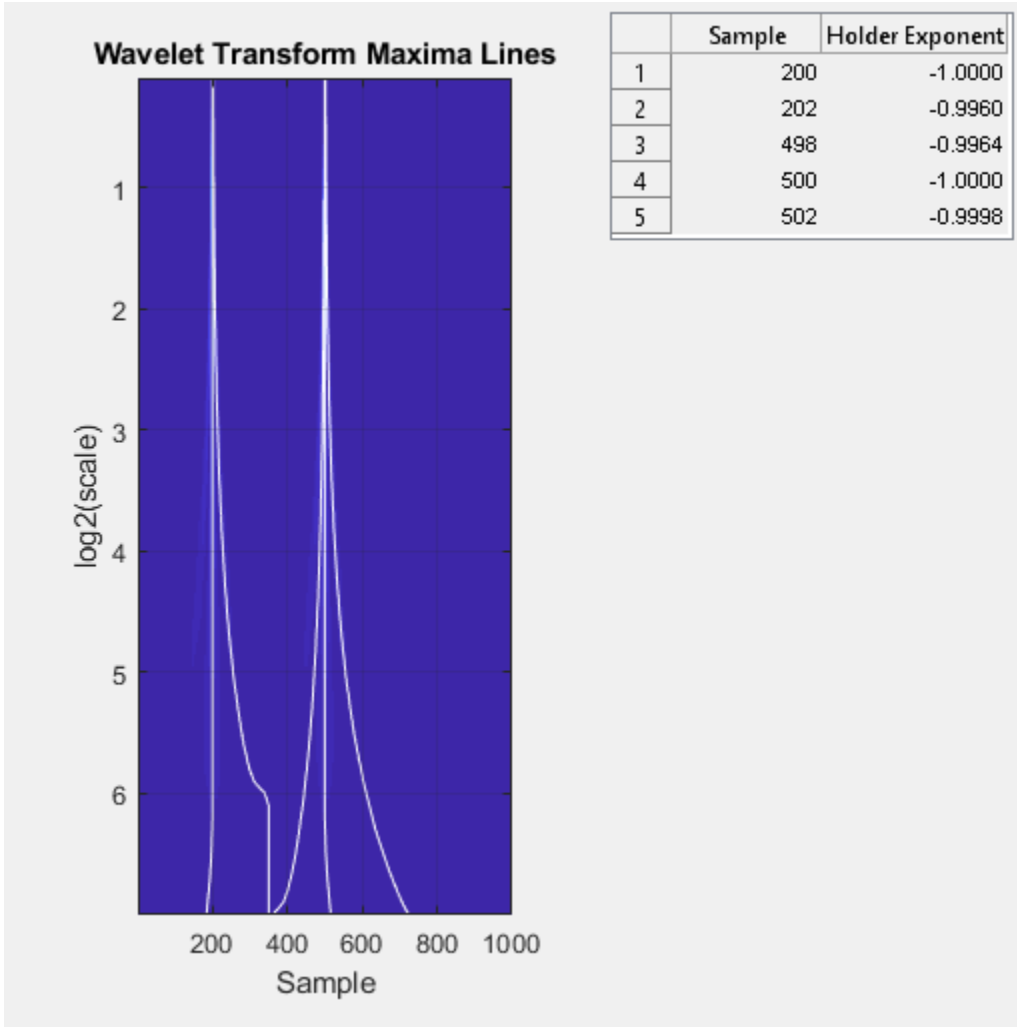
Create and plot two delta functions.

```
x = zeros(1e3,1);
x([200 500]) = 1;
plot(x)
grid on
xlabel('Sample')
ylabel('Amplitude')
```



Obtain the local Holder exponents using the default number of octaves, which in this case is 7. Plot the modulus maxima. A delta function has a Holder exponenent of &ndash1.

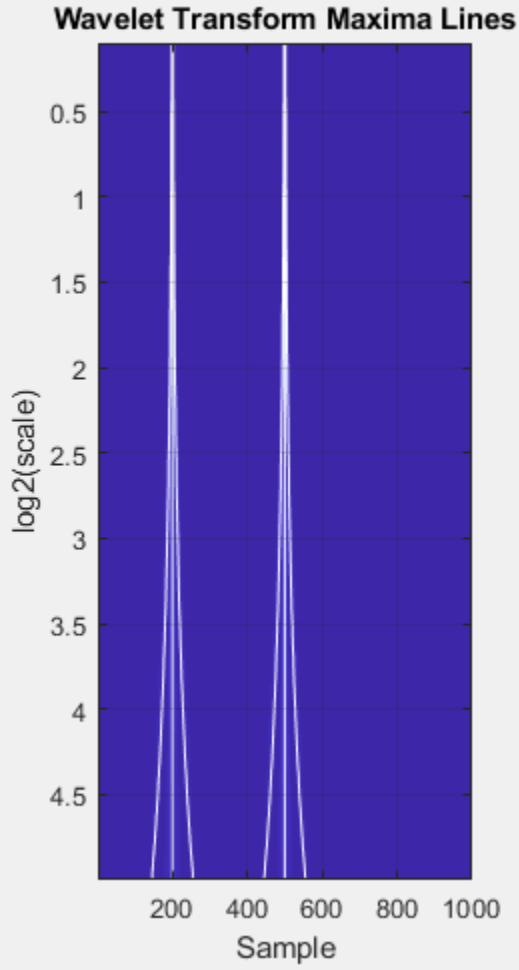
```
wtmm(x,'ScalingExponent','local');
```



	Sample	Holder Exponent
1	200	-1.0000
2	202	-0.9960
3	498	-0.9964
4	500	-1.0000
5	502	-0.9998

Obtain the local Holder exponents using 5 octaves and compare the modulus maxima plot to the plot using the default number of octaves.

```
wtmm(x,'ScalingExponent','local','NumOctaves',5);
```



	Sample	Holder Exponent
1	198	-0.9958
2	200	-1.0000
3	202	-0.9958
4	498	-0.9958
5	500	-1.0000
6	502	-0.9958

Reducing the number of scales provides more separation in frequency and less overlap between the modulus maxima lines of the delta functions.

Input Arguments

`x` — Input signal

real-valued vector

Input signal, specified as a real-valued vector with a minimum of 128 samples. The wavelet transform modulus maxima technique works best for data with 8000 or more samples.

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: `'VoicesPerOctave',18` estimates the global Holder estimate using 18 voices per octave.

`MinRegressionScale` — Minimum scale for regression

`4` (default) | scalar greater than or equal to 4

Minimum scale for regression, specified as the comma-separated pair consisting of `'MinRegressionScale'` and a scalar greater than or equal to 4. This scale is the smallest scale used by the regression. There must be at least two scales with more than 6 CWT maxima. `'MinRegressionScale'` applies only to global Holder exponents.

`VoicesPerOctave` — Number of voices per octave

`10` (default) | even integer from 8 to 32

Number of voices per octave, specified as the comma-separated pair consisting of `'VoicesPerOctave'` and an even integer from 8 to 32. The number of voices per octave and the number of octaves determine the number of scales used in the CWT.

`NumOctaves` — Number of octaves

minimum of `7` and `floor(log2(numel(x)/(3*sqrt(1.1666))))` (default) | integer greater than or equal to 4

Number of octaves, specified as the comma-separated pair consisting of `'NumOctaves'` and an integer. The number of octaves and the number of voices per octave determine the number of scales used in the CWT. The maximum number of octaves is less than or equal to `floor(log2(numel(x)/(3*sqrt(1.1666))))`. The `sqrt(1.1666)` factor is the standard deviation of the second derivative of a Gaussian wavelet. If you specify the number of octaves as greater than the maximum number of octaves, `wtmm` uses the maximum supported number of octaves.

`ScalingExponent` — Type of scaling exponents
`'global'` (default) | `'local'`

Type of scaling exponents, specified as a comma-separated pair consisting of `'ScalingExponent'` and either `'global'` or `'local'`. A global Holder exponent is used for monofractal signals, such as white noise, which are singular everywhere. Global holder exponents give a single estimate of degree of these singularities over the whole signal. Local Holder exponents are useful for signals with cusp singularities.

Output Arguments

`hexp` — Global Holder exponent
real scalar

Global Holder exponent, returned as a real scalar. Holder exponents are useful for identifying singularities, which are locations where a signal is not differentiable. A global Holder exponent uses a single value to estimate the degree of differentiability of all of the singularities of a signal. Signals with a global Holder exponent are monofractal signals.

`tauq` — Scaling exponents
column vector

Scaling exponents, returned as a column vector. The exponents are estimated for the linearly-spaced moments of the structure functions from –2 to +2 in 0.1 increments.

`structfunc` — Multiresolution structure functions
`struct`

Multiresolution structure functions for the global Holder exponent estimates, returned as a `struct`. The structure function for data `x` is defined as

$$S(q,a) = \frac{1}{n_a}\sum_{k=1}^{n_a}\left|T_x(a,k)\right|^q \simeq a^{\zeta(q)},$$

where a is the scale, q is the moment, T_x is the maxima at each scale, n_a is the number of maxima at each scale, and $\zeta(q)$ is the scaling exponent. `structfunc` is a structure array containing the following fields:

- `Tq` — Measurements of the input, `x`, at various scales. `Tq` is a matrix of multiresolution quantities that depend jointly on time and scale. Scaling phenomena in `x` imply a power-law relationship between the moments of `Tq` and the scale. `Tq` is an *Ns*-by-44 matrix, where *Ns* is the number of scales. The first 41 columns of `Tq` contain the scaling exponent estimates for each of the *q*th from -2:0.1:2, by scale. The last three columns correspond to the first-order, second-order, and third-order cumulants, respectively, by scale. For a monofractal signal, cumulants greater than the first cumulant are zero.
- `weights` — Weights used in the regression estimates. The weights correspond to the number of wavelet maxima at each scale. `weights` is an *Ns*-by-1 vector.
- `logscales` — Scales used as predictors in the regression. `logscales` is an *Ns*-by-1 vector with the base-2 logarithm of the scales.

`localhexp` — Local Holder exponent estimates

array of real values

Local Holder exponent estimates, returned as an *M*-by-2 array of real values, where *M* is the number of maxima. If no maxima lines converge to the finest scale in the wavelet transform, then `localhexp` is an empty array. The wavelet transform modulus maxima method (WTMM) identifies cusp-like singularities in a signal. To analyze multifractal signals, use `dwtleader`.

`wt` — Continuous wavelet transform

matrix

Continuous wavelet transform, returned as a matrix of real values. `wt` is a `numel(wavscales)-by-N` matrix where `N` is the length of the input signal `x`.

`wavscales` — Wavelet scales

column vector

Wavelet scales, returned as a column vector of real values. `wavscales` are the scales used to calculate the CWT.

Algorithms

The WTMM algorithm finds singularities in a signal by determining maxima. The algorithm first calculates the continuous wavelet transform using the second derivative of a Gaussian wavelet with 10 voices per octave. The wavelet that meets this criteria is the Mexican hat, or Ricker, wavelet. Then, the algorithm determines the modulus maxima for each scale. The WTMM is intended to be used with large data sets so that enough samples are available to determine maxima accurately.

The definition of the modulus maximum at point x_0 and scale s_0 is

$$|Wf(s_0, x)| < |Wf(s_0, x_0)|$$

where x is either in the right or left neighborhood of x_0. When x is in the opposite neighborhood of x_0, the definition is

$$|Wf(s_0, x)| \leq |Wf(s_0, x_0)|$$

. The algorithm for finding additional maxima repeats for values in that scale. Then, the algorithm continues up through finer scales, checking whether the maxima align between scales. If a maximum converges to the finest scale, it is a true maximum and indicates a singularity at that point.

When each singularity is determined, the algorithm then estimates its Holder exponent. Holder exponents indicate the degree of differentiability for each singularity, which classifies the singularity strength. A Holder exponent less than or equal to 0 indicates a discontinuity at that location. Holder exponents greater than or equal to 1 indicate that the signal is differentiable at that location. Holder values between 0 and 1 indicate continuous, but not differentiable locations. They indicate how close the signal at that sample is to being differentiable. Holder exponents close to 0 indicate signal locations that are less differentiable than locations with exponents closer to 1. The signal is smoother at locations with higher local Holder exponents.

For signals with a few cusp-like singularities and Holder exponents that have large variation, you set the algorithm to return local Holder exponents, which provide individual values for each singularity. For signals with numerous Holder exponents that have relatively small variations, you set the algorithm to return a global Holder exponent. A global Holder exponent applies to the whole signal. For signals with many singularities, you can reduce the number of maxima found by limiting the algorithm to start at or regress to a specific minimum or maximum scale, respectively. For detailed information about the WTMM, see [1] and [3].

References

[1] Mallat, S., and W. L. Hwang. “Singularity Detection and Processing with Wavelets.” *IEEE Transactions on Information Theory*. Vol. 38, No. 2, March 1992, pp. 617–643.

[2] Wendt, H. and P. Abry. “Multifractality Tests Using Bootstrapped Wavelet Leaders.” *IEEE Transactions on. Signal Processing*. Vol. 55, No. 10, 2007, pp. 4811–4820.

[3] Arneodo, A., B. Audit, N. Decoster, J.-F. Muzy, and C. Vaillant. “Wavelet-Based Multifractal Formalism: Application to DNA Sequences, Satellite Images of the Cloud Structure and Stock Market Data.” *The Science of Disasters: Climate Disruptions, Heart Attacks, and Market Crashes*. Bunde, A., J. Kropp, and H. J. Schellnhuber, Eds. 2002, pp. 26–102.

See Also

`dwtleader` | `wfbm`

Introduced in R2016b

wtreemgr

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`

Introduced before R2006a

wvarchg

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 ($0 \le$ `KOPT` $\le$ `K`). The vector `PTS_OPT` contains the corresponding change points.

For $1 \le$ `k` $\le$ `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` << length(`Y`) and $1 \le$ `D` << 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)
```

```
ans =
    'The estimated change points are 181 and 601
     '
```

References

Lavielle, M. (1999), “Detection of multiple changes in a sequence of dependent variables,” *Stoch. Proc. and their Applications*, 83, 2, pp. 79–102.

Introduced before R2006a